TI-PUF: Toward Side-Channel Resistant Physical Unclonable Functions

Anita Aghaie, Amir Moradi

Abstract—One of the main motivations behind introducing PUFs was their ability to resist physical attacks. Among them, cloning was the major concern of related scientific literature. Several primitive PUF designs have been introduced to the community, and several machine learning attacks have been shown capable of modeling such constructions. Although a few works have expressed how to make use of Side-Channel Analysis (SCA) leakage of PUF constructions to significantly improve the modeling attacks, little attention has been paid to provide corresponding countermeasures.

In this paper, we present a generic technique to operate any PUF primitive in an SCA-secure fashion. We, for the first time, make it possible to apply a provably-secure *masking* countermeasure – Threshold Implementation (TI) – on a strong PUF design. As a case study, we concentrate on the Interpose PUF and based on practical experiments on an FPGA prototype, we demonstrate the ability of our construction to prevent the recovery of intermediate values through SCA measurements.

Index Terms—Physical Unclonable Function, Side-Channel Analysis, Threshold Implementation, Masking, Modeling, Machine Learning

1 Introduction

Physical Unclonable Functions (PUFs) are mostly known as chip fingerprints by employing their inherent physical variations which are also used for key generation in cryptographic applications [19], [37], [22]. These one-way physical functions behave uniquely and (in the ideal case) are reliable to generate an unpredictable output as a response to the given random input as a challenge [14]. Among different kinds of PUF, silicon PUFs [14] absorbed more attention of researchers as they mainly allow querying Challenge-Response Pairs (CRPs) within the chip independent of any external actuation and without requiring any analog signal [9].

An ideal PUF promises to generate a unique unpredictable response $\boldsymbol{r} \in \{0,1\}^m$ of a random challenge $\boldsymbol{c} \in \{0,1\}^n$ under a deterministic unclonable function inspiring from the Integrated Circuit (IC) manufacturing process. The unclonability of PUFs guarantees that building two PUF instances that have the exact same characteristics would be impractical.

PUFs are usually divided into two categories as *weak* PUFs and *strong* PUFs. The first category, also known as Physically Obfuscated Key (POK), has a small and limited number of CRPs, so that is applicable for cryptographic key generations. Instead, the second category has the exponential challenge-response space, which is appropriate in authentication protocols as well as key generation [19], [25], [36], [38]. Strong PUFs support multiple readings of the same response for one randomly chosen challenge.

The concern of designing a good strong PUF includes not only lightweight and qualitative features but also resistance against various attacks. However, these two parameters are mostly in a close trade-off, i.e., a highly-secure PUF which provides resistance against modeling and Side-Channel Analysis (SCA) attacks may not be considered as a too lightweight construction. In addition, it should be given thought how to achieve a proper uniqueness and uniformity for a small PUF on hardware platforms like an FPGA in which the designer should consider placement and routing as well as SCA protection [29].

With respect to the design and application of PUFs, there exist multiple PUF primitive options such as Arbiter PUF (APUF) [14], Ring-Oscillator PUF (ROPUF) [43], and Bistable Ring PUF (BR PUF) [7]. To exemplify, several delay-based PUFs are made of APUF as a basic element, e.g., XOR APUF [43], Feed Forward PUF [22], and the recently-introduced Interpose PUF [29]. Such strong PUF primitives promise to provide tamper-evident feature and satisfy the strict avalanche criterion [12], [10]. It is also notable to recall *composite* PUFs that build a PUF instance with combining a couple of PUF primitives [41], [30].

Due to improved various active and passive attacks on strong PUFs, the research attention moved toward other PUF design approaches to harden PUF primitives against modeling as well as physical attacks. One of such directions is the development of PUF protocols such as Lockdown [49], Slender PUF[26], and Noise Bifurcation[50]. In addition, the PUF protocols can even be helpful to increase the resistance against reliability attack [2] by altering the PUF enrollment phase and adding random nonce. To be more exemplified, there are more observations through the PUF-based security protocol applications such as identification, key exchange, or oblivious transfer [6], [14], [33], [35]. Besides, controlled PUF design can be counted one of such approaches which utilize input and output layers to increase the resistance against modeling attacks [15], [9].

The work described in this paper has been supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.
The authors would like to acknowledge Ulrich Rührmair, who motivated the topic of this work during his stay at Ruhr University Bochum.

- *Anita Aghaie and Amir Moradi are with the Ruhr University Bochum, Horst Görtz Institute for IT-Security, Bochum, Germany.*
  E-mail: firstname.lastname@rub.de

1.1 Related Works

Similar to cryptographic primitives, designing PUFs has encountered the problem of various cryptanalytic attacks being able to diminish their promises. Following the state of the art [28], the PUF security threats can be categorized in three groups: cryptanalysis attacks, machine learning based modeling attacks (MA), and SCA-based machine learning attacks. The attacks in the first category are not applicable to PUF primitives such as APUF or ROPUF, whereas they target controlled PUFs or composite PUFs (see more details in [28], [30]).

The second category involves the most threatening attack vector against PUF primitives. In MAs, it is supposed that the adversary simulates a learned software model of a PUF by means of information extracted from a set of collected CRPs. In other words, the attacker tries to model a PUF that can be used to imitate its intrinsic physical behavior [29], [22], [23]. There is a back-and-forth story for many PUF primitives and protocols which claimed on machine learning resistance such as PUF-FSM [13] and Slender PUF protocol [26]. Nevertheless, other research works like [8] present that these candidates are still required to be improved in terms of pure MA immunity.

The third category, i.e., SCA-based MAs (the focus of this paper), can be considered as the most potent attacks against the security of PUFs [28]. In such attacks, information about intermediate values (e.g., the response of each APUF primitive in an XOR APUF) are extracted by power analysis or timing analysis that help the attacker to emulate the targeted PUF function with modeling algorithms.

To cope with SCA leakages, two techniques are proposed in [39], [24]. One of them, which targets power analysis attacks, adds an extra arbiter cell to the end of each APUF primitive to produce complementary responses. Borrowed from the concept of dual-rail logics [27], this can mitigate the leakage but cannot entirely avoid it. Dual-rail logics for sure harden the SCA attacks; they can – in the best way – lead to an increase in the number of required SCA measurements. This shortcoming is due to the difference between the capacitive load of dual rails, which cannot be ideally equal due to slight differences in their routing originating from layout and process variations. Further, it has been shown in [21] that Electro-Magnetic (EM) SCA attacks can defeat dual-rail logics with moderate effort. Further, in such a dual-rail solution, due to unreliability[1] there are probable cases where the output of the dual-rail arbiters are not always complementary. Hence, the main goal of dual-rail logic (always having '10' or '01' at the dual-rail arbiter) is not always met, and the SCA leakages should show a data dependency (e.g., by collecting more SCA measurements). However, masking schemes (which we follow in this work) do not face such issues, and can protect against both power and EM SCA attacks. By constructing an isochronous circuit for the XOR network, the second approach of [39] and [24] aims at defeating timing attacks targeting the time that an APUF requires to issue the response. The ideal goal is to make a circuit with constant delay independent of its input. To the best of our knowledge, none of such techniques has been practically evaluated.

Nevertheless, controlled PUFs with input/output layers and composite PUFs are threatened by cryptanalysis and modeling attacks as well as those which make use of SCA information [40]. In other words, SCA information of the PUF primitives would allow having some information about the intermediate values, which again makes the modeling attacks possible. For example, we refer to [39], [24], [11], [3], [44]. The countermeasures discussed in these schemes focus on the input and output layers to enable the integration of an SCA-protection technique. They mainly argue that SCA attacks on PUF primitives are impractical due to a high level of noise in the measurements, which is in contrast to what we practically show in this paper.

In terms of reliability of PUFs (that can be somehow considered as side-channel information), there is a threatening modeling attack known as CMA-ES reliability attack [2], [1]. Apart from Lockdown protocol solutions [9] which inherently prevent the adversary from collecting reliability information, general approaches to increase the reliability (e.g., majority voting) have been considered to defeat CMA-ES [47]. Although such techniques can mitigate the internal noise sources, they almost have no effect on external ones. Therefore, they may just harden the attacks, but cannot provide full protection.

1.2 Our Contributions

Before introducing our achievements, the specification of the adversary model considered in this work is given. Note that the given definitions hold for almost all strong PUFs which have a publicly-accessible CRP interface.

- The attacker has access to the net-list of the PUF and its input/output layers, knows all implementation details, can with no particular limit send challenges and receive responses.
- While the challenge is processed, the attacker can measure the SCA leakage of either the entire design, i.e., the chip (by measuring power consumption) or somehow a part of the design (by measuring EM radiations by localized facilities).
- The attacker is not able to directly probe the intermediate values of the chip.
- The adversary does not have access to and cannot predict the random values which are generated inside the chip (a common and essential assumption of secret sharing).

We present a technique which enables operating any PUF primitive in an SCA-protected fashion. The core idea is based on Boolean *masking* (a proper SCA countermeasure), and its secure realization in hardware, i.e., Threshold Implementation (TI) [32]. Since the underlying function of PUF primitives (e.g., an APUF) is not known before fabrication, it does not seem possible to design a circuit which realizes the corresponding masked representation. Here we illustrate a mechanism to operate any arbitrary function (like a PUF primitive) in a masked form fulfilling all TI requirements. Further, we employ an SCA-resistant technique to improve the reliability.

Although the SCA-security of our construction comes at the cost of area and timing overheads, we believe that area

[1] For some challenges, the PUF primitive does not provide the same response when the same challenge is repeatedly given. This is referred to as PUF unreliability.

footprint is not of major concerns anymore in modern nanoscale technologies[2]. For practical evaluations we consider a recently published strong PUF primitive (Interpose PUF [29]). By means of FPGA-based experiments, we present its susceptibility to SCA-based MAs, and examine its resistance against SCA modeling attacks when it is plugged into our SCA-protected construction.

Organization. In Section 2, we give the preliminary knowledge on PUF primitives required to follow the underlying concept of the paper. Section 3 exhibits the practical result of our SCA-based modeling attacks on an instance of Interpose PUF [29]. We present the concept of Boolean masking and its application on arbitrary functions including PUF primitives in Section 4. This is followed by the application of our scheme on the same instance of Interpose PUF and its practical SCA evaluations in Section 5, while we conclude our research in Section 6.

2 PUF Preliminaries

This section briefly gives the required knowledge of PUF primitives used in our investigations. Afterward, the modeling attack(s), which we use in our analyses, are shortly described.

Notations. We denote binary or real random variables $\in \mathbb{F}_2 / \in \mathbb{R}$ with *italic*, vectors $\in \mathbb{F}_2^{n>1} / \in \mathbb{R}^{n>1}$ with ***italic bold***, elements in a vector with superscripts, and functions with *italic sans serif* font.

2.1 Arbiter PUF

The most common delay-based silicon strong PUF is known as Arbiter PUF (APUF) [22], [14], [43], [29] consisting of n stages, each of which controlled by a challenge bit $c^{i<n}$. Triggered by an electrical signal, a race is initiated simultaneously in two different delay-based challenge-dependent paths formed by these stages to reach a flip-flop. This delay-based PUF is modeled by a linear additive delay model (shown below) presented and applied in many related articles [22], [14].

$$\Delta = w^0\Phi^0 + ... + w^i\Phi^i + ... + w^n\Phi^n = \boldsymbol{w}^T \cdot \boldsymbol{\Phi}, \quad (1)$$

where $\Phi^n = 1$. The Vector $\boldsymbol{w} : \langle w^0, \ldots, w^n \rangle$ plays the role of weights which convey the physical characteristics of the underlying APUF and should be learned through the learner function and $\boldsymbol{w}^T$ stands for $\boldsymbol{w}$ transposed. The same way, $\boldsymbol{\Phi} : \langle \Phi^0, \ldots, \Phi^n \rangle$ is considered as the feature vector which is derived from the given challenge $\boldsymbol{c} : \langle c^0, \ldots, c^{n-1} \rangle$ and computed as follows.

$$\Phi_{i\in\{0,\ldots,n-1\}} = \prod_{j=i}^{n-1} (1 - 2 \cdot c^j) \quad (2)$$

Then based on this linear additive model, the final response of an arbiter PUF with n stages and 1-bit response r is computed. To this end, a unit sign function $\Theta(.)$ is used which determines which delay line propagators earlier to the arbiter.

$$r = \Theta(\Delta) = \begin{cases} 1 & \text{if } \Delta \geq 0, \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

2.2 XOR Arbiter PUF

XOR arbiter PUFs are more prevalent in strong PUF applications due to their boosted resistance against modeling attacks by means of combining the output of k arbiter PUFs by an XOR gate. The aforementioned additive linear model in Equation (1) is also applicable to model XOR Arbiter PUFs. Equation (4) represents this parallel attribute for a k-XOR APUF, where delay differences Δ are multiplied involving k weight vectors $\boldsymbol{w}^i$ of each individual APUF and k feature vectors $\boldsymbol{\Phi}^i$ derived from the corresponding challenges [38].

$$r = \Theta\Big(\prod_{i=1}^{k} \Delta^i\Big) = \Theta\Big(\prod_{i=1}^{k} \boldsymbol{w}^{i^T} \cdot \boldsymbol{\Phi}^i\Big) \quad (4)$$

2.3 Interpose PUF

With combining the concepts of APUF and XOR APUF, the recent modeling-secure PUF, called Interpose PUF, (x, y)-IPUF, is made of two XOR APUF layers as shown in Figure 1. The top layer consists of x instances of n-bit XOR APUFs (x-XOR APUF) with a 1-bit response, so-called r^t. This 1-bit response interposes to a determined position (ideally $\frac{n}{2}$) of the challenge of the bottom y-XOR APUF layer. Therefore, the bottom layer has $n + 1$ bits challenge and 1-bit final response r^b [29]. The entire n-bit challenge $\boldsymbol{c} : \langle c^0, \ldots, c^{n-1} \rangle$ is given to all x-XOR APUF of the top layer. The challenge of the entire y-XOR APUF of the bottom layer is formed by $\langle c^0, \ldots, c^{\frac{n}{2}-1}, r^t, c^{\frac{n}{2}}, \ldots, c^{n-1} \rangle$ for the most ideal case, i.e., the interpose position $i = \frac{n}{2}$.

This PUF is claimed to be highly secure against MAs depending on the challenge length n, the position of the interpose bit, and the number of applied APUFs in each layer (x, y) [29]. However, it is recently shown that even this construction is vulnerable to MAs up to (8,8)-IPUF through splitting the layers and learning one separately in an iterative manner [48]. It is notable to point out that in spite of this new modeling attack, IPUF's security does not cover the third category of attacks, i.e., SCA-based MAs [29].

2.4 Modeling Attacks on PUFs

As described in Section 1.1, MAs are one of the most threatening attacks on PUFs, which are also appeared in the form of improved by SCA leakages extracted when CRPs are processed. In several works such as [38], [45], it is efficiently shown that in classical MAs (without reliability information) the adapted logistic regression outperforms other techniques such as Support Vector Machine (SVM) and Artificial Neural Networks (ANNs) which do not benefit from the precise modeling of the underlying PUF primitives. Due to this fact, we express a brief introduction of the well-considered logistic regression.

Logistic Regression (LR) known as a classification algorithm is also exploited as a supervised machine learning framework to assign input data to a discrete set of classes [5]. It is shown

[2] In a 2 mm × 2 mm chip fabricated by an ASIC 40 nm standard cell library, the available area is more than 10 million GE, while a small AES encryption circuit needs around 2000 GE.

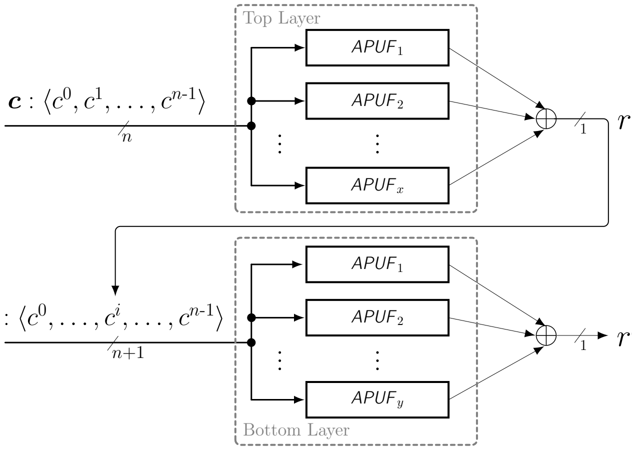

Top Layer
$APUF_1$
$APUF_2$
$APUF_x$
$\boldsymbol{c} : \langle c^0, c^1, \ldots, c^{n\text{-}1} \rangle$
n
1
r^t
$APUF_1$
$APUF_2$
$APUF_y$
$\boldsymbol{c} : \langle c^0, \ldots, c^i, \ldots, c^{n\text{-}1} \rangle$
$n{+}1$
1
r^b
Bottom Layer

Fig. 1. (x, y)-IPUF.

in [38] that there is an adapted format of logistic regression to model the PUF process as learning the weights especially in the case of a 1-bit response. Through logistic regression, the adversary aims at learning the APUF's weights $\boldsymbol{w}$ so that each challenge $\boldsymbol{c}$ has a probability $p(\boldsymbol{c}, r|\boldsymbol{w})$ to produce one-bit response r [38]. In order to classify the given challenges to possible responses $0/1$, we need an activation function for the weight vector $\boldsymbol{w}$ which is the encoding of the inherent PUF's parameters like the delay of the PUF lines.

Supposing that the PUF output is predicted by $f(\boldsymbol{c}, \boldsymbol{w})$, the logistic sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$ is applied as an activation function on $f(.,.)$ to update the randomly-initialized weight vector by $p(\boldsymbol{c}, r|\boldsymbol{w}) = r \cdot \sigma\big(f(\boldsymbol{c}, \boldsymbol{w})\big) + (1 - r) \cdot \Big(1 - \sigma\big(f(\boldsymbol{c}, \boldsymbol{w})\big)\Big)$ [38]. The decision boundary for the aforementioned function is determined when $f(.,.) = 0$ in the case of equal output probabilities. Through getting maximum likelihood for a selected training set $\mathcal{Q}$ of CRPs, the boundary decision for the weights $\boldsymbol{w}$ is determined in such a way that it leads to the minimum log-likelihood. In order to reduce the distance between initialized weights and the optimal ones, a gradient function can be applied to minimize the logistic regression cost function of weights for the selected training set $\mathcal{Q}$:

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_{\forall (\boldsymbol{c}, r) \in \mathcal{Q}} -\ln\big(p(\boldsymbol{c}, r|\boldsymbol{w})\big). \tag{5}$$

The Resilient back Propagation (Rprop) is well investigated and advantageous for gradient descent among other optimization algorithms like conjugate gradient. Rprop provides more advantages such as fast convergence speed and no need to chose a learning rate. Nevertheless, it can sound more complex [5], [20]. Since the learning weights process should be optimized frequently during the iterations, gradient information of weights on set $\mathcal{Q}$ as $\bigtriangledown l(\mathcal{Q}, \boldsymbol{w})$ can be helpful for the optimization methods like Rprop.

$$\bigtriangledown l(\mathcal{Q}, \boldsymbol{w}) = \sum_{\forall (\boldsymbol{c}, r) \in \mathcal{Q}} \big(\sigma(f(\boldsymbol{c}, \boldsymbol{w}) - r)\big) \bigtriangledown f(\boldsymbol{c}, \boldsymbol{w}) \tag{6}$$

To sum up, logistic regression sounds more appropriate compared to other learners like SVM due to its high learning speed, its stability through a large number of XOR APUFs, and since it does not require to be separated linearly in the feature space [39]. It is worth to mention that if the convergence speed or accuracy in the optimization algorithm Rprop through LR does not achieve the local minima, the training or test process should be restarted. Regarding these facts, the Artificial Neural Networks (ANNs) operate similarly to LR considering an activation function and an additive linear model for APUFs.

3 SCA Analysis on Original Design

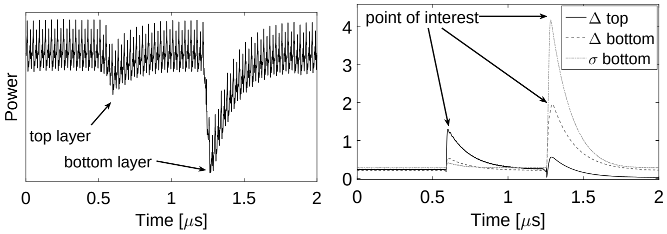
In order to practically show the effectiveness of SCA-based MAs, we have taken an exemplary instance of Interpose PUF with $x = 1$ and $y = 5$ with the challenge length $n = 64$, so-called $(1, 5)$-IPUF. More precisely, the top layer consists of a single 64-bit APUF, and the bottom layer a 65-bit 5-XOR APUF. The interpose bit is set right in the middle of the 65-bit challenge of the bottom layer. Based on the analyses reported in the original article [29], $(1, 5)$-IPUF provides a high level of security against modeling attacks. We performed our analyses on the original design provided by the authors of [29]; the HDL designs are accessible through the authors' GitHub[3].

We implemented one instance of $(1, 5)$-IPUF on a Spartan-6 FPGA, and conducted SCA analysis. To this end, we made use of a SAKURA-G platform [42] dedicated to SCA evaluations. The power consumption of the target FPGA is monitored at the output of the embedded amplifier, which magnifies the voltage drop over a shunt resistor at the V_{dd} path. The power traces are collected by a digital oscilloscope at the sampling rate of 2.5 GS/s, while the FPGA is driven by a 24 MHz clock. Following the recommendations given in [29], we put a considerable delay between the termination of the top layer and the activation of the bottom layer to make sure that the interpose bit is stable while the bottom later is triggered. Further, to conduct a proper analysis, we routed out the output of all APUF instances. In other words, the target FPGA sends out the response of the bottom later r^b as well as r^t and 5 response bits of the APUF instances of bottom layer. This is essential for the analysis to examine how accurately the intermediate values can be recovered through SCA analysis. Such additional outputs, of course, are not present in an actual design.

Since the power consumption of an APUF instance is relatively low (as the trigger signal just traverses through two lines to reach the arbiter cell with no glitch or extra toggle), it has been predicted in many articles that power analysis on such PUF primitives is unlikely possible [13]. According to the adversary model defined in Section 1.2, the adversary is able to give the same challenge to the design repeatedly. Hence, to overcome the low signal (or high noise issue) we repeated every measurement 1000 times and get an averaged power signal, one of which is shown in Figure 2(a), where the time instances corresponding to the activation of top and bottom layers are marked. We collected 100 000 such averaged signals (so-called collected signals), for each of which the design (on the target FPGA) is provided by a random 64-bit challenge.

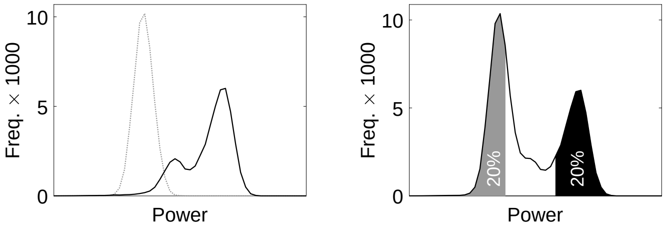
Top Layer. By classifying the collected signals into two groups based on the response of the top layer r^t, we obtain

[3] https://github.com/scluconn/DA_PUF_Library



(a) An averaged power signal (b) Difference of means

Fig. 2. SCA of original $(1, 5)$-IPUF, 100 000 averaged signals, each of which obtained by repeating 1000 times.



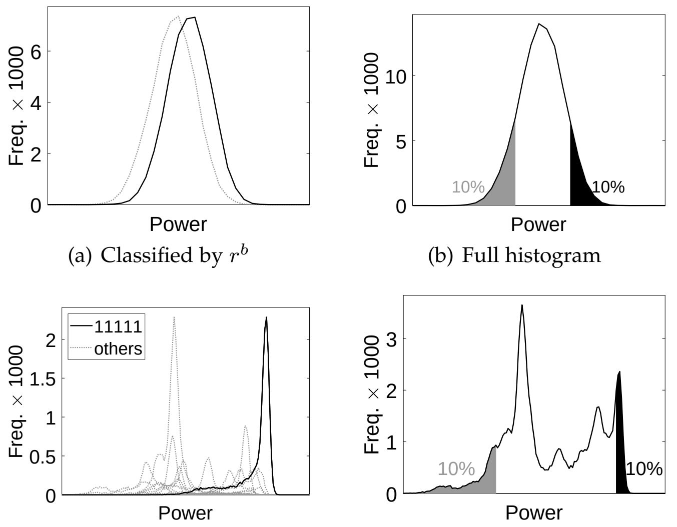
(a) Classified by r^t (b) Full histogram

Fig. 3. SCA of the top layer of original $(1, 5)$-IPUF using 100 000 averaged signals, histograms at the point of interest identified in Figure 2(b).

two mean traces, whose difference identifies at which time instances the power consumption depends on the output of the single APUF in the top layer (as shown in Figure 2(b)). Hence, in Figure 3(a), we present two histograms of the collected signals at a certain sample point corresponding to the evaluation of the top layer. It can be seen that the histograms include areas (the tails) which have a little overlap with each other. Therefore, the adversary can extract a subset of challenges associated with the tails of the full histogram (Figure 3(b)), and predict r^t based on being in the left or right tail. In our experiments, we defined the threshold of 20% at each tail to extract the challenges with guessed r^t. Note that such conjectures are not fully noise-free, i.e., there are (but not many) challenges in both tails with wrong guessed r^t. However, this does not strongly affect the performance of the modeling attack which learns the weights of the APUF of the top layer.

Attack. We conducted an LR attack (as described in Section 2.4) using the extracted challenges with the predicted r^t. Although – as stated – such predictions are slightly noisy, the attack required around 3000 CRPs to achieve the accuracy of 99%.

Bottom Layer. Having access to r^t or being able to predict it with a high accuracy (as we showed above how to do), the adversary can conduct a modeling attack on the bottom layer since it turns to an XOR APUF with a challenge of $n + 1 = 65$ bits, where all challenges are known. In the shown practical results, the top layer consists of only a single APUF. Therefore, we extend our SCA investigations on the bottom layer, where a 5-XOR APUF is employed. To this end, we performed two different analyses: one based on the output of the XOR APUF (r^b) and another one based on the response of all APUFs in the bottom layer. Figure 2(b) shows the corresponding difference of means signal Δ (based on



(a) Classified by r^b (b) Full histogram

(c) Classified by all APUFs in the bottom layer (d) Full histogram

Fig. 4. SCA of the bottom layer of original $(1, 5)$-IPUF using 100 000 averaged signals, histograms at the point of interest identified in Figure 2(b).

r^b) and the standard deviation signal σ (based on all APUFs in the bottom layer).

In Figure 4(a) and Figure 4(b), the result of the analyses based on r^b is shown. It can be seen that – similar to that on the top layer – the adversary can extract a subset of challenges with highly accurately guessed r^b. In Figure 4(c) and Figure 4(d), we represent a similar analysis result but based on the response of all APUFs in the bottom layer. In contrary to what has been reported based on simulation in [39], we observed that the Hamming weight of the XOR input (of the bottom layer) is not detectable by analyzing the SCA leakages. Instead, we noticed that the histogram of the case when all APUFs' responses are '1' is distinguishable from the others.

Attacks. In our experiments – knowing all challenge bits – having only the r^b, the weights of the 65-bit 5-XOR APUF of the bottom layer have been learned by an LR attack using 15 000 CRPs reaching the accuracy of 86%. In case of being able to distinguish the full one '11111' from the others, the adopted LR attack utilizes the same CRP size 15 000 but achieves a higher accuracy of 96%. It is noteworthy to mention that to adopt the LR algorithm, the logistic regression optimization has been changed to minimize the squared error between the given (extracted through SCA) and modeled responses leading to a different gradient formula instead of Equation (6) (see more details in [39]).

These analyses show that when the top layer is not a single APUF (i.e., $x > 1$), SCA information makes it still possible to conduct modeling attacks dedicated to the top layer. We should note that for IPUFs with a large x, the accuracy of the model obtained for the top layer might be not as good as that for small x as the SCA distributions may even overlap more (see Figure 4). Nevertheless, the SCA information decreases the required number of CRPs to model XOR APUFs. This is of high interest since the CRP space applied in classical MAs exponentially increases by a higher number of APUF instances in XOR APUFs. As stated before, such an SCA-boosted divide-and-conquer scenario allows the adversary to learn the weights of all APUF instances.

Remark. We should emphasize that the selected 20% and 10% thresholds are empirical and should be found by trial and error. Looking at the full histogram Figure 3(b), it can be straightforwardly predicted that it is made of two (slightly overlapping) distributions, hence a significant threshold 20% can be selected. In contrast, it can be seen in Figure 4(b) and Figure 4(d) that the full histogram does not show a clear distinction between two distributions. Therefore, a smaller threshold should be selected to minimize the wrong predictions. Due to the empirical nature of this approach, the attack should be made by a selected threshold; if the accuracy of the constructed model is not satisfactory, the attack should be repeated by a different threshold.

4 Protection Technique

In this section before illustrating the concept behind our construction, we first restate the basics of hardware masking as an SCA countermeasure. We limit ourselves to the essentials necessary for the underlying concept of our proposed scheme.

As given in Section 2, we denote elements in a vector with superscripts, and functions with *italic sans serif* font. Additionally, we denote shares of a random variable with subscripts, binary matrices $\in \mathbb{F}_2^{n>1} \times \mathbb{F}_2^{m>1}$ with ***CAPITAL ITALIC BOLD*** and Boolean functions with larger than one output bit with *CAPITAL ITALIC SANS SERIF* font.

4.1 Threshold Implementation

The most widely studied SCA countermeasure, i.e., masking, follows the concept of secret sharing as well as multi-party computation. In the s-th order Boolean masking, the secret value $\boldsymbol{c}$ is represented by $s+1$ shares $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_{s+1})$ in such a way that $\boldsymbol{c} = \bigoplus_{\forall i} \boldsymbol{c}_i$. The initial masking requires s independent and uniformly-distributed masks $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_s$ to form the shares as $\boldsymbol{c}_{i \leq s} = \boldsymbol{m}_i$ and $\boldsymbol{c}_{s+1} = \boldsymbol{c} \oplus \bigoplus_{\forall i} \boldsymbol{m}_i$. Application of a binary linear function $\mathsf{L}(.)$ on $\boldsymbol{c}$ in a masked form can be easily achieved by applying the same function on all shares as $\mathsf{L}(\boldsymbol{c}) = \bigoplus_{\forall i} \mathsf{L}(\boldsymbol{c}_i)$. This also holds for any affine function $\mathsf{A}(\boldsymbol{c}) = \mathsf{L}(\boldsymbol{c}) \oplus \boldsymbol{\alpha}$ if the constant $\boldsymbol{\alpha}$ is applied an odd number of times. The challenge is how to apply a non-linear function $\mathsf{F}(.)$ in such a masked form. For simplicity, without losing generality, suppose that $\boldsymbol{c}$ is a vector of 3 bits $\langle w, y, z \rangle$ and $\mathsf{f}(.)$ a coordinate function $\mathbb{F}_2^3 \mapsto \mathbb{F}_2$ with the algebraic degree of 3. Algebraic Normal Form (ANF) of exemplary function $\mathsf{f}(\boldsymbol{c})$: `11011100` is written as

$$\mathsf{f}\big(\boldsymbol{c} : \langle w, y, z \rangle\big) = 1 \oplus y \oplus wy \oplus wyz. \tag{7}$$

Representing every variable with $s+1 = 3$ shares leads to

$$\begin{aligned} \mathsf{f}(\boldsymbol{c}) = 1 \oplus \quad & y_1 \oplus y_2 \oplus y_3 \oplus \quad w_1y_1 \oplus w_1y_2 \oplus w_1y_3 \oplus \\ & w_2y_1 \oplus w_2y_2 \oplus w_2y_3 \oplus w_3y_1 \oplus w_3y_2 \oplus w_3y_3 \oplus \\ & w_1y_1z_1 \oplus w_1y_1z_2 \oplus w_1y_1z_3 \oplus w_1y_2z_1 \oplus w_1y_2z_2 \oplus w_1y_2z_3 \oplus \\ & w_1y_3z_1 \oplus w_1y_3z_2 \oplus w_1y_3z_3 \oplus w_2y_1z_1 \oplus w_2y_1z_2 \oplus w_2y_1z_3 \oplus \\ & w_2y_2z_1 \oplus w_2y_2z_2 \oplus w_2y_2z_3 \oplus w_2y_3z_1 \oplus w_2y_3z_2 \oplus w_2y_3z_3 \oplus \\ & w_3y_1z_1 \oplus w_3y_1z_2 \oplus w_3y_1z_3 \oplus w_3y_2z_1 \oplus w_3y_2z_2 \oplus w_3y_2z_3 \oplus \\ & w_3y_3z_1 \oplus w_3y_3z_2 \oplus w_3y_3z_3. \end{aligned} \tag{8}$$

The terms in the above equation should be split into $s'+1$ parts to represent $\mathsf{f}(.)$ in a masked form with $s'+1$ shares. To achieve d-th order security, one share of each variable should be missing in every d parts of the resulting split. This condition, known as non-completeness, is originally defined by TI [32], [31] forcing at least $s+1 = td+1$ input shares to achieve d-th order security for a function with algebraic degree t. This implies at least $s+1 = 3$ shares for the smallest non-linear function, an AND gate. The resulting split should also achieve uniformity, i.e., shared output $(\mathsf{f}_1(.), \ldots, \mathsf{f}_{s'+1}(.))$ should be indistinguishable from the output of $\mathsf{f}(.)$ being masked in $s'+1$ shares using s' independent and uniformly-distributed masks. For a given function, it is not straightforward to find a uniform split (see [32], [4]). Alternatively, additional masks (so-called fresh masks) can be added to achieve the uniformity. In a conservative way, by means of s' fresh masks $\boldsymbol{\gamma}$ we can re-mask the shared output as $\mathsf{f}_{i \leq s'}(.) \oplus \boldsymbol{\gamma}_i$ and $\mathsf{f}_{s'+1}(.) \oplus \bigoplus_{\forall i} \boldsymbol{\gamma}_i$.

The high number of required input shares of TI can be relaxed by computing the shared output in two steps which (most of the time) enforce the use of fresh masks. In the case of the above example, every term in Equation (8) can independently be XORed with a fresh mask and stored into a register, i.e., 40 fresh masks and 40 register cells. If the XOR of all fresh masks is also stored in a register, the output of the 41 registers form a uniform sharing of $\mathsf{f}(.)$ with 41 shares. The output of the register cells can be classified into any number of $s'+1$ groups; the XOR of all cells in each group would also result in an $s'+1$ uniform sharing of $\mathsf{f}(.)$. Of course, this conservative way of using fresh masks and registers can be optimized as some terms can be combined before being re-masked without any effect on the desired security order, e.g., y_1 and w_1y_1 and $w_1y_1z_1$. This has been extensively investigated in a couple of related articles [18], [17], [34], [16]. It indeed allows us to use $d+1$ inputs (and output) shares for d-th order security without being bounded by the algebraic degree t of the underlying function.

4.2 Application on PUF

The main motivation behind a PUF primitive is to avoid knowing the function it realizes before fabrication, and every fabricated PUF primitive should be different from the others, i.e., uniqueness. Therefore, it does not seem possible to construct a masked PUF primitive. Instead, we here introduce a mechanism that allows us to operate a PUF primitive in a masked way. Back to the example given in Section 4.1, we can add some terms an even number of times (highlighted in gray) to Equation (8) without affecting its functionality as follows.

$$\begin{aligned} \mathsf{f}(\boldsymbol{c}) = & (1 \oplus y_1 \oplus w_1y_1 \oplus w_1y_1z_1) \oplus (1 \oplus y_2 \oplus w_1y_2 \oplus w_1y_2z_1) \oplus \\ & (1 \oplus y_3 \oplus w_1y_3 \oplus w_1y_3z_1) \oplus (1 \oplus y_1 \oplus w_2y_1 \oplus w_2y_1z_1) \oplus \\ & (1 \oplus y_2 \oplus w_2y_2 \oplus w_2y_2z_1) \oplus (1 \oplus y_3 \oplus w_2y_3 \oplus w_2y_3z_1) \oplus \\ & (1 \oplus y_1 \oplus w_3y_1 \oplus w_3y_1z_1) \oplus (1 \oplus y_2 \oplus w_3y_2 \oplus w_3y_2z_1) \oplus \\ & (1 \oplus y_3 \oplus w_3y_3 \oplus w_3y_3z_1) \oplus (1 \oplus y_1 \oplus w_1y_1 \oplus w_1y_1z_2) \oplus \\ & (1 \oplus y_2 \oplus w_1y_2 \oplus w_1y_2z_2) \oplus (1 \oplus y_3 \oplus w_1y_3 \oplus w_1y_3z_2) \oplus \\ & (1 \oplus y_1 \oplus w_2y_1 \oplus w_2y_1z_2) \oplus (1 \oplus y_2 \oplus w_2y_2 \oplus w_2y_2z_2) \oplus \\ & (1 \oplus y_3 \oplus w_2y_3 \oplus w_2y_3z_2) \oplus (1 \oplus y_1 \oplus w_3y_1 \oplus w_3y_1z_2) \oplus \\ & (1 \oplus y_2 \oplus w_3y_2 \oplus w_3y_2z_2) \oplus (1 \oplus y_3 \oplus w_3y_3 \oplus w_3y_3z_2) \oplus \\ & (1 \oplus y_1 \oplus w_1y_1 \oplus w_1y_1z_3) \oplus (1 \oplus y_2 \oplus w_1y_2 \oplus w_1y_2z_3) \oplus \end{aligned}$$

$$\begin{aligned}&(1 \oplus y_3 \oplus w_1y_3 \oplus \boldsymbol{w_1y_3z_3}) \oplus (1 \oplus y_1 \oplus w_2y_1 \oplus \boldsymbol{w_2y_1z_3}) \oplus \\ &(1 \oplus y_2 \oplus w_2y_2 \oplus \boldsymbol{w_2y_2z_3}) \oplus (1 \oplus y_3 \oplus w_2y_3 \oplus \boldsymbol{w_2y_3z_3}) \oplus \\ &(1 \oplus y_1 \oplus w_3y_1 \oplus \boldsymbol{w_3y_1z_3}) \oplus (1 \oplus y_2 \oplus w_3y_2 \oplus \boldsymbol{w_3y_2z_3}) \oplus \\ &(1 \oplus y_3 \oplus w_3y_3 \oplus \boldsymbol{w_3y_3z_3})\end{aligned}$$

Based on Equation (7), we can therefore write

$$\begin{aligned}f(\boldsymbol{c}) =& f(\langle w_1, y_1, z_1\rangle) \oplus f(\langle w_1, y_2, z_1\rangle) \oplus f(\langle w_1, y_3, z_1\rangle) \oplus \\ & f(\langle w_2, y_1, z_1\rangle) \oplus f(\langle w_2, y_2, z_1\rangle) \oplus f(\langle w_2, y_3, z_1\rangle) \oplus \\ & \cdots \\ & f(\langle w_3, y_1, z_3\rangle) \oplus f(\langle w_3, y_2, z_3\rangle) \oplus f(\langle w_3, y_3, z_3\rangle) \\ =& \bigoplus_{\forall i,j,k} f(\langle w_i, y_j, z_k\rangle).\end{aligned} \tag{9}$$

This means that we can use the same function $f(.)$ to realize it in a masked form. However, this is correct only for an odd number of input shares $s+1$. In case of an even number (exemplary 2), Equation (9) would contain only the cubic terms (see Equation (7)). The constant, all linear and quadratic terms are canceled out, and the correctness property (defined by TI [32]) is not fulfilled.

Observation 1. For any arbitrary coordinate function with n-bit input $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$, if the input $\boldsymbol{c}$ is masked in an odd number of $s+1$ shares, the XOR sum of applying $f(.)$ on all $(s+1)^n$ shared input combinations equals $f(\boldsymbol{c})$.

Therefore, we can securely operate any arbitrary function; the minimum number of $s+1=3$ input shares provides at most 2nd-order security. As given in Section 4.1, this requires $(s+1)^n$ fresh masks as well as register cells to maintain both non-completeness and uniformity. As stated, in such a scenario, the output can be represented with any arbitrary number of shares $s'+1 \leq (s+1)^n$.

A PUF primitive with n challenge bits realizes a coordinate function $\mathbb{F}_2^n \mapsto \mathbb{F}_2$. Therefore, the above observation also holds for a PUF primitive. Since a PUF primitive cannot be instantiated multiple times with the same underlying function, we must serially re-use the same PUF primitive $(s+1)^n$ times to cover all shared input combinations. In other words, multiple instances of $f(.)$ can be used in Observation 1, but the nature of PUF primitives deactivates such an optimization.

Limitations. Serially using a PUF primitive $(s+1)^n$ times (and XORing the masked results, see Equation (9)) leads to a high latency even for small n and $s+1$. Exemplary, for an 8-bit PUF primitive and minimum number of input shares $s+1=3$, the PUF primitive should operate $3^8 = 6561$ times and the (re-masked) results should be XORed. Since any PUF primitive faces unreliability due to its physical characteristics, XORing 6561 PUF outputs will lead to extreme unreliability, regardless of its high latency.

Back to the main idea, i.e., Equation (9), we highlight that all shared input combinations $\forall i, j, k\ \langle w_i, y_j, z_k\rangle$ are applied to the function $f(.)$. Since it is based on an odd number of input shares $s+1$, the original (unmasked) input $\boldsymbol{c} : \langle w, y, z\rangle$ is definitely one of such shared input combinations. For example with 3 shares, $\boldsymbol{0}$ is represented by $(\boldsymbol{0}, \boldsymbol{0}, \boldsymbol{0})$, $(1, \boldsymbol{0}, 1)$, $(1, 1, \boldsymbol{0})$, or $(\boldsymbol{0}, 1, 1)$, and $\boldsymbol{1}$ by $(\boldsymbol{1}, 0, 0)$, $(0, 0, \boldsymbol{1})$, $(0, \boldsymbol{1}, 0)$, or $(\boldsymbol{1}, \boldsymbol{1}, \boldsymbol{1})$. It can be seen that the original (unmasked) value always appears in one of the shares. This means that although Equation (9) realizes a masked way of operating the function $f(.)$, it indeed hides the calculation of the original input $\boldsymbol{c} : \langle w, y, z\rangle$ among the others depending on the given masks. This fact does not hold for even number of input shares, which also justifies Observation 1.

We further should point out that – as defined in Section 1.2 – the challenge $\boldsymbol{c}$ is known to the adversary. This allows us to share only a few challenge bits as protecting the output of the PUF primitive is of our interest. Therefore, based on the trade-off between security and latency (resp. reliability), the designer can select how many challenge bits should be masked. This way, PUF primitives with large challenge sizes can also be used as long as the number of masked challenge bits is limited.

4.3 Realization

In order to realize the aforementioned scheme for a given PUF primitive, we illustrate our construction by an n-bit coordinate function and $s+1=3$ shares. To this end, we define an at-run-time user-adjustable signal $\boldsymbol{p}$, that identifies which challenge bits should be masked. As shown in Figure 5, the n-bit challenge $\boldsymbol{c}$ is masked by two n-bit masks $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$ to form a 3-share challenge $(\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3)$. A priori, multiplying $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$ by $\boldsymbol{p}$ (bit-wise AND) would allow us only to mask the identified challenge bits.

Each 3 corresponding bits (3 LSBs, 3 second bits, ..., 3 MSBs) are given to a 3-to-1 multiplexer (MUX) controlled by a 2-bit signal $\boldsymbol{s}^{i=1,\ldots,n}$. In order to efficiently control the MUXes, we make use of cascaded 2-bit Linear Feedback Shift Registers (LFSRs) initiated with '11' (see Figure 5). If p^i='0' (i.e., no masking for the i-th challenge bit), the i-th LFSR constantly states at '11'. Otherwise, the LFSR generates the cyclic sequence '11', '01', '10' and enables the next LFSR when it moves from '10' to '11'.

The selected n-bit input is given to the PUF primitive whose output is re-masked by means of fresh single-bit masks γ_1 and γ_2. This updates the register cells which store the XOR sum of re-masked output of the PUF primitive applied to shared input combinations. The shared final output $(\omega_1, \omega_2, \omega_3)$ is taken from the register cells which fulfills $f(\boldsymbol{c}) = \bigoplus_{\forall i} \omega_i = \omega$. This process is in line with the observation given in Section 4.2 satisfying non-completeness and uniformity (if the fresh masks $\boldsymbol{\gamma}$ are uniformly distributed).

We should highlight that when the LFSRs change, a leakage depending on the input might be exhibited. For clarification, consider the first MUX receiving LSBs (c_1^1, c_2^1, c_3^1) and controlled by $\boldsymbol{s}^1$. When $\boldsymbol{s}^1$ changes from '11' to '01', the output of the MUX switches from c_1^1 to c_2^1. However, by changing from '01' to '10', due to a race condition it may happen that the MUX switches from c_2^1 to c_1^1 shortly before giving c_3^1 as output. This can lead to a first-order leakage about c^1. A similar issue has formerly been observed in [46], where a solution based on gray codes has been given. However, in our application scenario, the given input $\boldsymbol{c}$ is not secret. It is either the given challenge or derived by the known input layer. Based on the adversary model defined in Section 1, the input layer is known to the attacker; hence, $\boldsymbol{c}$ is also of his possession. Therefore, the aforementioned issue does not pose any security weakness in our application.

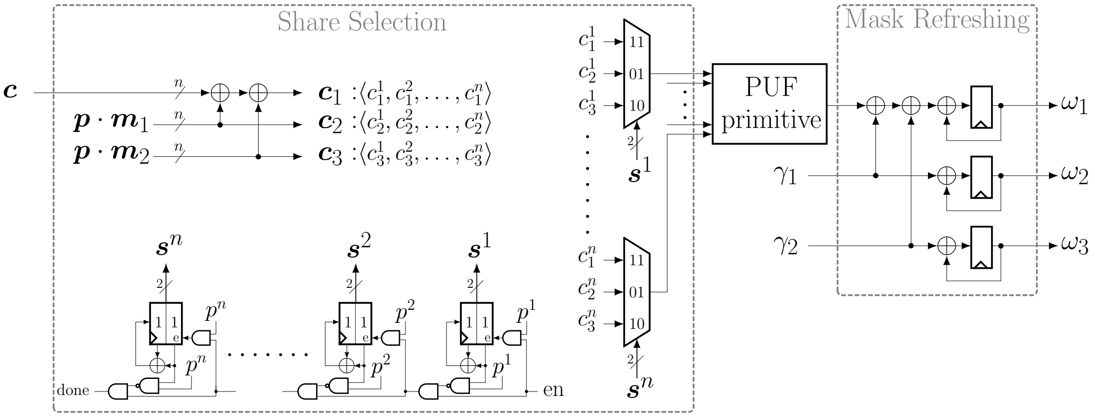
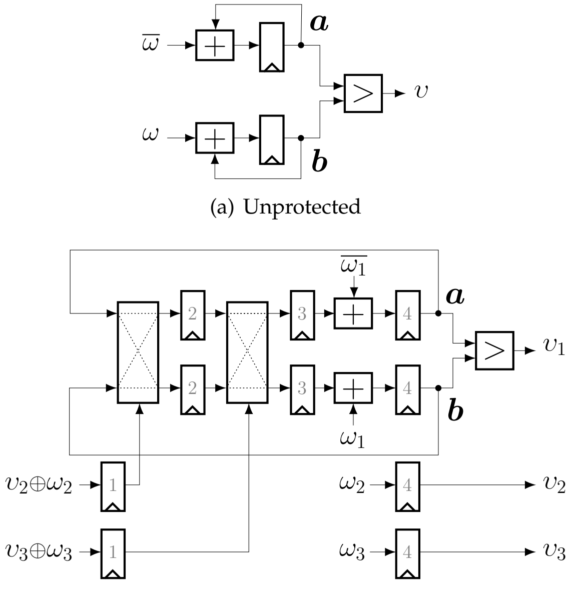


Fig. 5. Secure evaluation, the construction to operate an n-bit PUF primitive in a masked way with 3 shares.

4.4 Reliability

The reliability of PUF primitives is of serious concern, which becomes more problematic when the output of a couple of PUF primitives are combined, e.g., in XOR PUFs. Our construction which operates a PUF primitive in a secure way, can be seen as an XOR PUF with $(s+1)^h$ instances where $h = w(\boldsymbol{p})$, i.e., Hamming weight of signal $\boldsymbol{p}$. By increasing h the reliability is obviously decreased. A trivial solution – as majority voting – is to apply the given challenge $\boldsymbol{c}$ for a couple of times, and count by means of two counters $\boldsymbol{a}$ and $\boldsymbol{b}$ how many times the output $f(\boldsymbol{c}) = \omega = 0$ and how many times $\omega = 1$, respectively. The larger counter would vote for the probable reliable output $\upsilon = 1$ if $\boldsymbol{b} > \boldsymbol{a}$ (see Figure 6(a)). Application of this technique on our construction – where the output is masked –is not straightforward. In other words, the majority voting mechanism should also be adjusted based on the underlying number of output shares. To this end, we start with the 3-share case, i.e., the output of the majority voting is represented by three shares $(\upsilon_1, \upsilon_2, \upsilon_2)$. The goal is to mask the underlying counters. Since we can always write $\upsilon_1 = \upsilon \oplus \upsilon_2 \oplus \upsilon_3$, following the concept of Boolean masking and considering υ_2 and υ_3 as single-bit masks, the counters $\boldsymbol{a}, \boldsymbol{b}$ are swapped if $\upsilon_2 \oplus \upsilon_3 = 1$. Therefore, at any time $\left(\upsilon_1 = \begin{cases} 1 \text{ if } \boldsymbol{b} > \boldsymbol{a} \\ 0 \text{ else} \end{cases}, \upsilon_2, \upsilon_3\right)$ represent the voting output with three shares. The given challenge $\boldsymbol{c}$ is supplied to our construction (Section 4.3) for a couple of times, each time with newly generated masks $\boldsymbol{m}_i$ (and newly generated $\boldsymbol{p}$) leading to the shared output $(\omega_1, \omega_2, \omega_3)$ (see Figure 5). Suppose that counters $\boldsymbol{a}$ and $\boldsymbol{b}$ contain some values associated with the masks (υ_2, υ_3) that should be updated by the newly given tuple $(\omega_1, \omega_2, \omega_3)$. First, $\boldsymbol{a}$ and $\boldsymbol{b}$ are swapped if $\upsilon_2 \oplus \omega_2 = 1$ and again if $\upsilon_3 \oplus \omega_3 = 1$. Then, based on ω_1 one of the counters is incremented resulting in updated counters $\boldsymbol{a}'$ and $\boldsymbol{b}'$ with $\upsilon_2' = \omega_2$ and $\upsilon_3' = \omega_3$ as the updated associated masks. By the swap based on $\upsilon_2 \oplus \omega_2$ and $\upsilon_3 \oplus \omega_3$, we indeed replace the masks (υ_2, υ_3) with (ω_2, ω_3). This process is depicted in Figure 6(b). As shown in the diagram, the result of each swap should be stored in registers. Otherwise, all shares ω_1, ω_2, and ω_3 are involved in a combinatorial circuit violating the non-completeness property of TI. Further, the gray numbers written inside the registers in Figure 6(b) show the order which should be followed when updating the registers. This scheme can straightforwardly be extended to a higher number of shares.



(a) Unprotected

(b) Secure with 3 shares

Fig. 6. Compensating unreliability by majority voting.

5 Design and Analysis

We applied our construction (explained in Section 4) on the selected case study, i.e., $(1,5)$-IPUF, discussed and presented in Section 3. To this end, we built a circuit (shown in Figure 7) consisting of a single instance of 'Share Selection', 'Mask Refreshing', and 'Secure Majority Voting' modules. Since the top and bottom layers have to be evaluated serially, the aforementioned modules are shared. For the 'Secure Majority Voting' module, we adjusted the counters/comparators to deal with 4-bit values, i.e., the user can at most repeat top/bottom evaluation $l = 15$ times to compensate the unreliability. The masked result of the top layer (after repeating l times) is stored in the masked $\boldsymbol{r}^t$ register, and later is used during the evaluation of the bottom layer (also l times).

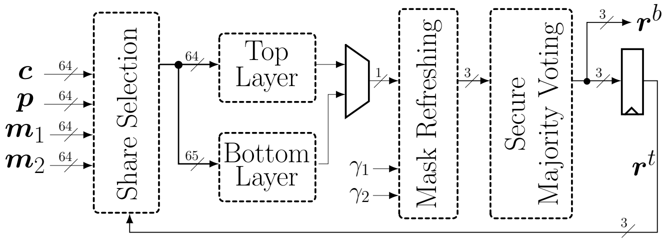


Fig. 7. Application of our construction on Interpose PUF.

TABLE 1
Performance figures of our construction (for Interpose PUF).

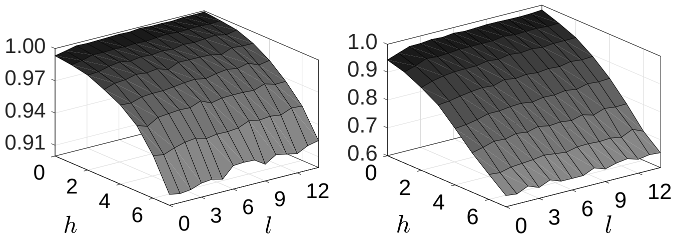
FPGA		**ASIC**		**Latency**
(Spartan-6)		(IBM 130 nm)		
[*FFs*]	[*LUTs*]	[*GE*]	freq. [*MHz*]	clock cycles [#]
178	383	2336	72	$4 + (l + 1)(3^h \cdot 16 + 10)$

In addition to the FPGA-based practical experiments, we synthesized our construction using the IBM 130 nm standard library. Our developed HDL codes are accessible through GitHub[4]. In Table 1, we report the corresponding performance figures including the area footprint and latency. Note that the $(1, 5)$-IPUF and the random number generators are excluded in the reported area. As stated before, the latency of our construction depends exponentially on $h = w(\boldsymbol{p})$ (the number of challenge bits which should be masked) and linearly on l (the number of repeats in the majority voting module). Hence, selecting a small h is of reasonable choices; otherwise, the very high latency might be not affordable.

5.1 Reliability

We would like to highlight that we have not constructed a new PUF to extensively examine its reliability. Instead, we propose a construction that allows operating any PUF primitive in an SCA-secure way. Since in our construction, every PUF primitive is evaluated 3^h times to form a single-bit masked output (by XORing all 3^h results), increasing h would negatively affect the reliability. Therefore, here we practically evaluate the added value of the secure majority voting module. Indeed, we do not give any claim on the reliability of our construction, as it is not a PUF primitive. The unreliability comes from the underlying PUF primitive and is directly transferred to the unreliability of the entire design, that is amplified by our construction depending on h and can (to some extent) be compensated based on l.

We have practically examined this issue using the aforementioned SAKURA-G platform based on a Spartan-6 FPGA. We have supplied the implementation with 1000 randomly-chosen challenges for different values of $l \in \{0, \ldots, 15\}$ and $h \in \{0, \ldots, 7\}$, and repeated this process 10 times. As the reference response we took those belonging to $(l = 15, h = 0)$ and extracted the reliable output as those with at least 6 times the same output. Based on this, we calculated the reliability for other settings of (l, h) which are shown in Figure 8. The results imply that increasing h makes sense for single APUF (top layer), while the reliability suddenly drops for $(1, 5)$-IPUF (final output) for high h. Further, the majority voting can slightly compensate for the unreliability, but it does not fully solve the problem due to the nature of XOR APUF. Hence, both latency and reliability indicate that a small $h \leq 3$ should be chosen, which can still reach a reliability of 90% for high l. This experiment was performed at room temperature (25 °C) when the FPGA's supply voltage was set to 1.1 v. We have repeated this experiment on 3 other FPGAs of the same kind at supply voltages 1 v, 1.1 v, 1.2 v, and temperatures 25 °C, 40 °C and 80 °C. The results are very similar to those shown in Figure 8. The top layer just showed a slightly lower reliability at higher temperatures.



(a) Top layer

(b) Final output

Fig. 8. Reliability of the protected implementation for different number of masked challenge bits $h = w(\boldsymbol{p})$ and number of repeats l.

5.2 Analysis

In order to examine the resistance of our construction to SCA attacks, similar to the analysis formerly illustrated in Section 3 we collected 100 000 averaged power signals, each of which obtained by repeating the measurements 1000 times for a given challenge. For the entire measurements, we set $(l = 4, h = 1)$, i.e., only one challenge bit is masked and the majority voting module decides between 5 runs (for each of the top and bottom layers). A sample collected signal is shown in Figure 9. For each run, the masks $(\boldsymbol{m}_1, \boldsymbol{m}_2)$ and (γ_1, γ_2) as well as $\boldsymbol{p}$ are randomly generated inside the target FPGA by means of a couple of randomly-seeded 31-bit LFSRs with feedback polynomial $x^{31}+x^{28}+1$. For more clarification, during the entire process of collecting signals, $\boldsymbol{m}_1$, $\boldsymbol{m}_2$, $\boldsymbol{p}$ have been generated $(l+1) \cdot 2 \cdot 1000 \cdot 100\,000 = 1\,000\,000\,000$ times, while fresh masks (γ_1, γ_2) were updated at each clock cycle.

In order to conduct the same analysis as we have done for the unprotected implementation (Section 3), we routed out the **masked** output of the top layer $\boldsymbol{r}^t$. This allows us to unmask the output of both top and bottom layers by the PC which communicates with the FPGA to obtain the unmasked values r^t and r^b. It is an essential task, otherwise unmasking inside the FPGA would strongly exhibit leakage about both r^t and r^b. Note that due to the underlying construction (Figure 7) where the APUFs of the bottom layer are not separately evaluated, we could not route out the output of all APUF instances (in contrast to the unprotected analyses in Section 3).

Classifying the collected signals based on the unmasked values r^t and r^b led to the difference of mean signals represented in Figure 9. As marked on the curves, there exist $(l + 1) \cdot 3^h = 15$ points of interest which show a slight dependency of collected signals to the output of the top layer r^t. This turns to $(l + 1) \cdot 3^{h+1} = 45$ points of interest for the

[4] https://github.com/emsec/TI-PUF

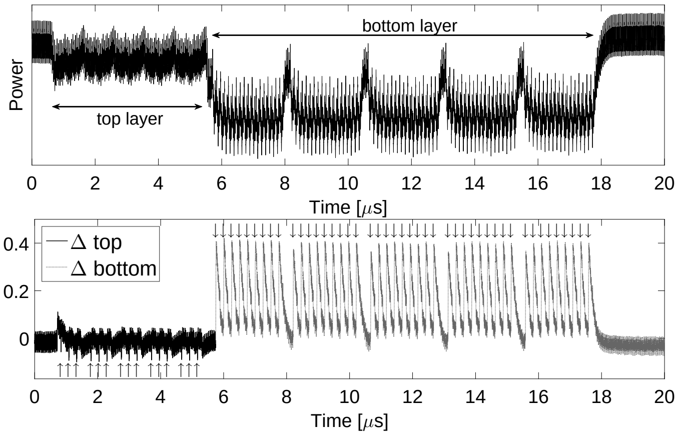
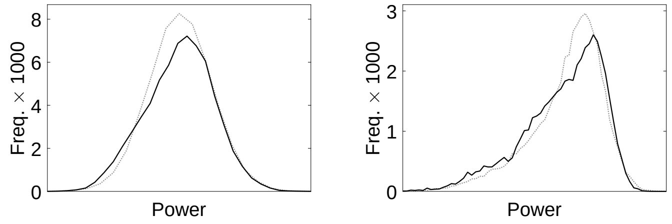


Fig. 9. SCA of our protected $(1,5)$-IPUF, 100 000 averaged signals, each of which obtained by repeating 1000 times, (top) an averaged signal, (bottom) difference of means.



(a) Classified by unmasked r^t at sample point 2 μs

(b) Classified by unmasked r^b at sample point 6 μs

Fig. 10. SCA of our protected $(1,5)$-IPUF using 100 000 averaged signals, histograms at an exemplary point of interest identified in Figure 9.

bottom layer. This increase is due to the fact that during the evaluation of the bottom layer r^t (which is always provided by 3 shares independent of the given $\boldsymbol{p}$) is used as a masked challenge bit. We highlight the different scale of the y-axis in difference-of-mean signals of the unprotected and protected implementations shown in Figure 2(b) and Figure 9(bottom).

We analyzed the histogram of the collected signals at all points of interest marked in Figure 9. They all share roughly the same shape; one example for each layer is shown in Figure 10. It can be seen that the histograms classified either by unmasked r^t or unmasked r^b are not easily distinguishable from each other. This means that by observing the full histograms the adversary is not able to predict the intermediate values (unmasked r^t or r^b) even for a subset of the collected signals. In other words, the SCA leakages cannot provide more information to boost the modeling attacks.

5.3 Comparison

We provide this comparison to be fair in the investigation of our proposed method. Although our TI-based approach applies on any PUF function in the scope of PUF primitives, several PUF protocols claim physical resistance, including SCA resistance like immunity to power or timing attacks and reliability information leakage. It is worth to point out that these two different scopes are not well fitted to compare as each PUF primitive can be utilized independently in various applications such as authentication protocol and key generation. As mentioned in Section 1.1, up to now most PUF primitives have been found vulnerable to modeling attacks and hence have been employed in PUF-based authentication protocols to establish a higher level of security. To exemplify, we can consider a PUF primitive as security primitive like cryptographic ciphers, which are applied in different security protocols. However, in the case of authentication applications, the client can choose between these options in terms of area consumption, latency, and security level, while all of them are in a trade-off. In other words, a high security level always comes at the cost of overheads.

Among various physical resistant PUF-authentication protocols, the most well-observed ones can be recalled as Slender PUF [26], Noise Bifurcation [50], and the Lockdown protocol [49]. The first one, is a PUF-authentication protocol which takes benefit of true random number generators (TRNGs) to send out a random substring of PUF's response as the output accompanying a part of challenge randomly produced inside the chip. However, there can be a chance of physical information leakage through unreliable responses applied in CMA-ES reliability attack [2], [9]. Moreover, in the case of small random substrings, SCA attacks can be probably possible with brute force on the small sent-out part of the response.

Noise Bifurcation protocol that is almost similar to the previous procedure has the main difference with Slender PUF in terms of masking. To be more precise, the token (client) sends out a random substring PUF response to the server in the form of randomly chosen bits from a subdivided partition of PUF response. Afterward, the server applies the masking procedure on the predicted PUF response and retain the actual response. Although this protocol uses a more obfuscated link between the client and the server through the authentication process, the protocol is already attacked [45]. The same probabilistic aforementioned physical attacks are applicable to this protocol as well. Additionally, this protocol as well as Slender PUF allows the collection of unlimited number of CRPs, which is an essential fact for most of SCA attacks.

The more recent scheme, Lockdown protocol [49] (in two versions) applies a custom approach to limit the number of CRPs, thereby providing robustness against MAs. The first version of Lockdown protocol is not practically immune to physical attacks. An SCA adversary is capable of using a fixed challenge for an unlimited number of times to force a token to evaluate the subdivided PUF responses [9]. However, the second version, which makes use of an internal TRNG on the client side as a part of the given challenge to the strong PUF primitive, is yet known to be resistant against SCA attacks. Since it avoids collecting repeated challenges, the adversary is even not able to increase the signal-to-noise ratio in SCA measurements.

5.4 Strength and Limitations

Below we list the strength and limitations of our construction.

- Our proposed technique is a generic SCA countermeasure. In other words, assuming a PUF primitive $f(\boldsymbol{c}) = r$, our technique enables giving a masked representation of the challenge $\boldsymbol{c}$ to the circuit and receive a masked representation of the response r. This does not change the functionality of the underlying PUF and does not alter the robustness or weakness of the design to pure MAs (those which do not use SCA leakages).

- It is not dedicated to any scenario (protocol/strong PUF/weak PUF/etc.). As an example, it can be applied to delay-based PUF protocols and strong PUFs consisting of APUF components.
- It deals with the evaluation of the underlying PUF primitive for the given challenge; it does not deal with either the way the challenges are provided or the way the response is used. In short, any co-ordinate function can be plugged into our construction. If a PUF primitive is a realization of a co-ordinate function, our solution stays applicable to that PUF primitive.
- However, our proposed approach is still not a comprehensive solution against PUFs' threats, especially against the most common threats, i.e., ML attacks. It targets only those which make use of SCA leakages.
- TI-PUF, which is a masking countermeasure against SCA attacks, deals with the reliability shortage, the area overhead, and the cost of latency.
- Despite the fact that such overheads are the unavoidable cost to achieve security, finding a reasonable point in the trade-off has been shown possible. In short, the designer is suggested to select a small $\boldsymbol{h}$ and a large $\boldsymbol{l}$ as long as the resulting latency is affordable.

6 Conclusions

In this work, we have presented a mechanism which allows us to SCA-securely operate any PUF primitive. We should highlight that our introduced concept is independent of the way a PUF primitive is used in an application. Examples include PUF-based protocols, weak PUFs, and strong PUFs. Based on this concept, we have shown the application of our construction on an Interpose PUF and represented its corresponding practical SCA analyses. We first practically showed how SCA leakages can extract intermediate values (i.e., interpose bit) of an original (unprotected) implementation leading to successful straightforward modeling attacks. We further illustrated the result of the same analyses when our construction prevents the exhibition of exploitable SCA leakages.

The main message of this research is a hope to construct SCA-resistant PUFs based on sound and widely-studied countermeasures. This, for sure, comes at the cost of latency and area. We believe that our work initiates further research in this area.

References

[1] G. T. Becker, "On the Pitfalls of Using Arbiter-PUFs as Building Blocks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 8, pp. 1295–1307, 2015.

[2] G. T. Becker, "The Gap Between Promise and Reality: On the Insecurity of XOR Arbiter PUFs," in *Proceedings Cryptographic Hardware and Embedded Systems - CHES 2015*, ser. Lecture Notes in Computer Science, vol. 9293. Springer, 2015, pp. 535–555.

[3] G. T. Becker and R. Kumar, "Active and Passive Side-Channel Attacks on Delay Based PUF Designs," *IACR Cryptology ePrint Archive*, vol. 2014, p. 287, 2014.

[4] T. Beyne and B. Bilgin, "Uniform First-Order Threshold Implementations," in *Selected Areas in Cryptography - SAC 2016*, ser. Lecture Notes in Computer Science, vol. 10532. Springer, 2017, pp. 79–98.

[5] C. M. Bishop, *Pattern recognition and machine learning, 5th Edition*, ser. Information science and statistics. Springer, 2007.

[6] C. Brzuska, M. Fischlin, H. Schröder, and S. Katzenbeisser, "Physically Uncloneable Functions in the Universal Composition Framework," in *Advances in Cryptology - CRYPTO 2011*, ser. Lecture Notes in Computer Science, vol. 6841. Springer, 2011, pp. 51–70.

[7] Q. Chen, G. Csaba, P. Lugli, U. Schlichtmann, and U. Rührmair, "The Bistable Ring PUF: A new architecture for strong Physical Unclonable Functions," in *IEEE International Symposium on Hardware-Oriented Security and Trust - HOST 2011*. IEEE Computer Society, 2011, pp. 134–141.

[8] J. Delvaux, "Machine-Learning Attacks on PolyPUFs, OB-PUFs, RPUFs, LHS-PUFs, and PUF–FSMs," *IEEE Transactions on Information Forensics and Security*, 2019, to appear.

[9] J. Delvaux, "Security analysis of PUF-based key generation and entity authentication," Ph.D. dissertation, Ph. D. Thesis, Katholieke Universiteit Leuven (KULeuven), Leuven, Belgium, 2017.

[10] 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*, ser. Lecture Notes in Computer Science, vol. 8731. Springer, 2014, pp. 451–475.

[11] J. Delvaux and I. Verbauwhede, "Side channel modeling attacks on 65nm arbiter PUFs exploiting CMOS device noise," in *Symposium on Hardware-Oriented Security and Trust – HOST 2013*. IEEE Computer Society, 2013, pp. 137–142.

[12] F. Ganji, "PUFmeter," 2018, technical Report, Technical University Berlin, https://trust-hub.org/resource/software/PUFmeter_version_00.pdf.

[13] Y. Gao, H. Ma, S. F. Al-Sarawi, D. Abbott, and D. C. Ranasinghe, "PUF-FSM: A Controlled Strong PUF," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 37, no. 5, pp. 1104–1108, 2018.

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

[15] B. Gassend, M. van Dijk, D. E. Clarke, E. Torlak, S. Devadas, and P. Tuyls, "Controlled physical random functions and applications," *ACM Trans. Inf. Syst. Secur.*, vol. 10, no. 4, pp. 3:1–3:22, 2008.

[16] H. Groß, R. Iusupov, and R. Bloem, "Generic Low-Latency Masking in Hardware," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 2, pp. 1–21, 2018.

[17] H. Groß, S. Mangard, and T. Korak, "Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order," in *Theory of Implementation Security - TIS@CCS 2016*. ACM, 2016, p. 3.

[18] ——, "An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order," in *Topics in Cryptology - CT-RSA 2017*, ser. Lecture Notes in Computer Science, vol. 10159. Springer, 2017, pp. 95–112.

[19] C. Herder, M. Yu, F. Koushanfar, and S. Devadas, "Physical Unclonable Functions and Applications: A Tutorial," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1126–1141, 2014.

[20] C. Igel, M. Toussaint, and W. Weishui, "Rprop using the natural gradient," in *Trends and applications in constructive approximation*. Springer, 2005, pp. 259–272.

[21] V. Immler, R. Specht, and F. Unterstein, "Your Rails Cannot Hide from Localized EM: How Dual-Rail Logic Fails on FPGAs," in *Cryptographic Hardware and Embedded Systems - CHES 2017*, ser. Lecture Notes in Computer Science, vol. 10529. Springer, 2017, pp. 403–424.

[22] 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 applications," in *VLSI Circuits. Digest of Technical Papers*. IEEE, 2004, pp. 176–179.

[23] 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 Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 10, pp. 1200–1205, 2005.

[24] A. Mahmoud, U. Rührmair, M. Majzoobi, and F. Koushanfar, "Combined Modeling and Side Channel Attacks on Strong PUFs," *IACR Cryptology ePrint Archive*, vol. 2013, p. 632, 2013.

[25] M. Majzoobi, F. Koushanfar, and S. Devadas, "FPGA PUF using programmable delay lines," in *Information Forensics and Security - WIFS 2010*. IEEE, 2010, pp. 1–6.

[26] M. Majzoobi, M. Rostami, F. Koushanfar, D. S. Wallach, and S. Devadas, "Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by Substring Matching," in *2012 IEEE*

Symposium on Security and Privacy Workshops, CA, USA, 2012. IEEE Computer Society, 2012, pp. 33–44.

[27] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.

[28] P. H. Nguyen and D. P. Sahoo, "Lightweight and Secure PUFs: A Survey (Invited Paper)," in *Security, Privacy, and Applied Cryptography Engineering - SPACE*, ser. Lecture Notes in Computer Science, vol. 8804. Springer, 2014, pp. 1–13.

[29] P. H. Nguyen, D. P. Sahoo, C. Jin, K. Mahmood, U. Rührmair, and M. van Dijk, "The Interpose PUF: Secure PUF Design against State-of-the-art Machine Learning Attacks," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2019, no. 4, pp. 243–290, 2019.

[30] P. H. Nguyen, D. P. Sahoo, D. Mukhopadhyay, and R. S. Chakraborty, "Cryptanalysis of Composite PUFs (Extended abstract-invited talk)," in *VLSI Design and Test - VDAT 2014*. IEEE, 2014, pp. 1–2.

[31] S. Nikova, C. Rechberger, and V. Rijmen, "Threshold Implementations Against Side-Channel Attacks and Glitches," in *Information and Communications Security - ICICS 2006*, ser. Lecture Notes in Computer Science, vol. 4307. Springer, 2006, pp. 529–545.

[32] S. Nikova, V. Rijmen, and M. Schläffer, "Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches," *J. Cryptology*, vol. 24, no. 2, pp. 292–321, 2011.

[33] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical One-Way Functions," *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.

[34] O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs, and I. Verbauwhede, "Consolidating Masking Schemes," in *Advances in Cryptology - CRYPTO 2015*, ser. Lecture Notes in Computer Science, vol. 9215. Springer, 2015, pp. 764–783.

[35] U. Rührmair, "Oblivious Transfer Based on Physical Unclonable Functions," in *Trust and Trustworthy Computing, Third International Conference, TRUST,2010*, ser. Lecture Notes in Computer Science, vol. 6101. Springer, 2010.

[36] U. Rührmair, S. Devadas, and F. Koushanfar, *Security Based on Physical Unclonability and Disorder*. Springer New York, 2012, pp. 65–102.

[37] U. Rührmair and D. E. Holcomb, "PUFs at a glance," in *Design, Automation & Test in Europe Conference & Exhibition – DATE 2014*. European Design and Automation Association, 2014, pp. 1–6.

[38] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *Computer and Communications Security, CCS 2010*. ACM, 2010, pp. 237–249.

[39] U. Rührmair, X. Xu, J. Sölter, A. Mahmoud, M. Majzoobi, F. Koushanfar, and W. P. Burleson, "Efficient Power and Timing Side Channels for Physical Unclonable Functions," in *Cryptographic Hardware and Embedded Systems - CHES 2014*, ser. Lecture Notes in Computer Science, vol. 8731. Springer, 2014, pp. 476–492.

[40] D. P. Sahoo, P. H. Nguyen, D. B. Roy, D. Mukhopadhyay, and R. S. Chakraborty, "Side Channel Evaluation of PUF-Based Pseudorandom Permutation," in *2017 Euromicro Conference on Digital System Design (DSD)*. IEEE Computer Society, 2017, pp. 237–243.

[41] D. P. Sahoo, S. Saha, D. Mukhopadhyay, R. S. Chakraborty, and H. Kapoor, "Composite PUF: A new design paradigm for Physically Unclonable Functions on FPGA," in *Hardware-Oriented Security and Trust - HOST 2014*. IEEE Computer Society, 2014, pp. 50–55.

[42] SAKURA, "Side-channel Attack User Reference Architecture," http://satoh.cs.uec.ac.jp/SAKURA/index.html.

[43] G. E. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," in *Proceedings of the 44th Design Automation Conference, DAC, 2007*. IEEE, 2007, pp. 9–14.

[44] Tebelmann, Lars, Pehl, Michael, Immler, and Vincent, "Side-Channel Analysis of the TERO PUF," in *Constructive Side-Channel Analysis and Secure Design*, I. Polian and M. Stöttinger, Eds. Springer, 2019, pp. 43–60.

[45] J. Tobisch and G. T. Becker, "On the Scaling of Machine Learning Attacks on PUFs with Application to Noise Bifurcation," in *Radio Frequency Identification Security and Privacy Issues - RFIDsec 2015*, ser. Lecture Notes in Computer Science, vol. 9440. Springer, 2015, pp. 17–31.

[46] F. Wegener and A. Moradi, "A First-Order SCA Resistant AES Without Fresh Randomness," in *Constructive Side-Channel Analysis and Secure Design - COSADE 2018*, ser. Lecture Notes in Computer Science, vol. 10815. Springer, 2018, pp. 245–262.

[47] N. Wisiol and M. Margraf, "Why attackers lose: design and security analysis of arbitrarily large XOR arbiter PUFs," *J. Cryptographic Engineering*, vol. 9, no. 3, pp. 221–230, 2019.

[48] N. Wisiol, C. Mühl, N. Pirnay, P. H. Nguyen, M. Margraf, J. Seifert, M. van Dijk, and U. Rührmair, "Splitting the Interpose PUF: A Novel Modeling Attack Strategy," *IACR Cryptology ePrint Archive*, vol. 2019, p. 1473, 2019.

[49] M. M. Yu, M. Hiller, J. Delvaux, R. Sowell, S. Devadas, and I. Verbauwhede, "A Lockdown Technique to Prevent Machine Learning on PUFs for Lightweight Authentication," *IEEE Trans. Multi-Scale Computing Systems*, vol. 2, no. 3, pp. 146–159, 2016.

[50] M. M. Yu, D. M'Raïhi, I. Verbauwhede, and S. Devadas, "A noise bifurcation architecture for linear additive physical functions," in *2014 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST, VA, USA, 2014*. IEEE Computer Society, 2014, pp. 124–129.