2011 IEEE Symposium on Security and Privacy

A Formal Foundation for the Security Features of Physical Functions

Frederik Armknecht*, Roel Maes†, Ahmad-Reza Sadeghi‡, François-Xavier Standaert§, and Christian Wachsmann¶

* *University of Mannheim, Germany*
armknecht@uni-mannheim.de

† *K.U. Leuven, ESAT/COSIC, IBBT, Leuven, Belgium*
roel.maes@esat.kuleuven.be

‡ *TU Darmstadt and Fraunhofer SIT Darmstadt, Germany*
ahmad.sadeghi@trust.cased.de

§ *Université catholique de Louvain, Belgium*
fstandae@uclouvain.be

¶ *TU Darmstadt, Germany*
christian.wachsmann@trust.cased.de

***Abstract*—Physical attacks against cryptographic devices typically take advantage of information leakage (e.g., side-channels attacks) or erroneous computations (e.g., fault injection attacks). Preventing or detecting these attacks has become a challenging task in modern cryptographic research. In this context intrinsic physical properties of integrated circuits, such as Physical(ly) Unclonable Functions (PUFs), can be used to complement classical cryptographic constructions, and to enhance the security of cryptographic devices. PUFs have recently been proposed for various applications, including anti-counterfeiting schemes, key generation algorithms, and in the design of block ciphers. However, currently only rudimentary security models for PUFs exist, limiting the confidence in the security claims of PUF-based security primitives. A useful model should at the same time (i) define the security properties of PUFs abstractly and naturally, allowing to design and formally analyze PUF-based security solutions, and (ii) provide practical quantification tools allowing engineers to evaluate PUF instantiations.**

In this paper, we present a formal foundation for security primitives based on PUFs. Our approach requires as little as possible from the physics and focuses more on the main properties at the heart of most published works on PUFs: robustness (generation of stable answers), unclonability (not provided by algorithmic solutions), and unpredictability. We first formally define these properties and then show that they can be achieved by previously introduced PUF instantiations. We stress that such a consolidating work allows for a meaningful security analysis of security primitives taking advantage of physical properties, becoming increasingly important in the development of the next generation secure information systems.

***Keywords*-Physically Unclonable Function (PUF), Formal Security Model, Robustness, Unclonability, Unpredictability**

I. Introduction

Research on Physically Unclonable Functions (PUFs) was initiated by the work of Pappu [1], [2] and aims to constructively exploit the (random) physical variations that can be found in various objects. The core concept put forward by PUFs is the use of unique physical properties in order to identify a device rather than assigning an arbitrary identity to it upon creation, like a barcode, an electronic product code, or a cryptographic key. In fact, the underlying principles of PUFs were known and applied much earlier, mainly in different attempts to design anti-counterfeiting mechanisms (see, e.g., [3], [4], [5], [6]). Following these seminal contributions, it was quickly realized that PUFs can be used beyond identification and anti-counterfeiting applications for which they were initially designed. For example, PUFs can be used to generate and securely store strong cryptographic keys (see, e.g., [7], [8]). They can even be an integral part of cryptographic primitives, emerging hardware-entangled cryptography [9], where security is based on the physical properties of PUFs instead of mathematical problems. Today, there are already several PUF-based security products aimed for the market (e.g., RFID, IP-protection, anti-counterfeiting solutions) [10], [11]. More generally, the exploitation of physics is an appealing solution in the evolutionary perspective of information technologies for at least two reasons: first, computing and communication devices tend to become smaller and smaller, and their deep integration leads to the apparition of many physical effects that were typically unnoticed in former technologies (including manufacturing variability or quantum effects). In this context, it appears natural to exploit the physics, rather than systematically struggling against it in order to minimize parasitic effects. Second, the increasing development of distributed (e.g., Cloud) computing and the growing interconnection of billions of objects within the emerging "Internet of Things" also creates an important trust and security challenge. In this context, the ability to equip each object or computing device with a unique identity, which can be used as a trust

1081-6011/11 $26.00 © 2011 IEEE
DOI 10.1109/SP.2011.10

anchor in higher level security architectures, would be of considerable help, and is typically what could be provided by PUFs at low manufacturing costs [12].

Quite naturally, exploiting physical properties in security systems also raises important formalization problems. The core issues are to determine which properties of physical objects need to be defined, and to find efficient ways to guarantee them in practice. In other words, one of the main challenges for using PUFs in future security applications is to properly integrate them in complex systems, where some of their physical properties can be a real advantage compared to purely algorithmic solutions. In this respect, useful and reasonable security definitions of PUFs should be both (i) sound for cryptographers, in order to allow the analysis of PUF-based cryptographic systems, and (ii) empirically verifiable by engineers, such that the security levels guaranteed by the physics can be evaluated (or at least be lower bounded). These challenges give a strong motivation for introducing a security model for PUFs that unifies previous formalization attempts and at the same time satisfies (i) and (ii). For this purpose, our rationale is based on the following observations:

1) It is generally difficult to argue about the physical properties of an object, e.g., compared to classical cryptography, where explicit security parameters can do an excellent job in this respect.
2) It is generally unknown if the properties expected for PUFs, such as unpredictability or unclonability, relate to any exponentially hard problem. While this situation can be unsatisfying from a theoretical point of view, it is in fact similar to the situation of many primitives used in applied cryptography. For example, there is no exponential hardness problem on which current block ciphers are based, e.g., the AES is only *expected* to provide a security level of roughly 2^{128} operations.
3) The interface of PUFs to the outside world usually does not directly access the physics but uses some mathematical post-processing of the PUF outputs (which we denote as extractor algorithm).

As a consequence of (1) and (2), our focus is to start with a set of three basic properties allowing the design of hybrid systems combining PUFs with classical algorithms, and to formalize PUFs by security notions similar to those of, e.g., block ciphers, with constant security levels that can be properly quantified by engineers in a physical counterpart to cryptanalysis. First, PUFs must be robust, i.e., able to provide stable outputs, since non-robust PUFs would significantly harm the efficiency of the underlying system. Robustness essentially captures the resilience of a PUF system to noisy measurements. Next, we investigate formal definitions of unclonability, which is a central property of PUFs that cannot be guaranteed by purely algorithmic solutions. Having improved arguments of unclonability, quantified within a sound model, would better motivate the use of PUFs in many security applications. Third, we propose a definition of unpredictability of PUF outputs, which is the weakest cryptographic property that could be expected from PUFs. While unpredictability could also be guaranteed by algorithmic means, we believe that the inherent physical randomness provided by PUFs is worth to be exploited as well. As a consequence of (3), we finally propose to define these cryptographic properties as function of the extractor algorithm instead of a plain PUF. By applying our framework and definitions to PUF instances, we show that our abstractions are useful and properly capture the physical properties for security purposes.

The rest of the paper is structured as follows: in Section II we analyze previous approaches to the formalization of the security properties of physical functions and point out their weaknesses and drawbacks. Then, we present our general framework for the formalization of physical functions in Section III and define robustness in Section IV, physical unclonability in Section V, and unpredictability in Section VI. Finally, we conclude in Section VII.

II. Related Work

This section gives a comprehensive but concise overview of different constructions of physically unclonable functions and the attempts to formalize their properties. A more extensive discussion on all known types of PUFs and their defining properties is provided by Maes et al. [13].

A. A History of PUFs

The initial idea of Pappu [1], [2] was to use optically transparent tokens, which are randomly doped with light scattering particles, as unique and practically *unclonable* identifiers, e.g., as an alternative to smart cards. An incident laser beam on the token creates an unpredictable speckle pattern, which is highly sensitive to the random arrangement of the scattering particles and to the relative orientation of the token to the laser beam. The unclonable pattern is captured and processed into a unique identifier. This construction became known as the optical PUF and is well studied. Tuyls et al. [14] showed bounds on the difficulty of learning the optical PUF in an attempt to make an algorithmic model, and Pappu already showed that the tokens are tamper-evident, i.e., an invasive attack on the token, e.g., in order to learn the positions and sizes of the scatters, significantly changes the arising speckle patterns and hence can be detected. Despite these very interesting properties, the optical PUF has a number of practical drawbacks: the high sensitivity on the laser orientation poses very high mechanical constraints on the construction and its strenuous readout setup, which limits its use in cryptographic and security applications (e.g., identification schemes).

Following the introduction of the optical PUF, many attempts to construct more practical PUFs were introduced.

The general trend is to embed PUFs in silicon integrated circuits (ICs). The rationale behind this is that the PUF's outputs or *responses* can be used directly on the chip, particularly in combination with a secret key generation algorithm, to enable more elaborate security applications. In the *coating PUF* [15], a silicon chip is covered with a randomized dielectric coating that affects the exact capacitance values of underlying metal sensors, leading again to unique and practically unclonable PUF responses. It was shown that this construction provides tamper-evidence even for other components of the chip containing the PUF, which is a very desirable property for security-critical integrated circuits. However, the amount of unique information provided by a coating PUF is limited by the number of sensors that can be placed on the chip. Moreover, providing randomized coating is a highly specialized and thus an additional and costly step in the already complex manufacturing flow of an IC.

Further integration of PUFs on ICs was proposed by Gassend et al. [16]. The idea is to exploit the *intrinsic* randomness introduced during the fabrication of the chip. Typical production flows of silicon ICs suffer from manufacturing variability, i.e., uncontrollable processes that typically have an effect at the submicron structures of the IC, which causes every IC to behave slightly different. Gassend et al. [16], [17] first showed that unique effects of manufacturing variability can be detected by observing the frequency of identically designed asynchronous oscillating loops. More elaborate ring-oscillator based constructions were proposed later and their statistical properties were extensively studied [17], [18], [19]. Similarly to these *ring oscillator PUFs*, it was shown in [20] that manufacturing variability also affects the outcome of a race condition between two identical delay lines, the so-called *arbiter PUF*. By using a challengeable delay circuit, the number of responses of a single delay-based PUF can be made exponentially large in the dimensions of the PUF. However, it was realized early that a relatively small number of challenge-response pairs enables to learn these PUFs to such detail that unknown responses can be predicted with great accuracy [20], [21], [22]. A number of attempts to harden the learning of delay-based PUFs [23], e.g., through the use of non-linear elements, have not been able to completely prevent sophisticated learning algorithms from predicting responses with non-negligible advantage [24].

Another approach towards using manufacturing variability as a source for on-chip embedded PUFs makes use of bi-stable memory cells. When such a cell converges from an unstable to one of both stable states, it will often have a preference of one state over the other. This effect, which was first observed in the power-up behavior of SRAM cells by Guajardo et al. [25] and Holcomb et al. [26], originates from the mismatch of the symmetry of the memory cell, which is again caused by silicon manufacturing variability. This construction has been called *SRAM PUF*. Similar behavior has been observed in other bi-stable memory structures such as flip-flops [27], [28] and latches [29], [30]. The amount of unique responses of a memory-based PUF is limited by the number of its memory cells but the information density is much higher compared to, e.g., a coating PUF. Opposed to most delay-based PUFs, it is fairly safe to assume that each response of a memory PUF, originating from an individual element, is independent of the others and hence unlearnable.

The latter two PUF proposals, based on silicon delay elements and bi-stable memory structures, have also been labelled *intrinsic PUFs*. This is due to the fact that they exploit the *intrinsic* device uniqueness caused by manufacturing variability, as opposed to, e.g., optical PUFs and coating PUFs, where randomness is explicitly introduced during the manufacturing process of the PUF. Additional properties of intrinsic PUFs are that the whole PUF, including the component that measures the PUF responses, is embedded into the device and can be built using the standard device manufacturing flow without the need for custom processes. Intrinsic PUFs are particularly well suited for applications where physics is used to generate secret data (e.g., cryptographic keys), since the PUF response must not leave the device.

Extrinsic (e.g., optical) PUFs have advantages in certain scenarios as well, e.g., in anti-counterfeiting applications, where the ability to directly observe the PUF answers (and the measurement process) can increase the confidence of the verifier. Notable examples of such extrinsic PUFs are based on the uniqueness of the structure of regular paper [31], [32].

B. Former Formal PUF Modelling Attempts

PUF behavior can be explained rather intuitively from physical processes, however, when PUFs should be used for security purposes, a formal model is often required in order to bootstrap mathematical reductions for higher level security claims. Throughout literature, a number of attempts towards formalizing a PUF definition exist. We briefly introduce them and point out why none of them captures the full spectrum of proposed PUFs and their properties, either by being too restrictive, i.e., excluding certain PUFs, or by being too ad-hoc, i.e., listing perceived and even assumed properties of certain PUFs instead of providing a more general model. A similar overview and discussion has been given by Rührmair et al. [33]. However, we do not completely follow all their arguments and moreover point out why the new models they propose are still insufficient.

Pappu [1] describes the optical PUF as a *physical one-way function* (POWF), taking a laser orientation as challenge and producing a speckle pattern as response. The first part of the definition of a POWF states that it is a deterministic physical interaction that is evaluable in constant time but cannot be inverted by a probabilistic polynomial time adversary with a non-negligible probability. The second part of the definition focusses on the unclonability of the POWF: both simulating

a response and physically cloning the POWF should be hard. The POWF definition was the first formal attempt of defining a PUF and solely reflects the optical PUF, which at that time was the only known PUF. As other PUFs were introduced shortly after, it became clear that this definition was too stringent, in particular regarding the one-wayness assumption. Whilst the optical PUF has a very large range of possible outputs (speckle patterns), many of the discussed intrinsic PUFs on ICs have a small output length of only one or a few bits. In the latter case, one-wayness does not hold any longer, since inverting such a PUF with non-negligible advantage becomes trivial. It is also noteworthy to mention that, as also pointed out by Rührmair et al. [33], for many security applications one-wayness of the PUF is not a necessary condition. A final issue with the POWF definition is that it lacks any notion of *noise*, in fact it even describes a POWF as a deterministic interaction. This is contradicted by the fact that virtually all PUF proposals, including the optical PUF, produce *noisy* responses due to uncontrollable physical effects during the response measurement.

With the introduction of delay-based intrinsic PUFs, Gassend et al. [16], propose the definition of *physical random functions* to describe PUFs. In brief, a physical random function is defined as a function embodied by a physical device, which is *easy to evaluate* but *hard to predict* from a polynomial number of challenge-response pairs (CRPs). Note that this definition replaces the very stringent one-wayness assumption from POWFs by a more relaxed unpredictability assumption. However, it was quickly realized that due to the linearity of the internal PUF delays, simple delay-based PUFs and in particular arbiter PUFs are relatively easy to model from a limited number of CRPs [20]. Rührmair et al. [24] show that even more elaborate delay-based PUF constructions can be modelled using advanced machine learning techniques. Once such a model is made, prediction of unknown responses becomes trivial. It is clear that for these PUFs the level of unpredictability is reduced significantly when an adversary learns many CRPs. Also note that the later introduced memory-based intrinsic PUFs only possess a polynomial number of CRPs and hence do not classify as physical random functions since they can be easily modelled through exhaustive readout. Moreover, the definition of physical random functions also does not capture the possibility of noisy responses.

With the introduction of memory-based intrinsic PUFs, Guajardo et al. [25] further refine the formal specification of PUFs. They describe PUFs as inherently unclonable physical systems with a challenge-response behavior. It is *assumed* that (i) different responses are independent of each other, (ii) it is difficult to come up with unknown responses, and (iii) tampering with the PUF substantially changes its challenge-response behavior. For the first time, it is made explicit that PUF responses are observed as noisy measurements. This definition also comes with a division in *strong* and *weak* PUFs, depending on how many CRPs an adversary is allowed to obtain in order to model the PUF. If the number is exponentially large in some security parameter, the PUF is called a strong PUF, otherwise the PUF is called weak. It can be argued that some of the assumptions made in this description do not have a solid experimental basis, in particular regarding the tamper-evidence assumption, which has not been tested in practice for any of the intrinsic PUF proposals. Also, strong PUFs seem to be difficult to characterize in general, as the idea of a security parameter is specific to each PUF instance, and no practical procedure is proposed to exhibit the required exponential behavior in practice. Moreover, the terms weak and strong PUFs are confusing w.r.t. the classical cryptographic notions of weak and strong pseudorandom functions, where weak and strong do not refer to an amount of CRPs but to the ability of the adversary to select his queries adaptively.

Following a similar analysis as above, Rührmaier et al. [33] proposed yet a further refinement of a formal PUF definition. They keep the distinction between strong and weak PUFs by Guajardo et al. [25] and build upon these definitions. Both strong and weak PUFs are now defined by means of a security game with an adversary. Weak PUFs are called *obfuscating PUFs* and are basically considered as *physically obfuscated keys*. The main statement in the definition of obfuscated PUFs is that an adversary cannot learn the key after having had access to the PUF for a limited amount of time. Strong PUFs are defined similarly, but here the adversary needs to come up with the response to a randomly chosen challenge after having had access to the PUF and a *PUF oracle* for some limited amount of time. Some issues are again left unresolved in this formalization: first, despite building upon the work by Guajardo et al. [25], responses are not considered to be noisy. Next, the use of a PUF oracle in the definition of a strong PUF seems questionable. It is argued that this oracle is introduced to circumvent any kind of practical access restriction to the PUF. However, if a PUF-based system is secured against any attacks possible "by the current state of technology", the access to such an oracle provides an unrealistic advantage to the adversary, which weakens the proposed definition.

Finally, Armknecht et al. [9] introduce another PUF model that, as opposed to most of the previous proposals, was not described following the introduction of a new PUF construction but rather in an attempt to use existing PUFs as cryptographic building blocks in a block cipher, i.e., in an application of *hardware entangled cryptography*. For this goal, the previously discussed definitions proved to be insufficient. Armknecht et al. [9] make a distinction between algorithmic and physical properties of a PUF. From the algorithmic side, a PUF is said to be a *noisy function* for which the distribution of responses is indistinguishable from a random distribution with a certain amount of min-entropy. From the physical side, a PUF is assumed to be

physically unclonable and tamper-evident, i.e., any physical attack against the PUF will irreversibly and randomly change its challenge-response behavior. We already pointed out the lack of experimental evidence for tamper-evidence of intrinsic PUFs in practice, and the same argument applies to this definition. Contrarily to most of the previous definitions, here PUFs are explicitly defined as noisy functions, where the noise error of the output stays within a certain bound.

As pointed out in this section, existing approaches to model the security properties of PUFs have several weaknesses and drawbacks. In the following, we formalize the security features of physical functions in accordance to existing literature on PUFs and propose a general security framework for physical functions, which modularly captures the most important properties of PUFs and allows for a meaningful security analysis of PUF-based constructions.

III. Framework

A. Background and Rationale

In this section, we explain the components and procedures relevant for deploying physical functions (PF). Observe that we focus not only on PUFs but on physical functions in general, where unclonability is only one possible security property. Before we provide formal definitions, we give an overview of our framework, which is depicted in Figure 1 that shows all components necessary for creating, evaluating and post-processing the output of a physical function. In the following, we explain each of these components separately.

1) Physical Function: A *Physical Function* (PF) consists of a *physical component* p that can be stimulated with some *challenge signal* $\tilde{x}$, which makes p respond with a corresponding *response signal* $\tilde{y}$. In addition to the physical component p, a PF contains an *evaluation procedure* Eval that, on input a digital representation x of $\tilde{x}$, stimulates the physical component with $\tilde{x}$ and obtains the resulting response signal $\tilde{y}$. Finally, Eval returns a digital representation y of $\tilde{y}$. The challenge-response behavior of a PF heavily relies on the properties of the physical component p, uncontrollable random noise (e.g., thermal noise and measurement uncertainties), and an evaluation parameter α_{PF} (e.g., a quantization factor) chosen by the PF manufacturer. Observe that the same physical component p can yield completely different PFs if combined with different evaluation procedures. This fact should be representable by a comprehensive model.

2) Extraction Algorithm: Although the notion of a physical function suggests differently, a PF is not a function in the classical sense. The main difference is that, when challenged with the same challenge x twice, a PF may produce different responses y. This is because the challenge-response behavior of a PF heavily relies on the physical properties of its physical component p, which is subject to uncontrollable random noise. The effects of noise can be removed up to a certain threshold by an *extraction algorithm* Extract, which maps slightly different responses y to the same challenge x to a unique output z according to some *extraction parameter* α_{EX}, which is typically chosen by the PF manufacturer or the PF user (i.e., the entity that integrates the PUF into a higher-level protocol or algorithm). We assume that the extraction parameter specifies both the deployed extraction algorithm and all possible parameters (e.g., number of output bits) of the chosen Extract algorithm. The Extract algorithm can be executed in two different modes: *setup* and *reconstruction*. If a challenge x is requested for the first time, setup mode is used to generate an output z and some appropriate *helper data* h'. Later, when challenge x is requested again together with helper data $h = h'$, the reconstruction mode is used to recreate z. The purpose of the helper data h' is to twofold [34]: (i) h' supports the extraction algorithm Extract in recreating the same output z for a challenge x, and (ii) h' allows to bind given values (e.g., cryptographic keys) to a PUF.

3) Physical Function System: As explained above, a PF is usually coupled with an appropriate extraction algorithm. Indeed, in a typical application scenario, a user will be only aware of the challenges given to the PF and the output returned by the extraction algorithm. Furthermore, for almost all relevant security notions, both the deployed PF and the extraction algorithm determine whether a security property is given or not. Therefore, it is a natural choice to abstract away the physical function PF and the extraction algorithm Extract and consider their combination as one single building block. We term this a *Physical Function System* (PF system). Consequently, we will mostly refer to PF systems only and refer to the underlying PF or extraction algorithm only if necessary.

4) Creation Process: The creation of the physical component p of a physical function PF is the result of a *creation process* Create, usually performed by the manufacturer of PF. The result of this process depends on a creation parameter α_{CR} that is chosen by the PF manufacturer and some uncontrollable production variability.

5) Physical Function Infrastructure: We call the combination of all components described in (1) to (4) a *Physical Function Infrastructure* (PFI). We stress that within a PFI the creation, evaluation and extraction parameters are *fixed*. Furthermore, we assume that these parameters uniquely specify the deployed procedures, e.g., α_{PF} defines the full details of the Eval procedure.

B. Formalization

In the following, we formalize the concepts described above. We start by introducing our notation.

1) Notation: Let A be a probabilistic procedure. Note that with procedure we denote a probabilistic polynomial time algorithm that may involve some physical process (e.g., the evaluation of a PF). Then $y \leftarrow \mathsf{A}(x)$ refers to the event that on input x, procedure A assigns its output to variable y.

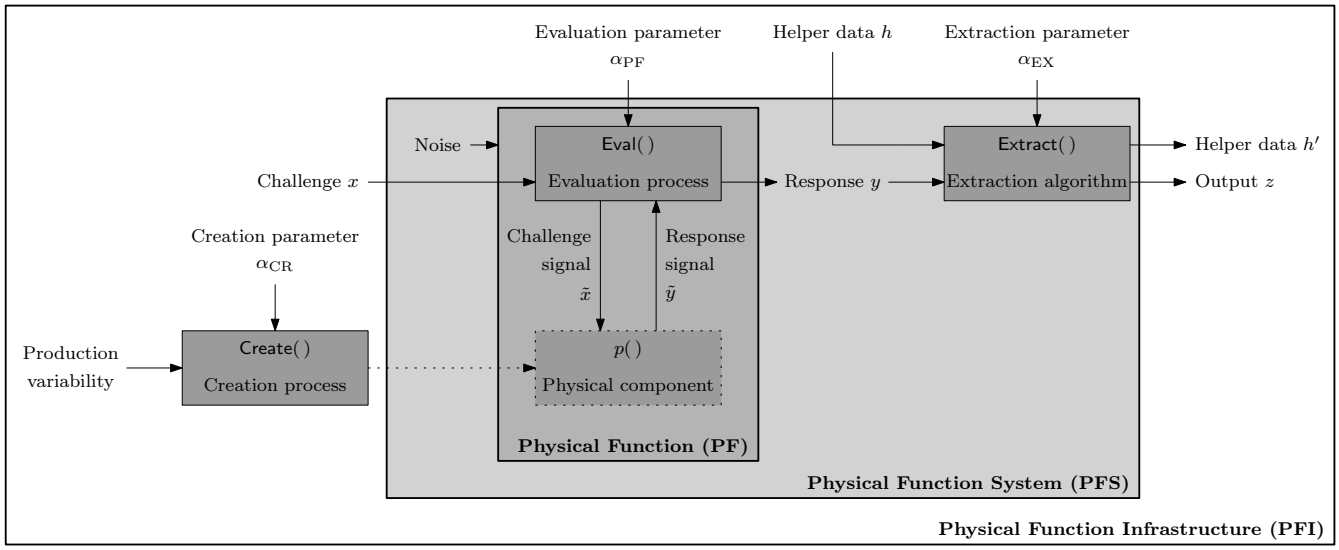


Figure 1. Generic framework for physical functions.

The term $[\mathsf{A}(x)]$ denotes the set of all possible outputs of A on input x that appear with a probability larger than 0. Let E be some event (e.g., the result of a security experiment), then $\Pr[E]$ denotes the probability that E occurs. Moreover, for a set $\mathcal{S}$, the expression $s \xleftarrow{\$} \mathcal{S}$ refers to the event that s has been randomly sampled from $\mathcal{S}$. We denote with ϵ the empty string, and with $\mathrm{HW}(x)$ the Hamming weight of a bitstring x, i.e., the number of non-zero bits of x.

2) Definitions: We now formally define the components and procedures within a physical function infrastructure as explained in Section III-A.

Definition 1 (Physical Function): A *physical function* PF is a probabilistic procedure

$$\mathsf{PF}_{p,\alpha_{\mathrm{PF}}} : \mathcal{X} \to \mathcal{Y} \tag{1}$$

where $\mathcal{X}$ denotes the set of challenges and $\mathcal{Y}$ the set of responses. Internally, a PF is the combination of a physical component p and an evaluation procedure Eval, i.e.,

$$y \leftarrow \mathsf{PF}_{p,\alpha_{\mathrm{PF}}}(x) = \mathsf{Eval}_p(\alpha_{\mathrm{PF}}, x) \tag{2}$$

Usually, the specification of p and α_{PF} will be discarded in our notation, that is we simply write PF instead of $\mathsf{PF}_{p,\alpha_{\mathrm{PF}}}$.

Definition 2 (Physical Function System): A *physical function system* PFS is a probabilistic procedure

$$\mathsf{PFS}_{p,\alpha_{\mathrm{PF}},\alpha_{\mathrm{EX}}} : \mathcal{X} \times (\mathcal{H} \cup \{\epsilon\}) \to \mathcal{Z} \times \mathcal{H}, \tag{3}$$

where $\mathcal{X}$ is the set of challenges, $\mathcal{H}$ the set of helper data values, ϵ the empty string, and $\mathcal{Z}$ the set of outputs.

Internally, a PF system is the combination of a physical function $\mathsf{PF} = \mathsf{PF}_{p,\alpha_{\mathrm{PF}}}$ (Definition 1) and an extraction algorithm $\mathsf{Extract}$ (see Section III-A), i.e.,

$$\begin{aligned}(z, h') \leftarrow &\mathsf{PFS}_{p,\alpha_{\mathrm{PF}},\alpha_{\mathrm{EX}}}(x, h) \\ &= \mathsf{Extract}_{\alpha_{\mathrm{EX}}}(\mathsf{PF}_{p,\alpha_{\mathrm{PF}}}(x), h)\end{aligned} \tag{4}$$

Hereby, we require that if $h \neq \epsilon$, then $h' = h$. Only in case $h = \epsilon$, a new helper data h' is generated for x. in the following, we omit the internal components and abbreviate $\mathsf{PFS} = \mathsf{PFS}_{p,\alpha_{\mathrm{PF}},\alpha_{\mathrm{EX}}}$.

Note that $h = \epsilon$ means that $\mathsf{Extract}$ should be executed in setup mode to generate a new helper data h w.r.t. challenge x. In case $h \neq \epsilon$, $\mathsf{Extract}$ should be executed in reconstruction mode to recreate output z associated with challenge x and helper data h. Note that, for the sake of consistent notation, in this case we require $h' = h$ to be returned by $\mathsf{Extract}$.

Definition 3 (Creation Process): A creation process Create is a probabilistic procedure that, on input of a creation parameter α_{CR}, produces a physical component p (Definition 1).

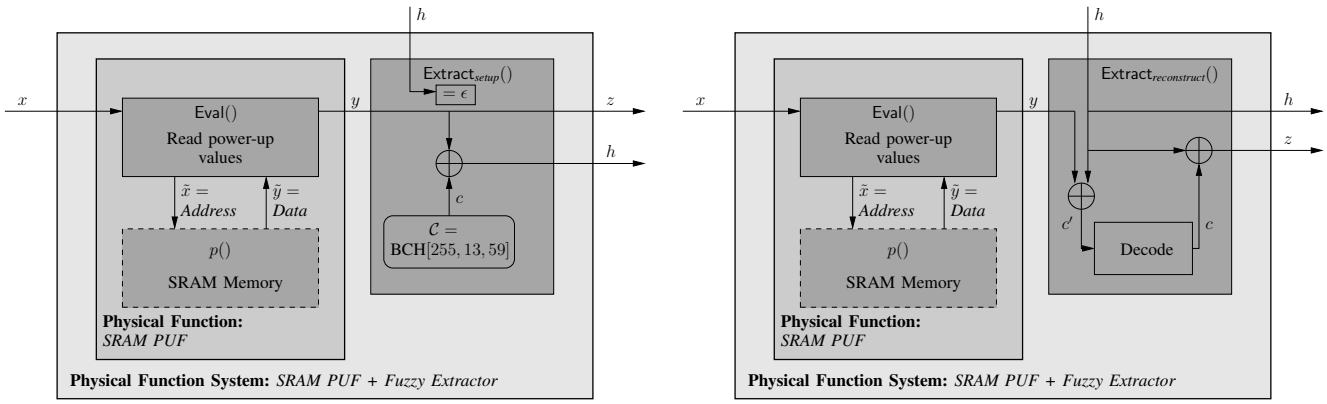
Definition 4 (Physical Function Infrastructure): A *physical function infrastructure* $\mathcal{F}$ refers to a fixed creation process Create (Definition 3) and the set of all PF systems PFS (Definition 2), where the physical component p is the result of Create, i.e.,

$$\mathcal{F}_{\alpha_{\mathrm{CR}}} = (\mathsf{Create}, \{\mathsf{PFS}_{p,\alpha_{\mathrm{PF}},\alpha_{\mathrm{EX}}} : p \leftarrow \mathsf{Create}(\alpha_{\mathrm{CR}})\}) \tag{5}$$

where α_{CR}, α_{PF} and α_{EX} are fixed.

C. Example

As a practical example of an implementation of a PF system, we consider an SRAM PUF [25] as physical function and a *fuzzy extractor* [34] as the $\mathsf{Extract}$ procedure. This example is used throughout this paper to illustrate the practical relevance of the introduced formal properties (see Figure 2). However, the generic nature of the introduced model allows to apply it to a very broad class of physical functions including all known types of PUF constructions.



(a) Setup mode implementation of the PF system.

(b) Reconstruction mode implementation of the PF system.

Figure 2. An illustrative and practical example of a PF system implementation based on an SRAM PUF and a fuzzy extractor.

The behavior of an SRAM PUF is based on the power-up values of SRAM memory cells. An SRAM memory address range is considered as the challenge x to the PUF, and the power-up values of these cells are considered as the PUF response y. SRAM memory cells are typically implemented on a silicon chip using a CMOS[1] production process. The parameter α_{CR} of the SRAM PUF creation process includes, among others, the different design options for an SRAM cell and the controllable parameters of the CMOS production process. The evaluation parameter α_{PF} describes the settings of the measurement process, e.g., typically the resolution of the analog-to-digital converter. In the SRAM PUF, this resolution is fixed by construction to produce 1-bit values, hence α_{PF} is considered to be fixed by design. Finally, the extraction parameter α_{EX} is used to describe the settable options of the extractor such as the correction capabilities of the used error correcting codes. For instance, in the example described in Figure 2, α_{EX} would be fixed and specifies a BCH$[255, 13, 59]$ error correcting code.

Experimental results by Guajardo et al. [25] show that under normal conditions, consecutive power-ups of the same SRAM memory differ on average only in 3.57% of the bits, which can rise up to 12% when large environmental variations are taken into account. Two power-ups of distinctive but identically implemented memories differ in 49.97% of the bits, which is extremely close to the optimum of 50%. These figures point out clearly the identifying properties of this structure. For the extraction procedure, a *fuzzy extractor* is used as introduced by Dodis et al. [34]. A further discussion on the goal and operation of a typical fuzzy extractor can be found in Section IV-C. The implementation of this SRAM PUF-based PF system is shown in Figure 2.

[1]Complementary Metal-Oxide-Semiconductor

IV. Robustness

A. Rationale

As explained in Section III, a PF might respond to the same challenge with different responses, when queried several times. However, if these responses are "similar", it is possible to overcome this problem by using an appropriate extraction algorithm. By robustness, we refer to the property that former outputs of a PF system can be reconstructed at a later time. Obviously, a certain level of robustness is a necessary prerequisite for using PF systems as functions in the classical sense.

Robustness could refer to at least two properties: (i) the ability to reconstruct the output of a PF system that has been produced by the setup mode, or (ii) the ability to always recreate the same output in reconstruction mode (that may be different from the output in setup mode). We decided for the first option for two reasons: first, one can show that a high probability for (i) implies also a high probability for (ii). Furthermore, (i) directly reflects the basic criterion that is necessary in a typical PUF-based key generation scenario.

B. Formalization

Following the consideration mentioned above, we formally define the robustness of a PF system as follows:

Definition 5 (Robustness): Let PFS be a PF system (Definition 2) and let $x \in \mathcal{X}$ be a challenge. The *challenge robustness* of PFS w.r.t. x is defined as the probability

$$\rho_{\mathsf{PFS}}(x) := \Pr\left[(z,h) \leftarrow \mathsf{PFS}(x,h) : (z,h) \leftarrow \mathsf{PFS}(x,\epsilon)\right] \tag{6}$$

This means that robustness is the probability that an output generated by $\mathsf{Extract}$ in reconstruction mode matches the output generated earlier by $\mathsf{Extract}$ in setup mode.

In practice, the best estimate of the challenge robustness is the sample mean over many evaluations of the same challenge on the same PF system. For cases where it is important

that each challenge has at least a certain robustness, the notion of *minimum robustness* is introduced:

Definition 6 (Minimum Robustness of a PF System): The *minimum robustness* of a PF system PFS (Definition 2) w.r.t. to a set of challenges $\mathcal{X}' \subseteq \mathcal{X}$ is defined as

$$\rho_{\mathsf{PFS}}^{\min} := \min\{\rho_{\mathsf{PFS}}(x) : x \in \mathcal{X}'\} \tag{7}$$

In some cases it may be difficult to estimate the minimum robustness. Actually, from a practical point of view, it can be sufficient that the average challenge robustness over many challenges of a PF system is high enough. This is where the notion of *average robustness* comes in:

Definition 7 (Average Robustness of a PF System): The *average robustness* of a PF system (Definition 2) w.r.t. a set of challenges $\mathcal{X}' \subseteq \mathcal{X}$ is defined as

$$\rho_{\mathsf{PFS}}^{\text{avg}} := \sum_{x \in \mathcal{X}'} \Pr[x \xleftarrow{\$} \mathcal{X}'] \cdot \rho_{\mathsf{PFS}}(x) \tag{8}$$

So far we considered PF systems, where the underlying physical function PF is fixed. Moreover, it is important to consider the probability of finding PF systems with a certain minimum/average robustness within a given PF infrastructure. The corresponding terminology is given in the following definitions:

Definition 8 (Minimum Robustness of a PF Infrastructure): Consider a PF infrastructure $\mathcal{F} = (\mathsf{Create}, \mathcal{P})$ (Definition 4), where $\mathcal{P} = \{\mathsf{PFS}_{p,\alpha_{\mathrm{PF}},\alpha_{\mathrm{EX}}} : p \leftarrow \mathsf{Create}(\alpha_{\mathrm{CR}})\}$. The minimum robustness of $\mathcal{F}$ is defined as

$$\rho_{\mathcal{F}}^{\min} := \min\left\{\rho_{\mathsf{PFS}}^{\min} : \mathsf{PFS} \in \mathcal{P}\right\} \tag{9}$$

Analogously, we define the average robustness of a PF infrastructure as follows:

Definition 9 (Average Robustness of a PF Infrastructure): Let $\mathcal{F} = (\mathsf{Create}, \mathcal{P})$ be a PF infrastructure (Definition 4), where $\mathcal{P} = \{\mathsf{PFS}_{p,\alpha_{\mathrm{PF}},\alpha_{\mathrm{EX}}} : p \leftarrow \mathsf{Create}(\alpha_{\mathrm{CR}})\}$. The average robustness of $\mathcal{F}$ is defined as

$$\rho_{\mathcal{F}}^{\text{avg}} := \sum_{\mathsf{PFS} \in \mathcal{P}} \Pr[\mathsf{PFS} \xleftarrow{\$} \mathcal{P}] \cdot \rho_{\mathsf{PFS}}^{\text{avg}} \tag{10}$$

Here, $\mathsf{PFS} \xleftarrow{\$} \mathcal{P}$ denotes the event that a random physical component p has been created, i.e., $p \leftarrow \mathsf{Create}(\alpha_{\mathrm{CR}})$, and that a PF system PFS has been generated based on p, i.e., $\mathsf{PFS} = \mathsf{PFS}_{p,\alpha_{\mathrm{PF}},\alpha_{\mathrm{EX}}}$ for a fixed α_{PF} and α_{EX}.

C. Example

Consider the practical PF system implementation (based on the SRAM PUF) described in Section III-C. For assessing robustness, we are interested in the difference between responses to the same challenge on the same PF, which in this case is caused by thermal noise and uncontrollable environmental variability. At large environmental fluctuations, the average percentage of differing bits in an SRAM PUF response can go up to 12% [25], which is too high for any practical cryptographic application. Industry-grade implementations require an average robustness of at least $1-10^{-6}$ to $1-10^{-9}$ or even higher. To achieve this with an SRAM PUF as physical function, an appropriate extraction algorithm must be used.

Different techniques are possible to decrease the error rate of PF responses. The typical choice is a *fuzzy extractor* [34], which is an algorithm that can be used to increase the robustness and the unpredictability of the responses of the PF system. In this section, we focus on the former process, which is called *information reconciliation*. The goal of an information reconciliation algorithm is to generate with high probability in the reconstruction phase the same output as in the setup phase, ensuring a high robustness level as defined in Definition 5. The inputs to the reconciliation algorithm, which are the responses of the physical function, i.e., y_{setup} and $y_{reconstruct}$, are not necessarily equal but can be distorted by noise. A secondary requirement is that the information reconciliation algorithm preserves as much of the information as possible that is present in the input[2]. This is necessary to provide acceptable levels of unclonability and unpredictability as defined in Section V and VI, respectively.

Obtaining a reliable and information-bearing result from a noisy measurement implies the use of error-correcting codes, which are the basis for most information reconciliation algorithms. Directly correcting the response of a physical function is not possible since this typically is not a noisy version of a code word but an arbitrary noisy vector. Most information reconciliation algorithms deploy a clever technique, which allows the use of decoding algorithms on arbitrary words. A relatively simple but powerful construction is the code-offset method proposed by Dodis et al. [34]. The idea is to transform an arbitrary bit vector y_{setup}, which represents the response of a physical function during the setup phase, to a random code word c of a predefined error correcting code. In the reconstruction phase, the same transformation maps the noisy version of the PF response $y_{reconstruct}$ to a noisy version of the code word, provided that the transformation is transitive and isometric. The noisy code word can be decoded to the correct code word c if the amplitude of the noise is smaller than the code's error correcting capability. The original PF evaluation during the setup phase can now be easily recovered by applying the inverse transition on the corrected code word c. Note that the random transformation of a code word is chosen in the setup phase and needs to be known in the reconstruction phase. For the code-offset method, this transformation is defined by the vector difference (offset) h between the PF response y_{setup} in the setup phase and the corresponding code word c. This offset vector is called *helper data*. The helper data

[2]A trivial algorithm, which gives a constant output regardless of the input, achieves perfect robustness but is not considered a good information reconciliation algorithm since all information of the input is lost.

h does not disclose the full output z of the PF system[3] to challenge x and hence can be stored in plaintext format, e.g., in a public database or in an external non-volatile memory.

A practical example of a PF system based on an SRAM PUF and an Extract algorithm using the code-offset technique is shown in Figure 2. The SRAM PUF generates 255-bit responses with an average bit-error rate of 12% between consecutive responses to the same challenge of the same PUF. The Extract algorithm transforms the PF responses to a random code word of a BCH$[255, 13, 59]$ error-correcting code by computing the bitwise exclusive-or of the PF response and an offset, as depicted in Figure 2(a). Assuming a binomial distribution for the number of errors in a single response, one can calculate that $\Pr\left[\mathrm{HW}(y_{setup} \oplus y_{reconstruct}) \leq 58\right] < 10^{-6}$. Hence, in order to achieve an average robustness $\rho_{\mathsf{PFS}}^{\mathrm{avg}} > 1 - 10^{-6}$ in this example, all occurrences of 58 or less bit errors must be correctable. The Extract algorithm achieves this since all occurrences of 59 or less bit errors in a code word of the BCH$[255, 13, 59]$ code can be corrected successfully by the decoder in the reconstruction phase, as shown in Figure 2(b). By using this extraction algorithm, a 12%-noisy physical function can be used to construct a PF system with an average robustness of $1 - 10^{-6}$.

The code-offset method is powerful and generic. It can be used as an Extract algorithm for every type of PUF implementation, provided that a suitable error-correcting code is available and a transitive and isometric transformation from PUF responses to a random code words exists. Besides error-correcting codes, other signal processing techniques can be used to reduce the amount of noise of responses [35]. However, some degree of error-correction is usually inevitable. Note that most known PUF constructions have an average bit error probability of their responses of less than 10%.

V. Physical Unclonability

A. Rationale

As this work is motivated by the increasing usage of *physically unclonable* functions, it is a natural choice to include unclonability into the model, which is the key property of PUFs that cannot be achieved by algorithmic solutions. In this section, we formally define the notion of *physical* unclonability. We stress that we consider only clones on the physical level and exclude mathematical clones. This restriction is motivated by the fact that an adversary in general has different possibilities for creating (i.e., cloning) a PF system that shows the "same" behavior as another PF system. For instance, the adversary could choose an Extract algorithm that maps all inputs to the same output. Clearly, two different PF systems using this Extract algorithm would behave exactly the same, independent of the underlying PFs. It is obvious that protection against such attacks can only be provided by mechanisms outside of the PF system. In general, while physical unclonability is an intrinsic feature, this is not true for mathematical unclonability, which hence is outside of the scope of a PF security model. We propose a definition of physical unclonability that can informally be stated as follows: *A PF system* PFS' *is a physical clone of another PF system* PFS *if both PF systems show the same behavior* and *deploy the same* Extract *algorithm.* By the second condition, we guarantee that we consider clonability on a physical level only.

It remains to discuss how to formalize the notion of "same behavior". Recall that PFs are assumed to be noisy in general, which raises the question of when two PFs can be considered being the same. A good starting point is to consider at first only one PF system. Recall that the extraction procedure is deployed to make a PF system "as deterministic as possible". Nonetheless, in certain cases, the same PF system might produce the same output twice only with a certain probability. We referred to this probability as the robustness of the PF system and termed it $\rho_{\mathsf{PFS}}(x)$ in dependence of the considered challenge x (see Definition 5). Intuitively, a clone PFS' cannot be more similar to the corresponding original PF system PFS than PFS itself. On the other hand, any PF system should be formally seen as a clone of itself. Therefore, the robustness marks a natural upper bound on "how similar a clone can become" and it seems to be natural to integrate the notion of robustness into the definition of clones.

Another aspect that needs to be considered is the following: depending on the use case, only the responses of PFS to a subset of challenges might be known at all. Thus, any other PF system PFS' that coincides on this subset of challenges could be seen as a clone. Therefore, it is sufficient that the definition of a clone captures only the set of challenges $\mathcal{X}' \subseteq \mathcal{X}$ that are relevant w.r.t. the underlying use case.

Note that a cloning attack might have different meanings:

- *Selective cloning* refers to the event that for a *given* PF system PFS a clone PFS' is constructed.
- *Existential cloning:* means that two arbitrary PF systems PFS and PFS' are produced, where one is the clone of the other.

The difference between selective and existential cloning is that in the latter case no "original PF system" is given and instead, the adversary is free to choose which PF system is cloned. Observe that this classification has some similarities to the security properties established for digital signatures and message authentication codes (MACs).

B. Formalization

We start with formalizing the notion of a clone:

Definition 10 (Physical Clone): Let α_{PF} and α_{EX} be a fixed evaluation and extraction parameter, respectively. Moreover, let $\mathsf{PFS} = \mathsf{PFS}_{p,\alpha_{\mathrm{PF}},\alpha_{\mathrm{EX}}}$ and $\mathsf{PFS}' =$

[3] An outsider only learns the offset h but not the code word c itself.

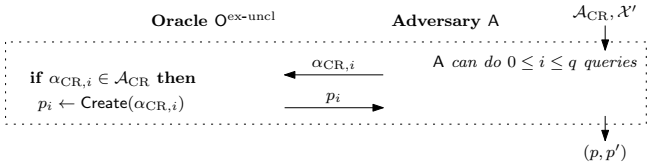


Figure 3. Existential unclonability security experiment $\mathbf{Exp}_{\mathsf{A}}^{\text{ex-uncl}}(q)$.

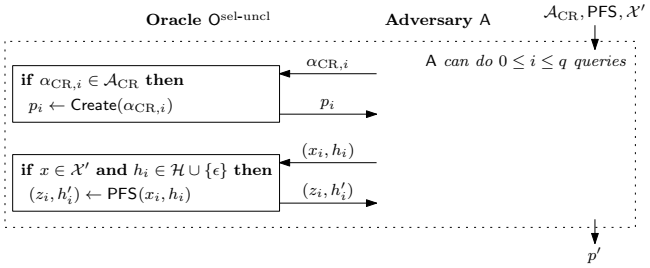


Figure 4. Selective unclonability security experiment $\mathbf{Exp}_{\mathsf{A}}^{\text{sel-uncl}}(q)$.

$\mathsf{PFS}_{p',\alpha_{\text{PF}},\alpha_{\text{EX}}}$ be two PF systems (Definition 2), that are identical except of their physical component, i.e., $p \neq p'$. We define that PFS' is a δ-*clone* of PFS w.r.t. $\mathcal{X}' \subseteq \mathcal{X}$ if for all $x \in \mathcal{X}'$ it holds that

$$\begin{aligned} \Pr\left[(z,h) \leftarrow \mathsf{PFS}'(x,h) : (z,h) \leftarrow \mathsf{PFS}(x,\epsilon)\right] \\ \geq \delta \cdot \rho_{\mathsf{PFS}}(x) \end{aligned} \tag{11}$$

For simplicity, we write $\mathsf{PFS}' \overset{\delta,\mathcal{X}'}{\equiv} \mathsf{PFS}$ if Eq. 11 holds.

Next, we formalize both notions of unclonability by means of two security experiments that specify the capabilities and the goal of the adversary A. On a high level, the adversary A is capable of creating arbitrary physical components, which in turn determine PF systems. In practice, A will be limited to a certain set of creation processes, e.g., by increasing the sensitivity of his production facility. We capture this formally by allowing A to choose the creation parameter α_{CR} from a set $\mathcal{A}_{\text{CR}}$ of possible creation parameters. In practice $\mathcal{A}_{\text{CR}}$ is expected to be small. We start by defining existential unclonability, where A must produce two *arbitrary* clones. In this scenario, which is depicted in Figure 3, A can query the Create process for $\alpha_{\text{CR}} \in \mathcal{A}_{\text{CR}}$ to create physical components p (see Definition 3).

Note that a physical function p implicitly defines a PF system $\mathsf{PFS} = \mathsf{PFS}_{p,\alpha_{\text{PF}},\alpha_{\text{EX}}}$ for some fixed evaluation and extraction parameter α_{PF} and α_{CR}, respectively (see Definition 2). Typically, only adversaries for which the time and computational effort are bounded are relevant for practice. Hence, we assume that A can do at most $q \geq 2$ queries to Create.

Definition 11 (Existential Physical Unclonability): Let $\mathcal{A}_{\text{CR}}$ be a set of creation parameters and let α_{PF} and α_{EX} be fixed parameters for the evaluation and extraction procedures, respectively. Note that this implicitly defines a family $\mathcal{F}_{\mathcal{A}_{\text{CR}}} := \{\mathcal{F}_{\alpha_{\text{CR}}} : \alpha_{\text{CR}} \in \mathcal{A}_{\text{CR}}\}$ of PF infrastructures (Definition 4).

A family of PF infrastructures $\mathcal{F}_{\mathcal{A}_{\text{CR}}}$ is called (γ, δ, q)-*cloning-resistant* w.r.t. $\mathcal{X}' \subseteq \mathcal{X}$, if

$$\begin{aligned} \Pr\big[\mathsf{PFS}'_{p',\alpha_{\text{PF}},\alpha_{\text{EX}}} \overset{\delta,\mathcal{X}'}{\equiv} \mathsf{PFS}_{p,\alpha_{\text{PF}},\alpha_{\text{EX}}} : \\ (p,p') \leftarrow \mathbf{Exp}_{\mathsf{A}}^{\text{ex-uncl}}(q); \\ p \in [\mathsf{Create}(\alpha_{\text{CR}})]; \alpha_{\text{CR}} \in \mathcal{A}_{\text{CR}}; \\ p' \in [\mathsf{Create}(\alpha'_{\text{CR}})]; \alpha'_{\text{CR}} \in \mathcal{A}_{\text{CR}}\big] \leq \gamma \end{aligned} \tag{12}$$

This means: the probability that A generated, as output of the security experiment depicted in Figure 3, two physical components p and p' which (i) imply clones on the PF system level and (ii) that have been created using creation parameters $\alpha_{\text{CR}} \in \mathcal{A}_{\text{CR}}$, is less than γ.

Note that Definition 11 covers different situations:

- *Honest manufacturer:* This case reflects the probability that an honest manufacturer creates two clones by coincidence and captures clonable PFs. In the case of $\mathcal{A}_{\text{CR}} = \{\alpha_{\text{CR}}\}$, i.e., where only one creation parameter is involved, the set $\mathcal{F}_{\mathcal{A}_{\text{CR}}}$ "collapses" to a single PF infrastructure $\mathcal{F}_{\alpha_{\text{CR}}}$. Likewise, A can perform Create only with this specific creation parameter. In other words, A is restricted to actions that an honest manufacturer could do within $\mathcal{F}_{\alpha_{\text{CR}}}$.
- *Malicious manufacturer:* This case covers the scenario, where $\mathcal{A}_{\text{CR}}$ contains more than one possible choice for the creation parameter α_{CR}, which allows A to influence the Create process in order to create a clone.

Finally, we formalize selective physical unclonability in terms of the security experiment depicted in Figure 4. The difference to the security experiment of existential unclonability is that the adversary A is *given* a PF system PFS for which A must create a clone. Therefore, in addition to queries to Create, A is allowed to query PFS with challenges $x \in \mathcal{X}'$. Again, we consider only restricted adversaries A that can do at most $q \geq 1$ queries to Create and PFS.

Definition 12 (Selective Physical Unclonability): Let $\mathcal{A}_{\text{CR}}$ be a set of creation parameters and let α_{PF} and α_{EX} be fixed parameters for the evaluation and extraction procedures, respectively. Moreover, let $\mathcal{F}_{\mathcal{A}_{\text{CR}}} := \{\mathcal{F}_{\alpha_{\text{CR}}} : \alpha_{\text{CR}} \in \mathcal{A}_{\text{CR}}\}$ be the corresponding set of PF infrastructures (Definition 4). Further, let PFS be a PF system (Definition 2) within the family of PF infrastructures $\mathcal{F}_{\mathcal{A}_{\text{CR}}}$, i.e., $\mathsf{PFS} \in [\mathsf{Create}(\alpha_{\text{CR}})]$ for some $\alpha_{\text{CR}} \in \mathcal{A}_{\text{CR}}$. We denote with A the adversary.

PFS is called (γ, δ, q)-*cloning-resistant* w.r.t. $\mathcal{X}' \in \mathcal{X}$, if

$$\Pr\left[\mathsf{PFS}'_{p',\alpha_{\mathrm{PF}},\alpha_{\mathrm{EX}}} \overset{\delta,\mathcal{X}'}{\equiv} \mathsf{PFS}_{p,\alpha_{\mathrm{PF}},\alpha_{\mathrm{EX}}} : \begin{array}{r} p' \leftarrow \mathbf{Exp}_{\mathsf{A}}^{\text{ex-uncl}}(q); \\ p' \in [\mathsf{Create}(\alpha_{\mathrm{CR}})]; \\ \alpha_{\mathrm{CR}} \in \mathcal{A}_{\mathrm{CR}} \end{array}\right] \leq \gamma \tag{13}$$

C. Example

As an example, we consider the implementation of an SRAM PUF as described in Section IV-C. We consider the case of existential physical unclonability by an honest manufacturer ($\mathcal{A}_{\mathrm{CR}} = \{\alpha_{\mathrm{CR}}\}$).

Experiments by Guajardo et al. [25] show that the relative amount of differing bits between two responses coming from distinct SRAM PUFs is on average close to one half. Independent experiments by Holcomb et al. [26] confirm that for common SRAM implementations, power-up states of different instances differ on average by approximately 40%. The exact relative difference depends on the way the considered SRAM cells are designed (which is specified by α_{CR}). It is expected that this probability will always tend to 50%. As a safe margin, we consider an SRAM PUF with an average relative difference of 40% between responses coming from distinct PUFs and a 12% bit error rate between responses coming from the same PUF. In both cases the challenge is fixed. It is reasonable to assume that the 40% differing bits are independent and uniformly distributed over all the responses, i.e., there is no particular bit position in a response that is more likely to differ between two PUFs than any other. This is explained by the random manufacturing processes affecting each SRAM memory cell independently and is confirmed by extensive experiments [25]. As discussed in Section IV-C, applying information reconciliation in $\mathsf{Extract}$ can turn this 12%-noisy SRAM PUF into a PF system with a robustness of $\rho_{\mathsf{PFS}}^{\mathrm{avg}} > 1 - 10^{-6}$. For simplicity, we consider clones w.r.t. *average* robustness rather than the individual challenge robustness. To assess the cloning-resistance of this SRAM PUF, we calculate the probability that an honest manufacturer produces two clones "by accident", relative to the average robustness and a particular challenge set $\mathcal{X}'$.

Consider an SRAM PUF that accepts 8-bit challenges and produces 255-bit responses We assess the physical unclonability of this PUF w.r.t. a challenge set consisting of a single challenge, i.e., $\mathcal{X}' = \{x\}$. Therefore, we estimate the probability of producing the same output to the same challenge on two independently created PF systems (i.e., $q = 2$). We denote $y \leftarrow \mathsf{PF}_{p,\alpha_{\mathrm{PF}}}(x)$, $(z, h) \leftarrow \mathsf{Extract}(y, \epsilon)$ and $y' \leftarrow \mathsf{PF}_{p',\alpha_{\mathrm{PF}}}(x)$. The event, where $\mathsf{PFS}_p(x) = \mathsf{Extract}(y, \epsilon)$ and $\mathsf{PFS}_{p'}(x) = \mathsf{Extract}(y', h)$ produce the same output z, only happens when y and y' are by accident similar enough such that the error correcting capability of $\mathsf{Extract}$ corrects y' to y.

To calculate the probability of this event, we start by evaluating the probability of a particular creation event (see Eq. 12) and determine to what extent this creation event produces a pair of clones according to Definition 10. We first introduce the following notation: $\Delta_y = y \oplus y'$ is the offset between the two expected responses of different SRAM PUFs solely caused by the random manufacturing variability affecting the Create process. Moreover, by e we denote the error vector representing the effect of random noise occurring in the Eval process of a single SRAM PUF. p_y and p_e are the respective probabilities of a bit of Δ_y or e being one. Note that in our example $p_y = 40\%$ and $p_e = 12\%$. We also use $\mathbf{f}_{\mathsf{bino}}(t; n, p_i)$ and $\mathbf{F}_{\mathsf{bino}}(t; n, p_i)$, respectively, as the probability distribution and the cumulative distribution function of the binomial distribution in t with parameters n and p_i. We start by upper bounding $\mathrm{HW}(\Delta_y)$ with $\mathrm{HW}(\Delta_y) \leq 50$. This bound determines the probability of these "clones" being created according to Eq. 12: $\Pr[\mathrm{HW}(\Delta_y) \leq 50] = \mathbf{F}_{\mathsf{bino}}(50; 255, p_y) = 2.66 \cdot 10^{-12}$, which is taken over the randomness of the Create process. It is yet to be evaluated to what extent the two PF systems based on these SRAM PUFs are considered clones according to Definition 10. Both PF systems produce the same output z if both SRAM PUF responses (this time including bit errors) differ by no more than the error correcting capability of the $\mathsf{Extract}$ algorithm (i.e., 59 bits), given that the expected difference is 50 bits. The probability of this event corresponds to the left-hand side of Eq. 11 and is calculated as

$$\begin{aligned} &\Pr[\mathrm{HW}(\Delta_y \oplus e) \leq 59 : \mathrm{HW}(\Delta_y) \leq 50] \\ &= \sum_{i=0}^{50} \Pr[\mathrm{HW}(\Delta_y \oplus e) \leq 59 : \mathrm{HW}(\Delta_y) = i] \\ &\qquad \cdot \Pr[\mathrm{HW}(\Delta_y) = i : \mathrm{HW}(\Delta_y) \leq 50] \\ &= \sum_{i=0}^{50} \mathbf{F}_{\mathsf{bino}}\left(59; 255, \frac{i}{255} - 2 \cdot \frac{i}{255} \cdot p_e + p_e\right) \\ &\qquad \cdot \frac{\mathbf{f}_{\mathsf{bino}}(i; 255, p_y)}{\mathbf{F}_{\mathsf{bino}}(50; 255, p_y)} = 0.11 \geq 0.11 \cdot \rho_{\mathsf{PFS}}^{\mathrm{avg}} \end{aligned}$$

The last equality uses the fact that the considered Hamming weights are binomially distributed and evaluates the bit probability of the exclusive-or sum of two independent random bit vectors.

It follows that the considered PF infrastructure $\mathcal{F}_{\alpha_{\mathrm{CR}}}$ of this example, which is based on an SRAM PUF and a fuzzy extractor, is $(2.66 \cdot 10^{-12}, 0.11, 2)$-cloning resistant against an honest manufacturer. In practice, this means that with probability $2.66 \cdot 10^{-12}$, a manufacturer produces two PF systems that generate the same output on the same challenge with probability 0.11. Other values for (γ, δ) and $q = 2$ can be obtained by considering different bounds for Δ_y. Smaller bounds on Δ_y will result in increasingly larger chances of producing the same output but at much smaller probability to

Table I
DIFFERENT LEVELS OF $(\gamma, \delta, q = 2)$-CLONING-RESISTANCE FOR THE EXAMPLE PF INFRASTRUCTURE DISCUSSED IN SECTION V-C.

$\max \Delta_y$	γ	δ
0	$2.68 \cdot 10^{-57}$	1.00
10	$1.32 \cdot 10^{-41}$	0.9998
20	$2.39 \cdot 10^{-31}$	0.986
30	$1.75 \cdot 10^{-23}$	0.83
40	$2.96 \cdot 10^{-17}$	0.43
50	$2.66 \cdot 10^{-12}$	0.11
100	0.43	$3.01 \cdot 10^{-7}$
110	0.86	$1.49 \cdot 10^{-7}$
120	0.99	$1.29 \cdot 10^{-7}$
255	1.00	$1.28 \cdot 10^{-7}$

create such a PUF. At the other end of the spectrum are pairs of PF systems, which are very likely to be constructed but very unlikely to produce the same output. This is illustrated in Table I, which shows a few other unclonability levels of the considered PF systems based on SRAM PUFs.

For other PUF types, a comparable quantitative analysis of existential unclonability can be made based on statistical data and estimated distributions of PF responses. The assumed distribution of responses and noise will often be normal rather than uniform, as it is the case for SRAM PUFs. Another issue could be that responses to different challenges and/or on different physical functions are not independent. Due to their construction, it is reasonable to assume response independence for SRAM PUFs but in general this is not the case for other PUF types. In such cases an additional post-processing step is required in the extraction algorithm that *amplifies* the randomness in the PFS output and removes dependencies between different PF instantiations and responses. This step is called *privacy amplification* and is implemented by means of an adequate compressing (e.g., a hash) function. Note that a typical fuzzy extractor implementation already includes a privacy amplification step [34]. We stress that the results obtained for unclonability are almost always based on estimated distribution parameters. Hence, a statistical analysis of the accuracy of these results is preferred. Based on such an analysis, adequate safety margins should be taken into account when assessing the security of a PF system.

Discussing unclonability against malicious manufacturers and adversaries in general is often very difficult in practice. The reason is that, in practice it is often not possible to take the effort and implications of all possible technical capabilities of such adversaries into account. For such cases, we suggest a more ad-hoc approach that measures the cloning-resistance of a PF system against a malicious adversary by the efficiency of the best known cloning attack that can be performed by that adversary. This means that the cost and effort of the adversary need to be considered as an additional parameter of cloning resistance. This approach is very similar to the cryptanalysis approach of standard cryptographic primitives such as block ciphers and hash functions, where the security is measured based on the computational effort to perform the best known attack possible against the primitive.

VI. UNPREDICTABILITY

A. Rationale

One common application of PUFs is to use them to securely generate secret values (e.g., cryptographic keys). Examples include secure key storage [7], [8], [36] or hardware-entangled cryptography [9]. Such applications implicitly require that the adversary cannot predict the output of a PF system. Moreover, for typical PUF-based challenge-response identification protocols, e.g., as presented by Gassend et al. [37], it is important that the adversary cannot predict the response to a new challenge from previously observed challenge-response pairs. Therefore, the notion of *unpredictability* is an important property that needs to be included into a model for physical functions. Classically, the notion of unpredictability of a random function f is formalized by the following security experiment consisting of a *learning* and a *challenge* phase: in the learning phase, A learns the evaluations of f on a set of inputs $\{x_1, \ldots, x_n\}$ (which may be given from outside or chosen by A). Then, in the challenge phase, A must return $(x, f(x))$ for some $x \notin \{x_1, \ldots, x_n\}$.

Given that this formalization is common and widely accepted in cryptography, one may be tempted to adapt it in a straightforward manner to PUFs. This would mean to take the same definition but to consider PF systems instead of functions. However, this approach not always makes sense: first, the output of a PF system depends on a challenge x *and* some helper data h. Thus, the helper data h must be taken into account as well. Moreover, we stress that different applications may require different variants of unpredictability. For instance, the concept of PUF-based secure key storage [7] is to use a PF system for securely storing a cryptographic secret k. This secret k is usually derived from the output z of a PF system for some input x. In some cases, x is public and/or possibly fixed for all instantiations. Note that in such a scenario it is required that each device generates a different secret k for the same challenge x. Hence, the outputs of different devices (i.e., their PF systems) should be independent. This requirement is captured by the following security experiment: given the outputs $\mathsf{PFS}_1(x, \epsilon), \ldots, \mathsf{PFS}_n(x, \epsilon)$ of a set of PF systems to a fixed challenge x within the learning phase, the adversary A has to predict the output $\mathsf{PFS}(x, \epsilon)$ for another PF system $\mathsf{PFS} \notin \{\mathsf{PFS}_1, \ldots, \mathsf{PFS}_n\}$ in the challenge phase.

Clearly, there is a fundamental difference between the classical definition of unpredictability and this security experiment: in the original definition of unpredictability described in the previous paragraph, A is given the evaluation of *one* PF system on *many* challenges, while in the latter

experiment A learns the evaluation of *many* PF systems on *one* fixed challenge.

Obviously, a useful definition of unpredictability of a PF system should cover both unpredictability in the original sense *and* independence of the outputs of different PF systems. Therefore, we define a security experiment that involves the following sets:

- Let $\mathcal{P}_L$ be the set of PF systems that are allowed to be queried by A in the learning phase
- Let $\mathcal{P}_C$ be the set of PF systems that are allowed to be queried by A in the challenge phase
- Let $\mathcal{X}$ be the set of challenges that are allowed to be queried during the whole experiment

Now we consider two extreme cases:[4]

1) *Independence of the outputs of a single PF system:* Consider the case, where $\mathcal{P}_L = \mathcal{P}_C = \{\mathsf{PFS}\}$ consists of one single PF system only, while $\mathcal{X}$ contains several challenges. During the learning phase, the adversary A learns $\mathsf{PFS}(x_i)$ for several challenges $x_i \in \mathcal{X}$. Later, in the challenge phase, A has to predict $\mathsf{PFS}(x)$ for a new challenge $x \in \mathcal{X}$. It is easy to see that this is the direct translation of the classical unpredictability experiment described at the beginning o this section to the scenario of physical function systems.
2) *Independence of the outputs of a different PF systems:* Now consider the scenario, where $\mathcal{X} = \{x\}$ consists of one single challenge only, while $\mathcal{P}_L$ and $\mathcal{P}_C$ contains several PF systems. In this case, during the learning phase, A learns $\mathsf{PFS}_i(x)$ for several different PF systems $\mathsf{PFS}_i \in \mathcal{P}_L$. Afterwards, in the challenge phase, A has to predict $\mathsf{PFS}(x)$ for a *new* PF system $\mathsf{PFS} \in \mathcal{P}_C$ that has not been queried before. Note that this reflects the requirements of PUF-based secure key storage [7].

The definition of unpredictability should cover both extreme and all intermediate cases.

B. Formalization

We now are ready to define unpredictability. The definition is based on the security experiment $\mathbf{Exp}_{\mathsf{A}}^{\text{w-uprd}}$ shown in Figure 5.

Definition 13 (Weak Unpredictability): Let $\mathcal{P}_L, \mathcal{P}_C \subseteq \mathcal{P}$ be subsets of the set of all possible PF systems. Let $T = \emptyset$ and $q \in \mathbb{N}$ with $q \geq 0$. With A we denote the adversary that takes part in the security experiment depicted in Figure 5. A PF system is *weak* (λ, q)*-unpredictable* if

$$\Pr\left[z = z' : (z, z') \leftarrow \mathbf{Exp}_{\mathsf{A}}^{\text{w-uprd}}(q)\right] \leq \lambda \cdot \rho_p(x) \quad (14)$$

Note that the robustness of a PF system PFS is an upper bound for the predictability of the outputs of PFS. For instance, a true random number generator is a PF system

[4]For the sake of readability, we omit the helper data here.

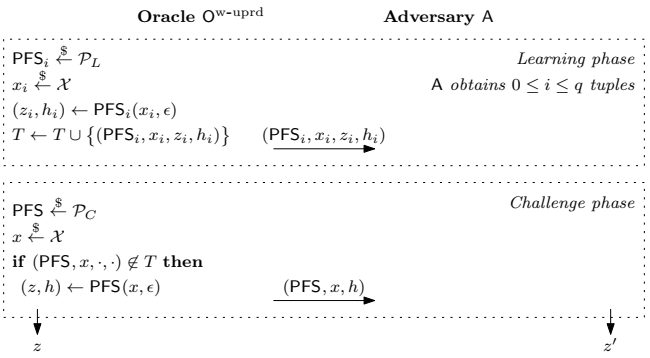


Figure 5. Weak unpredictability security experiment $\mathbf{Exp}_{\mathsf{A}}^{\text{w-uprd}}(q)$.

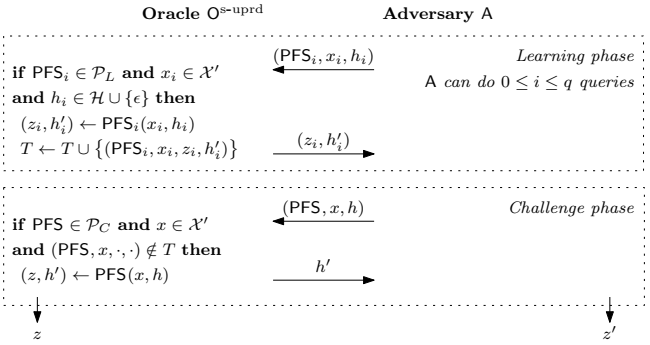


Figure 6. Strong unpredictability security experiment $\mathbf{Exp}_{\mathsf{A}}^{\text{s-uprd}}(q)$.

with very low reliability and thus, its outputs are highly unpredictable.

While stronger notions of unpredictability exist (see below), the consideration of weak unpredictability is nonetheless important for at least the following reasons: (i) weak unpredictability is an established property in cryptography and has been used for stronger constructions, e.g., see [38], and (ii) PF constructions may be weakly unpredictable only, e.g., arbiter PUFs, and hence should be covered by the model. Some use cases require a stronger notion of unpredictability, where the adversary is allowed to adaptively query the PF system in the challenge phase. We therefore define strong unpredictability based on the security experiment $\mathbf{Exp}_{\mathsf{A}}^{\text{s-uprd}}$ depicted in Figure 6.

Definition 14 (Strong Unpredictability): Let $\mathcal{P}_L$ be the set of PF systems that are allowed to be queried by A in the learning phase and let $\mathcal{P}_C$ be the set of PF systems that are allowed to be queried by A in the challenge phase. Moreover, let $T = \emptyset$ and $q \in \mathbb{N}$ with $q \geq 0$. With A we denote the adversary that takes part in the security experiment as shown in Figure 6. A PF system is *strong* (λ, q)*-unpredictable* if

$$\Pr\left[z = z' : (z, z') \leftarrow \mathbf{Exp}_{\mathsf{A}}^{\text{s-uprd}}(q)\right] \leq \lambda \cdot \rho_p(x) \quad (15)$$

C. Example

In general, there are no straightforward methods to strictly bound the unpredictability of a PF system. However, unpredictability can be assessed w.r.t. the best known attacks against the security experiments defined in Definition 13 and 14. This is very similar to measuring the security of a classical computational cryptographic primitive, where security is measured based on the effort needed for its best known cryptanalysis. However, in contrast to most classical primitives, physical functions do not have a well-defined algorithmic description against which cryptanalysis can be launched. In order to win the unpredictability experiment, the adversary needs to apply different methods, e.g., using additional information about the implementation of the physical function, or alternatively taking advantage of previously unknown statistical deviations or dependencies in the PF responses. The former method is used for modelling attacks against delay-based PUFs [20], [24], [22], where the adversary exploits the linearity of the delay circuits to build an accurate mathematical model of the PUF. The vulnerability of abusing any statistical deviation is assessed by running statistical tests on the outputs of the PF system and can be prevented by applying an appropriate privacy amplification algorithm on the PF system output (see Section V-C) [34].

For the SRAM PUF of our example, the first type of attack is considered to be infeasible. The reason is that learning the physical implementation of the SRAM cells in such detail that allows predicting their power-up state is infeasible in practice. Regarding the second class of attacks, an interesting result has been obtained for SRAM PUFs. A particular statistical test, called the context-tree weighting method (CTW) [39], has been performed on experimental SRAM PUF data to estimate its *min-entropy* content. Min-entropy is a notion from information theory measuring the uncertainty any adversary has about a particular value[5]. This means that min-entropy provides a strict lower bound for the unpredictability against *any* adversary. It must be noted that, in general, estimating the min-entropy of a physical function response is very difficult and relies on statistical tests that offer only limited assurance. For SRAM PUFs, min-entropy was nonetheless estimable due to their simple structure. Moreover, it was shown that their relative min-entropy content is relatively high (up to 95%). This can be explained by two reasons:

1) The *bias* of the bits of the response of a typical SRAM PUF is very low, i.e., on average the number of ones and zeros in an SRAM PUF response is almost equal.
2) Since every single SRAM PUF response bit is generated by a spatially separate piece of physical material (i.e., an individual silicon SRAM cell), it is very reasonable to assume that the produced bits are to a large extent statistically independent.

It is clear that both effects greatly increase the unpredictability of SRAM PUF responses.

Following this, we assume the SRAM PUF discussed in the previous example sections to have a min-entropy of 80%, which is a very safe estimation given the experimental observations discussed in the previous paragraph. This means that every 255-bit response of the SRAM PUF contains on average 204 bits of min-entropy, i.e., any adversary can guess the correct SRAM PUF response with a probability of at most 2^{-204}, which is negligible. However, we must assess the unpredictability of the PF system output. For that we need to take the extractor algorithm and the helper data into account. The fuzzy extractor construction, as described in Section IV-C and shown in Figure 2, outputs besides the SRAM PUF response $z = y$ also some helper data h, which is the offset between y and a random codeword c from the BCH$[255, 13, 59]$ error correcting code. Since there are only 2^{13} possible codewords c and since the helper data h is stored or transferred publicly, the adversary knows that the 255-bit output of the PF system has the form $z = c \oplus h$, which can only take one of 2^{13} possible values. Hence, the effective uncertainty of the adversary about z is at most 13 bits, and due to the 80% relative min-entropy, we estimate it to be 10.4 bits per 255-bit response. This is a significant reduction, which illustrates the cost we have to pay for achieving high robustness. Indeed, the extent of this *min-entropy loss* is directly related to the required error correction capability of the underlying error correction code. On the other hand, the remaining min-entropy provides a strong lower bound for the predictability. More in detail, no matter how strong the adversary is, its best guess of z will be correct only with a probability of at most $2^{-10.4}$. Hence, the example PF system based on an SRAM PUF and a fuzzy extractor is $(2^{-10.4}, 2^8 - 1)$-unpredictable for $\mathcal{P}_L = \mathcal{P}_C = \{\mathsf{PFS}\}$, both in the weak and the strong sense. This means that, even if all but one of the PF system outputs is learned, an adversary cannot predict the last output with a probability greater than $2^{-10.4}$. Since the assumptions made above can be generalized to many SRAM PUFs, equivalent unpredictability bounds hold for $\mathcal{P}_L = \mathcal{P}_C = \mathcal{P}$.

If the PF system is used to generate a secret key, it must be evaluated on different challenges multiple times to collect enough min-entropy. For instance, to generate a 128-bit key using the PF system in our example, it is required to obtain at least 13 challenge-response-pairs leading to $13 \cdot 255 = 3315$ response bits containing $13 \cdot 10.4 = 135.2$ bits of min-entropy. In order to obtain the 128-bit key from the PF responses, these 3315 bits have to be compressed by an appropriate *strong extractor*. This can be done using a universal hash function [40]. Typically, the strong extractor is integrated into the PF system as part of the $\mathsf{Extract}$ algorithm to ensure that the PF system generates full entropy outputs. This process is called privacy amplification and is

[5]Formally, the min-entropy of a distribution D over a set $\mathcal{X}$ is defined as $H_\infty(\mathsf{D}) = -\log_2 \max\{\Pr[x \leftarrow \mathsf{D} : x \in \mathcal{X}]\}$.

typically part of a fuzzy extractor construction [34].

In general, it is more difficult to obtain strong quantitative unpredictability bounds for other PUF types since there are no known methods to estimate the min-entropy content of their responses. Moreover, statistical dependencies between responses on different challenges and of different PUF instantiations need to be taken into account but can be difficult to detect. Note that min-entropy provides a very strict *information-theoretical* lower-bound on unpredictability, even against computationally unlimited adversaries. However, typically the computational power of the adversary is limited in practice. This means that even PUFs with a low min-entropy content can still produce unpredictable responses if their simulation is computationally complex. Alternatively, the unpredictability of a particular type of PUF can be assessed w.r.t. the effort of the best known modeling attack, as is done for symmetric cryptographic primitives (see Section V-C).

VII. Conclusion

Physically Unclonable Functions (PUFs) have been proposed in literature to exploit physical characteristics for security purposes. Various practical instantiations of PUFs exist, ranging from real-life products to theoretical PUF-based primitives and protocols (e.g., for identification and authentication). In view of the very different physical features they are based on, PUFs have mainly been developed in independent models and under different assumptions that are specialized for the corresponding applications. This absence of a unifying view typically makes the integration of PUFs in secure information systems a difficult task, hence limiting their further development and deployment. In this paper, we consequently formalized the security features of physical functions, in accordance to the existing literature on PUFs. More precisely, we proposed a new general security model for physical functions, that modularly captures the most important properties required for the integration of PUFs into cryptographic primitives and security applications. Our current model focuses on the minimum requirements on PUFs and can be easily extended by defining additional security-relevant properties required by future use cases.

In fact, the extension of the model to other security properties is one of the important remaining challenges, e.g., for covering tamper-evidence, meaning the property that unauthorized manipulations of PUFs are detectable. Another challenge is to develop new cryptographic mechanisms based on PUFs where the security can be reduced to the (alleged) properties of the deployed PUFs. Moreover, our present examples are mainly based on SRAM PUFs. Hence, a scope for further research would be to analyze other PUF types w.r.t. the properties formalized in our model.

Acknowledgements

We thank Pim Tuyls for several useful discussions and the anonymous reviewers for their helpful comments. This work has been supported in part by the European Commission under grant agreement ICT-2007-238811 UNIQUE and ICT-2007-216676 ECRYPT NoE phase II, and in part by the IAP Programme P6/26 BCRYPT of the Belgian State. François-Xavier Standaert is an associate researcher of the Belgian Fund for Scientific Research (FNRS-F.R.S.). Roel Maes is funded by a specialisation grant (nr. 073369) of the Flemish Agency for Innovation by Science and Technology (IWT).

References

[1] R. S. Pappu, "Physical one-way functions," Ph.D. dissertation, Massachusetts Institute of Technology, March 2001.

[2] R. S. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, pp. 2026–2030, 2002.

[3] D. Bauder, "An anti-counterfeiting concept for currency systems." Sandia National Labs, Albuquerque, NM, Tech. Rep. PTK-11990, 1983.

[4] K. Tolk, "Reflective particle technology for identification of critical components." Sandia National Labs, Albuquerque, NM, Tech. Rep. SAND-92-1676C, 1992.

[5] Commission on Engineering and Technical Systems (CETS), *Counterfeit Deterrent Features for the Next-Generation Currency Design*. The National Academic Press, 1993.

[6] K. Lofstrom, W. R. Daasch, and D. Taylor, "IC identification circuit using device mismatch," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2000, pp. 372–373.

[7] B. Škorić, P. Tuyls, and W. Ophey, "Robust key extraction from physical uncloneable functions," in *Applied Cryptography and Network Security (ACNS)*, ser. LNCS, vol. 3531, 2005, pp. 407–422.

[8] D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas, "Extracting secret keys from integrated circuits," *IEEE Transactions on VLSI Systems*, vol. 13, no. 10, pp. 1200–1205, 2005.

[9] F. Armknecht, R. Maes, A.-R. Sadeghi, B. Sunar, and P. Tuyls, "Memory leakage-resilient encryption based on physically unclonable functions," in *Advances in Cryptology (ASIACRYPT)*, ser. LNCS, vol. 5912, 2009, pp. 685–702.

[10] Verayo, Inc., "Verayo product page," http://www.verayo.com/product/products.html, November 2010.

[11] Intrinsic ID, "Intrinsic ID product page," http://www.intrinsic-id.com/products/, November 2010.

[12] K. Kursawe, A.-R. Sadeghi, D. Schellekens, B. Skoric, and P. Tuyls, "Reconfigurable physical unclonable functions — Enabling technology for tamper-resistant storage," in *IEEE Workshop on Hardware-Oriented Security and Trust (HOST)*, 2009, pp. 22–29.

[13] R. Maes and I. Verbauwhede, "Physically unclonable functions: A study on the state of the art and future research directions," in *Towards Hardware-Intrinsic Security*, ser. Information Security and Cryptography, D. Basin, U. Maurer, A.-R. Sadeghi, and D. Naccache, Eds. Springer Berlin Heidelberg, 2010, pp. 3–37.

[14] P. Tuyls, B. Skoric, S. Stallinga, T. Akkermans, and W. Ophey, "An information theoretic model for physical uncloneable functions," in *IEEE Symposium on Information Theory (ISIT)*, 2004.

[15] P. Tuyls, G.-J. Schrijen, B. Škorić, J. van Geloven, N. Verhaegh, and R. Wolters, "Read-proof hardware from protective coatings," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, vol. 4249, 2006, pp. 369–383.

[16] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," in *ACM Conference on Computer and Communications Security (CCS)*, 2002, pp. 148–160.

[17] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Design Automation Conference*, 2007, pp. 9–14.

[18] C.-E. D. Yin and G. Qu, "LISA: Maximizing RO PUF's secret extraction," in *IEEE Symposium on Hardware-Oriented Security and Trust (HOST)*, 2010, pp. 100–105.

[19] A. Maiti, J. Casarona, L. McHale, and P. Schaumont, "A large scale characterization of RO-PUF," in *IEEE Symposium on Hardware-Oriented Security and Trust (HOST)*, 2010, pp. 94–99.

[20] J. W. Lee, D. Lim, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas, "A technique to build a secret key in integrated circuits for identification and authentication application," in *Symposium on VLSI Circuits*, 2004, pp. 176–159.

[21] E. Ozturk, G. Hammouri, and B. Sunar, "Physical unclonable function with tristate buffers," in *IEEE Symposium on Circuits and Systems (ISCAS)*, 2008, pp. 3194–3197.

[22] L. Lin, D. Holcomb, D. K. Krishnappa, P. Shabadi, and W. Burleson, "Low-power sub-threshold design of secure physical unclonable functions," in *ACM/IEEE international symposium on Low power electronics and design (ISLPED)*, 2010, pp. 43–48.

[23] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Techniques for design and implementation of secure reconfigurable PUFs," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 2, no. 1, pp. 1–33, 2009.

[24] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *ACM conference on Computer and communications security (CCS)*, 2010, pp. 237–249.

[25] J. Guajardo, S. S. Kumar, G. J. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, vol. 4727, 2007, pp. 63–80.

[26] D. E. Holcomb, W. P. Burleson, and K. Fu, "Initial SRAM state as a fingerprint and source of true random numbers for RFID tags," in *Conference on RFID Security (RFIDSec)*, 2007.

[27] R. Maes, P. Tuyls, and I. Verbauwhede, "Intrinsic PUFs from flip-flops on reconfigurable devices," in *Workshop on Information and System Security (WISSec)*, 2008, p. 17.

[28] V. van der Leest, G.-J. Schrijen, H. Handschuh, and P. Tuyls, "Hardware intrinsic security from D flip-flops," in *ACM Workshop on Scalable Trusted Computing (STC)*, 2010, pp. 53–62.

[29] Y. Su, J. Holleman, and B. Otis, "A 1.6pJ/bit 96% stable chip-ID generating circuit using process variations," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2007, pp. 406–611.

[30] S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls, "Extended abstract: The butterfly PUF protecting IP on every FPGA," in *IEEE Workshop on Hardware-Oriented Security and Trust (HOST)*, 2008, pp. 67–70.

[31] J. D. R. Buchanan, R. P. Cowburn, A.-V. Jausovec, D. Petit, P. Seem, G. Xiong, D. Atkinson, K. Fenton, D. A. Allwood, and M. T. Bryan, "Forgery: 'fingerprinting' documents and packaging," *Nature*, vol. 436, no. 7050, p. 475, 2005.

[32] P. Bulens, F.-X. Standaert, and J.-J. Quisquater, "How to strongly link data and its medium: the paper case," *IET Information Security*, vol. 4, no. 3, pp. 125–136, 2010.

[33] U. Rührmair, J. Sölter, and F. Sehnke, "On the foundations of physical unclonable functions," Cryptology ePrint Archive, Report 2009/277, 2009.

[34] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in *Advances in Cryptology (EUROCRYPT*, ser. LNCS, vol. 3027, 2004, pp. 523–540.

[35] P. Tuyls, B. Skoric, and T. Kevenaar, Eds., *Security with Noisy Data — On Private Biometrics, Secure Key Storage, and Anti-Counterfeiting*. Springer-Verlag, 2007.

[36] P. Tuyls and L. Batina, "RFID-tags for anti-counterfeiting," in *The Cryptographers' Track at the RSA Conference (CT-RSA)*, ser. LNCS, vol. 3860. Springer Verlag, 2006, pp. 115–131.

[37] B. Gassend, D. Lim, D. Clarke, M. van Dijk, and S. Devadas, "Identification and authentication of integrated circuits," *Concurrency and Computation: Practice and Experience*, vol. 16, no. 11, pp. 1077–1098, 2004.

[38] K. Pietrzak, "A leakage-resilient mode of operation," in *Advances in Cryptology (EUROCRYPT)*, ser. LNCS, A. Joux, Ed., vol. 5479. Springer, 2009, pp. 462–482.

[39] F. Willems, Y. Shtarkov, and T. Tjalkens, "The context-tree weighting method: basic properties," *IEEE Transactions on Information Theory*, vol. 41, no. 3, pp. 653–664, 1995.

[40] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," in *ACM Symposium on Theory of Computing (STOC)*, 1977, pp. 106–112.