Cryptanalysis of Strong Physically Unclonable Functions

Liliya Kraleva [1], Mohammad Mahzoun [2], Raluca Posteuca [1], Dilara Toprakhisar [1], Tomer Ashur[3] and Ingrid Verbauwhede[1]

[1]imec-COSIC, KU Leuven, Belgium
firstname.lastname@esat.kuleuven.be
[2]Eindhoven University of Technology, Netherlands
m.mahzoun@tue.nl
[2]Cryptomeria, Belgium
tomer@cryptomeria.tech

Abstract

Physically Unclonable Functions (PUFs) are being proposed as a low cost alternative to permanently store secret keys or provide device authentication without requiring non-volatile memory, large e-fuses or other dedicated processing steps. In the literature, PUFs are split into two main categories. The so-called strong PUFs are mainly used for authentication purposes, hence also called authentication PUFs. They promise to be lightweight by avoiding extensive digital post-processing and cryptography. The so-called weak PUFs, also called key generation PUFs, can only provide authentication when combined with a cryptographic authentication protocol. Over the years, multiple research results have demonstrated that Strong PUFs can be modeled and attacked by machine learning techniques. Hence, the general assumption is that the security of a strong PUF is solely dependent on its security against machine learning attacks. The goal of this paper is to debunk this myth, by analyzing and breaking three recently published Strong PUFs (Suresh *et al.*, VLSI Circuits 2020; Liu *et al.*, ISSCC 2021; and Jeloka *et al.*, VLSI Circuits 2017). The attacks presented in this paper have practical complexities and use generic symmetric key cryptanalysis techniques.

1 Introduction

Physically Unclonable Functions (PUFs) are the method of choice for hardware applications requiring device authentication. Since securely storing a secret key in an Integrated Circuit (IC) is expensive and simply hard-coding it is vulnerable to physical attacks, PUFs offer a third option: as the manufacturing process of

ICs is subject to environmental variances, one can parameterize a cryptographic algorithm by harvesting the resulting randomness.

PUF taxonomy distinguishes between two types of Physically Unclonable Functions, namely Weak- and Strong PUFs. While both types are described in the literature as *Challenge-Response Protocols*, they differ by the challenge domain's size, i.e., the number of *Challenge Response Pairs* (CRPs). Weak PUFs support a relatively small number of CRPs, while the number of CRPs supported by a Strong PUF is much larger. Thus, Weak PUFs are usually used for storing a (small number of) cryptographic key(s), whereas Strong PUFs are often perceived as a building block in an authentication protocol.

The focus of this paper is on analysing the security of Strong PUFs as a device implementing a random n-to-m function. Broadly speaking, the workings of such a device consists of an n-bit challenge, and an m-bit response; in Strong PUF-literature, typically, $m = 1$. To compute the response, the PUF uses a finite amount of intrinsic randomness harvested from some physical properties of the hardware implementing it (*e.g.* the start-up value of a SRAM or the delay of a multiplexer in case of an arbiter PUF or the frequency of a ring oscillator). Following a sequence of successful attacks using Machine Learning (ML) techniques, the most recent trend is to build Strong PUFs from an IC template cascading non-linear components. The behavior of the non-linear components is determined by the intrinsic randomness mentioned above and the intuition behind this approach is that the cascade amplifies the non-linear effect. It appears from the literature that this approach indeed thwarts machine learning attacks; however, as we show in this paper it is not enough by itself to provide the desired properties.

1.1 Previous Work

The first PUF allowing for a large number of CRPs (thus initiating the research line on Strong PUFs) was the *arbiter PUF* introduced in [6]. The arbiter PUF exploits signal delay variations unique to each integrated circuit to parameterize a challenge-response protocol (*i.e.*, assigning a unique behavior to each device). The paper further introduces the adversary objectives and capabilities. It is assumed that the adversary has physical access to the IC and their goal is to clone the PUF. The PUF is considered secure if the adversary is unable to:[1]

- Apply an exhaustive search over the entire CRP space;
- Produce a counterfeit which perfectly simulates the behaviour of the attacked PUF;
- Apply a timing attack based on a measurement of the delays in the attacked strong PUF, followed by a prediction of the outputs;

[1]We stress that these attack scenarios are reproduced from [6] and are therefore informal. We provide a formal discussion in Section 5

- Apply non-invasive attacks (*e.g.*, algorithmic attacks); in this case the adversary models the PUF and tries to predict the response associated with a specific challenge with "very high" probability.[2]

A series of strong PUFs were subsequently proposed in the literature, *e.g.*, ring oscillator [17], XOR-Arbiter [17], or the OT-based PUFs [15]. At CCS'10, Rührmair *et al.* [16] developed a machine learning model attempting to predict the response of previously published strong PUFs. This resulted in breaking all Strong PUFs published until that point.

Other important work in the field of Strong PUFs' analysis is presented by Delvaux et al. in [4], in which the authors analyse eight Strong PUFs in an integrated framework. This work was then extended in [5], in which eleven more Strong PUFs were added to the integrated framework, showing numerous security and practicality issues for all the nineteen analysed primitives. Both papers focus on the analysis of Strong PUFs as authentication protocols, underlining that the CRP size of all the analysed Strong PUFs is not suitable for ensuring a sufficient level of security. The conclusion of these two papers is that "proper compensation seems to be in conflict with the lightweight objective" of a Strong PUF.

Moreover, [3] presents an analysis of five Arbiter-PUF based authentication protocols, using machine learning techniques, concluding that the use of lightweight obfuscation logic provides insufficient protection against machine-learning attacks for all five analysed primitives.

Consequently, the primary focus in PUF-design has shifted towards explicitly showing resistance to ML-modeling. A recent trend in this direction are *cascaded Strong PUFs* suggested in [19]. The idea is to use a composition of random sub-functions (*i.e.*, a cascade), conjecturing that the overall non-linear effect thwarts ML-modeling. The three PUFs we investigate in this paper are all of this type. Besides the three primitives addressed in this paper, many more strong PUFs are proposed in the literature, always aiming at increasing machine learning resistance. One such example is the "Double-Arbiter" PUF [12], which introduces a new type of Strong PUF based on the Arbiter PUF. The aim of this new primitive is to increase the unpredictability of the response, which is measured as the tolerance of the primitive to machine learning attacks. Another example of a cascaded PUF is the LP-PUF introduced in [20]. The LP-PUF consists of three layers, namely an Arbiter layer, a mixing layer and a XOR layer, leading to the generation of 1-bit responses. As the author observes, the structure of the LP-PUF resembles the Substitution-Permutation-Network (SPN), a technique used for the design of block ciphers. The same observation could apply for many cascaded Strong PUFs presented in the literature, leading to a natural inclusion of symmetric-key cryptanalysis techniques in the security evaluation of a Strong PUF. Although ML represents an important technique to asses the security of a primitive, it represents only a first step in the security analysis. ML is a black box approach and it does not take into account

[2]The term *very high probability* is used in the original paper. We interpret it to mean better than a random guess.

the structure of a Strong PUF. In this work we go beyond this approach, by using tools from symmetric-key cryptanalysis and by taking insights from the description of the primitive. More precisely, in this paper we choose to analyse three representative strong PUFs (Suresh *et al.* [18], VLSI Circuits 2020; Liu *et al.* [11], ISSCC 2021; and Jeloka *et al.* [7], VLSI Circuits 2017) because they have nice CMOS circuit implementations.

1.2 Our Contribution

This paper serves as a tutorial for the design of secure PUFs from a symmetric-key point of view. We motivate the need for such a tutorial by investigating—from an algorithmic point of view—three recently published cascaded Strong PUFs and show that they all exhibit undesirable properties undermining their security.

We stress that our aim with this paper is to provide general guidelines to the design of secure algorithms and that we did not attempt to provide a thorough cryptanalysis of the three algorithms. Indeed, more powerful attacks probably exist, and directly fixing the issues we raise is unlikely to result in secure devices.

2 Suresh *et al.*

In this section we present details of the strong PUF proposed by Suresh *et al.* in [18]. We then show that the cascaded nature of this algorithm collapses the 2^{128} CRP domain into a much smaller subspace of equivalence classes.

2.1 Description

Suresh et al. [18] offer a cascaded algorithm in 14nm CMOS with a claimed $10^{28} \approx 2^{93}$ CRP space. It takes a 128-bit challenge and returns a 1-bit response. The algorithm is abstracted into three layers as depicted in Figure 1.

The first layer consists of 64 random 4-to-1 functions (denoted ES1 boxes in the original paper). It takes a 256-bit input and returns a 64-bit output. The second layer involves 16 AES Sboxes, each taking an 8-bit input and returning an 8-bit output. The third layer again consists of 64 random 4-to-1 functions (denoted ES2 boxes in the original paper). Finally, the 1-bit response is taken as the parity of the third layer's output.

The original paper also refers to the first and third layers as Stage 1 and Stage 2, respectively. In lieu of a complete specification, which was not provided in [18], we proceed by assuming that the bit permutations are according to the wirings illustrated in Figure 1.

2.2 Intuition

According to [18], the PUF offers a challenge space in the range 10^{28}–$10^{31} \approx 2^{93}$–2^{103}. As this is the only quantifiable claim, we understand it to be the advertised se-

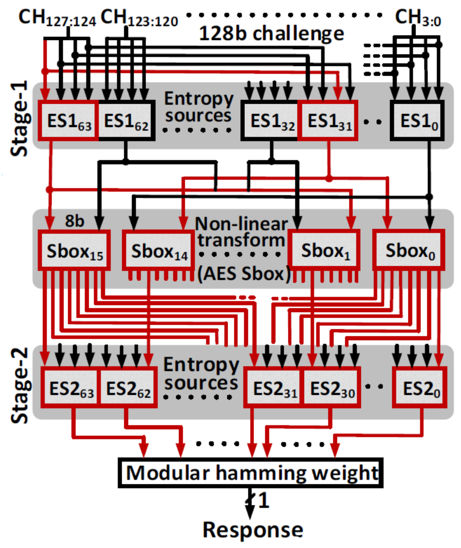

Figure 1: A schematic description of the PUF. The figure is copied without any modifications from [18] and we use it under the provisions of fair use.

curity. Since the response of the PUF consists of only a single bit, predicting the output with probability better than $\frac{1}{2}$ amounts to a successful attack.

Our first observation is that the first layer is an entropy choke point, *i.e.*, no matter how much randomness was invested in it, it cannot cascade more than 64 bits of entropy to the rest of the device. We say that two inputs are in the same *collision class* if they result in the same output after the first layer (Stage 1). Such two values will result in the same computation throughout the rest of the device and subsequently the same response for both inputs.

Our second observation is that each pair $(ES1_{32+n}, ES1_n)$ of ES1 functions is isolated from all other ES1 functions. Therefore, we consider an alternative representation where the first layer consists of 32 functions each mapping a 4-bit input to a 2-bit output. Then the output values are 00, 01, 10, and 11 and they induce four equivalence classes each containing four values on average.

2.3 Recovering the Equivalence Classes

By using the two observations presented in the previous section we show how $2^{11.8} \approx 10^{3.55}$ queries are enough to group the 128-bit input into 2^{64} sets. Each of these sets contains on average 2^{64} values all resulting in the same response. Thus, learning the response to one challenge leaks the response to all the other $2^{64} - 1$ values from the same equivalence class.

Without loss of generality, we consider the equivalence classes of the pair $(ES1_{63}, ES1_{31})$. The adversary fixes the last 124 bits of the input to an arbitrary value and iterates the first four. Two sets S^0 and S^1 are initialized and each 4-bit input is added to the set S^i if and only if the response is i. However, those are not yet the desired equivalence classes. Having the same response for two different plaintexts can be caused by any of the following three reasons:

- a collision after Stage 1;
- a collision after Stage 2, without a collision after Stage 1;
- the same Hamming weight after Stage 2, regardless of the state after Stage 1 or Stage 2.

The last two cases are *false positives* that need to be filtered out. In order to do so, the last 124 bits are fixed to a different arbitrary value.

Note that the elements in the same class will always be mapped together to one of the sets (but not necessarily the same one for different values of the last 124 bits). The equivalence classes are then computed by identifying which values are always mapped together to the same set. The probability of recovering the correct equivalence classes is approximately 99% when the process is repeated 7 times. For comparison, even three repetitions result in a success probability of 65% to identify the right set. The probability was computed empirically, using Monte Carlo simulations.

The complexity of determining an equivalence class for a single pair of ES1 functions with 99% success rate is $7 \cdot 16 = 2^{6.8} \approx 10^{2.05}$chosen queries. This is repeated for each pair of ES1 functions independently, leading to an overall complexity of $32 \times 2^{6.8} = 2^{11.8} = 10^{3.55}$ chosen challenges for recovering all pair-equivalence classes with expected success probability of $(1-0.01)^{32} = 0.73$ (or 73%).

The next step after recovering all pair-equivalence classes is to link them into state-equivalence classes. We define a state equivalence class as a set of challenges for which the state after Stage 1 is equal, therefore leading to the same response for both inputs.

Since each pair-equivalence class has an average of four elements, the average number of elements in one state-equivalence class is $4^{32} = 2^{64} \approx 10^{19.27}$. Additionally, the state after Stage 1 is 64-bit long, leading to $2^{64} \approx 10^{19.27}$ different state-equivalence classes.

At this point, learning the response to any challenge leaks the response to all other challenges within the same state-equivalence class. Note that the classes are built such that on observing a new challenge it requires negligible effort to identify which state-equivalence class it belongs to.

2.4 Discussion

Our approach exploits the fact that the input space collapses from 2^{128} to 2^{64} after Stage 1. We see that by investing a relatively small amount of effort (the analysis of $2^{11.8}$ chosen challenges) an adversary can learn the outputs of Stage 1 for any new challenge, effectively removing the first layer of random functions. Moreover, if the adversary learns the response associated to a challenge, then they also know that the same response is associated to all the challenges in the same state-equivalence class with the initial one. This is an undesirable behavior; or alternatively, in case this is an acceptable behavior, it can be achieved using cheaper components.

Note that we did not investigate the properties of the other layers, and it is likely that this basic attack can be improved further.

3 Liu *et al.*

The second design we analyse is [11] due to Liu et al. We show that this device is vulnerable to two generic attacks. The first attack can be applied if the the device exhibits an inherent bias, which can easily be detected and exploited. The second attack is independent on the bias of the device, and allows the adversary to guess the responses associated to a group of well chosen challenges.

3.1 Description

Liu et al. propose a cascade of 5-to-5-bit random functions formed in two layers of five with a bit-permutation between them, resulting in a 25-to-25-bit function which we denote by $S(x)$; see Phase 1 in Figure 2. $S(x)$ is used to digest the 100-bit input by first splitting it into four 25-bit blocks, then consuming the blocks iteratively with a feed-backward operation from each block to the next one; see Phase 2 in Figure 2.

A finalization function, which we denote by $C(x)$ is used to compute the response. First, a sequence of 5-to-1-bit random functions is applied to the output coming from the last call to $S(x)$, resulting in a 5-bit output. Then, the parity of these five bits is returned as the response; see Phase 3 in Figure 2.

Formally, denote by $x = (x_0, x_1, x_2, x_3)$ the 100-bit challenge, where x_i is the i-th part of length 25. Then,

$$\mathrm{PUF}(x) = C(S(S(S(S(x_0) \oplus x_1) \oplus x_2) \oplus x_3)) \tag{1}$$

is a succinct description of the function.

As before, we assume that the adversary does not have access to the description of the internal functions. Again the adversary seeks to predict the 1-bit response associated to a newly seen challenge with a probability better than $\frac{1}{2}$.

3.2 Intuition

From the formal description in Equation (1) arises a natural observation:

Observation 1. Fix the first 75 bits of the challenge, namely x_0, x_1 and, x_2; then, Equation (1) is reduced to

$$PUF(x) = C(S(c \oplus x_3)), \tag{2}$$

where $c = S(S(S(x_0) \oplus x_1) \oplus x_2)$ is fixed but unknown.

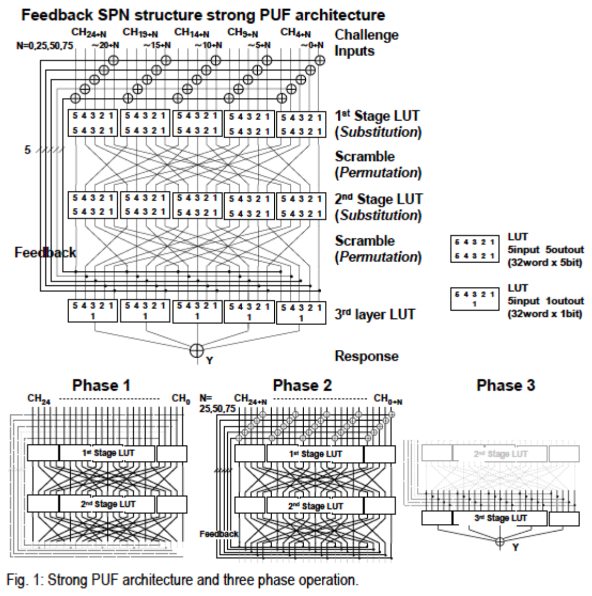
Feedback SPN structure strong PUF architecture
Challenge Inputs
N=0,25,50,75
1st Stage LUT (Substitution)
Scramble (Permutation)
2nd Stage LUT (Substitution)
Scramble (Permutation)
Feedback
3rd layer LUT
Response
LUT 5input 5outout (32word x 5bit)
LUT 5input 1outout (32word x 1bit)
Phase 1
Phase 2
Phase 3
Fig. 1: Strong PUF architecture and three phase operation.

Figure 2: A schematic description of the PUF. The figure is copied without any modifications from [11] and we use it under the provisions of fair use.

For brevity, we define an auxiliary function

$$f(x) = C(S(x)). \tag{3}$$

We underline that $PUF(x) = f(x \oplus c)$, where $c = S(S(S(x_0) \oplus x_1) \oplus x_2)$.

Definition 3.1. We define a map of a function f as the ordered set $M_f = \{(x, f(x)), \forall x\}$, where the order is defined as follows:

$$(x_1, f(x_1)) < (x_2, f(x_2)) \Leftrightarrow x_1 < x_2.$$

Note that the map M_f was constructed such that the input before the last application of the S function takes all possible values. In particular, all the maps have the same elements, but arranged in a different order. This property is formally described in the following general observation:

Observation 2. Let x_0, x_1, and x_2 be randomly chosen and fixed. Then, the map $M_f^c = \{(x \oplus c, f(x \oplus c), \forall x\}$ is an affine translation by $c = S(S(S(x_0) \oplus x_1) \oplus x_2)$ of the map M_f.

By constructing one arbitrary M_f in full, the adversary learns its distribution, *i.e.* the number of 0 or 1 responses. Due to Observation 2, any map M_f^c has the same distribution as M_f. If the device exhibits an inherent bias, then the distribution of M_f can be trivially used by the adversary to predict the output to any challenge not in M_f with probability better than $\frac{1}{2}$. For example, if the number of 0 responses associated to M_f is $2^{25} - 2^{20}$, then the probability of having a 0 response to an arbitrary challenge is $\Pr = \frac{2^{25}-2^{20}}{2^{25}} = 1 - 2^{-5} = 0.96$ (96%).

3.3 Attack Description

To construct M_f the adversary first chooses an arbitrary 75-bit value which they use to fix $x_0||x_1||x_2$. Then, iterating over the remaining 25 bits, the adversary queries the PUF $2^{25} \approx 10^{7.53}$ times and records the responses in M_f.

Constructing M_f requires $2^{25} \approx 10^{7.53}$ chosen challenges. The memory complexity for storing M_f can be optimized to 2^{22} bytes, *i.e.*, 4.19MB by querying the challenges in a natural order and indexing the responses accordingly.

Fixing M_f as a reference system, and recalling Observation 2, we see that determining c is sufficient for translating CRPs from M_f to M_f^c. To do so, the adversary observes 25 CRPs from the target equivalence class (*i.e.*, these 25 CRPs share the same value in the first 75 bits). Then, by means of exhaustive search on c, the adversary filters candidates where $M_f[i \oplus c] \neq M_f^c[i]$ using the 25 queries. This exhaustive search could be viewed as solving a system of equations with 25 bit unknown values. Therefore, the minimum number of equations such that this system is independent is 25. Our experiments show that 25 CRPs are enough for the correct c to be the only surviving candidate with high probability. At this point the adversary can determine with full certainty that $M_f^c[i] = M_f[i \oplus c]$ for all values of i. This way the adversary can learn the responses to all the remaining challenges from M_f^c.

3.4 Discussion

Note that the fact that the device is biased is not a problem in itself. Daemen and Rijmen analyzed in [1] the bias of ideal m-to-n-bit functions and showed that they are approximately normal distribution with mean 0 and variance 2^{-n}. For an ideal 100-to-1-bit function, such a bias would not be detectable. However, what Observation 2 shows is that this device actually models a random 25-to-1-bit function (in the best case scenario); making it significantly easier to detect the bias.

A second observation that we did not pursue in this paper is that the cascade structure will amplify the bias introduced by the random 5-to-5-bit functions. Each of these functions is an entropy choke point in itself and the cascade will result in bias that is even larger than what is predicted by [1] for the ideal case.

Moreover, we see again that by investing a relatively small amount of effort (the processing of $2^{25} + 25$ chosen challenges and an exhaustive search over a space of 2^{25}) an adversary can predict the response to unknown queries with high probability. This is again an undesirable behavior; or alternatively, in case this is an acceptable behavior, it can be achieved using cheaper components.

Note that we did not investigate the properties of the other layers, nor the ones of the component functions, and it is likely that this basic attack can be improved further.

4 Jeloka *et al.*

The third design we analyse was introduced by Jeloka *et al.* in [7]. We show that the responses of this device preserve input correlations with high probability.

4.1 Description

Jeloka *et al.* propose an SRAM-based PUF with a claimed CRP space that grows exponentially in the number of rows and the challenge length. The device has a secret initial state $S \in \mathbb{F}_2^{n\times m}$ and a secret matrix of powers $P \in \mathbb{Z}_n^{n\times m}$. Each column of P is viewed as a permutation on the integers $\{0, 1, \ldots, n-1\}$. Larger numbers are associated with "more power". A challenge $C = \{r_1, \cdots, r_t\} \subseteq \{0, 1, \ldots, n-1\}$ is defined as a sequence of rows, where each two consecutive rows (r_i, r_{i+1}) "fight" and the cell with bigger associated power wins the fight and overwrites its value over that of the other cell. The fight takes place independently for each column. More precisely, in a fight between row i and row j, for each column k, if $P_{i,k} > P_{j,k}$ then $S_{j,k} \leftarrow S_{i,k}$; otherwise $S_{i,k} \leftarrow S_{j,k}$. The corresponding response to a challenge C denoted by $F(C)$, represents the m bits of the last row in the challenge.

[7] describes the PUF in an abstract way for any dimensions (n, m) and challenge length t (see Algorithm 1). The authors suggest that $n = m = 64$ and $t = 6$ are sufficient to attain some unspecified level of security.

Algorithm 1: Psuedo-code for the PUF from [7]

```
Input: $C = \{r_1, \ldots, r_t\}$
Output: $F(C) := T_{r_t}$
$T \leftarrow S$
for $(r_i, r_{i+1}) \in C$ do
    for $0 \leq j \leq m-1$ do
        if $P_{r_i,j} > P_{r_{i+1},j}$ then
            $T_{r_{i+1},j} \leftarrow T_{r_i,j}$
        else
            $T_{r_i,j} \leftarrow T_{r_{i+1},j}$
        end
    end
end
Return $T_{r_t}$
```

4.2 Intuition

We show how two challenges that are "similar" will result in the same response with high probability. Concretely, we show that two challenges (sequences) differing only in their first value will produce the same 64-bit response with probability $2^{-0.77}$. Note that since the response is 64-bit long, one would expect this probability to be 2^{-64}.

4.3 Attack Description

First, we analyze the state S with a single column. In the case where $m = 1$, the response $F(C)$ to the challenge C is a 1-bit value $F(C) = b$.

Observation 3. Let $\bar{X} = (x_1, x_2, x_3, x_4)$ and $F(a, \bar{X})$ be the result of the fight between the rows a, x_1, x_2, x_3, x_4 (a, is the first row of the fight and the output is x_4). Let $G(\bar{X})$ be the result of the fight without considering the first row a. Then

$$F(a, \bar{X}) \neq G(\bar{X}) \iff F(a, \bar{X}) = S_a \text{ and } G(\bar{X}) \neq S_a$$

And the probability that a changes the response is

$$\Pr\left[F(a, \bar{X}) \neq G(\bar{X})\right] = \frac{1}{2} \cdot \frac{1}{5!} = \frac{1}{240}$$

The only case in which the result of two functions is different is when the response is S_a and the powers have the form $P_a > P_{x_1} > P_{x_2} > P_{x_3} > P_{x_4}$ with $p = \frac{1}{120}$ and $F(\bar{X}) \neq S_a$ with $p = \frac{1}{2}$.

Observation 3 shows that the first row in the challenge has small influence on the final response. Therefore, two challenges that only differ in the first row have different responses with the following probability

$$\Pr\left[F(a, \bar{X}) \neq F(b, \bar{X}) = \frac{1}{120}\right]$$

The cases that $F(a, \bar{X}) \neq F(b, \bar{X})$ are as follows:

1. $F(a, \bar{X}) = S_a$ and $F(b, \bar{X}) \neq S_a$ with probability $\frac{1}{240}$.
2. $F(a, \bar{X}) = F(\bar{X})$ and b is the strongest: $F(b, \bar{X}) \neq F(\bar{X})$ with probability $\frac{1}{240}$.

So the total probability that both queries give the same result in a single column is $1 - \frac{1}{120} = \frac{119}{120}$. Since the powers in different columns are independent, the probability of a correct guess for $m = 64$ columns is $\frac{119}{120}^{64} = 2^{-0.77}$.

4.4 Discussion

In this case we see that the PUF does not offer good diffusion properties and that each query allows to predict with high probability the response to multiple other queries. This is an undesirable property as one can expect from an authentication device to produce an uncorrelated response even for correlated entries.

For brevity, we did not model the case where the two challenges differ in positions other than the first one or when they differ in more than one position.

As a general observation we offer that the probability for collision is higher when the change occurs in earlier positions. This violates Jeloka *et al.*'s claim that longer sequences result in better security.

5 Strong PUFS When Viewed as Symmetric-key Algorithms

The work presented in this paper highlights a gap between two communities concerned with the development and implementation of secure cryptographic algorithms. The disparity is reflected in different areas such as design strategies, security analysis, and even in the way a new algorithm is presented to a larger audience. We discuss some of these differences, hoping to initiate a larger discussion and exchange of ideas between the two communities.

5.1 Abstraction Level

The design process resulting in a secure device involves several levels of abstraction. In the context of this paper, we focus on three of those:

- *Mathematical level.* In this abstraction level, the mathematical properties of abstract classes of functions are investigated. This kind of work involves for example methods from probability theory, combinatorics, statistics, algebra (both linear and modern), Fourier analysis, complexity theory, etc., and is normally published in pure mathematics- or mathematically-oriented cryptography- venues;

- *Algorithmic level.* At this level, the abstract functions are used as conceptual building blocks to form an algorithmic model. In addition to understanding the properties of the building blocks, the designer must also understand how they interact when combined together (for security reasons), and the platform that they will be running on (for efficiency reasons). In addition to the algorithm description itself, the algorithmic model includes a well-defined adversarial model and security claims pertaining to it. This is at the core of what cryptographers do, and a standard practice is to submit such works to cryptographic venues for peer-review and 3rd party evaluation;

- *Implementation level.* Having completed the vetting process of the algorithmic model, the algorithm is implemented first in simulation and later in tangible form. Even assuming the ideal security of the algorithm, the implementation process itself is susceptible to issues undermining the device's security (*e.g.*, timing attacks, side-channels). It is therefore not enough to understand the efficiency metrics and one must also understand the idiosyncrasies of secure implementations.

Strong PUFs aim to achieve goals on the implementation level (*e.g.*, secure key storage). The PUFs we looked into stem from a deep understanding of the execution platform, namely IC, and are therefore very efficient. However, the mathematical and algorithmic levels have been systematically overlooked. This is why Rührmair et al. and Delvaux et al. were able to use machine learning attacks in [3, 16] and why this paper can attack more recent works.

5.2 Adversarial Models

To be able to speak of the security offered by an algorithm, the conservation must obtain a shared understanding of what "security" is. As the notion of security only makes sense with respect to the existence of a bad actor (*i.e.*, an adversary), it makes sense to start the discussion there. An inherent imbalance between defenders and attackers (designer and adversaries, respectively) is that attack methods are plentiful, and it is enough for one of them to succeed for the defender to fail their role. Since attackers are creative and resourceful, it is futile to attempt predicting what methods they will use. Instead, cryptographers work with adversarial models. An adversarial model presumes only the capabilities of the adversary, but not how the adversary will use these capabilities.

Depending on the capabilities, an adversary can be classified as either *passive* or *active*. Whereas a passive adversary is capable of only observing the communication channel, an active adversary is additionally able to delete, add, and alter the data sent over the channel. In the context of PUF design, we identified the following relevant models from [8, 13]:

- **Passive adversaries**
    - *Ciphertext-only attack:* the adversary can only observe the outputs of the system; thus, the outputs of a secure cryptosystem should provide no information regarding the corresponding inputs, or the secret key/randomness; this model is the easiest to carry out in practice, since the only requirement is passively eavesdropping the communication channel.
    - *Known-plaintext attack:* the adversary is in possession of some inputs and the corresponding outputs generated under the same secret key.
- **Active adversaries**
    - *Chosen-plaintext attack:* the adversary is capable of obtaining the outputs corresponding to inputs of the adversary's choice.
    - *Adaptive chosen-plaintext attack:* the adversary may select the inputs depending on the received outputs from the previous requests.

Since in many cases the input or parts thereof are public (*e.g.*, HTTP headers) known plaintext attacks are often regarded practical. Furthermore, in 2-party authentication protocols such as the ones considered for strong PUFs, both active models also appear reasonably feasible.

5.3 Algorithmic Description

Symmetric-key algorithms normally define the input, output, and key spaces, and an algorithmic description to produce the corresponding output given the input and the secret key. A strong PUF can be modeled in the same way by considering the challenge as the input, the PUF's response as the output, and the intrinsic randomness as the key.

A fundamental assumption in cryptography is the reputed *Kerckhoffs' principle*, which states that "a cryptosystem should be secure even if everything about the system, except the key, is public knowledge". In the algorithmic description, the key is seen as part of the input that is unknown to the adversary.

It follows from Kerckhoffs' principle that the algorithmic description can be provided independent of the key (just like an IC can be manufactured irrespective of the values it will receive through its input wires).

The dissemination of new algorithms is a sort of "conversation" between the designers and their audience. To support this conversation, the designers provide a *design rationale* motivating their decisions. A reference implementation, and/or test vectors are provided to alleviate any ambiguity in the algorithmic description. If additional data was generated or used by the designers in the design process, it is also provided for examination and reproducibility.

5.4 Security Claims

When the key is modeled as an additional input, it becomes apparent that any algorithm using a finite number of secret bits is vulnerable to *brute force attacks* since it is always possible (but not necessarily feasible) to exhaustively iterate the key space. From the observation that the key size provides an upper bound on the effort required for attacking the algorithm arises an intuitive definition to what constitutes an attack.

Definition 5.1. A cryptosystem is said to be *broken* if an adversary can achieve their goals with less effort that would be required if they used a brute force attack.

Translating this to the case of Strong PUFs, let $\mathcal{C}$ be an arbitrary[3] n-bit challenge, $f(\mathcal{C})$ its 1-bit response, and $\mathcal{Q}$ the set challenges that have already been made and whose responses are known to the adversary. A reasonable security claim would be that if $\mathcal{C} \notin \mathcal{Q}$, no adversary can predict $f(\mathcal{C})$ with probability better than $\frac{1}{2}$, even when $\mathcal{C}$ is chosen after observing all the responses to the challenges in $\mathcal{Q}$ (adaptive chosen challenge); Formally,

$$\Pr[f(\mathcal{C})|\mathcal{C}, Q] = \frac{1}{2}. \tag{4}$$

[3]We note the terms *arbitrary* and *random* are not interchangeable in cryptography. A value is said to be *random* if it is sampled rigorously from a given distribution, usually the uniform one; it is said to be *arbitrary* when there is no importance to how it was sampled. For example, one's birthday is an arbitrary value, but for an encryption algorithm to be secure, the key must be chosen randomly from the uniform distribution of k-bit vectors.

Another way to view this security claim is through the notion of advantage: for any n-bit uniformly random challenge $\mathcal{C}$,

$$\mathbf{ADV} = \frac{1}{2} + \frac{|Q|}{2^n}. \tag{5}$$

Equation (5) captures the intuitive notion that the only way for an adversary to gain any knowledge is by querying device on that specific challenge.[4]

With an adversarial model, an algorithmic description, and a clear notion of security, the designer can provide *security claims*. Such security claim for the PUF presented in Section 2 can take the following form "*the Strong PUF can resist any chosen plaintext attack that runs in time less than* 2^{50} *time and requires less than* 2^{93} *chosen queries*". If an attacker generates all the state-equivalence classes, i.e. requiring 2^{64} queries, they can guess the response associated to any future challenge with probability 1. Therefore, such an attack which uses fewer queries and running time than what is permissible, is interpreted as breaking the algorithm.

A subtle point that we have seen overlooked is that in addition to being correct, the security claim must also be sensible. For example, since a PUF does not have a way to verify that a challenge has been received from a valid server (rather than from an adversary), it does not make sense to ignore chosen and adaptive-chosen challenge attacks. Likewise, our observations in this paper do not invalidate the claims about resistance to ML-attacks, yet these PUFs are not secure and should not be deployed in field settings.

Finally, we note that in light of the Strong PUFs we found in the literature, it does not make sense to consider the complexity of a brute-force attack as a function in the secret/random material. As the amount of randomness used is much larger than the CRP-space, the adversary can clone the PUF trivially by querying it completely. Thus Equations 4–5 are the more natural choice.

5.5 Concluding Remarks on Security-Efficiency Trade-offs

Surveying the recent literature on Strong PUFs, we noticed that the general trend in designing them is to employ a series of random-based operations, such as random Boolean functions, Sboxes, or linear layers. While the need for efficient algorithms is understandable, the security/efficiency trade-off must be handled carefully, as operations resulting in an insecure mechanism are by definition inefficient. Moreover, in symmetric-key cryptography, two of the most important properties that are analysed in a new design are the confusion and the diffusion ensured by the component functions of a cipher. These aspects are covered in a series of books such as [2, 9, 10].

In the PUFs we surveyed in this paper we see that the adversarial model is not stated, and instead, heuristic are used to assess the device's security. Among symmetric-key cryptographers, this approach is considered obsolete. Modern

[4]Cryptographers sometimes use the word *leakage* to describe "the knowledge an adversary may gain". However, this term is already loaded with meaning in the electrical engineering community, hence we omit it to avoid confusion.

techniques for the design of symmetric-key algorithms build on over 50 years of research in this domain to offer well understood trade-offs between efficiency and security. In other words, it is unlikely that a casual, non-systematic approach would yield a secure algorithm (regardless of its efficiency).[5].

We refer readers interested in understanding the state of the art in lightweight cryptography to the Wiki maintained by the cryptography group in the University of Luxembourg [14] noting that after more than a decade of research into this area it is unlikely that the state-of-the-art can be significantly improved without a paradigm shift. Interestingly, we observe that the amount of randomness exploited in PUF designs far exceeds what is common in symmetric-key cryptography. Whereas contemporary symmetric-key primitives have key sizes ranging between 80–256 bits, the algorithms we surveyed above use randomness that is measured in the order of thousands of bits. More randomness is usually associated with better security through an increase in the key size. It would be interesting to explore in future work if a different trade-off can be obtained by fixing the security level and somehow exploiting the additional randomness to improve efficiency.

6 Conclusion

PUF design share many common characteristics with the design of symmetric-key cryptographic algorithms. Motivated by undesirable properties we found in three recently published Strong PUFs, we attempted to provide in this paper a tutorial to the approach taken by symmetric-key researchers to ensure the security of their algorithms. We hope that this paper would serve as a starting point for discussion between the two communities.

7 Acknowledgements

This research was partially supported by CyberSecurity Research Flanders with reference number VR20192203 and by the Research Council of the KU Leuven C16/15/058. At the time of the research Tomer Ashur was an FWO post-doctoral fellow under Grant Number 12ZH420N. Liliya Kraleva was a PhD fellow by Research Foundation Flanders (FWO). This project was partially supported by Intel through the Intel Project on Cryptographic Frontiers.

References

[1] J. Daemen and V. Rijmen. Probability distributions of correlation and differentials in block ciphers. *J. Math. Cryptol.*, 1(3):221–242, 2007.

[5]For a version of this message, see Bruce Schneier's blogpost `https://www.schneier.com/blog/archives/2015/05/amateurs_produc.html`

[2] J. Daemen and V. Rijmen. *The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition.* Information Security and Cryptography. Springer, 2020.

[3] J. Delvaux. Machine-Learning Attacks on PolyPUFs, OB-PUFs, RPUFs, LHS-PUFs, and PUF–FSMs. *IEEE Transactions on Information Forensics and Security*, 14(8):2043–2058, 2019.

[4] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede. Secure lightweight entity authentication with strong pufs: Mission impossible? In *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 451–475. Springer, 2014.

[5] J. Delvaux, R. Peeters, D. Gu, and I. Verbauwhede. A Survey on Lightweight Entity Authentication with Strong PUFs. *ACM Comput. Surv.*, 48(2):26:1–26:42, 2015.

[6] J. W. Lee and D. Lim and B. Gassend and G. E. Suh and M. van Dijk and S. Devadas. A technique to build a secret key in integrated circuits for identification and authentication applications. In *2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No.04CH37525)*, pages 176–179, 2004.

[7] S. Jeloka, K. Yang, M. Orshansky, D. Sylvester, and D. Blaauw. A sequence dependent challenge-response PUF using 28nm SRAM 6T bit cell. In *2017 Symposium on VLSI Circuits*, pages C270–C271, 2017.

[8] J. Katz and Y. Lindell. *Introduction to Modern Cryptography.* Chapman and Hall/CRC Press, 2007.

[9] J. Katz and Y. Lindell. *Introduction to Modern Cryptography.* CRC Press, 2014.

[10] L. Knudsen and M. Robshaw. *The Block Cipher Companion.* Springer, 01 2011.

[11] K. Liu, Z. Fu, G. Li, H. Pu, Z. Guan, X. Want, X. Chen, and H. Shinohara. 36.3 A Modeling Attack Resilient Strong PUF with Feedback-SPN Structure Having $<$ 0.73% Bit Error Rate Through In-Cell Hot-Carrier Injection Burn-In. In *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, volume 64, pages 502–504, 2021.

[12] T. Machida, D. Yamamoto, M. Iwamoto, and K. Sakiyama. A New Arbiter PUF for Enhancing Unpredictability on FPGA. *The Scientific World Journal*, 2015, 2015.

[13] A. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[14] University of Luxembourg. Lightweight Block Ciphers, 2017. `https://www.cryptolux.org/index.php/Lightweight_Block_Ciphers`.

[15] U. Rührmair. Oblivious Transfer Based on Physical Unclonable Functions. In *Trust and Trustworthy Computing*, pages 430–440. Springer Berlin Heidelberg, 2010.

[16] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber. Modeling Attacks on Physical Unclonable Functions. In *CCS '10:: Proceedings of the 17th ACM conference on Computer and communications security*, page 237–249, New York, NY, USA, 2010. Association for Computing Machinery.

[17] G. E. Suh and S. Devadas. Physical Unclonable Functions for Device Authentication and Secret Key Generation. In *2007 44th ACM/IEEE Design Automation Conference*, pages 9–14, 2007.

[18] V. B. Suresh, R. Kumar, M. Anders, H. Kaul, V. De, and S. Mathew. A 0.26% BER, 10^{28} Challenge-Response Machine-Learning Resistant Strong-PUF in 14nm CMOS Featuring Stability-Aware Adversarial Challenge Selection. In *IEEE Symposium on VLSI Circuits, VLSI Circuits 2020, Honolulu, HI, USA*, pages 1–2. IEEE, 2020.

[19] A. Vijayakumar, V. C. Patil, C. B. Prado, and S. Kundu. Machine learning resistant strong PUF: Possible or a pipe dream? In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 19–24, 2016.

[20] N. Wisiol. Towards attack resilient arbiter puf-based strong pufs. Cryptology ePrint Archive, Paper 2021/1004, 2021.