Modeling Attack Security of Physical Unclonable Functions based on Arbiter PUFs

vorgelegt von

Master of Science (SUNY)

Nils Wisiol

ORCID: 0000-0003-2606-614X

an der Fakultät IV – Elektrotechnik und Informatik

der Technischen Universität Berlin

zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften

– Dr. rer. nat. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr.-Ing. Jan Nordholz
Gutachter: Prof. Dr. Jean-Pierre Seifert
Gutachter: Prof. Dr. Marian Margraf
Gutachter: Prof. Dr. Stefan Katzenbeisser
Gutachter: Prof. Debdeep Mukhopadhyay, Ph.D.

Tag der wissenschaftlichen Aussprache: 25. April 2022

Berlin 2022

The concept of Physical Unclonable Functions (PUFs) is an attempt to base cryptography on physical possession. This is in contrast to conventional cryptography, where the essential difference of participants in a cryptographic protocol is their knowledge of secret information such as keys and nonces.

The literature has a rich body of suggestions for the realization of PUFs. This thesis explores designs of variants and compositions of the Arbiter PUF, which has been introduced in 2002 as a CMOS-compatible, electrical PUF design, and has received much research attention since then, albeit being insecure with respect to modeling attacks.

After revisiting modeling attacks on the Arbiter PUF and XOR Arbiter PUF, we demonstrate attacks against the Lightweight Secure XOR Arbiter PUF, Feed-Forward Arbiter PUF, and the Interpose PUF. We introduce two novel PUF designs, the Beli PUF and the LP-PUF, and analyze their security against modeling attacks. We concluding that the LP-PUF is resilient against currently known modeling attacks.

Physical Unclonable Functions (PUFs) sind ein Versuch, Kryptographie auf der Basis von physikalischem Besitz aufzubauen, anstatt, wie bisher üblich, Teilnehmer eines kryptographischen Protokolls anhand ihrer Kenntnis oder Unkenntnis eines kryptographischen Geheimnisses wie beispielsweise Schlüsseln oder Nonces zu unterscheiden.

In der Literatur gibt es viele Vorschläge für die Implementierung von PUFs. Diese Dissertation untersucht PUF Designs, welche auf Variationen und Kombinationen des Arbiter PUF Designs beruhen. Arbiter PUFs wurden 2002 als CMOS-kompatibles, elekrtisches PUF Design vorgestellt und haben in der Wissenschaft viel Aufmerksamkeit erhalten, wenn auch die Arbiter PUF unsicher hinsichtlich Modellierungsangriffen ist.

Nach der Vorstellung bisheriger Modellierungsangriffe auf die Arbiter PUF und XOR Arbiter PUF zeigen wir Angriffe auf die Lightweight Secure XOR Arbiter PUF, Feed-Forward PUF und Interpose PUF. Anschließend stellen wir zwei neue PUF Designs, die Beli PUF und die LP-PUF vor und untersuchen ihre Sicherheit gegen Modellierungsangriffe. Wir folgern, dass die LP-PUF resilient gegen bekannte Modellierungsangriffe ist.

Contents

1. Introduction

In all cryptographic applications deployed today, what distinguishes the legitimate user from an adversary is the knowledge of secret information, usually called the *secret key*. Such secret keys are found everywhere where cryptography is used, including in computers of microscopic scale embedded in digital door keys, credit cards, and passports. As cryptographic algorithms and implementations matured over the years, attacks based on weaknesses of the used ciphers and software code became harder, and gaining access to the secret key itself became an important attack strategy, as a revealed secret key causes all security guarantees of the employed scheme to collapse [AK96; Mah97; KK99].

To remove this weakness, a branch of research on *Physical Unclonable Functions (PUFs)* emerged [Pap+02], where the difference of the legitimate user and adversary is not defined by knowledge, but by possession of a physical object. The physical object, called PUF token, is assumed to exhibit highly individual physical behavior when prompted with an electric or optical input. Further, it is assumed to be *(physically) unclonable*, meaning that it is inherently impossible to produce two exactly identical tokens. The envisioned secret-free cryptography shall be based on this unique response behavior of each individual unclonable token.

While such PUF-based, secret-free cryptography is by definition immune against attacks that recover any secret key, it may be possible to study the individual behavior of a PUF token and extrapolate its behavior using a mathematical model, in which case it is impossible for a remotely connected party to tell original PUF token and mathematical model apart. In this case, the security of any cryptographic application would collapse, just like in the case of a leaked secret key.

An important tool to create such models from observed behavior is *machine learning*, a highly parameterized and generic approach to create predictions from observed data by using specialized software. In the past two decades, machine learning has emerged as the tool of choice for *PUF modeling attacks* [Gas+04; Rüh+10], where an attacker creates a model of a PUF token which can be used to predict PUF responses with high accuracy, thereby breaking the token's security. Studies on modeling attacks hence represent an essential part of research on PUFs and secret-free cryptography.

In the past 20 years, a large variety of PUF designs and attack strategies have been developed. In this thesis, we revisit milestone designs and attacks, present new analysis strategies, and conclude with a novel PUF design secure against all known modeling attacks.

In Chapter 2, we formally introduce the notion of Physical Unclonable Functions and discuss desired security properties, most importantly, existential unforgeability under chosen and known message attacks.

In Chapter 3, we discuss the design rationale of the Arbiter PUF and XOR Arbiter

PUF, and detail on a variety of different attacks on this PUF design. This includes a discussion of a mathematical model for XOR Arbiter PUF behavior, comparison of different machine learning attacks (including different attacker models), a summary of physical attacks, an extension of the XOR Arbiter PUF that enables mitigation of some attacks by increasing security parameters.

As a different extension of the XOR Arbiter PUF, we discuss XOR Arbiter PUFs with input transformations in Chapter 4. Here, we introduce the correlation attack on the Lightweight Secure PUF and propose a mitigation against it, the Permutation PUF.

In Chapter 5, we discuss the Interpose PUF design as an extension of the XOR Arbiter PUF and demonstrate the Splitting Attack, which reduces the security level of the Interpose PUF to the level of an XOR Arbiter PUF (with respect to the corresponding attacker model).

Subsequently, we discuss modeling attacks on Feed Forward Arbiter PUFs in Chapter 6.

In Chapter 7, we discuss the Beli PUF, an extension of the Arbiter PUF design with respect to its inner works (as opposed to composing larger PUFs out of Arbiter PUFs). We derive a model of the Beli PUF and demonstrate a modeling attack based on it.

Chapter 8 proposes the LP-PUF, a PUF design that we show is resilient against all modeling attacks discussed in this thesis, both for the known and chosen message attack model.

Chapter 9 gives an overview over pypuf, a Python library that includes most code and data used in this thesis. Finally, Chapter 10 summarizes our work and suggests future research directions.

List of Publications

This dissertation is based on the following publications. The contributions of the author are detailed below.

- [WM19] Nils Wisiol and Marian Margraf. "Why Attackers Lose: Design and Security Analysis of Arbitrarily Large XOR Arbiter PUFs". In: *Journal of Cryptographic Engineering* 9.3 (Sept. 2019), pp. 221–230. ISSN: 2190-8516. URL: `https://doi.org/10.1007/s13389-019-00204-8` (visited on 10/11/2019)

  The authors of the paper jointly developed the theorems and proofs. The author of this dissertation further contributed implementation, data collection, and data analysis of the empirical PUF stability values.

- [Wis+20a] Nils Wisiol et al. "Breaking the Lightweight Secure PUF: Understanding the Relation of Input Transformations and Machine Learning Resistance". In: *Smart Card Research and Advanced Applications*. Ed. by Sonia Belaïd and Tim Güneysu. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 40–54. ISBN: 978-3-030-42068-0

  In this work, the author of this dissertation provided an efficient implementation of the attack, developed the analysis methodology, collected and analyzed the exper-

imental data, and developed the countermeasure for the attack, the Permutation PUF.

- [WP20] Nils Wisiol and Niklas Pirnay. "Short Paper: XOR Arbiter PUFs Have Systematic Response Bias". In: *Financial Cryptography and Data Security*. Ed. by Joseph Bonneau and Nadia Heninger. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 50–57. ISBN: 978-3-030-51280-4

  The author of this dissertation contributed data collection and analysis as well as a theoretical explanation for the observed phenomenon. He further added the corresponding interpretation and drew the suggested conclusions.

- [Wis+20b] Nils Wisiol et al. "Splitting the Interpose PUF: A Novel Modeling Attack Strategy". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (June 2020), pp. 97–120. ISSN: 2569-2925. URL: `https://tches.iacr.org/index.php/TCHES/article/view/8584` (visited on 09/01/2020)

  The author of this dissertation developed the attack strategy of this work jointly with Christopher Mühl. He contributed the formalization of the attack, as well as the implementation, data collection, data analysis, and interpretation of the attack complexity. The authors of this paper jointly contributed the interpretation of the results.

- [Wis+21a] Nils Wisiol et al. *Neural-Network-Based Modeling Attacks on XOR Arbiter PUFs Revisited*. Tech. rep. 555. 2021. URL: `https://eprint.iacr.org/2021/555` (visited on 04/28/2021)

  The author of this dissertation discovered the discussed implementation error as well as the replication experiments (with the exception of the replication of Mursi et al. [Mur+20], which was developed jointly). He further developed the improved LR attack, conducted the hyperparameter optimization, data collection, and data analysis. He also contributed the presentation of the results.

- [Agh+22] Anita Aghaie et al. "Security Analysis of Delay-Based Strong PUFs with Multiple Delay Lines". Jan. 2022

  In this work, Johannes Tobisch and the author of this dissertation jointly authored the mathematical model of the Beli PUF and the modeling attacks. The author was responsible for the formal analysis of the Beli PUF delay model as well as for implementation and analysis of the modeling attacks.

- [Wis21] Nils Wisiol. *Towards Attack Resilient Arbiter PUF-Based Strong PUFs*. Tech. rep. 1004. 2021. URL: `https://eprint.iacr.org/2021/1004` (visited on 08/19/2021)

  This work was authored without collaboration.

- [Wis+21b] Nils Wisiol et al. *Pypuf*. Zenodo. June 2021. URL: `https://zenodo.org/record/3901410` (visited on 07/07/2021)

The author of this dissertation is the maintainer of pypuf and, at the time of writing (Version 3.2.1), contributed virtually all simulation, attack, and metric implementations.

Additionally, the author also collaborated in the following publications.

- [WM18] Nils Wisiol and Marian Margraf. *Attacking RO-PUFs with Enhanced Challenge-Response Pairs*. Tech. rep. 862. 2018. URL: `https://eprint.iacr.org/2018/862` (visited on 10/11/2019)

  The author developed and formalized the attack.

2. Physical Unclonable Functions

Physical[1] Unclonable Functions (PUFs) have been proposed as a method to tie cryptography to a computing device or possession of a physical token. While the exact notion varies, PUFs are usually physical objects that represent a noisy implementation of a mathematical function. Several different use cases have been envisioned, including FPGA IP-protection [SS06; Gua+07], secure key storage [HBF07], and authentication [Pap+02]; also using PUFs as cryptographic primitives has been considered [Brz+11].

In the literature, several different notions of PUFs exist. In this thesis, we opt to use the notions given by Armknecht et al. [Arm+16], which summarize and generalize earlier notions found in the literature [Pap+02; Gas+02; Gua+07; Arm+09; Arm+11; Brz+11; Mae13]. In this framework, a *PUF token* (also *PUF instance* or, abbreviated, *PUF*) is modeled as a *probabilistic mapping* $f : \mathcal{D} \to \mathcal{R}$, often with $\mathcal{D} = \{-1, 1\}^n$ and $\mathcal{R} = \{-1, 1\}$. The notion of probabilistic mapping is used in favor of using functions to stress that due to noise, the same input may yield different responses on different occasions. We write $r \leftarrow f(c)$ to denote that a *response* (a.k.a. *output*, *behavior*) $r \in \mathcal{R}$ is generated by PUF f on *challenge* (a.k.a. *input*, *stimulus*) $c \in \mathcal{D}$.

We define a *PUF design* to be the list of all PUFs originating from the same manufacturing process along with the according probability distribution.

Some authors distinguish *strong* and *weak* PUFs depending on the size of their domain $\mathcal{D}$, where in the case of strong PUFs, the size of the domain is exponential in the security parameter, and sub-exponential otherwise. PUFs considered in this work have exponentially-sized domains, i.e. are considered strong PUFs.

2.1. Security Properties and Metrics

Armknecht et al. [Arm+16] define a variety of PUF security properties with respect to a security parameter λ and attackers using probabilistic polynomial time algorithms. Not all PUF applications require that all of the following properties hold; security definitions predating these general notions considered only subsets.

We re-state some of the security requirements defined by Armknecht et al. [Arm+16] and derive the security metrics of *reliability*, *uniqueness*, and *bias* to be used in this work. The reason to use *metrics* instead of *requirements* is that we are usually unable to conduct an asymptotic analysis of the relevant quantities, but have to resort to simulations fixed

[1] Sometimes, a distinction between *Physical* Unclonable Functions and *Physically* Unclonable Functions is made, where the former refers to physical functions that are unclonable (in some sense), and the latter refers to functions (in some sense) which are physically unclonable. We note that both notions can be abbreviated as PUF; the precise meaning of PUF in the context of this work is given by the formal definitions in this chapter.

to security parameters from a certain set. Motivated by the structure of the Arbiter PUF (introduced in Chapter 3), we simplify the notions to the case $\mathcal{D} = \{-1, 1\}^n$ and $\mathcal{R} = \{-1, 1\}$. In this work, the challenge length n is closely related to the security parameter λ. For the Arbiter PUF, we have $\lambda = n$; for other PUF designs, n is one of several security parameters.

For the sake of easier notation, we chose to model bits as values from $\{-1, 1\}$ rather than $\{0, 1\}$. All results can be transformed into $\{0, 1\}$ notation by "encoding" bits with the function $\rho : \{-1, 1\} \to \{0, 1\}$, where $\rho(1) = 0$ and $\rho(-1) = 1$. This way, we can write $\rho(b) = 1/2 - b/2$ and have the convenient property $\rho(b_1 \cdot b_2) = \rho(b_1) \oplus \rho(b_2)$, i.e. a group homomorphism, where $\oplus$ denotes addition modulo 2 and $\cdot$ denotes multiplication over $\mathbb{Z}$.

Definition 1 (Intra-Distance Requirement)**.** [Arm+16] Whenever a single PUF is repeatedly evaluated with a fixed input, the maximum distance between the corresponding outputs is at most δ_1. That is for any PUF f and any $c \in \mathcal{D}$, it holds that

$$\Pr\left[\max_{i \neq j} \{\mathrm{d}(r_i, r_j)\} \leq \delta_1 \mid c \in \mathcal{D}, r_i \leftarrow f(c) \text{ for } 1 \leq i \leq t\right] = 1 - \epsilon(\lambda).$$

We restrict the intra-distance requirement to PUFs mapping n bits to 1 bit and set $\delta_1 = 0$ and $t = 2$ to obtain the notion of *reliability*.

Definition 2 (Reliability)**.** For a fixed PUF instance f and challenge c, we define *reliability* to be the probability that f will return the same response when queried twice with c, i.e.

$$\Pr\left[r_1 = r_2 \mid r_i \leftarrow f(c), i \in \{1, 2\}\right].$$

For a fixed PUF instance f, we define the *reliability* $R(f)$ to be

$$R(f) = \mathrm{E}_{c \in \mathcal{D}}\left[\Pr\left[r_1 = r_2 \mid r_i \leftarrow f(c), i \in \{1, 2\}\right]\right].$$

Some reliability of the PUF is needed for it to be practically usable. Higher reliability values are desirable, with values of 1 meaning perfectly reliable. For low reliability, PUF responses are no longer reproducible, depriving the PUF of one of its key features. Note that this definition explicitly includes the case that different challenges c may have different reliability values, which plays an important role on reliability-based attacks on XOR Arbiter PUFs (Section 3.8).

Definition 3 (Inter-Distance II Requirement)**.** [Arm+16] Whenever multiple PUFs are evaluated on a single, fixed input, the minimum distance among them is at least δ_3. That is for any PUF f_i for $1 \leq i \leq m$ and any $c \in \mathcal{D}$, we have

$$\Pr\left[\min_{i \neq j} \{\mathrm{d}(r_i, r_j)\} \geq \delta_3 \mid c \in \mathcal{D}, r_i \leftarrow f_i(c) \text{ for } 1 \leq i \leq m\right] = 1 - \epsilon(\lambda).$$

We restrict the inter-distance II requirement to PUFs mapping n bits to 1 bit and set $\delta_3 = 1$ and $m = 2$ to obtain the notion of *uniqueness*.

Definition 4. [Uniqueness] For a PUF design $\mathcal{P}$, we define the *uniqueness* $\mathrm{U}(\mathcal{P})$ to be

$$\mathrm{U}(\mathcal{P}) = 1 - 2\,\mathrm{E}_{f_i \sim \mathcal{P}}\left[\left|\frac{1}{2} - \Pr_{c \in \mathcal{D}}\left[r_1 = r_2 \mid r_i \leftarrow f_i(c)\right]\right|\right].$$

The uniqueness measures the difference in behavior of randomly chosen PUF tokens of the same PUF design. If they show correlation or similarity in their behavior, an attacker can use this fact to guess PUF responses of one instance with the assistance of another, unrelated PUF instance. The optimal value of $\mathrm{U}(\mathcal{P})$ hence is 1, arising from the case when $r_1 = r_2$ happens with probability $1/2$.

Definition 5. For a PUF design $\mathcal{P}$, we define the *bias* $\mathrm{B}(\mathcal{P})$ to be

$$\mathrm{B}(\mathcal{P}) = \mathrm{E}_{f \sim \mathcal{P}}\left[\mathrm{E}_{c \in \mathcal{D}}\left[f(c)\right]\right],$$

with zero being the optimal value, given that the responses are from $\{-1, 1\}$. (For PUFs with more than one response bit, which are not studied in this work, Armknecht et al. [Arm+16] generalize this to the notion of *min-entropy*.) A low bias is required to reduce the probability that the attacker guesses correctly by choosing the most likely response.

Armknecht et al. [Arm+16] define the notion of *unforgeability* capturing the idea that the PUF's responses should not be predictable based on observed examples of its behavior. While this is also referred to as *unpredictability*, the notion as defined by Armknecht et al. is closely related to the security notions in the context of digital signatures schemes, where "unforgeability" is used. The notion can be parameterized in two dimensions, one specifying precisely which responses of the PUF are unforgeable, the other specifying which capabilities the attacker precisely has available. We give an informal overview over the different flavors of unforgeability, referring the reader to Armknecht et al. [Arm+16] for precise definitions.

In its strongest sense, unforgeable means that the attacker is unable to predict the behavior of the PUF for *any* (unseen) challenge. This case is referred to as *existential unforgeability* (*EUF*). The weaker version of *universal unforgeability* (*UUF*) asserts that the attacker is unable to predict the behavior of the PUF for a challenge chosen uniformly at random.

There are two important classes of attacks the adversary can use to predict ("forge") PUF responses. The weaker attack is the *known message attack* (*KMA*), where the attacker has knowledge of the challenge to the PUF, but cannot choose. The stronger attack setting is the *chosen message attack* (*CMA*), where the attacker can choose the input to the PUF.

We obtain the four PUF unforgeability notions of EUF-CMA, EUF-KMA, UUF-CMA, and UUF-KMA, with EUF-CMA being the strongest notion of security among the four, and UUF-KMA being the weakest.

Even stronger than EUF-CMA unforgeability are the notions of *indistinguishability* (different PUF instances cannot be distinguished based on their observed behavior) and *pseudorandomness* (PUF instances cannot be distinguished from random functions based on their behavior) [Arm+16].

Related to the notion existential unforgeability (EUF) is the question if observed behavior reveals information about the PUF behavior on *related*, i.e. slightly modified, challenge inputs [GTS18].

Definition 6. For a PUF design $\mathcal{P}$, we define *bit sensitivity* $\mathrm{BS}_i(\mathcal{P})$ to be the probability that the response of the PUF changes when the i-th input bit is flipped (denoted $x^{\oplus i}$),

$$\mathrm{BS}_i(\mathcal{P}) = \mathrm{E}_{f\in\mathcal{P}}\left[\Pr_{c\in\mathcal{D}}\left[f(c) \neq f(c^{\oplus i})\right]\right].$$

A proper bit sensitivity for all input bit positions $i \in \{1,\ldots,n\}$ is necessary (but not sufficient) to prevent attackers from predicting responses when the response to a related challenge is known. The optimal value is $1/2$.

While bit sensitivity can only capture one aspect of existential unforgeability, it is an interesting value, as it also relates to the *learnability* of a PUF (when viewed as a Boolean function, Section 2.3.3). Furthermore, Arbiter PUFs have a non-optimal bit sensitivity, so this is an important metric for the analysis of PUF designs based on Arbiter PUFs.

2.2. Attacker Model

The standard attacker model for PUFs is based on the promise of increased hardware security of PUF tokens, compared to the traditional approach of cryptographic hardware extended by secure key storage. Hence, in the standard PUF attacker model, the adversary gets physical access to the PUF token for some limited amount of time [Wis+20b]. During this time, a large number of different attacks is conceivable. While this work considers attacks operating on the challenge-response behavior of the PUF, the assumed physical access to the PUF hardware also enables other important classes of attacks which we summarize in Section 2.4.

For the attacks using information about the challenge-response behavior of the PUF, we distinguish chosen message attacks (CMA) and known message attacks (KMA). While the physical access to the PUF generally motivates the study of CMAs, in some cases it is also appropriate to study the weaker KMAs, as done in prior works [Gas+02; Gua+07; Arm+11]. This can be motivated by suggestions for PUF-based authentication protocols which mitigate CMAs by technical means [Maj+12; Yu+16; Yu+14] or by the fact that some KMAs are successful even without choosing challenges (Table 10.1).

2.3. Modeling Attacks

Attacks that build a mathematical model from observed challenge-response behavior of the PUF are the main focus of this work. Models that can predict PUF responses with high accuracy can be used as drop-in replacement for the PUF token and can be used to make remote parties believe that the attacker is in possession of the PUF when they are actually not. Security against modeling attacks is captured by the notion of unforgeability as defined in Section 2.1. There exists a large body of research on PUF

modeling attacks in the literature, which can roughly be divided into modeling attacks using (practical) machine learning algorithms, modeling attacks using provable methods, and modeling attacks that use specialized machine learning algorithms.

2.3.1. Machine Learning Attacks

Machine learning is the field of using algorithms for making predictions based on observed data. Based on a number N of observed examples given as pairs of input $x_i \in \mathcal{D}$ and output $y_i \in \mathcal{R}$, called the training set $T = \{(x_i, y_i) \mid 1 \leq i \leq N\}$, a *machine learning algorithm* outputs a *model* $M : \mathcal{D} \to \mathcal{R}$. The model has *high accuracy* if the predictions $M(x)$ match observed data well, with x chosen uniformly at random in $\mathcal{D}$ but omitting known examples from T.

The concept of machine learning matches the attacker definition of the notion of PUF unforgeability (Section 2.1) and is thus well-suited to study PUF modeling attacks. Machine learning was also outside the context of PUFs portrayed as the "opposite" of cryptography [ODo14].

One of the first machine learning algorithms is the Perceptron [MP69], which is also used for the first PUF modeling attack [Gas+04]. While this modeling attack and follow-up work [Rüh+10] used machine learning algorithms that can only output models of a certain class of functions, with increasing popularity of *deep learning*, also PUF modeling attacks started to use machine learning algorithms that are not restricted to a certain class of models [Yas+16; SBC19; Mur+20; Wis+21a].

Machine learning modeling attacks are usually demonstrated on data obtained either by simulating PUF behavior, or by collecting data from a PUF prototype, or both. They are hence fixed to a certain set of used security parameters. The run time of the attacks is often measured as wall-clock time and thus highly dependent on the used software and hardware, operating system, and configuration.

2.3.2. Specialized Attacks

In contrast to fully generic approaches and approaches only restricted to a broad class of model functions, machine learning algorithms can also be combined with classical algorithms to design highly specialized attacks. This is often useful when attacking PUF embedded in additional logic or when attacking PUFs composed of several different PUF primitives. Delvaux [Del19] demonstrated a collection of such attacks, targeting five PUFs with accompanying obfuscation logic. In Section 3.8, we present a specialized reliability-based attack on the Arbiter PUF. Also the *Splitting Attack* on the Interpose PUF [Wis+20b] presented in Section 5.1 can be understood as a specialized attack in this sense.

2.3.3. Provable Attacks

A third important class of PUF modeling attacks is constituted by applying the theory of *probably approximately correct* (*PAC*) learning algorithms to mathematical models of PUFs. Let $\mathcal{B}_n$ denote the set of all Boolean functions $f : \{-1, 1\}^n \to \{-1, 1\}$. Informally,

a subset $\mathcal{C} \subseteq \mathcal{B}_n$ is said to be *PAC-learnable* if there exists an efficient probabilistic algorithm A that, given access to examples of a function $f \in \mathcal{C}$, outputs, with high probability, a hypothesis that achieves high prediction accuracy. $\mathcal{C}$ is referred to as the *concept class* of A.

For a more formal definition, we quantify the notions of high probability and high accuracy, as well as detail on the access that Algorithm A has on the function f.

Definition 7. [ODo14] A probabilistic algorithm A learns a concept class $\mathcal{C}$ with error $0 < \varepsilon < \frac{1}{2}$ if for any $f \in \mathcal{C}$, with probability at least 90%, A outputs a hypothesis h which satisfies

$$\Pr_{x \in \{-1,1\}^n} [f(x) \neq h(x)] \leq \varepsilon.$$

Algorithm A can either have access to *random examples*, i.e. A can draw pairs $(x, f(x))$ where x is uniformly random in $\{-1, 1\}^n$, or have access to *queries*, i.e. A can request the value $f(x)$ for any $x \in \{-1, 1\}^n$ of its choice.

The access models correspond to the definitions of known message attack (KMA) and chosen message attack (CMA) of Section 2.1. The choice of 90% is arbitrary; choosing other numbers such as 51% or larger will yield equivalent definitions [ODo14]. The study of PAC learning was initiated by Valiant [Val84] and first used in the context of PUFs by Ganji, Tajik, and Seifert [GTS16]. In contrast to empirical machine learning attacks, algorithms in the PAC learning framework are analyzed on concept classes derived from mathematical models of PUFs, rather than on simulated or real-world data. Hence, analysis of PAC learning algorithms can yield information on the asymptotic complexity of such modeling attacks, rather than just information on a fixed security parameters.

In follow up work, Ganji, Tajik, and Seifert [GTS18] used the PAC-framework to demonstrate modeling attacks against the Bistable Ring PUF using the LMN-Algorithm [LMN93] that are provably resilient to noisy examples in the training set. The proof relies on the assumption that the PUF exhibits only a very limited number of influential bits [Gan+16]. This assumption is justified by observing FPGA data provided by Yamamoto et al. [Yam+14].

The observation that certain properties of the Boolean function implemented by PUFs can give way to provable learnability enabled the use of testing function metrics to gauge the PUF modeling attack resilience [Ngu+16; GFS19].

Using a semi-automated approach, Chatterjee, Mukhopadhyay, and Hazra [CMH20] extended this approach by using metrics in the PAC-framework to analyze composite PUF designs for their PAC learnability. This way, they found the Interpose PUF (Chapter 5), the Feed-Forward Arbiter PUF (Chapter 6), and others to be PAC-learnable.

2.4. Hardware Security

As PUFs are considered an alternative to cryptography based on secret keys [Gas+02; Pap+02] to mitigate hardware attacks [AK96; Mah97; KK99], such attacks are included in the attacker model (Section 2.2) and need to be considered when assessing the security of PUFs.

Here, we review hardware attacks on SRAM PUFs [Gua+07] and (XOR) Arbiter PUFs (Chapter 3); later we restrict our attention to modeling attacks in the known message attack or chosen message attack model (Section 2.1) for the remainder of this work.

For SRAM PUFs[2], Helfmeier et al. [Hel+13] and Nedospasov et al. [Ned+13] demonstrated attacks that are not only able to extract the supposedly secret SRAM value using a variety of methods, but also showed that a SRAM *hardware clone* can be produced. This goes beyond modeling attacks presented in this work not only by using hardware attack surface, but also by producing a functionally identical piece of hardware, rather than just a simulation based on a model. Further extraction methods for SRAM PUFs were later presented by Lohrke et al. [Loh+16].

Tajik et al. [Taj+14] demonstrated an attack based on photon emission analysis on the Arbiter PUF (Chapter 3) that can create a high-precision model even when challenges and responses to the PUF are unknown. Their methodology has data complexity linear in the security parameter. For these reasons, their methodology must be considered a very powerful attack, albeit requiring some preparation on the hardware under attack.

Power and other side-channel attacks also can be threat to PUF implementations. Aghaie and Moradi [AM20] proposed a general technique to protect PUF implementations against attacks based on power side channel analysis.

Ruhrmair [Ruh20] offers a taxonomy with respect to hardware attacks on PUFs.

[2]Note that SRAM PUFs $f : \mathcal{D} \to \mathcal{R}$ have domains of small size and are thus trivially insecure in the EUF-KMA model considered in this work, but constitute an important area of research for PUF hardware attacks.

3. XOR Arbiter PUF

3.1. Arbiter PUF

An *Arbiter PUF* is a PUF based on electrical signal delays that vary from chip to chip. It was introduced by Gassend et al. [Gas+02], who argue that the due to mask, temperature, and pressure variations during manufacturing, differences in delays can exceed 5% of the total delay value and are increasing with shrinking device sizes.

To utilize this intrinsic variability of Arbiter PUFs, two electrical signals start at the same time and travel through a circuit of n *stages* for a chosen design parameter $n \in \mathbb{N}$. At each stage, one bit of the n-bit challenge determines whether the signals travel straight through or are interchanged. With the hardware implementation in mind, the stages are sometimes called *multiplexers*. At the end of this *arbiter chain*, an *arbiter* determines on which input a signal arrives first and outputs the result as the *response*. The schematics of an Arbiter PUF are shown in Figure 3.1.1.

Gassend et al. [Gas+04] showed how to model the behavior of Arbiter PUFs using the *additive delay model*. In this model, it is assumed that the total delay of the signal in the Arbiter PUF is the sum of all individual delays along its path. This enabled them to conduct a modeling attack using the Perceptron algorithm [MP69] and to demonstrate high prediction accuracy of their model. Even though their experiments were conducted on FPGAs, this indirectly confirms the accuracy of the additive delay model. A detailed evaluation of the parameters of the additive delay model was done by Lim et al. [Lim+05]. A CMOS implementation of the Arbiter PUF has been studied by Maes et al. [Mae+12]. Delvaux and Verbauwhede [DV13] and Tajik et al. [Taj+14] later confirmed the accuracy of the additive delay model also for proof-of-concept designs implemented in CMOS and FPGA technology, respectively; Utz, Tobisch, and Becker [UTB16] confirmed the

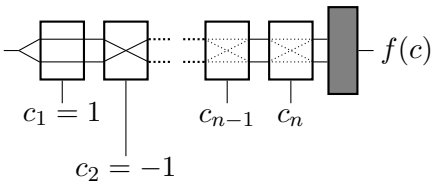


Figure 3.1.1.: Schematic representation of an Arbiter PUF. After setting up the challenge $c_1, \ldots, c_n$ on the bottom (one for each *stage*), an electrical signal is input on the left-hand side and travels though the circuit. The *arbiter* shown on the right-hand side (gray box) detects if a signal arrives at the top or bottom input first and outputs the response bit $f(c)$ accordingly.

accuracy of the model for commercial Arbiter PUFs.

The additive delay model of Arbiter PUFs is convenient to write when using $\{-1, 1\}$ as bit values (instead of the more traditional $\{0, 1\}$), as done by Wisiol et al. [Wis+21a].

An Arbiter PUF with n stages, once set up with a challenge $c \in \{-1, 1\}^n$, accumulates signal propagation delays for two electrical signals traveling though the stages of the Arbiter PUF. At the i-th stage, the additional delays for the two signals are either d_i^{TT} and d_i^{BB} (if $c = -1$ and the signals travel straight through) or d_i^{TB} and d_i^{BT} (if $c = 1$ and the signals are interchanged). We define the total accumulated delay *after* the i-th stage as d_i^{T} for the top output of that stage, and d_i^{B} for the bottom output of that stage. Additional delays are added at each stage, i.e.

$$d_i^{\mathrm{T}} = \begin{cases} d_{i-1}^{\mathrm{T}} + d_i^{\mathrm{TT}} & (c = -1)\,, \\ d_{i-1}^{\mathrm{B}} + d_i^{\mathrm{BT}} & (c = 1)\,, \end{cases}$$
$$d_i^{\mathrm{B}} = \begin{cases} d_{i-1}^{\mathrm{B}} + d_i^{\mathrm{BB}} & (c = -1)\,, \\ d_{i-1}^{\mathrm{T}} + d_i^{\mathrm{TB}} & (c = 1)\,. \end{cases}$$

The initial delays are zero, $d_0^{\mathrm{T}} = d_0^{\mathrm{B}} = 0$. The delay *difference* after the i-th stage is $\Delta D_i = d_i^{\mathrm{T}} - d_i^{\mathrm{B}}$; we abbreviate the delay difference after the n-th stage to $\Delta D_n = \Delta D_{\mathrm{Model}}$. This value depends on the given challenge, often denoted c, so we sometimes also write $\Delta D_{\mathrm{Model}}(c)$ to emphasize on this dependency.

Theorem 8. *For $n \in \mathbb{N}$ and given stage delay values $d_i^{\mathrm{TT}}, d_i^{\mathrm{TB}}, d_i^{\mathrm{BT}}, d_i^{\mathrm{BB}} \in \mathbb{R}^+$, with $1 \le i \le n$, there exists $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$ such that for all $c \in \{-1, 1\}^n$,it holds that*

$$\Delta D_{\mathrm{Model}}(c) = \langle w, x \rangle + b,$$

where $x = (x_i)_i = \left(\prod_{j=i}^{n} c_j\right)_i$. For even n we have

$$w = \frac{1}{2} \begin{pmatrix} d_1^{\mathrm{TT}} - d_1^{\mathrm{BB}} - d_1^{\mathrm{BT}} + d_1^{\mathrm{TB}} \\ -d_2^{\mathrm{TT}} + d_2^{\mathrm{BB}} + d_2^{\mathrm{BT}} - d_2^{\mathrm{TB}} + d_1^{\mathrm{TT}} - d_1^{\mathrm{BB}} + d_1^{\mathrm{BT}} - d_1^{\mathrm{TB}} \\ d_3^{\mathrm{TT}} - d_3^{\mathrm{BB}} - d_3^{\mathrm{BT}} + d_3^{\mathrm{TB}} - d_2^{\mathrm{TT}} + d_2^{\mathrm{BB}} - d_2^{\mathrm{BT}} + d_2^{\mathrm{TB}} \\ \vdots \\ -d_{n-2}^{\mathrm{TT}} + d_{n-2}^{\mathrm{BB}} + d_{n-2}^{\mathrm{BT}} - d_{n-2}^{\mathrm{TB}} + d_{n-3}^{\mathrm{TT}} - d_{n-3}^{\mathrm{BB}} + d_{n-3}^{\mathrm{BT}} - d_{n-3}^{\mathrm{TB}} \\ d_{n-1}^{\mathrm{TT}} - d_{n-1}^{\mathrm{BB}} - d_{n-1}^{\mathrm{BT}} + d_{n-1}^{\mathrm{TB}} - d_{n-2}^{\mathrm{TT}} + d_{n-2}^{\mathrm{BB}} - d_{n-2}^{\mathrm{BT}} + d_{n-2}^{\mathrm{TB}} \\ -d_n^{\mathrm{TT}} + d_n^{\mathrm{BB}} + d_n^{\mathrm{BT}} - d_n^{\mathrm{TB}} + d_{n-1}^{\mathrm{TT}} - d_{n-1}^{\mathrm{BB}} + d_{n-1}^{\mathrm{BT}} - d_{n-1}^{\mathrm{TB}} \end{pmatrix},$$
$$b = 1/2 \left(d_n^{\mathrm{TT}} - d_n^{\mathrm{BB}} + d_n^{\mathrm{BT}} - d_n^{\mathrm{TB}}\right);$$

Similar formulae exist for odd n.

The proof is relegated to Appendix A.

As the final response bit of the Arbiter PUF is determined by whether there is a signal at the top or bottom input of the arbiter element first, we can conclude that for an n-bit Arbiter PUF with physical parameters $w \in \mathbb{R}^n$ and given challenge $c \in \{-1, 1\}^n$, the response r will be

$$r = \operatorname{sgn} \Delta D_{\mathrm{Model}} = \operatorname{sgn} \left(\langle w, x \rangle + b\right),$$

using the above definition for $x \in \{-1,1\}^n$. Assuming there is no evaluation noise, from this model we can derive a function $f : \{-1,1\}^n \to \{-1,1\}$ which models the Arbiter PUF behavior by setting $f(c) = \langle w, x\rangle + b$.

The additive delay model can thus be understood as a hyperplane in n dimensions, dividing the edges of the Boolean cube $\{-1,1\}^n$ (and, by extension, $\mathbb{R}^n$) into two regions labeled by the -1 and 1 responses of f. The boundary between the two regions is a linear, or in case of nonzero bias, affine subspace of $\mathbb{R}^n$. In this sense, the model can be understood as *linear*; the behavior of the PUF can be described as *linearly separable*.

For a given challenge c, the corresponding value of x according to above definition is referred to as *feature vector,* as it facilitates the modeling of the Arbiter PUF as linearly separable function. Note that the values of w and b only depend on the physics of the PUF, whereas the values of x only depend on the given challenge.

The observation that Arbiter PUFs can be modeled as hyperplanes in $\mathbb{R}^n$ lead Gassend et al. [Gas+04] to use the Perceptron algorithm to create models of Arbiter PUF instances, as it is suitable to learn linearly separable functions.

If we assume certain properties of the Arbiter PUF manufacturing process, such as that the delays are distributed according to a Gaussian distribution [Gas+02], we can also derive statements about the distribution of the w and b in the class of all Arbiter PUFs.

Corollary 9. *Assuming the delays $d_i^{\mathrm{TT}}, d_i^{\mathrm{TB}}, d_i^{\mathrm{BT}}, d_i^{\mathrm{BB}}$ for an Arbiter PUF are distributed according to a Gaussian with all the same mean and variance $\sigma^2_{\mathrm{Model}}$, the weights w and bias b of $\Delta D_{\mathrm{Model}} = \langle w, x\rangle + b$ are distributed according to*

$$w_0 \sim \mathcal{N}\left(0, \sigma^2_{\mathrm{Model}}\right), \quad w_i \sim \mathcal{N}\left(0, 2\sigma^2_{\mathrm{Model}}\right), \quad b \sim \mathcal{N}\left(0, \sigma^2_{\mathrm{Model}}\right).$$

Implementation of arbiter chains in hardware produce noisy responses. To obtain valid predictions for the behavior and security of Arbiter PUFs, this noise needs to be modeled, but is not covered by the additive delay model. For any challenge $c \in \{-1,1\}^n$, we thus model the *noisy* PUF delay value ΔD as

$$\Delta D(c) = \Delta D_{\mathrm{Model}}(c) + \Delta D_{\mathrm{Noise}}, \tag{3.1.1}$$

where $\Delta D_{\mathrm{Noise}}$ is a Gaussian random variable with zero mean and a variance $\sigma^2_{\mathrm{Noise}}$ depending on measurement conditions and implementation. Note that this model does not make any assumptions about the distribution or model of $\Delta D_{\mathrm{Model}}(c)$. Delvaux and Verbauwhede [DV13] were the first to propose this model and proved its accuracy in a noise-based side-channel attack (Section 3.8). The accuracy of the model was later confirmed in follow-up studies [Taj+14; Bec15].

3.2. XOR Arbiter PUF

The successful modeling attack against the Arbiter PUF by Gassend et al. [Gas+04] using the Perceptron algorithm motivated the introduction of a new PUF design. As the Perceptron algorithm is restricted to modeling functions which are linearly separable but

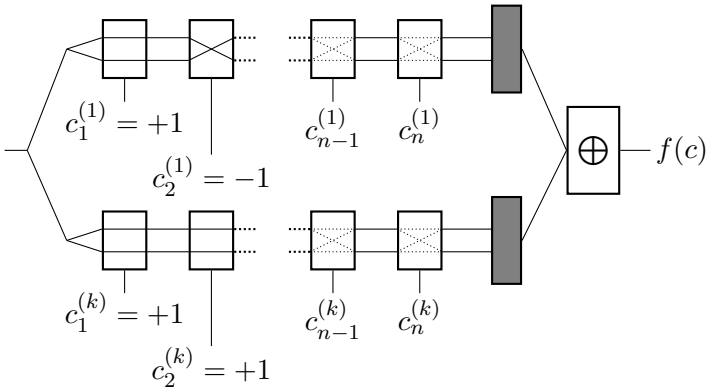


Figure 3.2.1.: Schematic representation of an 2-XOR Arbiter PUF; the scheme can be extended to include more arbiter chains. After the challenge is set up, a rising edge is input on the left-hand side, with the arbiters at the end of each chain (shown as gray rectangles) measuring if the top line or bottom line shows a signal first. During this process, the $n \cdot k$ challenge bits decide at each stage (white rectangles) if the signal paths are crossed or not. The arbiter result bits of each line are then XORed and output on the right-hand side.

cannot model the parity function [MP69], Suh and Devadas [SD07] proposed the XOR Arbiter PUF. In the XOR *Arbiter PUF*, a number of k Arbiter PUFs is implemented in parallel. Any given challenge is applied to all k Arbiter PUFs, which are evaluated separately. Then, the parity of the resulting k response bits is the final 1-bit response of the XOR Arbiter PUF. The schematics of an XOR Arbiter PUF are depicted in Figure 3.2.1.

While each Arbiter PUF has linearly separable responses, the XOR Arbiter PUF does not have this property. Instead, the decision boundary cannot be modeled by an n-dimensional hyperplane anymore. In this sense, adding XOR operations increases the *nonlinearity* of the model.

Nevertheless, the additive delay model can be extended to XOR Arbiter PUFs. Note that when using $\{-1, 1\}$ as bit values, the parity function can be written as the real product of bit values. Hence, using the noise-free variant of the model and writing $f^{(l)}$ with $1 \leq l \leq k$ and $f^{(l)}(c) = \operatorname{sgn}\left\langle w^{(l)}, x \right\rangle + b^{(l)}$ for the individual Arbiter PUFs in a k-XOR Arbiter PUF modeled by f, we obtain

$$\begin{aligned} f(c) &= \prod_{l=1}^{k} f^{(l)}(c) = \prod_{l=1}^{k} \operatorname{sgn}\left[\left\langle w^{(l)}, x \right\rangle + b^{(l)}\right] \\ &= \operatorname{sgn}\left[\prod_{l=1}^{k} \left(\left\langle w^{(l)}, x \right\rangle + b^{(l)}\right)\right]. \end{aligned} \tag{3.2.1}$$

3.3. Metrics

The quality of the XOR Arbiter PUF can be measured by the security metrics of bias, reliability, and uniqueness as introduced in Section 2.1. The unforgeability of XOR Arbiter PUFs will be discussed after this section.

3.3.1. Systematic Bias of XOR Arbiter PUFs

XOR Arbiter PUF bias can be analyzed by expanding the product of the additive delay model (Equation (3.2.1)) to observe the resulting threshold term. In order to focus on the *systematic* bias of XOR Arbiter PUF designs, we assume each Arbiter PUF to be independently chosen and *unbiased*[1].

The term $\prod_{l=1}^{k} \langle w^{(l)}, x \rangle = \prod_{l=1}^{k} \sum_{i=1}^{n} w_i^{(l)} x_i$ in Equation (3.2.1) is a polynomial of degree k over variables x_i that take values in $\{-1, 1\}$. Hence, $f(c)$ is a polynomial threshold function of degree at most k, including some monomials of the form $x_i^k \cdot \prod_{l=1}^{k} w_{l,i}$. If (and only if) k is even, these monomials contribute to the threshold term. When k is odd, no product will degenerate into a constant term, i.e. perfectly unbiased Arbiter PUFs will yield a perfectly unbiased XOR Arbiter PUF. As an example, a 2-XOR Arbiter PUF can then be modeled as

$$r(c) = r_1(c) \cdot r_2(c) = \operatorname{sgn}\left[\sum_{\substack{i,j \\ i \neq j}} w_{1,i} w_{2,j} x_i x_j + \sum_{\substack{i,j \\ i = j}} w_{1,i} w_{2,j}\right]. \tag{3.3.1}$$

It can be seen that even assuming unbiased building blocks, we obtain a non-zero threshold term of $\sum_{i=1}^{n} w_{1,i} w_{2,i}$. While the expectation of this value in the manufacturing process is zero, a high variance causes the 2-XOR Arbiter PUF to likely have significant bias. In other words, any 2-XOR Arbiter PUF consisting of two *unbiased* arbiter chains is biased with probability 1.

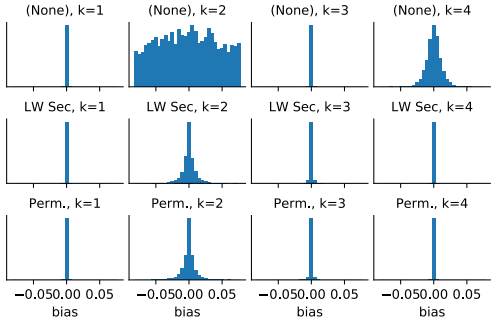
Theorem 10. *The responses of independently chosen unbiased Arbiter PUFs queried on the same challenge are not statistically independent.*

Proof. Let r_1, r_2 be models of unbiased Arbiter PUFs with parameters $w_i^{(1)}$ and $w_i^{(2)}$ for $1 \leq i \leq n$ chosen independently at random and $w_0^{(1)} = w_0^{(2)} = 0$ as defined in (Theorem 8). As demonstrated in Equation (3.3.1), the bias of $r_1(c) \cdot r_2(c)$ is non-zero and hence[2] $\Pr[r_1(c) \cdot r_2(c) = 1] \neq 1/2$. However, assuming statistical independence we have

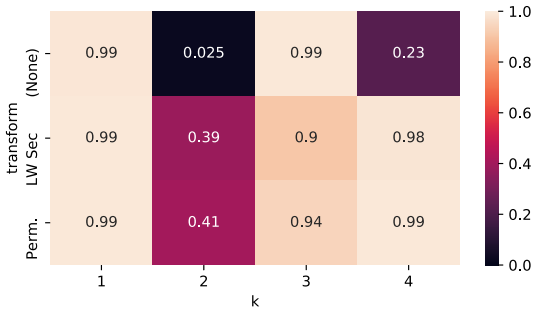
$$\Pr[r_1(c) \cdot r_2(c) = 1] = \Pr[r_1(c) = r_2(c)] = 1/2.$$

[1] The effect observed in this section can also be observed using Arbiter PUFs which are biased according to bias distribution as shown in Corollary 9. We chose to show the effect on unbiased Arbiter PUFs to exhibit the effect more clearly.

[2] An approximation of $\mathrm{E}_c[r(c)]$ in dependence of the bias values can be obtained using the Berry-Esseen-Theorem to approximate $\sum_{i,j} w_{1,i} w_{2,j} x_1 x_2$ for $i \neq j$ as a Gaussian random variable with variance σ^2 over uniformly chosen random challenges, resulting in $\mathrm{E}_c[r(c)] \approx \operatorname{erf}\left(\frac{\sum_{i=1}^{n} w_{1,i} w_{2,i}}{\sigma\sqrt{2}}\right)$; the value $\sum_{i=1}^{n} w_{1,i} w_{2,i}$ in turn follows (in the manufacturing random process) a distribution composed of



(a) Histogram of bias estimates. A bias value of zero represents perfectly unbiased responses.



(b) Proportion of instances that passed the NIST frequency test at significance level 1%.

Figure 3.3.1.: Analysis results for simulated 64-bit k-XOR Arbiter PUFs, k-Lightweight Secure PUFs (Chapter 4) and k-Permutation XOR Arbiter PUFs (Chapter 4) build from unbiased Arbiter PUFs. For each type and size, 5000 instances were sampled and queried with one million uniformly random challenges each.

□

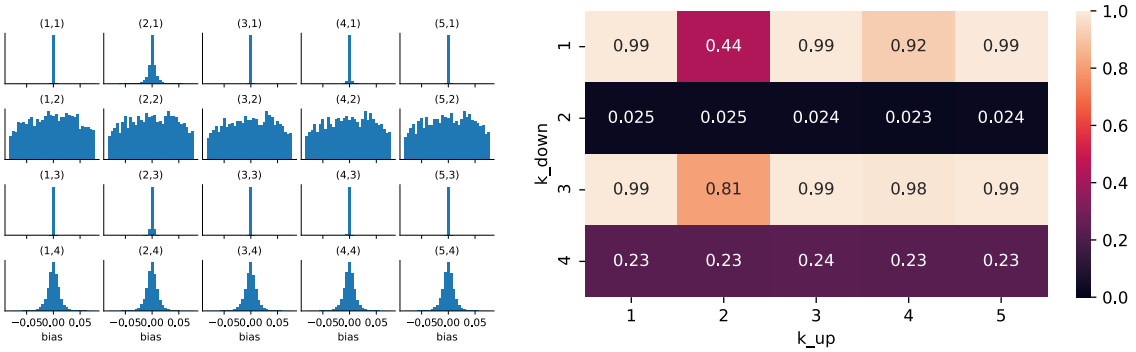
Such bias may be problematic for novel designs based on a single XOR Arbiter PUF, but with transformation of the challenge (Chapter 4) as well as for designs based on several XOR Arbiter PUFs, like the Interpose PUF (Chapter 5). As an example, Theorem 10 can be extended to cover all input transformations that result in the same challenge for each arbiter chain.

We emphasize that this analytical result holds regardless of any implementation weakness and must be considered a *systematic* weakness of the XOR Arbiter PUF design.

We confirmed the systematic XOR Arbiter PUF bias in simulations (Chapter 9) for different XOR Arbiter PUF sizes and input transformations, including the Interpose PUF. All simulations are based on the additive delay model with standard Gaussian weights and were conducted using *unbiased* arbiter chains. The distribution of the systematic bias is based on sampling 100 instances each; the bias of each instance is estimated using 1,000,000 responses to uniformly random challenges.

In Figure 3.3.1 we show the estimated bias distribution for XOR Arbiter PUFs and Lightweight Secure PUFs, which confirm our theoretical findings. As expected, the systematic bias is only present for PUFs with an even number of arbiter chains, while PUFs with an odd number of arbiter chains remain (systematically) unbiased. The bias variance becomes smaller as k increases. The statistical significance of these findings can be confirmed by applying a bias test like the one specified in NIST's SP-800-22 test suite [Ruk+10]: while our simulation passes the tests on 99% of XOR Arbiter PUF instances whenever k is odd; it fails for almost all instances when $k = 2$, and fails for the majority

the sum of product-normal distributions, which has increasing variance for increasing n. Extending the setting, for higher (but even) k the distribution narrows as the variance of the product-normal distribution narrows. The later effect can be observed in our simulations, cf. Figure 3.3.1.



(a) Histogram of bias estimates. A bias value of zero represents perfectly unbiased responses.

(b) Proportion of instances that passed the NIST frequency test at significance level 1%.

Figure 3.3.2.: Analysis results for simulated 64 bit $(k_{\text{up}}, k_{\text{down}})$-Interpose PUF instances build from unbiased Arbiter PUFs. For each size, 5000 instances were sampled and queried with one million uniformly random challenges each.

of instances when $k = 4$ (see Figure 3.3.1). This is also in line with our theoretical findings and simulation results, as we expect the effect to become smaller as k increases.

We hence recommend using an odd number of arbiter chains to avoid potential additional attack surface and especially discourage the use of two or four parallel chains. These recommendations also apply whenever (XOR) Arbiter PUFs are used as building blocks for larger PUFs, such as is the case in the Interpose PUF, as bias in intermediate values can result in increased predictability.

The bias distribution also suggests that the input transformation as done by the Lightweight Secure PUF [MKP08] compensates the systematic bias to some extend, which may be a contributing factor to the increased machine learning resistance [RBK10; Wis+19] of the Lightweight Secure PUF. On the other hand, the Lightweight Secure PUF and Permutation XOR Arbiter PUF (4.3) seems to introduce bias for the case $k = 3$. Such effects should be considered when designing novel input transformations.

Our findings also extend to the Interpose PUF [Ngu+19], which is a combination of two XOR Arbiter PUFs and was designed to be resilient against all state-of-the-art modeling attacks, while being CMOS-compatible. Consisting of two interposed XOR Arbiter PUFs, our simulation shows that the "down" XOR Arbiter PUF plays an important role for the systematic bias, while the "up" XOR Arbiter PUF only has minor influence on it (see Figure 3.3.2). We discuss the design of the Interpose PUF and applicable attacks in Chapter 5.

Given these findings, we provide additional evidence for the original author's advice to use the Interpose PUFs with an odd number of arbiter chains in the lower layer. Furthermore, as our findings are applicable any XOR Arbiter PUF, we extend the parameter recommendation to include the upper layer as well.

3.3.2. Implementation

Several academic works implement the Arbiter PUF using FPGA and/or CMOS technology and studied the resulting reliability, uniqueness, bias, and bit sensitivity metrics.

Lim et al. [Lim+05] examined the uniqueness of CMOS-implemented Arbiter PUFs with respect to PUFs implemented on the same wafer and implemented on different wafers and found no significant difference. Uniqueness values in their study averaged at 40-50%, significantly below the ideal value of 100% (Definition 4). In terms of reliability, they report values between 97% and 100% for a 64-bit Arbiter PUF, depending on operating voltage and temperature. Experiments have been done with 100,000 challenge-response pairs per PUF instance and per environmental condition, with 11 repeated measurements for determination of the reliability.

Maes et al. [Mae+12] report higher uniqueness values of 95% in their CMOS implementation at only slightly decreased reliability of 96% to 100% (again, depending on the temperature). However, their study only considers a relatively small number of 8192 challenge-response pairs.

Utz, Tobisch, and Becker [UTB16] analyzed the metrics of commercially available Arbiter PUFs and found a near-ideal average uniqueness of 99.8% (with narrow variance) based on 51,200 challenge-response pairs collected from 998 PUF tokens, each. They also found a high reliability 96.3%.

The reliability of XOR Arbiter PUFs decreases exponentially with the number of arbiter chains involved. This can be demonstrated using the additive delay model on the XOR Arbiter PUF, and has also been confirmed empirically by Zhou, Parhi, and Kim [ZPK17] on a very large data basis.

The bit sensitivity of Arbiter PUFs was studied by Siddhanti et al. [Sid+19] and is far from ideal. In particular, challenge bits to stages early in the delay lines have little influence on the response, whereas stages later in the delay lines have high influence. Near-optimal influence values of challenge bits are only found around the middle of the delay lines. This has led to the development of the Lightweight Secure PUF (Chapter 4).

Given the results above, the Arbiter PUF can be considered to fulfill fundamental PUF requirements, albeit the achieved reliability can be a concern, in particular in regard with usage in large PUF designs such as the XOR Arbiter PUF.

pypuf (Chapter 9) ships data measured from the Arbiter PUF implementation of Mursi et al. [Mur+20] and Aghaie and Moradi [AM21]. To the best of our knowledge, no other Arbiter PUF data is publicly available.

3.4. Logistic Regression Attack

After the successful attacks on the Arbiter PUF using the Perceptron algorithm [Gas+04], it was found that the Logistic Regression algorithm (LR) can model Arbiter PUFs using fewer examples during training [Söl09; Rüh+10]. In contrast to the Perceptron, LR also does not require the data to be linearly separable and can thus be used to model the XOR Arbiter PUF, which originally was suggested to mitigate the Perceptron modeling attack [SD07]. It has been confirmed that LR is also able to create models on data

obtained from real hardware [HMV12; Rüh+13b]. It can be scaled up to very high parameters, while the time and data complexity was observed to grow exponentially [TB15]. An open-source version of the LR attack, implemented in a modern machine learning framework, is available in pypuf (Chapter 9). In addition to differences caused by the new implementation, it also features some detail improvements [Wis+21a], on which we detail below.

For the purpose of the Logistic Regression Attack, we use the noise-free k-XOR Arbiter PUF model as stated in Equation (3.2.1),

$$f(c) = \operatorname{sgn} \prod_{l=1}^{k} \left(\left\langle w^{(l)}, x \right\rangle + b^{(l)} \right),$$

where $c \in \{-1, 1\}^n$ is the challenge and x the corresponding feature vector.

We assume that the attacker is given the size of the k-XOR Arbiter PUF. The value of n is known as the challenge-interface of the PUF is assumed to be public. Further, the attacker is given N examples of the PUF's challenge-response behavior as training set, $T = ((c_1, r_1), \dots)$, where $c_i \in \{-1, 1\}^n$, $r_i \in \{-1, 1\}$. This corresponds to the known message attack definition of Section 2.1.

The supervised learning process starts with a randomly initialized model, i.e. with $n \cdot k$ values drawn from a Gaussian distribution. We define the model function m as

$$\begin{aligned} \widehat{m}\left(w^{(1)}, \dots, w^{(k)}, c\right) &= \prod_{l=1}^{k} \left(\left\langle w^{(l)}, x \right\rangle + b^{(l)} \right); \\ m\left(w^{(1)}, \dots, w^{(k)}, c\right) &= \operatorname{sgn} \widehat{m}\left(w^{(1)}, \dots, w^{(k)}, c\right). \end{aligned}$$

Note that the model explicitly depends on the weights $w^{(l)}$ for $1 \leq l \leq k$.

The attacker is now faced with the task of finding weights $w^{(l)}$ such that the resulting model m models the examples in T as accurately as possible. For any given observed challenge c and response r, we define the *likelihood* that the model $\widehat{m}$ predicts the observed behavior correct to be

$$\frac{1}{1 + \exp\left(-r\widehat{m}\left(w^{(1)}, \dots, w^{(k)}, c\right)\right)}.$$

While $r \in \{-1, 1\}$, the model value $\widehat{m}$ can take values in $\mathbb{R}$. Whenever we have $r = \operatorname{sgn} \widehat{m}\left(w^{(1)}, \dots, w^{(k)}, c\right)$, the likelihood is above $1/2$ and rapidly approaches 1 with increasing absolute value of $\widehat{m}$. Likewise, whenever $r \neq \operatorname{sgn} \widehat{m}\left(w^{(1)}, \dots, w^{(k)}, c\right)$, the likelihood is below $1/2$ and rapidly approaches 0 with increasing absolute value of $\widehat{m}$.

To obtain a value that measures the overall correctness of the model across the training set, we define the *total likelihood* as the product all individual likelihoods, i.e.,

$$\prod_{(c,r) \in T} \frac{1}{1 + \exp\left(-r\widehat{m}\left(w^{(1)}, \dots, w^{(k)}, c\right)\right)}.$$

This is motivated by the fact that each correctly predicted response will contribute for this value to converge towards 1, whereas each incorrectly predicted value will contribute for this value to converge towards zero.

The task of finding weights $w^{(l)}$ that define a well-predicting model can now be reduced to the task of findings weights $w^{(l)}$ that increase the total likelihood as defined above.

In logistic regression, the weights $w^{(l)}$ are determined in an iterative manner by conducting a gradient ascent. For the gradient ascent, the derivative of the function to be optimized must be computed, which is difficult due to the large product term. This problem can be circumvented by considering the *log likelihood* instead. By applying the logarithm function, computing the derivative becomes easier, while the maxima of the function remain unchanged due to the monotonicity of the logarithm.

To summarize, the likelihood that the model predicts correctly is maximized when the log-likelihood

$$\begin{aligned} L\left(w^{(1)}, \ldots, w^{(k)}\right) &= \ln \prod_{(c,r)\in E} \frac{1}{1+\exp\left(-r\widehat{m}\left(w^{(1)}, \ldots, w^{(k)}, c\right)\right)} \\ &= \sum_{(c,r)\in E} \ln \frac{1}{1+\exp\left(-r\widehat{m}\left(w^{(1)}, \ldots, w^{(k)}, c\right)\right)} \end{aligned}$$

is maximized.

To find such a maximum, the gradient ascent algorithm starts at a random point and then "walks" on the surface defined by the log likelihood function towards the direction of steepest ascent. The stride length of such a step is determined by heuristic algorithms. The direction of the steps is computed using the gradient defined by

$$\operatorname{grad} L\left(w^{(1)}, \ldots, w^{(k)}\right) = \left(\frac{\partial L}{\partial w_1^{(1)}}, \ldots, \frac{\partial L}{\partial w_n^{(k)}}\right)',$$

where

$$\frac{\partial L}{\partial w_\nu^{(\mu)}} = \sum_{(c,r)\in T} \left(\frac{r \cdot \left(\prod_{j\in\{1,\ldots,k\}\setminus\{l\}} \left\langle w^{(l)}, x\right\rangle\right) \cdot x_\nu^{(\mu)}}{1+\exp\left(-r\cdot\widehat{m}\left(w^{(1)}, \ldots, w^{(k)}, c\right)\right)} \right).$$

To summarize, the task of finding weights $w^{(l)}$ that provide high-accuracy predictions for the given training set T can be reduced to iteratively computing the gradient of the log likelihood function and heuristically adjusting the weights into the direction of the steepest ascent. The process can be terminated when certain (heuristic) criteria are met, e.g. when the stride length becomes small or when the accuracy computed on either the training set or a separate validation set becomes high.

There exists several implementations of the LR algorithm for attacking Arbiter PUFs and XOR Arbiter PUFs. To our knowledge the first implementation was provided by Rührmair et al. [Rüh+10] and is implemented using Python and numpy. In this work, we present an implementation using the Keras machine learning framework.

The LR attack has become an important tool for the security analysis of delay-based PUFs and is used in a large number of different works, including in the recent proposal of the Interpose PUF [Ngu+19] and an attack on it [Wis+20b]. The Keras implementation of the LR attack in this work will serve as a baseline for comparing the performance of LR with various other attacks.

A large-scale study of Tobisch and Becker [TB15] determined training set sizes for the LR attack that yield optimal results, i.e., have the lowest training times, and minimal training set sizes, for which the attack was observed to work at least once, which we confirmed and will use for comparison. To provide for a fair comparison with the neural network attacks shown in Section 3.6, we are using our Keras-based implementation of the LR attack. Differences in run time hence may not only be caused by the usage of different CPUs, but also by optimization differences in the implementations. Nevertheless, we obtain the same number of required CRPs, which indicates that our implementation behaves similar to the one of Tobisch and Becker.

To increase the training performance of the original LR algorithm, we modified some details. First, to reduce the number of epochs required for training, we introduced the usage of mini batches, where the network is updated with the gradient not only after evaluating the complete training data, but several times in each epoch. This allows for convergence using fewer epochs, and thus (except in corner cases), for shorter training time, but one must be careful to not choose the batch size too small. Too small batch sizes can lead to noisy gradient values, which will in turn perform unhelpful updates on the network.

Second, we use Adam optimizer [KB17] instead of the originally used resilient backpropagation, as the latter works poorly together with the use of mini batches.

Third, we apply the tanh activation function to each of the k delay values computed by the respective arbiter chains, i.e. we change the model function[3] from

$$c \mapsto \tanh\left(\prod_{l=1}^{k}\left(\langle W_l, x\rangle + b_l\right)\right) \qquad \text{to} \qquad c \mapsto \tanh\left(\prod_{l=1}^{k}\tanh\left(\langle W_l, x\rangle + b_l\right)\right).$$

This change was motivated by the observation that in the traditional LR network, a single arbiter chain can have large influence on the absolute value of the final output. However, in the electrical circuit, no analogon to the absolute value exists. Instead, XOR Arbiter PUF model weights can be scaled using positive scalars without affecting the computed function. We speculate that different influences can hamper the training process, as weight updates during backpropagation may be applied predominantly to influential arbiter chains. Applying the tanh function ensures a more equalized influence of all arbiter chains on the final output.

By the introduction of the additional tanh activation function, our attack does not

[3]Note that tanh is implicitly used in the likelihood function above, as

$$\frac{1}{1+e^{-x}} = \frac{1}{2} + \frac{1}{2}\tanh\frac{x}{2}.$$

n	k	CRPs	success rate	duration (max. threads)	mean success accuracy	memory	[TB15]
64	4	30k	10/10	<1 min @ 4	95.5%		<1 min
64	5	260k	10/10	4 min @ 4	96.9%		<1 min
64	6	2M	20/20	<1 min @ 4	97.6%		1 min
64	7	20M	10/10	3 min @ 4	98.5%		55 min
64	8	150M	10/10	28 min @ 4	96.4%		391 min
64	9	500M	7/10	14 min @ 40	96.8%	132 GiB	*2266 min
64	10	1B	6/10	41 min @ 40	95.7%	197 GiB	-

Table 3.1.: Empirical results on learning simulated XOR Arbiter PUFs obtained using our Keras-based implementation of the LR attack. Reference values of Tobisch and Becker [TB15] use up to 16 cores. (⋆ Result obtained using a different number of CRPs.)

fulfill the definition of logistic regression anymore[4] . In a slight abuse of terminology, we will refer to this version as the *improved LR attack*. A sketch of the network structure used in the attack is displayed in Figure 3.6.4d.

Using our Keras-based implementation together with these improvements, we could increase the performance of the LR algorithm (with respect to wall clock time, i.e., elapsed real-time from beginning to end of the model training procedure), which is summarized in Table 3.1. We found that the largest proportion of the performance gain is due to Keras, which allows for optimized and highly parallel computing, and to a smaller extent due to our improvements. In a comparison of attack performance on 64-bit 5-XOR Arbiter PUFs, we found that our implementation of the original LR attack achieved a success rate of 16/20, while the improved version yielded 18/20 successful, with all other parameters being equal. We did not study in more detail the performance of the two versions depending on the choice of mini batch size, number of CRPs, learning rate, etc.

Similar to the attacks based on neural networks shown in Section 3.6, performance of the improved LR attack crucially depends on the choice of hyperparameters, in particular on a good combination of learning rate and batch size. The number of required epochs is also heavily influenced by any early stopping logic, which may depend on the validation accuracy or loss. We thus expect that the wall-clock performance and the number of required epochs can be further reduced, e.g., by using a systematic approach to find optimal hyperparameters. On the other hand, we expect that for the LR attack, the data complexity cannot significantly be reduced by hyperparameter tuning.

Our numbers confirm once more [Rüh+10; Wis+20b] that the LR attack requires a number of CRPs in the training set that grows exponentially with the number of employed XORs in the target XOR Arbiter PUF. In Fig 3.6.3, we show the required training set size. Based on a fitted function $k \mapsto \alpha \cdot e^k$ using the least squares method, we predict

[4]One can argue that already the LR attack on XOR Arbiter PUF hardly fits the textbook definition of a logistic regression.

the number of required CRPs for $k = 10$ is 1.3 billion, for $k = 11$ is 3.6 billion, and for $k = 12$ is 10^{12}.

The performance of the LR algorithm in the literature [RBK10; Rüh+13b; TB15] and our results show that Arbiter PUF and XOR Arbiter PUF cannot be considered EUF-KMA secure, unless the number of XORs k can be scaled significantly (Section 3.7).

3.5. Physical Attacks

For the security analysis of PUFs, as discussed in Section 2.2, the capabilities of attackers with physical access to the PUF must be considered.

To that end, Tajik et al. [Taj+14] analyzed an n-bit XOR Arbiter PUF implemented on a Complex Programmable Logic Device (CPLD) and analyzed the photonic emission during operation of the PUF, which is caused by the switching of n-type transistors belonging to the Arbiter PUF stages (realized as multiplexers) in the circuit. The observed measurements allowed the recovery of the delay differences of the Arbiter PUF. After observing the PUF responses for n linearly independent challenges, a system of linear equations can be used to recover all internal delays, effectively providing a model to predict the response to any challenge.

In a different attack, Tajik et al. [Taj+15] used a laser to inject a fault into an XOR Arbiter PUF, which lead to the deactivation of parts of the XOR Arbiter PUF and effectively degenerated it into a number of Arbiter PUFs that can be attacked separately using, e.g., the LR attack (Section 3.4). In this attacker model, the advantages provided by the XOR Arbiter PUF over the Arbiter PUF are removed.

Ganji et al. [Gan+15] use a similar approach, but do not require to inject a fault. By using a lattice basis reduction, they can use the information of *how many* Arbiter PUFs return a certain bit value in an XOR Arbiter PUF to reduce the complexity of the modeling attack. This information can be obtained using the photonic emission techniques also used by Tajik et al. [Taj+14].

These attacks demonstrate that the hope that the Arbiter PUF would be temper-evident, i.e. change it's behavior when attacked physically, cannot be fulfilled.

To mitigate the attacks to a certain extend, Sahoo et al. [Sah+16] suggested a fault-tolerant implementation of the Arbiter PUF.

Hardware attacks on Arbiter PUFs and XOR Arbiter PUFs are, to some extend, also applicable to PUF designs based on XOR Arbiter PUFs such as the ones discussed in this work.

3.6. Neural Network Attacks

This section studies how machine learning modeling attacks that are not restricted to a certain class of model functions can be used to model XOR Arbiter PUFs. This is in contrast to the modeling attack using logistic regression shown in Section 3.4, which is based on the physically motivated model function of the XOR Arbiter PUF (Equation (3.2.1)).

We revisit three neural-network attacks from the literature, replicate and discuss their results. Finally, in Section 3.6.3, we present the best-performing KMA modeling attack to XOR Arbiter PUFs.

The machine learning attacks in this section are all implemented using the Keras framework to provide a fair comparison of their properties and to compare the results to the improved logistic regression attack of Section 3.4. As the attacks run, compared to previous works [Rüh+10; TB15; Wis+19; Wis+20b], relatively fast, we focus our attention on the comparison of the data complexity, i.e., how much training data is required to obtain good modeling predictions.

In this work, we do not focus on the exact accuracy metrics of prediction quality for several reasons. First, the attacks that we study do not yield intermediate results when correctly parameterized, i.e., the attacks end either with accuracy 50% or close to 100%. Additionally, we observed that high-accuracy results can be further improved by letting the training continue for a couple more epochs. Second, to impersonate a PUF token, no extremely high accuracy is needed, and any prediction accuracy significantly better than 50% should be considered a security weakness of the PUF design [Del19]. Hence, instead of prediction accuracy, we focus on the attack *success rate*, i.e., the proportion of independently run attacks on independent PUF simulations that yielded prediction accuracy greater than 90%. Given that virtually all successful attacks yielded accuracy 95% or better and virtually all unsuccessful attacks yielded accuracy 55% or below, the success rate is very insensitive to the exact choice of this threshold. Nevertheless, for the sake of completeness, the prediction accuracy of obtained models is given as the average over several attacks on different PUF tokens.

To ensure that all of our results are reproducible in detail, we seed all involved pseudorandom number generators used for generating PUF token simulations, initializations of machine learning models, etc. with defined values.

3.6.1. Revisited: Santikellur et al.

To reduce the computational effort for modeling attacks on XOR Arbiter PUFs, Santikellur et al. [San+19] proposed to use an efficient CP-Decomposition Tensor Regression Network (ECP-TRN), which is parameterized by an integer rank R. To model a k-XOR n-bit Arbiter PUF, the proposed model computes the function

$$f(x) = \operatorname{sgn}\left[\sum_{i=1}^{R}\left(\alpha_i \cdot \prod_{l=1}^{k}\langle w_i^{(l)}, x\rangle + b_i^{(l)}\right)\right],$$

where $w_i^{(l)} \in \mathbb{R}^n$ and $b_i^{(l)} \in \mathbb{R}$. A drawing of the network structure is shown in Figure 3.6.4c. Due to the highly parallel structure of the network, the approach may benefit from performance improvements during training. The parameters to be trained are $\alpha_1, \ldots, \alpha_R$ and $w_{l,i}, b_{l,i}$, $l \in \{1, \ldots, k\}, i \in \{1, \ldots, R\}$. Hence, there are $nkR + kR + R$ trainable parameters.

Given this network structure, the network can be understood as an approach that trains R XOR Arbiter PUF models in parallel. The final response of the model is then

computed as the weighed sum of the R model outputs, then the sign of the response is returned. After the training completes, the network is filled with R sets of XOR Arbiter PUF weights, which raises the question how the *individual* prediction accuracy differs from the *overall* prediction accuracy reported by Santikellur et al.

For our experiments operating on simulations of 4, 5, and 6-XOR Arbiter PUFs with 64-bit challenge lengths, we found that in all successfully trained networks, exactly one of the R trained models showed high correlation with the simulation weights, whereas the other $R-1$ had no correlation. This finding was confirmed by the prediction accuracy: $R-1$ of the models in the successfully trained network had an individual prediction accuracy of around 50%, whereas exactly one had high prediction accuracy. Using the single model allowed for even higher prediction accuracy than using the fully trained network, as the noise of the $R-1$ low-correlation models is removed. We can conclude that the R-rank model of the ECP-TRN does not provide benefits over the 1-rank model.

As the final response of the ECP-TRN network is computed as the weighed sum of the R individual model responses, the individual models influence each others training process through the backpropagation algorithm. To examine if this interdependency during training provides an advantage to the modeling attack, we run many *attack attempts* on a single PUF under attack, i.e., we restart the training process with different initializations of the model, while keeping the PUF simulation and CRP set constant. In the case of many attack attempts, the training of each attempt is independent of the training process of the other attempts, above-mentioned interdependency is removed. This allows us to compare the performance metrics of rank R ECP-TRN attacks using a single attack attempt versus rank 1 ECP-TRN using R attack attempts. The results are displayed in Tab. 3.2.

In none of the cases that we studied, the training success rate of a rank R ECP-TRN could surpass the success rate of running R independent learning attempts using a rank 1 ECP-TRN.

Unfortunately, we were not able to replicate the ECP-TRN results of Santikellur et al. [San+19] exactly as published in the original paper. While the 64-bit 4-XOR case could be replicated, our experiments for 5-XOR and 6-XOR required significantly more CRPs for reliable convergence than originally claimed. For 7-XOR and larger, we failed to achieve any success using the proposed high-rank model. A discussion with the original authors also could not improve our results. We suspect that the reason for the larger requirement of CRPs is either caused by the different behavior of Keras internals compared to the original Tensorflow v1 implementation, or, considered more likely, by some differences in CRP generation. It may also play a role that by training R models in parallel, the original authors effectively report the maximum accuracy among (a large number of) R learning attempts, while our work reports data complexities for success rates mostly close to 100%. We further discuss this in Section 3.6.4.

3.6.2. Revisited: Aseeri et al.

After an attack on 3-XOR 64-bit Arbiter PUFs by Yashiro et al. [Yas+16] using a neural network with autoencoders and an attack by Alkatheiri and Zhuang [AZ17] on Feed-

n	k	[San+19] CRPs	our CRPs	R	attempts per run	total runs	run success rate	mean attempt success accuracy
64	4	40k	40k	1	5	10	100%	95.2%
64	4	40k	40k	5	1	10	100%	94.6%
64	5	80k	320k	1	10	10	90%	94.9%
64	5	80k	320k	10	1	10	90%	94.9%
64	6	400k	800k	1	10	10	80%	95.1%
64	6	400k	800k	10	1	10	80%	95.0%
64	7	800k	800k	1	100	5	20%	95.4%
64	7	800k	800k	1000	1	4	0%	—

Table 3.2.: Comparing single-attempt attacks using rank R ECP-TRN versus R-attempt attacks using rank 1 ECP-TRN. Our results indicate that the training of the ECP-TRN does not benefit from interaction of the models; but gives some indication to the contrary. Compared to the figures of Santikellur et al. [San+19], in some cases we increased the number of CRPs to obtain any successful results.

Forward Arbiter PUFs using a multilayer Perceptron, Aseeri, Zhuang, and Alkatheiri [AZA18] were the first ones to attack XOR Arbiter PUFs with more than four arbiter chains using neural networks.

While much of their work focused on the fact that their version of the modeling attack can be run on a regular laptop computer, i.e., on a machine without GPU, but with limited memory and just using a single core of a consumer CPU, their attack also achieves a significant reduction in both time and data complexity, compared to the then state-of-the-art LR attack by Tobisch and Becker [TB15].

Some attempts to replicate the work of Aseeri, Zhuang, and Alkatheiri failed [SBC19], likely because at the time, the source code of the attack was not publicly available yet. Consequently, the attack was not sufficiently considered in the security analysis of the Interpose PUF (cf. Chapter 5) [Ngu+19]. We take this as evidence that the publication of code that is the basis for claimed attack results should be strongly encouraged by the community.

The original implementation of this attack was done using scikit learn. As part of our comparison of neural network attacks, in this work, we reimplemented the network used by Aseeri et al. using the Keras machine learning framework and were able to replicate all of their results. An overview of our replicated results can be found in Table 3.3, including an extension to the 64-bit 9-XOR and 128-bit 8-XOR cases. While the original figures strictly use single-core performance on a consumer CPU, we used up to 40 cores in parallel. To allow for comparison, we include an estimation of the single-core performance of our implementation by multiplying the measured wall clock time with the maximal number of threads our experiment allowed. This overestimates the time required by our

n	k	CRPs	success rate	duration	avg. success accuracy	memory	single core duration this work⋆	[AZA18]
64	4	400k	10/10	<1 min	96.8%	<1 GiB	11 min	<1 min
64	5	400k	10/10	<1 min	96.7%	<1 GiB	17 min	<1 min
64	6	2M	9/10	<1 min	97.3%	1 GiB	8 min	7 min
64	7	5M	9/10	<1 min	97.3%	2 GiB	20 min	12 min
64	8	30M	10/10	3 min	98.1%	8 GiB	102 min	23 min
64	9	80M	9/10	86 min	98.2%	29 GiB	3438 min	-
128	4	400k	10/10	<1 min	96.8%	<1 GiB	17 min	1 min
128	5	3M	10/10	<1 min	97.1%	2 GiB	33 min	5 min
128	6	20M	10/10	<1 min	98.0%	10 GiB	28 min	19 min
128	7	40M	10/10	5 min	97.9%	20 GiB	181 min	90 min
128	8	100M	1/10	45 min	98.6%	50 GiB	1813 min	-

Table 3.3.: Extended results on learning simulated XOR Arbiter PUFs obtained using our Keras-based implementation of the multilayer Perceptron attack by Aseeri, Zhuang, and Alkatheiri [AZA18]. ⋆To allow for comparison with the original figures, we computed an approximation of the duration using a single core. The performance loss is caused by the lower single-core performance of our CPUs (Intel Xeon E5-2630 v4) compared to the Intel Core i7 used by Aseeri, Zhuang, and Alkatheiri. All of our experiments use up to 40 threads instead of just one as done by Aseeri, Zhuang, and Alkatheiri.

attack, especially for cases where multi-threading allows only for little speedup, i.e., for small training sets.

Aseeri, Zhuang, and Alkatheiri did not include arguments for the specific hyperparameter settings they used in their attack. We include a discussion of the multilayer Perceptron hyperparameters in Section 3.6.3. A comparing overview can be found in Table 3.5; a drawing of the network can be found in Figure 3.6.4a.

3.6.3. Revisited: Mursi et al.

3.6.3.1. Neural Network

In follow-up work to Aseeri, Zhuang, and Alkatheiri [AZA18] and Santikellur, Bhattacharyay, and Chakraborty [SBC19] (not to be confused with the ECP-TRN model), Mursi et al. [Mur+20] presented an enhancement of the multilayer Perceptron XOR Arbiter PUF modeling attack, claiming to reduce the data and time complexity of XOR Arbiter PUF modeling attack by several orders of magnitude. We falsify their empirical results, but show that their attack still requires fewer CRPs than other response-based XOR Arbiter PUF modeling attacks. Consequently, we are able to demonstrate that XOR Arbiter PUFs are insecure under EUF-KMA up to higher security parameters than previously known, posing a challenge to implementors who need to keep the noise low

enough to allow for the fabrication of XOR Arbiter PUFs with such large security parameters.

To attack an n-bit k-XOR Arbiter PUF, Mursi et al. propose to use a neural network that consists of three fully connected hidden layers of sizes $2^{k-1}, 2^k, 2^{k-1}$. We depict such a network in Figure 3.6.4b. By its design, this model uses fewer trainable parameters than the MLP-approach by Aseeri, Zhuang, and Alkatheiri and the high-rank approach by Santikellur et al., which can benefit training. Nevertheless, it uses orders of magnitude more parameters than the logistic regression attack (Section 3.4). For example, in the attack of a 64-bit 9-XOR Arbiter PUF, LR uses 585 trainable parameters, while the MLP attack in this section uses 66,560 trainable parameters.

Mursi et al. also use the tanh activation function for the hidden layers, compared to the usage of ReLU by Aseeri, Zhuang, and Alkatheiri. We surmise that this benefits training of the network as it enables weight update for neurons that compute a negative value. While Santikellur, Bhattacharyay, and Chakraborty [SBC19] argue that tanh suffers from the vanishing gradient problem, the successful use of tanh in the MLP attack can be explained by the relatively shallow three-layer structure of the neural network, which makes the vanishing gradient problem unlikely to appear [Mur+20]. A detailed comparison of hyperparameters as used in the different attacks showed in this section can be found in Table 3.5.

3.6.3.2. Replication and Results

To make the various neural-network-based attacks comparable, we reimplemented the attack by Mursi et al. using the Keras machine learning framework and found that their results could not be replicated. The difference in attack performance of our implementation and the original could be traced back to a bug in the CRP generator used by Mursi et al., which was based on a simulation of the delays. For each PUF instance, $4n$ delays were supposed to be drawn independently from a Gaussian distribution with mean 300 and variance 40; given a challenge, the delay difference can then be computed and converted into the PUF response. Due to the bug, about 20% of the randomly drawn delays were inadvertently set to zero. This was difficult to notice from the CRP data, as the bias was hardly influenced and the MLP-based attack does not recover the simulation delays or weights, but a neural network that is hard to be interpreted.

To study the attack by Mursi et al., we use our reimplementation of the neural network attack and the pypuf CRP generator [Wis+21b] used throughout this work. The CRP generation is, for performance reasons, based on the equivalent approach of using weights instead of delays as shown in Theorem 8 and Corollary 9. We found that while the attack performance reported by Mursi et al. significantly benefited from the faulty CRP generation, the results obtained on valid CRPs still improve on the LR attack in terms of data complexity by an order of magnitude, with increasing advantage for an increasing number of XORs, as can be seen in Figure 3.6.3. We also observed an improvement in run time. Detailed results are reported in Table 3.4.

For challenge lengths 128 and 256, we found that the data complexity grows fast with the number of XORs. Nevertheless, for 128 bit challenges, it remains below the figures

that Tobisch and Becker [TB15] reported for the LR attack; for 256 bit challenges we could not find numbers in the literature to compare to. However, the LR attack is known to have polynomially increasing data complexity in the challenge length [Wis+20b]. The steeply increasing required number of CRPs of the MLP attack could be caused by an inherent effect of the XOR Arbiter PUF structure, or by a mismatch of hyperparameters or neural network structure on the model. Considering everything, we conclude that there is no evidence that increasing the challenge length will be an effective defense against modeling attacks and let this question open to be studied in case sufficiently large XOR Arbiter PUFs can be built.

In light of the reduced data complexity, we come to the conclusion that a model with far more trainable parameters is able to outperform a model with fewer model parameters, which falsifies the claim by Nguyen et al. that the LR attack is the best performing among the XOR Arbiter PUF attacks [Ngu+19].

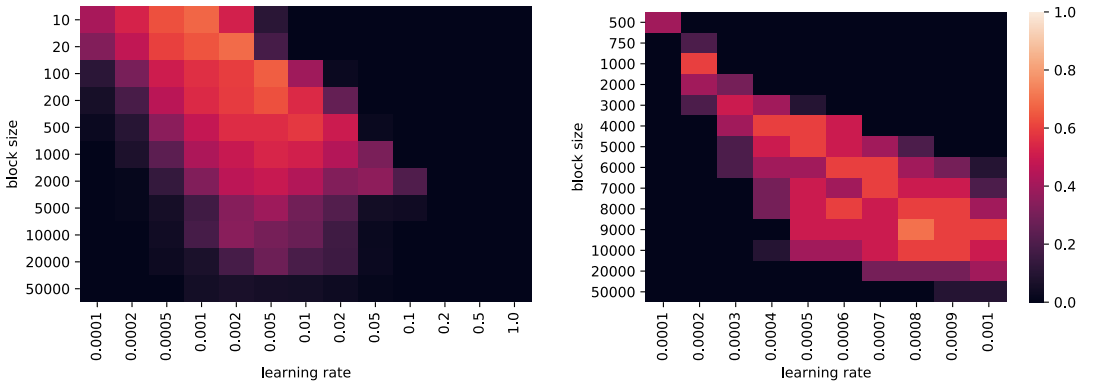
As a byproduct of our replication of the attack, we find that a relatively small proportion of zero-valued delays in the XOR Arbiter PUF can lead to a large loss of data complexity for the modeling attack. With this in mind, implementors of PUFs should treat any significant deviation from simulation-based attack results on real-world data with additional scrutiny on the validity of their implementation. In future security analyses of PUFs, the detailed validity of the simulation in use needs to be established, otherwise the analysis could over- or underestimate the PUF's security. Such validation is especially challenging when using generic models such as the MLP for modeling attacks. However, in case of the Arbiter PUF, several independent results confirming the validity of the additive delay model exist [Gas+04; DV13; Taj+14].

3.6.3.3. Hyperparameter Optimization

As a technical note, we found it difficult to configure the hyperparameters of the attack by Mursi et al. While the processing of the training data in mini batches provides benefits regarding the run time, and thus the development of the attack, it also requires to adjust the learning rate appropriately. In Figure 3.6.1 we report the success rate of MLP-based attacks on 4-XOR 64-bit and 8-XOR 64-bit for a large variety of different configurations of learning rate and batch size, showing that only an appropriate combination of those two hyperparameters will yield a successful attack. We speculate that on the one hand, for high learning rates on small batches, the gradient direction is too noisy to yield a meaningful update to the model, and on the other hand, that for low learning rates on large batches the learning process runs into the maximum number of epochs before a convergence could be achieved.

3.6.3.4. Noise Resilience

We found the attack to work in the presence of noise without significant changes in data complexity when compared to the noise-free case. This is somewhat surprising, as the MLP attack, in contrast to the LR attack, is not restricted to functions of a certain class. Nevertheless, the training converges to the desired XOR Arbiter PUF



(a) 4-XOR Arbiter PUF, 50k CRPs, 100+ runs each (b) 8-XOR Arbiter PUF, 6M CRPs, 10 runs each

Figure 3.6.1.: Success rate for training an 64-bit XOR Arbiter PUFs with the attack by Mursi et al. [Mur+20], with varying learning rates and block sizes.

model and approaches the maximum predictive power. Note that the noise resilience is not a prerequisite for a successful attack for an attacker in the chosen message attack model, as they could also query a challenge multiple times and remove noise efficiently by majority voting the responses. We report detailed results on the noise resilience in Table 3.4.

3.6.3.5. Application to Real-World Data

To confirm the lower data complexity of the MLP attack for realistic challenge-response and noise data, we compare the improved LR attack and MLP attack on the (XOR) Arbiter PUF data sets provided by Mursi et al. [Mur+20] and Aghaie and Moradi [AM21].

The data of Mursi et al. contains one instance of each a 64-bit k-XOR Arbiter PUF for $k \in \{4, \ldots, 9\}$, queried on the same set of 1 million (for $k \in \{4, 5, 6\}$) and 5 million challenges (for $k \in \{7, 8, 9\}$). This data set does not provide repeated measurements, hence no statement about the reliability can be made. Our statistical analysis of the responses confirmed the expectation of decreasing bias with an increase in the number of XORs; the 4-XOR Arbiter PUF has an average response of 0.03, the 9-XOR Arbiter PUF an average response of 10^{-5} (with its Boolean responses given as -1 and 1).

The data of Aghaie and Moradi [AM21] contains a 64-bit (1,5)-Interpose PUF queried on 1 million uniformly random challenges, each including the responses of all 6 individual Arbiter PUFs. The data also contains 11 repeated measurements of all challenges, which we only used to compute the reliability of the PUFs (all Arbiter PUFs had 99.7% or better). For the attacks, we just used the first of the 11 measurements, discarding the other 10. We confirmed the quality of the data by testing that all six involved Arbiter PUFs individually can be modeled using the delay model with high accuracy (all 98%). To use the Interpose PUF data in the context of XOR Arbiter PUF attacks, we discard the (64-bit) top layer and compose the five given 65-bit Arbiter PUFs of the Interpose

rel.	n	k	CRPs	success rate	duration (max. threads)	success accuracy	memory	[Mur+20] CRPs	[Mur+20] duration
1.00	64	4	150k	10/10	<1 min (40)	97.0%	1 GiB		
1.00	64	5	200k	10/10	<1 min (20)	97.3%	3 GiB	42k	<1 min
1.00	64	6	2M	10/10	<1 min (40)	97.5%	2 GiB	255k	2 min
1.00	64	7	4M	10/10	<1 min (40)	97.5%	2 GiB	680k	1 min
1.00	64	8	6M	7/10	13 min (4)	95.5%		1.7M	5 min
1.00	64	9	45M	10/10	16 min (40)	98.1%	14 GiB	4.2M	9 min
1.00	64	10	119M	7/10	291 min (40)	97.9%	41 GiB		
1.00	64	11	325M	10/10	1898 min (40)	98.1%	104 GiB		
1.00	128	4	1M	10/10	<1 min (40)	97.3%	1 GiB		
1.00	128	5	1M	10/10	<1 min (20)	97.4%	3 GiB		
1.00	128	6	10M	9/10	<1 min (20)	98.1%	5 GiB		
1.00	128	7	30M	10/10	2 min (20)	98.2%	20 GiB		
1.00	256	4	6M	10/10	1 min (40)	97.7%	6 GiB		
1.00	256	5	10M	10/10	2 min (20)	97.8%	10 GiB		
1.00	256	6	30M	0/7	— (20)	—	30 GiB		
1.00	256	7	100M	1/10		98.9%	99 GiB		
0.85	64	4	180k	9/10	<1 min (4)	90.9%	<1 GiB		
0.85	64	5	150k	10/10	<1 min (4)	91.2%	<1 GiB		
0.85	64	6	2M	10/10	<1 min (4)	91.8%	1 GiB		
0.85	64	7	4M	9/9	3 min (4)	91.6%	2 GiB		

Table 3.4.: Averaged empirical results on learning simulated XOR Arbiter PUFs obtained using our improved implementation of the neural network attack by Mursi et al. [Mur+20]. The learning was configured to stop at validation accuracy 95%, the variance of added noise was configured such that the simulation achieves the given reliability value ("rel.").

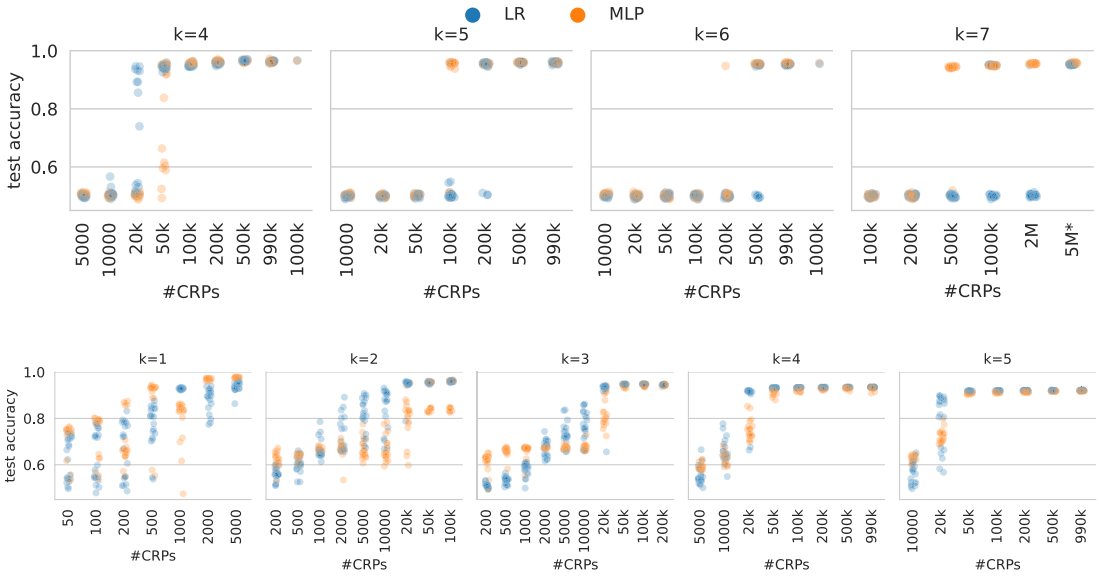


Figure 3.6.2.: Comparison of improved LR attack (Section 3.4) and our MLP attack (Section 3.6.3) on FPGA data from Mursi et al. [Mur+20] (top row) and Aghaie and Moradi [AM21] (bottom row). The results are in line with our results on simulated data: the MLP attack shows lower data complexity for XOR Arbiter PUFs with a large number of XORs k. (*Training set size was actually 4,990,000; 10,000 CRPs were used for testing.)

	[Rüh+10]	[AZA18]	[San+19]	[Mur+20]
Method	LR	MLP	TRN	MLP
Architecture	delay model	$(2^k, 2^k, 2^k)$	many delay mod.	$(2^{k-1}, 2^k, 2^{k-1})$
Hid. lay. activ.	—	ReLU	—	tanh
Optimizer	RProp	Adam	Adam	Adam
Loss function	BCE	BCE	BCE	BCE
Learning rate	RProp default	10^{-3}	(multiple)	adaptive
Initializer	Gaussian	Glorot Unif.	Glorot Normal	Gaussian

Table 3.5.: Parameter comparison of modeling attack methods on k-XOR Arbiter PUFs.

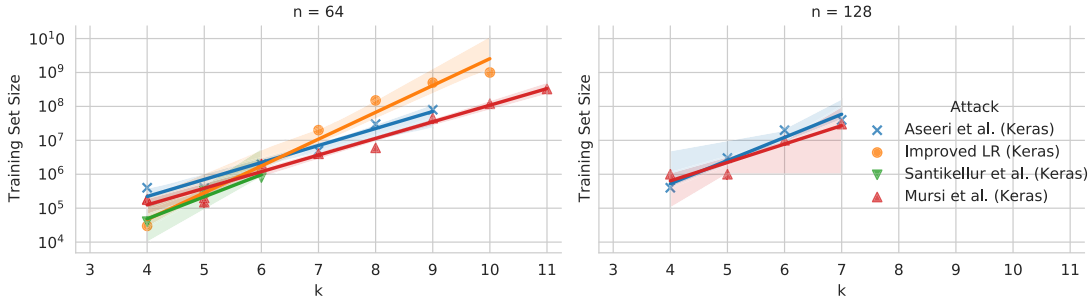
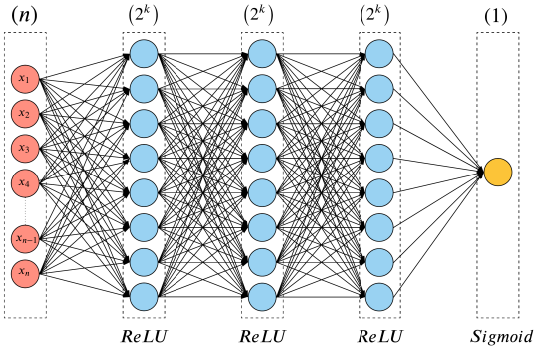


Figure 3.6.3.: Data complexity of our attacks on n-bit Arbiter PUF with k individual arbiter chains. Our implementation of the attack by Mursi et al. outperforms the improved LR attack by an order magnitude with regard to data complexity.

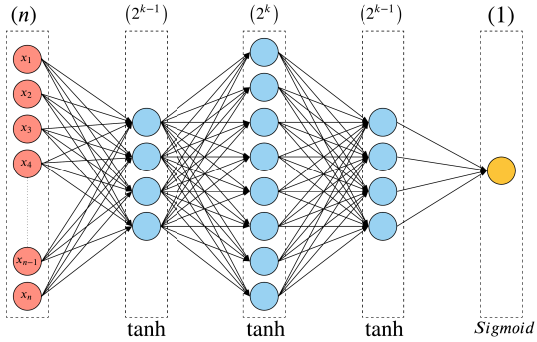
bottom layer to 65-bit k-XOR Arbiter PUFs for $k \in \{1, \ldots, 5\}$. The data shows that the involved Arbiter PUFs are heavily biased, with 3 out of 5 have an average response of 0.3 or higher, computed from Boolean responses given as -1 and 1.

We ran the LR and MLP attack with varying number of CRPs on the XOR Arbiter PUFs obtained from the data to compare the data complexity of the attacks. The detailed results are displayed in Figure 3.6.2. All attacks ran within a few seconds to few minutes each.

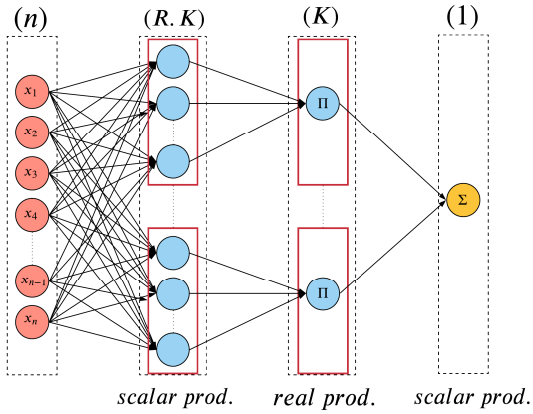
The results on the data of Mursi et al. show that for k-XOR Arbiter PUFs with $k = 6$ and $k = 7$, the data complexity of the MLP is much lower than that of the improved LR attack. On the other hand, the results on the data of Aghaie and Moradi show that the MLP attack is either unsuited or ill-parameterized for k-XOR Arbiter PUF with smaller numbers for k and provides no advantage or improved LR in data complexity. We conclude that our experiments on real-world data confirm our findings obtained with simulated data.



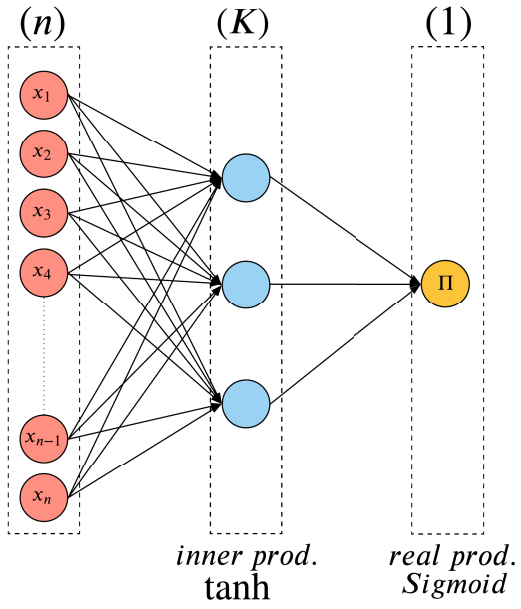
(a) The feed-forward neural network architecture for 3-XOR 64-bit using the model by Aseeri, Zhuang, and Alkatheiri [AZA18]. Since k is equal 3, the hidden layers consists of 8 neurons each. The architecture changes based on the number of streams in an k-XOR Arbiter PUF; the hidden layers use the ReLU activation function.



(b) The feed-forward neural network architecture for 3-XOR 64-bit using the model by Mursi et al. [Mur+20]. Since k is equal 3, the first and third hidden layers consists of 4 neurons each, the second layer however possesses 8 neurons. The architecture changes based on the number of streams in an k-XOR Arbiter PUF; the hidden layers use the tanh activation function.



(c) The neural network architecture used by Santikellur et al. [San+19], which is a parallel structure containing R models as used in the LR attack (Figure 3.6.4d) and computing the weighed sum of all outputs. Here shown for $R = 2$.



(d) The network architecture used by our improved version of the LR attack [Rüh+10]. The network very closely follows inspiration derived from a physical model of the XOR Arbiter PUF.

Figure 3.6.4.: Neural network comparison for XOR Arbiter PUF modeling attacks.

3.6.4. Comparison

Of the three studied neural-network-based modeling attacks and the improved LR attack used as a baseline for comparison, we found that the claims by Mursi et al. [Mur+20] could be traced back to an error in CRP generation, which leaves the work by Santikellur et al. [San+19] to claim, to the best of our knowledge, the lowest data complexity of XOR Arbiter PUFs in the literature on known message attacks. However, we were unable to replicate these attacks for the cases of 7-XOR and larger.

Among the attacks successfully replicated in this work, we found the attack by Mursi et al. [Mur+20] to have, despite a relatively high number of trainable parameters, the lowest data complexity. Being an enhancement of neural network attacks presented by Aseeri, Zhuang, and Alkatheiri [AZA18] and Santikellur, Bhattacharyay, and Chakraborty [SBC19] (not to be confused with the ECP-TRN), we attribute the advantage in data complexity to the choice of network size and hyperparameters, concluding that a further reduction of complexity may well be possible for more carefully optimized settings.

Comparing the data complexity of the MLP-attack by Mursi et al. to the results obtained with our improved version of the LR attack, we find that MLP only has advantages in data complexity for XOR Arbiter PUFs with more than four arbiter chains, but not for smaller designs. Next to the above-mentioned steep increase of data complexity for larger challenge lengths of the MLP-attack, we read this as evidence that the chosen neural network structure is not optimal with respect to arbitrary values of challenge length and number of XORs.

Comparing the complexity of the MLP-attack by Mursi et al. with the MLP-attack by Aseeri, Zhuang, and Alkatheiri, we find that the main differences of the attacks are the network shape and the choice of activation function in the hidden layers. As discussed above, we speculate that the ReLU activation function hampers weight updates during the backpropagation process and thus constitutes a disadvantage in the learning process.

For a detailed comparison of the four modeling attacks, we provide graphs of the network structure in Figure 3.6.4, display the chosen hyperparameters in Table 3.5, and provide an overview over the data complexities in Figure 3.6.3.

We do not include a detailed comparison of run times, as most of the attacks presented in this work run in minutes, which makes a valid comparison difficult. Furthermore, using the Keras-based implementation, our attacks can be run in a variety of different settings, i.e., on CPUs and GPUs, with and without multithreading, which will lead to different run times. In any event, none of the attack times reported in this work are prohibitively long for an attacker.

We conclude that the XOR Arbiter PUF is insecure under EUF-KMA for security parameters up to $k = 11$ when using 64-bit challenges and up to $k = 7$ when using 128-bit challenges, and an extension of the presented attacks beyond these parameter settings appears to be within reach.

We discuss applications of the neural-network attacks on the Interpose PUF in Section 5.3.

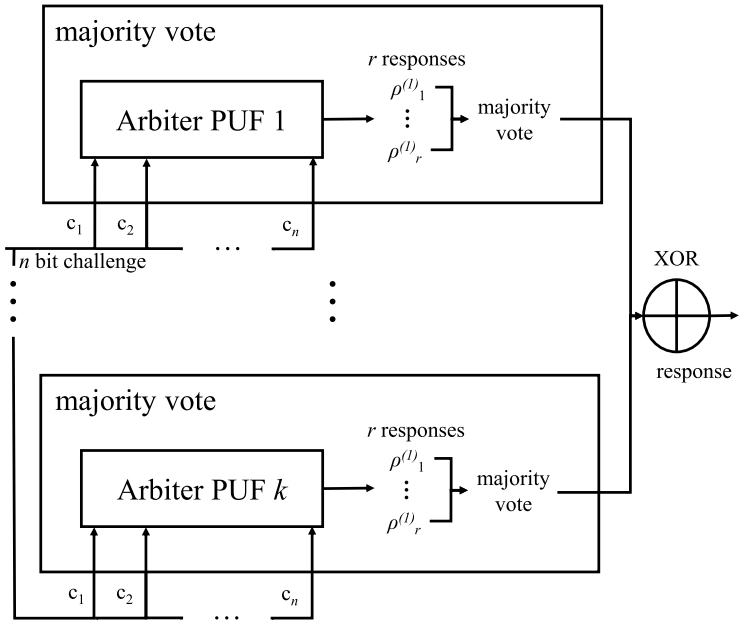


Figure 3.7.1.: A detailed logic block diagram of the proposed XOR Majority Vote Arbiter PUF design. For each arbiter chain, r responses to the given challenge are evaluated and passed to a majority vote. The result is then XORed and returned.

3.7. Arbitrarily Large XOR Arbiter PUFs

As a possible mitigation for the known message attacks of Section 3.4 and Section 3.6, an increase in the number k of used Arbiter PUFs in an XOR Arbiter PUF can be considered. As the run time of both attacks is expected to increase exponentially in this parameter, this has the potential to mitigate the attacks.

However, due to the design of the XOR Arbiter PUF, it is also known that the reliability of the Arbiter PUF decreases exponentially with the increase in k, effectively putting a limit on this parameter. While the exact limit depends on the noise of the particular implementation, the largest reported XOR Arbiter PUF in the literature uses 12 Arbiter PUFs [Yu+16].

In this section, we consider the XOR *Majority Vote Arbiter PUF*, which uses majority voting on Arbiter PUFs to reduce the noise. We demonstrate that the XOR Majority Vote Arbiter PUF is capable of producing low-noise responses for an arbitrarily high number k of Arbiter PUF by using only a limited number of votes in the majority vote process.

The XOR Majority Vote Arbiter PUF proposed in this section is a modification of the XOR Arbiter PUF design (Section 3.2) by Suh and Devadas [SD07]. In order to reduce the noise of Arbiter PUFs, responses can be determined by a majority vote process [MEK10] before XORing the individual response bits as shown in Figure 3.7.1. We show

that with a feasible, i.e. polynomial in the security parameters n and k, number of votes in the majority vote process, we can achieve a response stability as high as desired.

While the majority vote process requires some sort of volatile memory in the hardware implementation for vote counting, which increases the hardware attack surface (cf. Section 2.4 and Section 3.5), and assumes that subsequent evaluations of the Arbiter PUF are statistically independent, it demonstrates that this mitigation of the Logistic Regression and Neural Network Attacks is feasible.

3.7.1. Stability

To quantify the stability of Arbiter PUF behavior with respect to a given challenge for PUFs with a single output bit, we define the notion of *stability*. We assume that the response behavior of the PUF is *stateless*, i.e. does not depend on challenges and/or responses which have been previously been processed.

Definition 11. For a PUF $f : \mathcal{D} \to \{-1, 1\}$, we define the *stability* of f on a challenge c to be the probability to observe the more likely response when querying c once, i.e. $\mathrm{Stab}(c) = \Pr\left[r = \operatorname{sgn} \mathrm{E}\left[r' \mid r' \leftarrow f(c)\right] \mid r \leftarrow f(c)\right]$.

Using the definition of the expectation for $r \leftarrow f(c)$, i.e. $\mathrm{E}[r] = \Pr[r = 1] - \Pr[r = -1]$, we obtain that $\Pr[r = 1] = 1/2\,\mathrm{E}[r] - 1/2$ and $\Pr[r = -1] = 1/2 - 1/2\,\mathrm{E}[r]$. As $-1 \leq \mathrm{E}[r] \leq 1$, the stability of f on c can be written as

$$\mathrm{Stab}(c) = \frac{1}{2} + \frac{1}{2} \cdot \left|\mathrm{E}\left[r \mid r \leftarrow f(c)\right]\right|. \tag{3.7.1}$$

Note that stability as defined here is closely related to the reliability defined in Section 2.1 (Definition 2), but not identical: stability is the probability to see a response not altered by noise, while reliability is the probability to see the *same* response when querying the same challenge twice, i.e. it also includes the case of seeing two noise-affected responses in a row.

The exact relation of reliability and stability as defined in this work can be formalized as

Lemma 12. *In the setting of Definition 11, for any challenge c, we have*

$$\Pr\left[r_1 = r_2 \mid r_i \leftarrow f(c), i \in \{1, 2\}\right] = 1 + 2\,\mathrm{Stab}(c)^2 - 2\,\mathrm{Stab}(c) = \frac{1}{2} + \frac{1}{2}\left|\mathrm{E}\left[r \mid r \leftarrow f(c)\right]\right|^2.$$

Proof. Let $r^\star$ denote the more likely response of f to the challenge c, i.e. $r^\star = \operatorname{sgn} \mathrm{E}[f(c)]$. By case distinction, we have

$$\begin{aligned}
&\Pr\left[r_1 = r_2 \mid r_i \leftarrow f(c), i \in \{1, 2\}\right] \\
=&\Pr[r_1 = r^\star]\Pr[r_2 = r^\star] + (1 - \Pr[r_1 = r^\star])\,(1 - \Pr[r_2 = r^\star]) \\
=&1 + 2\,\mathrm{Stab}(c)^2 - 2\,\mathrm{Stab}(c).
\end{aligned}$$

By using Equation (3.7.1), we obtain

$$\Pr\left[r_1 = r_2 \mid r_i \leftarrow f(c), i \in \{1,2\}\right] = \frac{1}{2} + \frac{1}{2}\left|\mathrm{E}\left[r \mid r \leftarrow f(c)\right]\right|^2.$$

□

There is no wide consensus in the literature on the exact definitions and names of this formalization, and equivalent or similar notions can be found in the literature under the names *reliability* [Bec15], *repeatability* [DV13] and *robustness* [Arm+09].

3.7.2. Arbiter PUF

For Arbiter PUFs modeled using the noisy PUF delay model of Equation (3.1.1), we have $\operatorname{sgn}\mathrm{E}\left[r' \mid r' \leftarrow f(c)\right] = \operatorname{sgn}\mathrm{E}\left[\Delta D_{\mathrm{Model}}(c) + \Delta D_{\mathrm{Noise}}\right] = \operatorname{sgn}\Delta D_{\mathrm{Model}}(c)$ and the stability can thus be written as

$$\mathrm{Stab}(c) = \Pr_{\Delta D_{\mathrm{Noise}}}\left[\operatorname{sgn}\left(\Delta D_{\mathrm{Model}}(c) + \Delta D_{\mathrm{Noise}}\right) = \operatorname{sgn}\left(\Delta D_{\mathrm{Model}}(c)\right)\right].$$

Corollary 13. *For an Arbiter PUF instance, a challenge with model delay difference $\Delta D_{\mathrm{Model}}(c)$, and normally distributed noise $\Delta D_{\mathrm{Noise}}$ with zero mean and variance $\sigma^2_{\mathrm{Noise}}$, the probability that the PUF response is not influenced by noise is*

$$\mathrm{Stab}(c) = \frac{1}{2} + \frac{1}{2}\operatorname{erf}\left(\frac{|\Delta D_{\mathrm{Model}}(c)|}{\sqrt{2}\sigma_{\mathrm{Noise}}}\right). \tag{3.7.2}$$

While Corollary 13 gives information about the stability of a *single* challenge, in the context of PUF applications, often the average stability, or more detailed, the distribution of challenge stability is of interest. We can see that for Arbiter PUFs, the challenge stability depends on the distribution of $\Delta D_{\mathrm{Model}}(c) = \langle w, x\rangle + b$, with $w^n \in \mathbb{R}$ and $b \in \mathbb{R}$ chosen as determined in Corollary 9 and x defined as the feature vector determined by the given challenge c (cf. Theorem 8).

Under these assumptions, $\Delta D_{\mathrm{Model}}(c)$ is the sum of n random variables $w_i x_i$ and can thus be approximated by a Gaussian distribution using the Berry-Esseen Central Limit Theorem (CLT) [Ber41; Tyu10]. The error bound of this approximation is a random variable depending on the choice of w and b, with a narrowing variance and lowering mean as n becomes larger. That is, the approximation of $\Delta D_{\mathrm{Model}}$ becomes better for increasing n.

Assuming a Gaussian distribution of $\Delta D_{\mathrm{Model}}(c)$ for uniformly and randomly chosen c, we can approximate the probability that a chosen challenge has stability below a given threshold. In the following, we assume $\Delta D_{\mathrm{Model}} \sim \mathcal{N}(0, \sigma^2_{\mathrm{Model}})$ as a simplification although we actually have $\mathrm{E}\left[\Delta D_{\mathrm{Model}}\right] = b$, as b was in turn drawn from a distribution with mean zero. The simplification that $\Delta D_{\mathrm{Model}}$ follows a normal distribution simplifies the analysis drastically.

Lemma 14. *For any given Arbiter PUF with n stages, any probability $z \in \left[\frac{1}{2}, 1\right]$, and measurement and hardware conditions described by $\Delta D_{\text{Noise}} \sim \mathcal{N}(0, \sigma^2_{\text{Noise}})$, the probability that a uniformly, randomly chosen challenge c has stability lower than z is approximated by*

$$\begin{aligned} \text{Stab}_{\text{CDF}}(z) &= \Pr_{c \sim \{-1,1\}^n} [\text{Stab}(c) < z] \\ &= \text{erf}\left(\frac{\sigma_{\text{Noise}}}{\sigma_{\text{Model}}} \text{erf}^{-1}(2z - 1) \right). \end{aligned} \tag{3.7.3}$$

The proof uses the CLT approximation of ΔD_{Model} together with the essential Gaussian distribution fact that $\Pr[|\Delta D_{\text{Model}}(c)| < x] = \text{erf}(x/\sqrt{2}\sigma_{\text{Model}})$. The CLT also yields explicit error bounds, if needed.

In the simplest case, we can choose the threshold independently of n to be a constant, $z = 99\%$ to obtain the probability that a randomly chosen challenge has stability below 99%, i.e.

$$\Pr_{c \sim \{-1,1\}^n} [\text{Stab}(c) < 99\%] \approx \text{erf}\left(1.64 \frac{\sigma_{\text{Noise}}}{\sigma_{\text{Model}}} \right).$$

For any given constant threshold z, the probability that a uniformly and randomly chosen challenge has stability below z depends only on $\sigma_{\text{Noise}}/\sigma_{\text{Model}}$. A numeric evaluation can be found in Figure 3.7.2. We can interpret $\Pr_c[\text{Stab}(c) < z]$ as cumulative density function. The derivative then gives the probability density function, as shown in the same figure. We can see from the probability density that, while the majority of challenges will have high stability, there is a significant (i.e., polynomial) number of challenges that have stability close to $\frac{1}{2}$, as the probability density does not approach zero. This idea will be formalized in Theorem 16.

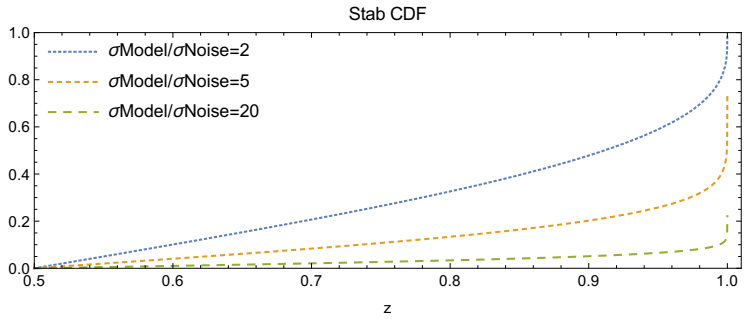
3.7.3. Majority Vote Arbiter PUF

The stability of an arbiter chain can be boosted by majority voting.

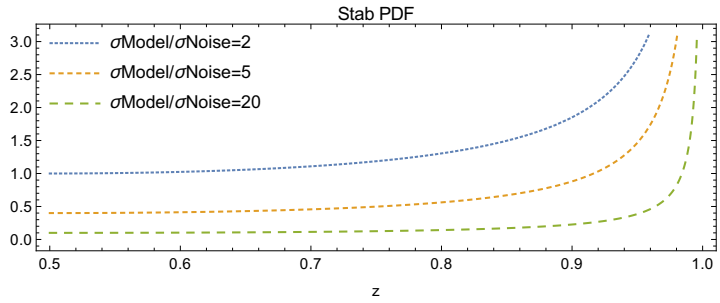
Definition 15. For a Majority Vote Arbiter PUF and any challenge $c \in \{-1, 1\}^n$, we define *majority vote stability using r votes,* $\text{Stab}^{(r)}_{\text{MV}}(c)$, as the probability that the result of majority vote results into a PUF response that is not altered by noise,

$$\text{Stab}^{(r)}_{\text{MV}}(c) = \Pr_{\Delta D_{\text{Noise}} \in \mathbb{R}^r} [\text{majority vote result} = \text{sgn}(\Delta D_{\text{Model}}(c))].$$

In the following, we show that for any monotone increasing polynomial $t(n)$, we can boost the stability for any challenge c that satisfies $\text{Stab}(c) \geq \frac{1}{2} + \frac{1}{t(n)}$ with a polynomial number of votes "exponentially close to 1" (formal notation follows). In Lemma 17, we will show that this prerequisite is fulfilled by "most" challenges. For any remaining challenges that do not fulfill the boosting requirement, we know that their stability will be increased through our process, although not necessarily up to the desired value. Hence the choice $t(n)$ can be thought of as the parameter that determines which challenges are



(a) Cumulative Distribution Function (CDF)



(b) Probability Density Function (PDF)

Figure 3.7.2.: CDF and PDF of the distribution of Arbiter PUF challenge stability under the assumption of normally distributed model delay values, both shown for $\sigma_{\text{Model}}/\sigma_{\text{Noise}} \in \{2, 5, 20\}$ based on Equation (3.7.3). The larger σ_{Model} is compared to σ_{Noise}, the higher the probability for high stability becomes. The CDF as proven in Lemma 14 is a central tool in our analysis; the PDF helps us to interpret measured stability frequency in simulations (see Fig. 3.7.3b).

boosted to exponential stability. To formalize the notion of "exponentially close to 1", we introduce the polynomial $t'(n)$ that determines the minimum stability for challenges selected by the choice of $t(n)$.

Theorem 16. *Consider an Arbiter PUF with n stages. Let $t(n)$ be any monotone increasing polynomial with $t(n) > 2$. Then for any polynomial $t'(n)$ and all challenges c satisfying* $\mathrm{Stab}(c) \geq \frac{1}{2} + \frac{1}{t(n)}$ *there exists a polynomial $r(n)$ such that when using $r(n)$ votes ($r(n)$ odd for all n), we have* $\mathrm{Stab}^{(r)}_{\mathrm{MV}}(c) > 1 - \frac{1}{2^{t'(n)}}$.

Proof. We consider the probability $q(n) = 1 - \mathrm{Stab}^{(r)}_{\mathrm{MV}}(c)$ that the result of majority voting does not match the model value, that is, the minority of votes show the model value. (For succinct presentation of the proof, we suppress the dependency on n in some places.)

By the majority vote, we get $q = \sum_{j=0}^{(r-1)/2} \binom{r}{j} \mathrm{Stab}(c)^j \left(1 - \mathrm{Stab}(c)\right)^{r-j}$. Considering the addends of the sum separately, for any $0 \leq j \leq \frac{r-1}{2}$, we set $m = \frac{r}{2} - j$ and rewrite

$$\begin{aligned} &\mathrm{Stab}(c)^j \left(1 - \mathrm{Stab}(c)\right)^{r-j} \\ &= \left(\mathrm{Stab}(c)(1 - \mathrm{Stab}(c))\right)^{r/2} \cdot \left(\frac{1 - \mathrm{Stab}(c)}{\mathrm{Stab}(c)}\right)^m . \end{aligned}$$

Note that the first factor is independent of j. As $\left(\frac{1-\mathrm{Stab}(c)}{\mathrm{Stab}(c)}\right)^m < 1$, we obtain the upper bound $q < \left(\mathrm{Stab}(c)\left(1 - \mathrm{Stab}(c)\right)\right)^{\frac{r}{2}}$. By definition of Stab and the prerequisite we have $q < \left(1 - \mathrm{erf}^2\left(\frac{|\Delta D_{\mathrm{Model}}|}{\sqrt{2}\sigma_{\mathrm{Noise}}}\right)\right)^{\frac{r}{2}}$ and choosing $r(n) = 2 \cdot \left\lceil \ln 2 \cdot t(n)^2 \cdot t'(n) \right\rceil + 1$, we obtain $q < 2^{-t'(n)}$. Finally, we have $\mathrm{Stab}^{(r)}_{\mathrm{MV}}(c) = 1 - q > 1 - \frac{1}{2^{t'(n)}}$. □

Theorem 16 shows that certain challenges can be boosted to stability exponentially close to 1 with a polynomial number of votes in the majority vote process. For applications of this work, it is essential that the portion of challenges that cannot be boosted is negligible. The next lemma shows that we can expect the number of challenges not satisfying the prerequisites of Theorem 16 to be small relative to $t(n)$.

Lemma 17. $\Pr[\mathrm{Stab}(c) < \frac{1}{2} + \frac{1}{t(n)}] < \frac{4}{\sqrt{\pi}} \cdot \frac{\sigma_{\mathrm{Noise}}}{\sigma_{\mathrm{Model}}} \cdot \frac{1}{t(n)}$.

The proof follows from Lemma 14 along with the standard bounds $\frac{2}{\sqrt{\pi}} \cdot x \cdot e^{-x^2} < \mathrm{erf}(x) < \frac{2}{\sqrt{\pi}} \cdot x$ (for $x > 0$) and $\mathrm{erf}^{-1} x < x$ (for $x \in (0, 1/2)$).

3.7.4. XOR Arbiter PUF

The Arbiter PUF stability boosted using majority voting as shown in Section 3.7.3 allows us to combine a polynomial number $k(n)$ of Majority Vote Arbiter PUFs into a XOR Majority Vote Arbiter PUF while maintaining high stability, as we show in this section.

Definition 18. Consider k arbiter chains using majority vote with r votes each. Let $\rho_i(c) \in \{-1, 1\}$ with $1 \leq i \leq k$ be the majority vote result of the i-th chain on input $c \in \{-1, 1\}^n$. Let $\Delta D^{(i)}_{\text{Model}}(c)$ be the noise-free delay difference of the i-th chain on input c. We define the *stability of the* XOR *Majority Vote Arbiter PUF* $\text{Stab}^{(r)}_{\text{XOR}}$ to be

$$\text{Stab}^{(r)}_{\text{XOR}}(c) = \Pr_{\Delta D_{\text{Noise}}}\left[\prod_{i=1}^{k} \rho_i(c) = \prod_{i=1}^{k} \operatorname{sgn} \Delta D^{(i)}_{\text{Model}}(c)\right].$$

The probability $\text{Stab}^{(r)}_{\text{XOR}}(c)$ is bounded from below by the probability that all individual arbiter chain response bits match their respective model value. Although this bound is not tight, as it disregards the cases where any even number of response bits are flipped, it still yields the desired exponential bound.

Theorem 19. *Let $t(n)$ be any monotone increasing polynomial with $t(n) > 2$. Then for any polynomial $t'(n)$ and for any challenge c with $\text{Stab}_i(c) \geq \frac{1}{2} + \frac{1}{t(n)}$ simultaneously in all arbiter chains $1 \leq i \leq k$, we have for an XOR Majority Vote Arbiter PUFs with $r(n) = 2 \cdot \lceil \ln 2 \cdot t(n)^2 \cdot t'(n) \rceil + 1$ votes and k arbiter chains that*

$$\text{Stab}^{(r)}_{\text{XOR}}(c) \geq 1 - \frac{1}{\frac{1}{k} 2^{t'(n)}}.$$

Proof. For any $1 \leq i \leq k$, we have

$$\text{Stab}^{(r)}_{\text{XOR}}(c) \geq \text{Stab}^{(r)}_{\text{MV}i}(c)^k \geq \left(1 - \frac{1}{2^{t'(n)}}\right)^k \geq 1 - \frac{k}{2^{t'(n)}},$$

ignoring correct answers caused by two (actually, an even number of) noisy responses in the first step and using Bernoulli's inequality in the last step. □

As before with Theorem 16, we cannot expect all challenges to be boosted to this stability. Instead, we discuss how many challenges we can expect to fulfill the prerequisites of Theorem 19.

Lemma 20. *For k given arbiter chains with challenge stability Stab_i, the probability for a uniformly and randomly chosen c to have $\text{Stab}_i(c) \geq \frac{1}{2} + \frac{1}{t(n)}$ simultaneously for all chains $1 \leq i \leq k$ is greater than $1 - \frac{4}{\sqrt{\pi}} \cdot \frac{\sigma_{\text{Noise}}}{\sigma_{\text{Model}}} \cdot \frac{k}{t(n)}$, under the condition that $\frac{4}{\sqrt{\pi}} \cdot \frac{\sigma_{\text{Noise}}}{\sigma_{\text{Model}}} < t(n)$.*

This lemma follows directly from the probability bound of Lemma 17 and Bernoulli's inequality.

Although we cannot show *all* challenges to have stability exponentially close to 1, Lemma 20 shows that the number of challenges we do not boost is (polynomially) converging to zero with growing n. While not an improvement, this is also not a significant degradation of stability, comparing to a single arbiter chain (see Lemma 17).

Putting all previous results together, we come to the following conclusion.

Corollary 21. *Choosing challenges randomly and uniformly, we have for the stability of the XOR Majority Vote Arbiter PUF with k Arbiter PUFs and n stages each using majority vote with r votes each, that for any constants $\alpha \in [0, \frac{1}{2}]$ and $\alpha' > 1$, there exists a number of votes $r \in O(\alpha^2 \cdot \alpha' \cdot k^2 \cdot \log k)$, such that*

$$\Pr\left[\mathrm{Stab}^{(r)}_{\mathrm{XOR}}(c) \geq 1 - \frac{1}{2^{\alpha'}}\right] \geq 1 - \alpha,$$

i.e. the proportion of challenges defined by α has stability exponentially close in α' to 1.

Proof. This result follows from Theorem 16, Lemma 17, and Theorem 19 by setting $t(n) = \frac{4}{\sqrt{\pi}} \cdot \frac{\sigma_{\mathrm{Noise}}}{\sigma_{\mathrm{Model}}} \cdot k \cdot \alpha$ and $t'(n) = \log_2 k + \alpha'$, where k is a monotone increasing polynomial in n. □

The choice of α affects the portion of challenges that will be boosted to exponential stability. Equivalently, α determines how many challenges we do not prove to be boosted to desired stability. The choice of α' sets up the boosting goal, i.e. how close to 1 we want the stability to be boosted to.

3.7.5. Number of Votes Required

For a real-world implementation, we need to know a lower bound on how many votes are required for the XOR Majority Vote Arbiter PUF to achieve the desired stability. In the simplest case, we require that all challenges c with $\mathrm{Stab}(c) \geq \frac{1}{2} + \frac{1}{10} = 60\%$ must have $\mathrm{Stab}^{(r)}_{\mathrm{XOR}}(c) > 99\%$. By Lemma 14, this captures most challenges, as

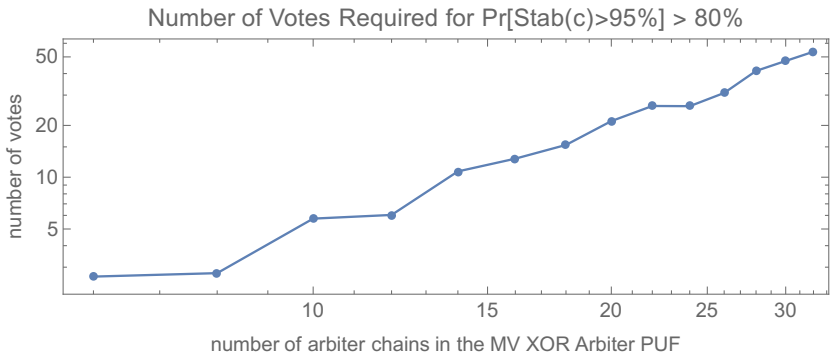
$$\Pr\left[\mathrm{Stab}(c) < \frac{1}{2} + \frac{1}{10}\right] = \mathrm{erf}\left(\frac{\sigma_{\mathrm{Noise}}}{\sigma_{\mathrm{Model}}}\,\mathrm{erf}^{-1}(0.2)\right) < \mathrm{erf}\left(0.2\frac{\sigma_{\mathrm{Noise}}}{\sigma_{\mathrm{Model}}}\right).$$

As an example, we obtain $\Pr[\mathrm{Stab}(c) < \frac{1}{2} + \frac{1}{10}] < 3\%$ for $\frac{\sigma_{\mathrm{Noise}}}{\sigma_{\mathrm{Model}}} = \frac{1}{10}$. From Theorem 19, we hence have $t(n) = 10$ and

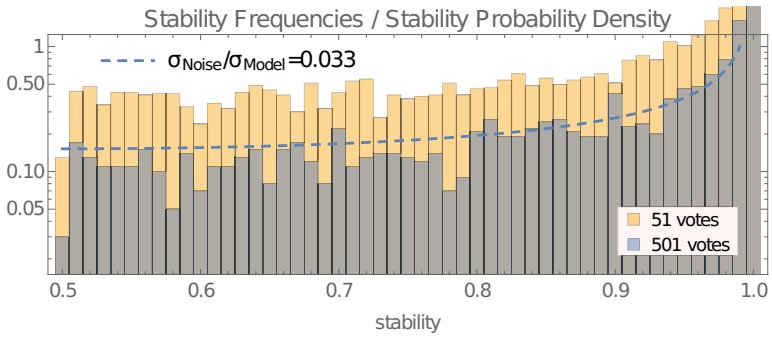
$$\mathrm{Stab}^{(r)}_{\mathrm{XOR}}(c) > 1 - \frac{1}{\frac{1}{k}2^{t'(n)}} = 99\%$$

and $t'(n) = \log_2(100k)$. Along with the number of votes $r(n)$ as defined in Theorem 16, we obtain $r(n) = 2 \cdot \lceil \ln 2 \cdot t(n)^2 \cdot t'(n) \rceil + 1 \approx 1 + 200 \log(100k)$; or $r(n) \approx 1400$ for $k = 24$. Notice that the upper bound grows with $O(\log k)$. Our simulations show that in fact much lower numbers for $r(n)$ suffice, see Section 3.7.6. The large values of $r(n)$ are hence due to the many non-tight bounds we use in the proof of Theorem 16 and Theorem 19.

Any implementation of this scheme will hence need to have k log-size memory units for storing the vote-count and will have approximately polynomial slowdown in evaluation, caused by the repeated evaluation of the Arbiter PUFs. The circuit size of the PUF overall will increase linearly with $O(k)$.



(a) The graph shows the minimum number of votes needed such that for a uniformly random challenge c we have $\Pr[\text{Stab}(c) \geq 95\%] \geq 80\%$ for different k, as determined by binary search on a simulation (Section 3.7.6). The simulation uses arbiter chain length of $n = 32$, however our results indicate that the number of required votes is independent of n. This log-log graph confirms the result that the number of votes required grows polynomially.



(b) The histogram shows the probability density of an XOR Majority Vote Arbiter PUF of size $k = 32$ and chain length of $n = 32$. We used 51 and 501 votes to boost stability to $\Pr[\text{Stab}(c) \geq 95\%] \geq 80\%$ and to the stability of the building block arbiter chains, respectively. The dashed line shows the theoretical stability probability density for a single arbiter chain (i.e. before majority vote and XOR) as used in this simulation ($\sigma_{\text{Noise}}/\sigma_{\text{Model}} = 0.033$). The graph confirms that a XOR Majority Vote Arbiter PUF built from these arbiter chains and the given number of votes can not only achieve a decent stability (at 51 votes), but also reach the same stability as a single arbiter chain (at 501 votes).

Figure 3.7.3.

3.7.6. Simulation

In this section we present software[5] simulation results on Arbiter PUF simulations that empirically confirm our theoretical results on the XOR Majority Vote Arbiter PUF.

To show that a high total stability can be reached using majority vote we determined the minimum number of votes required for acceptable stability for various numbers of parallel arbiter chains. Figure 3.7.3a shows detailed results of how many votes are needed for stable responses ($\Pr[\text{Stab}(c) \geq 95\%] \geq 80\%$), as determined by a binary search on the voting count for each k. The polynomially increasing number of votes required to fulfill the stability requirements for a useful PUF implementation shows that XOR Majority Vote Arbiter PUFs with high stability can be built arbitrarily large.

For a more precise exposition of the stability distribution of XOR Majority Vote Arbiter PUFs of size k we estimated stability values by a simulation. Our example uses $k = 32$ Majority Vote Arbiter PUFs with $n = 32$ stages each. The stability distribution shown in Figure 3.7.3b was achieved using $r \in \{51, 501\}$ votes. As estimated by the computation of the previous section, the number of votes needed to reach a stability of 95% with probability 80% is 51. Approximately 501 votes are needed to achieve a total stability comparable to the stability of a single Arbiter PUF that was used to build the XOR Majority Vote Arbiter PUF. This shows that even large XOR Majority Vote Arbiter PUFs can become stable using a feasible number of votes.

This allows us to conclude that the XOR Majority Vote Arbiter PUF, while exhibiting a larger hardware attack surface, can be considered secure against the known message attacks of Section 3.4 and Section 3.6. To the best of our knowledge, no other applicable known message attacks have been published, so that the XOR Majority Vote Arbiter PUF can be considered secure under EUF-KMA.

3.8. Reliability-Based Attacks

As we have seen in the previous sections, n-bit k-XOR Arbiter PUFs are insecure with respect to EUF-KMA for at least a certain set of security parameters. On the other hand, empirical study shows that the complexity of the Logistic Regression Attack (Section 3.4) and neural network attacks (Section 3.6) increases exponentially with the choice of the security parameter k. This raises the question if XOR Arbiter PUFs can be secure if the number of XORs k is chosen sufficiently high.

Becker [Bec15] answered this question with respect to EUF-CMA to the negative. In his work, he demonstrated a *reliability-based* attack against XOR Arbiter PUFs which takes advantage of the correlation of reliability of the response to a fixed challenge and the delay difference value for this challenge, which was first observed by Delvaux and Verbauwhede [DV13].

Both Delvaux and Verbauwhede and Becker use the following observation of the Arbiter PUF to recover information about the noise PUF delay value (Equation (3.1.1)).

[5] Code at `https://github.com/nils-wisiol/pypuf/tree/2017-why-attackers-lose`

Theorem 22. *For an Arbiter PUF f with noisy PUF delay value, the probability (taken over several evaluations of the same challenge) that f outputs -1 when queried with c is*

$$\Pr\left[-1 \leftarrow f(c)\right] = \frac{1}{2} + \frac{1}{2} \cdot \operatorname{erf}\left(\frac{-\Delta D_{\text{Model}}(c)}{\sqrt{2}\sigma_{\text{Noise}}}\right),$$

where $\Delta D_{\text{Model}}(c) = \langle w, x\rangle + b$ for intrinsic parameters $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$ and feature vector x derived from c, and σ^2_{Noise} is the variance of the noise as defined in Equation (3.1.1).

Proof. Assuming the noisy PUF delay value defined in Equation (3.1.1), $f(c)$ outputs -1 if and only if

$$\Delta D_{\text{Model}}(c) + \Delta D_{\text{Noise}} \leq 0.$$

As $\Delta D_{\text{Noise}} \sim \mathcal{N}\left(0, \sigma^2_{\text{Noise}}\right)$, the probability for this event is

$$\Pr\left[\Delta D_{\text{Noise}} \leq -\Delta D_{\text{Model}}(c)\right] = \frac{1}{2} + \frac{1}{2} \operatorname{erf}\left(\frac{-\Delta D_{\text{Model}}(c)}{\sigma_{\text{Noise}}\sqrt{2}}\right).$$

□

By the definition of the expected value, we have that

$$\Pr\left[-1 \leftarrow f(c)\right] = \frac{1}{2} - \frac{1}{2} \operatorname{E}\left[r \mid r \leftarrow f(c)\right].$$

As an attacker can approximate the value of $\operatorname{E}\left[r \mid r \leftarrow f(c)\right]$ by sampling $r \leftarrow f(c)$ many times, they can also approximate the value of $\Delta D_{\text{Model}}(c)$ by the means of Theorem 22 via

$$\Delta D_{\text{Model}}(c) = \sqrt{2}\sigma_{\text{Noise}} \operatorname{erf}^{-1}\left(\operatorname{E}\left[f(c)\right]\right). \tag{3.8.1}$$

This method of estimation is very precise, even for relatively small sample size, cf. Fig. 3.8.1.

As a result, Delvaux and Verbauwhede [DV13] were able to recover the intrinsic parameters $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$ of an n-bit Arbiter PUF by estimating $\Delta D_{\text{Model}}(c)$ with $\widehat{\Delta D_{\text{Model}}}(c)$ for n challenges $c_1, \ldots, c_n$ with linearly independent feature vectors $x_1, \ldots, x_n$ and then finding a solution to the resulting system of linear equations

$$\begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & \ddots & & \\ \vdots & & & \\ x_{n,1} & & & x_{n,n} \end{pmatrix} \begin{pmatrix} w_1 \\ \cdots \\ \\ w_n \end{pmatrix} + b = \begin{pmatrix} \widehat{\Delta D_{\text{Model}}}(c_1) \\ \vdots \\ \\ \widehat{\Delta D_{\text{Model}}}(c_n) \end{pmatrix}.$$

As the approximations are not perfect, the solution to this system may not provide a good model for the Arbiter PUF under attack. To improve accuracy, the attacker can use (much) more than n challenges and find a least squares solution to the resulting over-determined system of linear equations.

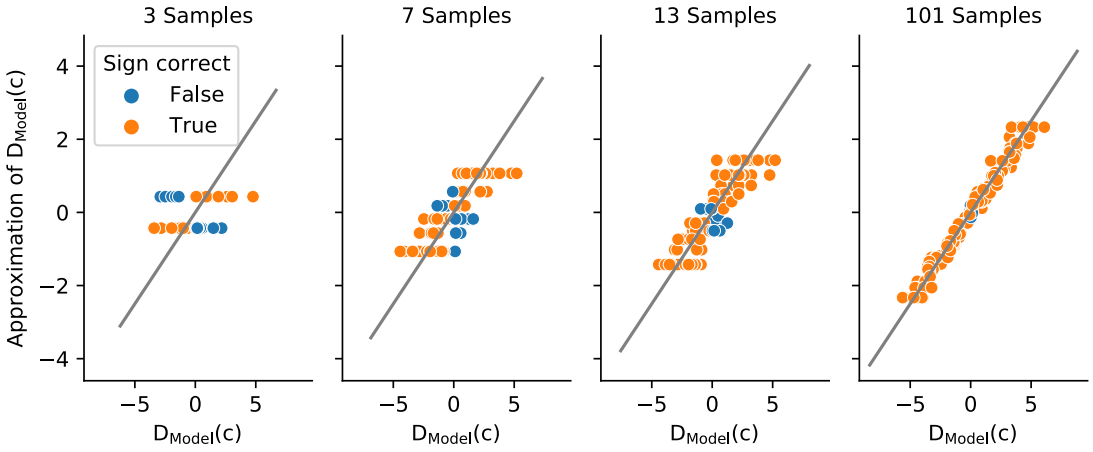


Figure 3.8.1.: Error made by estimation of $\Delta D_{\text{Model}}(c)$ when estimating via Equation (3.8.1) by sampling $\mathrm{E}[f(c)]$.

The general idea of this chosen message attack can be extended to also cover XOR Arbiter PUFs. Becker [Bec15] observed that for an XOR Arbiter PUF f that consists of Arbiter PUFs $f_1, \ldots, f_k$, and for a list of N challenges $c_1, \ldots, c_N$, the vector R defined by expected responses of f

$$R = (\mathrm{E}\left[r \mid r \leftarrow f(c_i)\right]) \in \mathbb{R}^N$$

correlates with the vector defined by the *individual* expected responses of the Arbiter PUFs,

$$R_j = (\mathrm{E}\left[r \mid r \leftarrow f_j(c_i)\right]) \in \mathbb{R}^N, \quad 1 \leq j \leq k.$$

Becker notes that this correlation is caused by the fact that the XOR Arbiter PUF will shown unstable responses for a given challenge c whenever *at least one* of the Arbiter PUFs shows an unstable response for c. We formalize this observation below. Based on this correlation, he randomly chooses k Arbiter PUF models and optimizes them individually such that R_j has high correlation with an sample-mean-based approximation of R. While the original work uses an evolution strategy for optimization, Tobisch, Aghaie, and Becker [TAB21] later showed that this can also be done by using a gradient-descent method.

This method was empirically shown to have runtime which polynomially increases with the number of used Arbiter PUFs k and thus demonstrates that the XOR Arbiter PUF is insecure under EUF-CMA in all studied parameter settings.

The XOR Majority Vote Arbiter PUF (Section 3.7) can also be considered vulnerable to this attack, as the attacker can query challenges many times to identify unstable responses. This fact along with the runtime being linear in the security parameters n and k deprives the design of its exponential advantage over the (chosen message) attacker.

4. XOR Arbiter PUFs with Input Transformation

In Chapter 3, we have shown that XOR Arbiter PUFs are vulnerable to known message attacks for any security parameter n and feasible security parameter k. While the XOR Majority Vote Arbiter PUF can mitigate the relevant known message attacks, we have shown that it cannot mitigate the reliability attack (Section 3.8) and thus XOR Arbiter PUF and XOR Majority Vote Arbiter PUF are insecure under chosen message attacks.

This section explores an extension of the XOR Arbiter PUF design space. It is based on the observation that in the original XOR Arbiter PUF design [SD07] as introduced in Chapter 3, all involved Arbiter PUFs receive the same challenge. In this chapter, we consider designs where the Arbiter PUFs receive individual challenges which have been derived from the given challenge.

4.1. Input Transformations: Classic vs. Random

When XOR Arbiter PUFs were proposed by Suh and Devadas [SD07], the first step was to provide all arbiter chains with the same challenge (here called *classic* design). Subsequently, Majzoobi, Koushanfar, and Potkonjak [MKP08] proposed to modify the challenge before feeding it into the individual arbiter chains, to let the PUF fulfill the strict avalanche criterion. Although initially designed to harden XOR Arbiter PUFs against chosen message attacks, it became clear that the design twist also has an impact on the known message logistic regression attack introduced by Sölter [Söl09] and Rührmair et al. [Rüh+10]. In this chapter, we generalize the idea of transforming challenges for each arbiter chain and call it *input transformation*. To shed some light on how machine learning hardness can be increased using an input transformation, we study the impact of input transformations on the success rate of the Logistic Regression Attack (Section 3.4).

We use the additive delay model introduced in Theorem 8 and Equation (3.2.1) and extend it to model responses as

$$f(c) = \operatorname{sgn}\left[\prod_{l=1}^{k}\left(\left\langle w^{(l)}, x^{(l)}\right\rangle + b^{(l)}\right)\right],$$

i.e., we extend Equation (3.2.1) to use k potentially different feature vectors $x^{(l)}$ derived by setting $x_i^{(l)} = \prod_{j=i}^{n} c_j^{(l)}$, where $c^{(l)}$ is the *individual* challenge for the l-th arbiter chain, computed from a single master-challenge $c \in \{-1, 1\}^n$. We call a list of functions

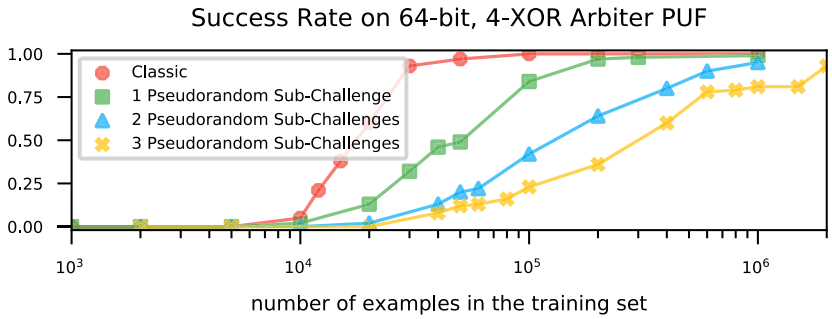


Figure 4.1.1.: Success rate of logistic regression attacks on simulated XOR Arbiter PUFs with 64-bit arbiter chains and four arbiter chains each, based on at least 250 samples per data point shown. Accuracy better than 70% is considered success, but we only observe accuracy around 50% and 99%. Four different designs are shown: of the four arbiter chains in each instance, an input transform is used that transforms zero, one, two, and three challenges pseudorandomly, keeping the remaining challenges unmodified.

$(\tau^{(1)}, ..., \tau^{(k)})$ with $\tau^{(l)} : \{-1, 1\}^n \to \{-1, 1\}^n$ that transform the master-challenge c into sub-challenges $\tau^{(l)}(c) = c^{(l)}$ the *sub-challenge generators*.

In this extended notation, we can write the classic design as $\tau^{(l)} = \mathrm{id}$ and hence have $c^{(l)} = \tau^{(l)}(c) = c$ and $x^{(l)} = x$ for all l.

For our analysis, the feature vector structure for a given input transformation is crucial. We hence denote the feature vector $x^{(l)}$ belonging to a given master-challenge c as $\sigma^{(l)}(c)$ and formally define *input transformation* to be the list of functions $(\sigma^{(1)}, ..., \sigma^{(k)})$ that transforms the master-challenge into the feature vectors.

For a given input transformation $(\sigma^{(1)}, ..., \sigma^{(k)})$, we can thus write the responses of f to challenge c as

$$f(c) = \mathrm{sgn}\left[\prod_{l=1}^{k} \left\langle w^{(l)}, \sigma^{(l)}(c) \right\rangle + b^{(l)}\right]. \tag{4.1.1}$$

4.1.1. Pseudorandom Input Transformation

We demonstrate the influence of input transformations on the learning hardness of logistic regression attacks in Figure 4.1.1. To contrast the classic design, where all arbiter chains receive the same challenge, we implemented a simulation of XOR Arbiter PUFs with pseudorandom sub-challenge generators, where all arbiter chains receive an individual pseudorandom challenge chosen by seeding the generator with the master-challenge and the index of the sub-challenge. For our implementation, we used a pseudorandom generator based on the Mersenne Twister. Assuming security of the pseudorandom generator, we can guarantee that the sub-challenges are chosen indistinguishable from truly random sub-challenges and feature vectors (for all polynomially time-bounded observers, i.e. including the machine learning attacker).

By the absence of any observable correlation, the pseudorandom input transformation is, while not being a reasonable real-world design choice, an extremal example among all input transformations. As elaborated in Section 4.2, the absence of correlation results in an increase of hardness for the Logistic Regression Attack.

The empirical results match this rationale: Figure 4.1.1 shows that, compared to the classic design, the required size of the training set to achieve a high success rate increases substantially. Figure 4.1.1 also shows designs in which only a subset of arbiter chains receive pseudorandom challenges, whereas the others receive the same unmodified challenge. For those designs, the required size of the training set is, as could be expected, in between the pure classic and the pure pseudorandom case.

4.1.2. Local Minima

Logistic regression uses gradient descent over the log-likelihood function L defined by the provided training set to conduct the modeling attack (Section 3.4). The algorithm's ability to find a "good" minimum depends, among many other parameters, on the algorithm's random initialization. Empirical results obtained by repeatedly attacking the same XOR Arbiter PUF show that the probability to guess successful initializations significantly changes with the input transformation in use (Figure 4.3.1).

Whenever an input transformation of an XOR Arbiter PUF sends the same challenge to several arbiter chains, this will be reflected in function L as symmetry. Using the classic input transformation, the attacker has at least $k!$ equally good minima[1] to choose from. This idea of L's symmetry can be generalized to the case where properties of the input transformation allow permutations of the original weights to approximate the XOR Arbiter PUF with mediocre accuracy, as we will show in Section 4.2.2. The approximating permutations can be observed as local minima in the logistic regression attack. On the contrary, using pseudorandom transformations, we can reduce the symmetries of L down to the minimum, hence increasing machine learning hardness and avoiding any intermediate solutions.

4.2. Input Transformations: Lightweight Secure

The Lightweight Secure PUF design was introduced by Majzoobi, Koushanfar, and Potkonjak [MKP08]. The design proposes an input transformation presented in two steps.

First, for the generation of the l-th sub-challenge $c^{(l)}$, the master-challenge is rotated by l bits, here denoted by $d^{(l)}$. Second, the sub-challenge $c^{(l)}$ will mostly be computed by XORing bits pairwise, such that it consists of three parts with length $n/2$, 1, and $n/2 - 1$, respectively.

[1] Strictly speaking, all models will have an infinite amount of local minima, as all weights in the model can be modified by a small value or scaled by a positive scalar without affecting the model's behavior. To fix above argument we can argue that additional symmetry causes the gradient descent to remain at a local minima with higher probability.

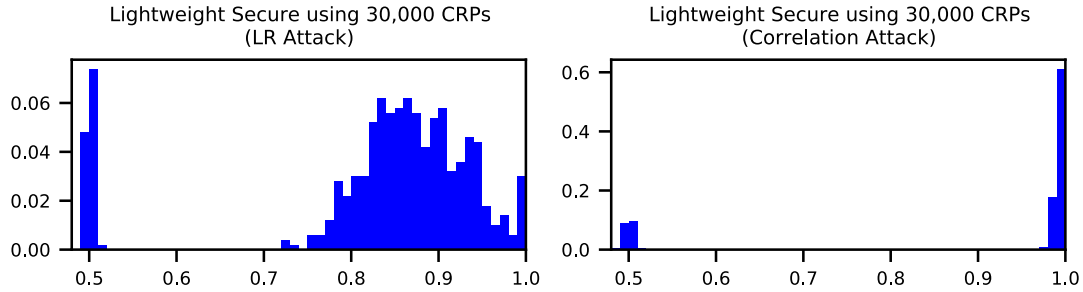


Figure 4.2.1.: Accuracy distribution for learning attempts on randomly chosen simulated 64-bit 4-XOR Lightweight Secure PUFs. Using the Logistic Regression Attack, many learning attempts end with an intermediate result, while all other input transformations studied in this work do not show such behavior. It can be seen that using our new correlation attack, the resulting model accuracy is increased significantly over the plain LR attack.

More specifically, we have

$$\begin{aligned} \left(c_1^{(l)}, ..., c_{n/2}^{(l)}\right) &= \left(d_1^{(l)}d_2^{(l)},\ d_3^{(l)}d_4^{(l)},\ \dots,\ d_{n-1}^{(l)}d_n^{(l)}\right), \\ \left(c_{n/2+1}^{(l)}\right) &= \left(d_1^{(l)}\right), \\ \left(c_{n/2+2}^{(l)}, ..., c_n^{(l)}\right) &= \left(d_2^{(l)}d_3^{(l)},\ d_4^{(l)}d_5^{(l)},\ \dots,\ d_{n-2}^{(l)}d_{n-1}^{(l)}\right). \end{aligned} \tag{4.2.1}$$

We will refer to the sub-challenges with $\tau^{(1)}(c), ..., \tau^{(k)}(c)$ and to the feature vectors they induce with $\sigma^{(1)}(c), ..., \sigma^{(k)}(c)$.

The transformation is chosen such that the Strict Avalanche Criterion is (almost) satisfied [MKP08], i.e., a single bit flip in the master challenge will result in bit flips in about 50% of the elements of each *feature* vector for each arbiter chain. If 50% of the *feature* vector bits flip, then the PUF output also flips with probability 50%.

In this work, we will not consider weaker versions of the Lightweight Secure PUF with multiple output bits.

4.2.1. Feature Vector Correlation

Using the Logistic Regression Attack against the XOR Arbiter PUF, we observe that each attack attempt either yields a near optimal model that has a predictive accuracy of around 99%, or yields a model that performs poorly in prediction, barely exceeding an accuracy of 50%, i.e., random guessing. Interestingly, this is not the case for the Lightweight Secure PUF, as can be observed in Figure 4.2.1. We found that the machine learning algorithm yielded models that performed clearly better than random guessing but did not achieve the desired accuracy of around 99%.

In empirical experiments done on simulations of the Lightweight Secure PUF (based on the additive delay model), we found that weight vectors of the intermediate solutions

consisted mostly of a permutation of the weight values of the PUF simulation. In fact, by permuting the individual weight vectors of the arbiter chains and rotating them for certain but distinct amounts, a close approximation of the original weight vectors could be constructed.

To give a theoretical basis to our attack, we formalize this observation by examining the impact of swapping and rotating two different weight vectors. Let $w^{(1)}, ..., w^{(k)}$ be the weight vectors of a Lightweight Secure XOR Arbiter PUF. Our observations suggest that this PUF can be approximated when the i-th and j-th weight vectors are swapped and shifted in a characteristic way. We call the weight vectors to be swapped $w = w^{(i)}$ and $v = w^{(j)}$ and the corresponding input transformation functions $\lambda = \sigma^{(i)}$ and $\mu = \sigma^{(j)}$. Note that this argument uses feature vectors, not sub-challenges. Consider the relevant part of the product in the XOR Arbiter PUF model (cf. (4.1.1) and (4.2.1)), omitting the bias terms corresponding to w and v:

$$\langle w, \lambda(c)\rangle \cdot \langle v, \mu(c)\rangle = \sum_{i,j} w_i \cdot v_j \cdot \lambda(c)_i \cdot \mu(c)_j$$

In the following, we compare this to the model where the weight vectors v and w are swapped and rotated by π and π^{-1}, respectively. That is, we substitute w by $\pi^{-1}(v)$ and substitute v by $\pi(w)$:

$$\begin{aligned}\left\langle \pi^{-1}(v), \lambda(c)\right\rangle \cdot \langle \pi(w), \mu(c)\rangle &= \sum_{i,j} \pi(w)_i \cdot \pi^{-1}(v)_j \cdot \mu(c)_i \cdot \lambda(c)_j \\ &= \sum_{i,j} w_i \cdot v_j \cdot \pi^{-1}(\mu(c))_i \cdot \pi(\lambda(c))_j \qquad \text{(re-numbering } i, j\text{)}\end{aligned}$$

To prove that the latter is an approximation of the original model, we studied the relationship of $\pi^{-1}(\mu(c))_i \cdot \pi(\lambda(c))_j$ and $\lambda(c)_i \cdot \mu(c)_j$ and found that for most pairs i, j, we have equality with significant probability for a uniformly random master-challenge c. The higher this probability, the better the approximation of the original model by the swapped and rotated version is.

The correlation of the Lightweight Secure input transformation $\sigma^{(1)}, \delta^{(2)}, ...$ can be measured by

$$\frac{1}{(n+1)^2} \sum_{i=1}^{n+1} \sum_{j=1}^{n+1} \Pr_c \left[\lambda(c)_i \cdot \mu(c)_j = \pi^{-1}(\mu(c))_i \cdot \pi(\lambda(c))_j\right], \tag{4.2.2}$$

where c is chosen uniformly at random. For each pair, there is exactly one rotation π which produces a significant correlation, cf. Table 4.1.

For example, consider the 64-bit 4-XOR Lightweight Secure PUF, where we write $(\sigma^{(1)}, \sigma^{(2)}, \sigma^{(3)}, \sigma^{(4)})$ for the input transformation and denote $\sigma^{(1)}$ as λ and $\sigma^{(2)}$ as μ. If we rotate the first feature vector $\lambda(c)$ by 32, say $\pi(\lambda(c))$, and the second feature vector $\mu(c)$ by the inverse of 33 positions to the right, say $\pi^{-1}(\mu(c))$, then we have high correlation as defined by Equation (4.2.2).

	1	2	3	4	5	6
1	—	32/98%	64/97%	31/95%	63/94%	30/92%
2	33/98%	—	32/98%	64/97%	31/95%	63/94%
3	1/97%	33/98%	—	32/99%	64/97%	31/95%
4	34/95%	1/97%	33/99%	—	32/98%	64/97%
5	2/94%	34/95%	1/97%	33/98%	—	32/98%
6	35/92%	2/94%	34/95%	1/97%	33/98%	—

Table 4.1.: Overview of correlations for a 64-bit 6-XOR Lightweight Secure Arbiter PUF. As an example, the feature vectors of the first and second arbiter chain show a correlation of 98% as defined in (4.2.2) with a rotation by 32 and 33 positions, respectively. Hence, the corresponding weight vectors can be swapped if they are rotated accordingly without significant change in the model accuracy.

As we can see, the correlation of the feature vectors leads to the fact that an approximation of the original model can be constructed by swapping two weight vectors and rotating them accordingly. Using this concept iteratively, any permutation of the weight vectors can be achieved.

Our empirical results in Figure 4.2.1 suggest that those partial solutions also generate local minima to which the regression algorithm converges. The combination of the information on the local minimum along with the correlation as outlined above can be used to stage a known message attack on the input transformation by Majzoobi, Koushanfar, and Potkonjak. This must be considered a key weakness of the Lightweight Secure transformation, as our empirical attack results show.

The cause for this symmetry lies in the definition of the input transformation and in the fact that results are XORed. There is a clear pattern and essentially every pair of PUFs can be exchanged by a rotated version, although the correlation decreases the further the PUF positions are apart from each other.

4.2.2. Improved Attack

As seen in the previous section, the Logistic Regression Attack on the k-XOR Lightweight Secure PUF often leads to local minima that model the PUF behavior only with a limited accuracy. In this section, we show how such a local minimum can be used to find a high-accuracy model.

If the attack has found a model with an intermediate accuracy in the range of 65%-98%, we assume that the initialization values for the attack lead to a swapped and rotated version of the high-accuracy version of the weights. Instead of restarting the attack from scratch with new initialization of the model weights until we find a high-accuracy solution and hence the correct ordering, our *correlation attack* guesses the correct ordering of the weights. To that end, we first generate the rotated weights for each possible permutation of the weight vectors in a brute-force manner and check their accuracy on a validation set. As a second step, the $2k$ most accurate rotated weights are used to restart the

n	k	CRPs	LR on Classic	LR on LW Secure	Correlation Attack on LW Secure	LR on Permutation PUF
64	4	12k	0m 33s	10m 11s	0m 58s	24m 50s
64	4	30k	0m 31s	3m 57s	0m 44s	4m 45s
64	5	300k	7m 03s	3h 03m	11m 07s	13h 59m
64	6	1M	42m 30s	8 days	1h 42m	(96h 00m)*
64	7	2M	75h 07m	(20 days)*	8 days	(16 days)*
128	4	1M	20m 31s	2h 53m	51m 23s	58m 38s
128	5	2M	1h 35m	35h 20m	3h 17m	(16 days)*

Table 4.2.: Expected time until the first success for attacks on classic XOR Arbiter PUF, Lightweight Secure XOR Arbiter PUF, and Permutation-Based XOR Arbiter PUF. An accuracy of at least 98% is considered success, all entries are based on 1000 samples. Runs with no success are marked with an asterisk (*).

logistic regression attack and refine the weights.

Although the first step has run time $O(k!)$, this procedure can outperform the simple restarting of the LR attack (Table 4.2) for practical values of k. Furthermore, the restarted logistic regression algorithm can use a much lower bound on the maximum number of epochs, discarding low-accuracy solutions rapidly. To achieve fast run times, we used a small validation set for the $k!$ accuracy computations. We empirically found that rotations with high initial accuracy have a higher chance to yield a high-accuracy solution, hence the ordering by initial accuracy helps to speed up the attack.

More specifically, we examined the ranking of the permutation that resulted in the highest accuracy solution for 1000 instances of 64-bit 6 XOR Lightweight Secure PUFs. In most cases, the best permutation was found among the first 10 candidates.

In Table 4.2 we give an overview over our correlation attack. To reflect the time unsuccessfully spent training a model, we define for chosen security parameters, training set size, and employed computing resources the *time until first success* as the expectation of time spend until a model with prediction accuracy higher than 98% is obtained. To empirically approximate the time until first success of our attacks, for each group of experiments we computed the mean time of unsuccessful runs t_{fail}, and the mean time of successful runs t_{success}, as well as the relative frequency of successful runs h_{success}. Assuming a Geometric distribution, we compute the expected number of required trials until success as $n_1 = 1/h_{\text{success}}$ and the expected time until first success t_1 as,

$$t_1 = (n_1 - 1) \cdot t_{\text{fail}} + t_{\text{success}};$$

for $h_{\text{success}} = 0$ we set $t_1 = \infty$. We point out that different instances of XOR Arbiter PUFs may differ in their resistance to modeling attacks [TB15], and t_1 only refers to the average time until success, not ruling out the possibility that some instances of the given size may be harder or easier to model. All results shown in this work are with respect to the time until first success.

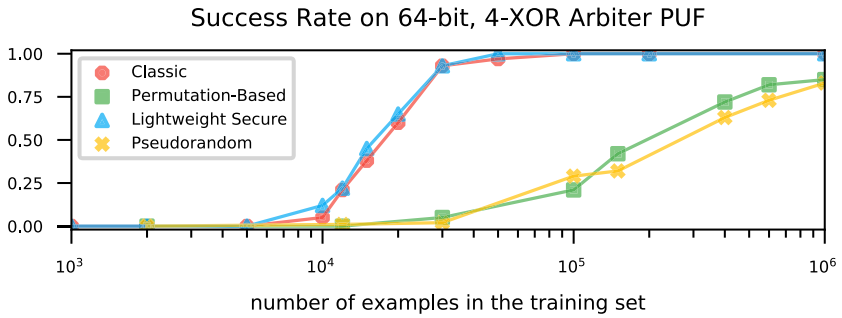


Figure 4.3.1.: Success rate of logistic regression attacks on simulated 64-bit 4-XOR Arbiter PUFs; accuracy above 70% is considered a success. Four different input transformations are shown: classic, Lightweight Secure [MKP08], the permutation-based input transformation proposed in this work, and a pseudorandom input transformation used as comparison. All data points are based off at least 80 samples. For a success threshold of 70%, Lightweight Secure and classic are equally hard to attack, whereas permutation-based and pseudorandom require significantly more CRPs.

Our results indicate that the Lightweight Secure PUF can be learned with much higher accuracy in less time than previously believed, with the security in some instances reduced to what the classic XOR Arbiter PUF provides. In contrast, the XOR Arbiter PUF with the permutation-based input transformation defined in Section 4.3 is considerably harder to attack using logistic regression and does not posses the attack surface we used in the correlation-based attack, i.e. does not show intermediate results.

4.3. Permutation PUF

The previous sections show that input transformations have an impact on the machine learning resistance. When using the same challenges for all arbiter chains as done in the classic XOR Arbiter PUF, there are multiple equivalent solutions as the order of the weight vectors $(w^{(1)}, ..., w^{(k)})$ does not matter. Using pseudorandom sub-challenges ensures that only one order is valid and hence reduces the number of global minima in the gradient descent of the LR attack. However, including a pseudorandom generator in a PUF increases hardware attack surface as well as chip area and power requirements and is thus usually not considered an option.

To compromise, Yu et al. [Yu+16] suggested to use challenges generated by a 256 bit Linear Feedback Shift Register (LFSR) to feed the four arbiter chains in the used 64-Stages 4-XOR Arbiter PUF. The area overhead of their 4-XOR Arbiter PUF is stated as 1024 Gate Equivalents (GE). The size of the LFSR was not provided, but assuming 4.5 GE for a flip-flop, the size of a 256 stage LFSR is comparable to that of the PUF circuitry[2]. Implementing a cryptographically secure pseudorandom generator will consume

[2]Although the Gate Equivalents for a PUF circuit can be a bit misleading as PUFs need special isolated

even more resources.

In the Lockdown Protocol [Yu+16] the LSFR is an essential part of the authentication protocol and hence needed anyways. But for other designs, especially if larger PUF instances are used with 128 stages, a more efficient input transformation is advised as the overhead is not negligible. However, our analysis of the Lightweight Secure PUF shows that this input transformation suffers a significant weakness and cannot provide security better than the original XOR Arbiter PUF design. The fact that feature vectors correlate in a certain way simplifies the machine learning attack to the point where no relevant advantage over the classic design is achieved.

To mitigate above correlation-based attack on the Lightweight Secure XOR Arbiter PUF, we propose an input transformation that is easier to implement than the Lightweight Secure PUF solution but does not show any indication of local minima. The idea is to use k different, fix-point-free permutations $\pi_1, \ldots, \pi_k$ as sub-challenge generators. We hence obtain the sub-challenges $c^{(l)} = \pi_l(c)$. As this input transformation can be implemented in wiring, no additional gate is used in the PUF design. Note that a permutation of the challenges *does not* result in a permutation of the feature vectors. More detailed, the permutation of $c^{(l)}$ ensures that for different $x_i^{(l)} = \prod_{j=i}^{n} c_j^{(l)}$, different parities of the master challenge c are used. As different parity functions are uncorrelated, pairs of feature vectors do not show significant correlation according to Equation (4.2.2) even if they are permuted. We call this family of input transformations the *permutation-based input transformations*.

We empirically confirmed that this approach does not show any of the local minima we observed for the Lightweight Secure PUF. The machine learning resistance was instead comparable to the results of pseudorandom inputs (Figure 4.3.1), which represent and upper bound on input transformation quality as argued in Section 4.1. Without observing local minima or correlations, the attack described in Section 4.2.2 cannot be applied.

Additionally, this input transformation comes at nearly zero resource overhead. Compared to using a pseudorandom input transformation, the permutation-based transformation is more efficient in terms of area and power and is also more efficient than the input transformation proposed for the Lightweight Secure PUF.

For a concrete instantiation of the Permutation PUF, we suggest to choose random permutation as sub-challenge generators such that no generator has a fix point, that is,

$$\forall_{1 \leq l \leq 10} \forall_{i \in [64]} : \quad \Pr_c \left[\sigma^{(l)}(c_i) \neq c_i \right] = 1/2,$$

and such that no two permutations share the same value on any coordinate, i.e.

$$\forall_{l \in [10]} \forall_{l' \in [10]-l} \forall_{i \in [64]} : \quad \Pr_c \left[\sigma^{(l)}(c_i) \neq \sigma^{(l')}(c_i) \right] = 1/2.$$

A concrete instantiation is shown in Appendix B.

routing compared to conventional digital circuits such as LFSRs.

5. Interpose PUF

The reliability-based attack by Becker [Bec15] outlined in Section 3.8 demonstrated that XOR Arbiter PUFs and XOR Majority Vote PUFs are insecure under EUF-CMA for any security parameters and created the need for a replacement.

Nguyen et al. [Ngu+19] picked up an idea to hide the challenge information from the attacker that first appeared in the context of the Feed-Forward Arbiter PUF [Gas+04]. The hope was to defeat the attacks on XOR Arbiter PUFs shown in Section 3.2 by not disclosing the complete challenges to the attacker.

The proposed *Interpose PUF* consists of a combination of two XOR Arbiter PUFs. It is defined by a challenge length n, the number k_{up} of arbiter chains in the first XOR Arbiter PUF, and the number k_{down} of arbiter chains in the second XOR Arbiter PUF. The first XOR Arbiter PUF, called *upper layer*, has challenge length n. When given an input, its 1-bit response is interposed in the middle bit position of the second XOR Arbiter PUF (the *lower layer*), while the other n bits a filled with the same challenge as given to the upper layer. This results in a challenge length of $n + 1$ bit for the lower layer. A schematic representation of the Interpose PUF is depicted in Figure 5.0.1.

By virtue of the interposed bit on the lower layer, the Logistic Regression Attack cannot directly be applied on the Interpose PUF, as information is missing from the training set. Applying the attack naively with omission of the interpose bit, constant or a random interpose bit (*linearization attack* [Ngu+19]) will result in a maximum accuracy of 75%. Furthermore, as the response bit of the upper layer will influence the bottom layer response for approximately half of all challenges, the reliability-based attack on XOR Arbiter PUFs [Bec15] is also mitigated by the design [Ngu+19].

In this chapter, we present the *Splitting Attack* which extends the Logistic Regression Attack of Section 3.4 to be applicable to Interpose PUF. For known message attackers, we demonstrate that this reduces the security level of the Interpose PUF to that of an XOR Arbiter PUF of similar size.

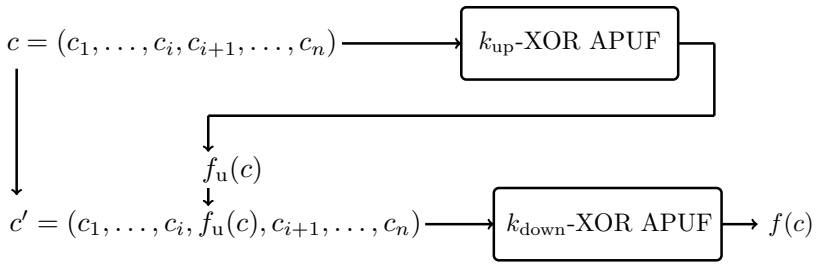


Figure 5.0.1.: Schematic view of an n-bit $(k_{\text{up}}, k_{\text{down}})$-Interpose PUF f on challenge c.

For chosen message attackers, Tobisch, Aghaie, and Becker [TAB21] demonstrated that reliability-based attack (Section 3.8) can also be extended to be applicable to the Interpose PUF.

5.1. Splitting Attack

The Splitting Attack is a modeling attack against the Interpose PUF that uses several runs of Logistic Regression Attack on training sets derived from the Interpose PUF. It returns a model for the full Interpose PUF comprised of two XOR Arbiter PUF models.

To describe the Splitting Attack, we first provide an intuition of the employed divide-and-conquer algorithm that separately models upper and lower layer of the Interpose PUF under attack. Section 5.1.1 describes how we obtain an initial high-accuracy model for the lower layer. Afterwards, Section 5.1.2 and Section 5.1.3 show how this paves the way to obtain a complete model.

Our proposed attack technique on the Interpose PUF uses a divide-and-conquer approach and is based on two crucial observations. First, when conducting the linearization attack, the resulting model will not only predict PUF responses with an accuracy up to 75%, but will contain all secret information about the lower layer of the Interpose PUF. That is, the reason for the relatively low accuracy of the linearization attack is not the missing information about the lower layer, but almost exclusively the missing challenge bit information. Second, the response bits of the upper layer can be heuristically guessed by the attacker for about half the known challenge-response pairs with high accuracy using Algorithm 5.1.

5.1.1. Initial Modeling of the Lower Layer via Random Interpose Bits

Using a given challenge-response set (C, R) collected from the Interpose PUF in the known message attack model, we argue why an attacker is capable of obtaining a high-accuracy model of the lower layer of the Interpose PUF. Possession of such a model subsequently enables the attacker to conduct the divide-and-conquer attack as described in the following sections.

Any challenge-response set (C, R) of the full Interpose PUF contains already n out of the $n+1$ challenge bits to the lower layer as well as the lower layer responses. Hence, the only information hidden from the attacker aside from the manufacturing imperfections are the challenge bits in the interpose position. However, our results show that this information is not required to obtain a high-accuracy model of the lower layer. Instead, the attacker can randomly guess the interpose bits, i.e., create the lower layer challenge-response set (C_d, R) by themselves where C_d is simply interposed with uniformly chosen random bits, i.e.

$$C_d = \left\{ \left(c_1, \ldots, c_{n/2}, \mathbf{c_r}, c_{n/2+1}, \ldots, c_n\right) \mid (c_1, \ldots, c_n) \in C, \mathbf{c_r} \sim_{\mathrm{u}} \{0,1\} \right\}.$$

As previous research has shown [MKP08; Ngu+19], the influence of the middle challenge bit of any XOR Arbiter PUF on the response bit is about 50%, i.e., in about half of

the challenges, the response bit will flip if the middle challenge bit is flipped. Applied to our situation, the response of the upper layer of the Interpose PUF will be irrelevant for the PUF's response in about 50% of cases. It follows that the information in (C_d, R) for that half of challenges – where the middle bit does not have an influence on the response – is correct. Furthermore, for the other half of challenges that do have an influence on the response bit, the attacker's guess will be correct with probability 50%, resulting in a total accuracy of 75% for the self-created CRPs (C_d, R) for the lower layer XOR Arbiter PUF.

Although the training set (C_d, R) has only an anticipated 75% accuracy on the lower layer XOR Arbiter PUF of the target Interpose PUF, the trained model obtained from learning with Logistic Regression will model the lower layer with very high accuracy. That is, the accuracy of the trained model surpasses the accuracy of the training set.

Recall that the Logistic Regression learning algorithm for XOR Arbiter PUFs uses a gradient descent algorithm to train an XOR Arbiter PUF model that agrees with the training set on as many as possible challenges (Section 3.4). This includes models that are "related" to physical parameters of the PUF under attack, such as ones with a permuted order of arbiter chains (as the XOR operation is commutative, cf. Chapter 4) and models where one half of the weights are negated (using negated interpose bits). We will refer to models using the original interpose bit as *non-negated*, and to models using negated interpose bits as *half-negated*. For both classes, small variations of the model weights and permutations of the arbiter chain order will agree with the training set on close to 75% of challenges.

On the other hand, with overwhelming probability, no unrelated XOR Arbiter PUF model will agree with this training set on a portion larger than 75% of its CRPs. This is due to the fact that the above constructed training set will very likely contain values that *cannot* be described with an XOR Arbiter PUF model[1]. Hence, the non-negated and half-negated models of the lower layer both constitute global minima in the Logistic Regression's loss function.

Figure 5.1.1 gives an overview of the lower layer's model accuracy when trained on the randomly interposed challenge-response set, which confirms that a high-accuracy model can be obtained from a partially guessed training set (C_d, R). The accuracy shown is with respect to both half-negated weights and non-negated weights, whichever is better. As we will see below, the random choice of a model class will not affect the final accuracy or run time of the attack in any way.

For variations of the Interpose PUF design it is important to note that this observation can (to some extend) be generalized to the case of multiple interposed bits and several layers of interposing[2]. In some extreme cases, we observed that the Logistic Regression algorithm is capable of recovering a significant proportion of the secret information of an

[1] To see this, recall that an Arbiter PUF f can be modeled as linear threshold functions $f(c) = \operatorname{sgn}\langle w, x\rangle + b$ (Theorem 8). LTFs are monotone in all input bits, but the above randomized challenge-response set (C_d, R) is likely not. Although the monotonicity argument gets weaker for products of k LTFs, randomized values are still likely to violate it.

[2] For a more rigorous treatment of feature and label noise in PUF modeling, we refer to Ganji et al. [GTS18].

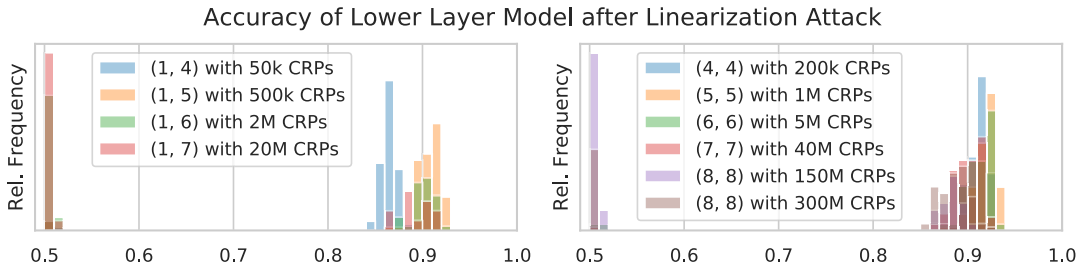


Figure 5.1.1.: Accuracy of the lower layer model $\hat{f}_{\mathrm{d}}$ after training using the CRP set (C_d, R) with randomly guessed interpose bits for $(1, k)$ and (k, k)-Interpose PUFs, as captured after execution of line 3 of Algorithm 5.2. Results shown are using the estimated best number of challenge-response pairs (see also Table 5.1); accuracy is relative to the PUF simulation's reliability. As with learning of XOR Arbiter PUFs, the probability to obtain a high-accuracy model of the lower layer depends on the size of the Interpose PUF and the training set size. (Our results are artificially capped at 95% to increase performance of the training algorithm.)

XOR Arbiter PUF even if half of all challenge bits in the training set were replaced with random bits. We hence recommend future PUF designs to be tested against this particular vulnerability by analyzing the correlation of the learned model with the simulation under test.

An important generalization of these findings is that a low accuracy of some training result (set) is not sufficient to even prove resilience against the LR machine learning algorithm. We extend and use these findings in Chapter 8.

5.1.2. Modeling of the Upper Layer

Algorithm 5.1 constructs a training set for the upper layer when given a model with decent accuracy for the lower layer. Intuitively, the algorithm first filters all challenges for the complete Interpose PUF where the response of the upper layer does not matter for the final response, as those challenge-response-pairs do not contain information about the upper layer. Second, for all remaining challenges, the model for the lower layer is evaluated on both possibilities, and the interpose bit producing the correct response is added to the training set of the upper layer. For all challenges where the model's prediction for the lower layer is correct, the heuristic will correctly determine the upper layer's response bit. We formally show the correctness, effectiveness and accuracy of this heuristic in the following.

Theorem 23. *Given an n-bit (k_u, k_d)-Interpose PUF f, a list of challenges C with corresponding responses R, and an ε-accuracy model $\hat{f}_d$ of the lower layer with $\varepsilon \geq 1/2$, Algorithm 5.1 will return a training set (C_H, R_H) for the upper layer with accuracy at least $2\varepsilon - 1$ and size expected to be at least $(\varepsilon - 1/2) \cdot |C|$.*

Algorithmus 5.1 Heuristic for creating upper-layer training sets

```
1: procedure HEURISTIC(C, R, f̂_d)
2:     initialize empty training set (C_H, R_H)
3:     for c, r in C, R do
4:         c^(+) ← (c_1, ..., c_{n/2}, +1, c_{n/2+1}, ..., c_n)
5:         c^(−) ← (c_1, ..., c_{n/2}, −1, c_{n/2+1}, ..., c_n)
6:         if f̂_d(c^(+)) = f̂_d(c^(−)) then
7:             continue
8:         end if
9:         if f̂_d(c^(+)) = r then
10:            add (c, 1) to (C_H, R_H)
11:        else
12:            add (c, −1) to (C_H, R_H)
13:        end if
14:    end for
15:    return (C_H, R_H)
16: end procedure
```

If $\hat{f}_d$ instead has accuracy ε on the half-negated lower layer, the training set is expected to have accuracy at most $2\varepsilon - 2$, i.e., it models the negation of the upper layer with accuracy at least $2\varepsilon - 1$; the expectation of the size remains the same.

Proof. For any given challenge $c \in \{-1,1\}^n$, let $c^{(+)}, c^{(-)} \in \{-1,1\}^{n+1}$ be defined as in Algorithm 5.1. We first give a lower bound for the probability that the learned model $\hat{f}_\mathrm{d}$ for the lower layer will predict both $c^{(+)}$ and $c^{(-)}$ correctly, based on a pigeon-hole-principle argument. Subsequently, we deduce the accuracy and expected size of the returned challenge-response set that is returned from HEURISTIC$(C, R, \hat{f}_\mathrm{d})$.

For any challenge $c \in \{-1,1\}^n$, we associate $c^{(+)}$ and $c^{(-)}$ with a pair $(c^{(+)}, c^{(-)})$. By construction, there are 2^n possible pairs containing all 2^{n+1} challenges of $n+1$ bits each. By prerequisite, we have that $\Pr_{c\in\{-1,1\}^{n+1}}[\hat{f}_\mathrm{d}(c) = f_\mathrm{d}(c)] = \varepsilon = 1/2 + \alpha$ with $0 \le \alpha \le 1/2$. That is, $\hat{f}_\mathrm{d}$ models at least $2^n + 2\alpha \cdot 2^n$ challenges correctly. Hence, by the pigeon hole principle, all correctly predicted challenges require at least $2\alpha \cdot 2^n$ pairs $(c^{(+)}, c^{(-)})$. That is,

$$\Pr_{c\in\{-1,1\}^n}\left[\hat{f}_\mathrm{d}(c^{(+)}) = f_\mathrm{d}(c^{(+)}) \text{ and } \hat{f}_\mathrm{d}(c^{(-)}) = f_\mathrm{d}(c^{(-)})\right] \ge 2\alpha = 2\varepsilon - 1.$$

I.e., if the model predicts $\hat{f}_\mathrm{d}(c^{(+)}) \neq \hat{f}_\mathrm{d}(c^{(-)})$, then with probability at least $2\varepsilon - 1$ we have $f_\mathrm{d}(c^{(+)}) \neq f_\mathrm{d}(c^{(-)})$ and let r' denote the unique bit that will produce the correct response, which is then indeed the correct response bit of the upper layer of the Interpose PUF. Hence, for each (c, r') added to the training set, the probability that the added example is correct is at least $2\varepsilon - 1$. If $\hat{f}_\mathrm{d}$ is instead an ε-accuracy model for the half-negated lower layer, then r' is the uniquely negated interpose bit that will give the correct response and the same argument applies.

Algorithmus 5.2 Divide-and-Conquer Interpose PUF Attack

1: **procedure** ATTACK($n, k_\mathrm{u}, k_\mathrm{d}, C, R$)
2: $C_d \leftarrow$ INTERPOSE(C, random bits) ▷ Guess training set for lower layer
3: $\hat{f}_\mathrm{d} \leftarrow \mathrm{LR}_{n+1}^{k_\mathrm{d}}(C_d, R)$ ▷ Train model for lower layer
4: **while** test accuracy below target **do**
5: $C_u, R_u \leftarrow$ HEURISTIC($C, R, \hat{f}_\mathrm{d}$) ▷ Create training set for upper layer
6: $\hat{f}_u \leftarrow \mathrm{LR}_n^{k_\mathrm{u}}(C_u, R_u)$ ▷ (Re-)train upper layer
7: $C_d \leftarrow$ INTERPOSE($C, \hat{f}_u(C)$) ▷ Create training set for lower layer
8: $\hat{f}_\mathrm{d} \leftarrow \mathrm{LR}_n^{k_\mathrm{d}}(C_d, R)$ ▷ Re-train lower layer
9: **end while**
10: **return** $\hat{f} : c \mapsto \hat{f}_\mathrm{d}(c_1, \ldots, c_{n/2}, \hat{f}_u(c), c_{n/2+1}, \ldots, c_n)$ ▷ Final Interpose PUF model
11: **end procedure**

As f_d is an XOR Arbiter PUF, we expect $\Pr[f_\mathrm{d}(c^{(+)}) \neq f_\mathrm{d}(c^{(-)})]$ to be $1/2$ on average (with little variance). Our model will thus predict this situation correctly on at least a $2\varepsilon - 1$ fraction of the cases, hence we expect the total number of challenge-response pairs returned by Algorithm 5.1 to be at least $(\varepsilon - 1/2) \cdot |C|$. □

5.1.3. Divide-and-Conquer Attack

The initial modeling of the lower layer and the heuristic to create a training set for the upper layer enable us to train a model for the upper layer and thereby launch a divide-and-conquer attack on the complete Interpose PUF. In this attack, we are able to model the upper and lower layer separately from each other. As can be seen from Figure 5.1.1 and Theorem 23, an initial accuracy of around $\varepsilon = 90\%$ and an application of the above heuristic will result in a training set for the upper layer of around $2\varepsilon - 1 = 80\%$. Note that the centering of the initial accuracy of the lower layer model at around 90% (as seen in Figure 5.1.1) is an artifact of our termination criterion of the used implementation of the Logistic Regression Attack. It is also possible to increase the initial accuracy close to 100% and conduct the attack with just training a single model for the lower layer, heuristically creating then a training set for the upper layer, and hereafter training a model for the upper layer. However, to heuristically increase performance, we use an iterative approach. We simply terminate each run of the Logistic Regression Attack earlier and repeat the process of training and re-training the upper and lower layer, until a high accuracy is achieved. In this process, while the initial training set for the lower layer was created using randomly guessed interpose bits, all following training phases of the lower layer use the upper layer model to predict interpose bits (cf. lines 2 and 7 in Algorithm 5.2).

For a $(k_\mathrm{up}, k_\mathrm{down})$-Interpose PUF with challenges of n-bit length, we conclude that launching the divide-and-conquer attack roughly requires the same computational effort as training a model for a $\max\{k_\mathrm{up}, k_\mathrm{down}\}$-XOR Arbiter PUF, although several iterations

of the attack[3] are required. This provides an informal reduction of the security of the Interpose PUF to the security of the XOR Arbiter PUF in the known message attacker model. This reduction is supported by both theoretical considerations and empirical results as presented in Section 5.2.

One caveat of our reduction lies in the nature of the heuristic in Algorithm 5.1: the training set for the upper layer is at most half the size of all challenges available to the attacker. While for $k_{\text{down}} > k_{\text{up}}$, this does not pose any challenge to the attacker, but for designs with $k_{\text{up}} \geq k_{\text{down}}$, this effectively forces the attacker to collect twice as many challenge-response pairs, compared to attack an XOR Arbiter PUF. On the other hand, relying on strict lower bounds for the number of challenge-response pairs is anyhow problematic, as Tobisch and Becker [TB15] have shown.

As noted in Section 5.1.1, the lower layer can randomly be trained in a half-negated fashion, which will result in a training set with very *low* accuracy for the upper layer of around 10%. This in turn will result in the training of a model for the upper layer that will predict the *negated* response of the actual PUF, and both effects will cancel out. Therefore, the total accuracy of the trained model will not be affected by the random choice of half-negated or non-negated model for the lower layer, and indeed the attacker has no way of knowing which option correctly reflects the physics of the Interpose PUF and which one only mathematically matches the challenge-response behavior.

5.2. Results and Performance Analysis

An overview of the attacks can be found in Table 5.1.

We studied n-bit challenge $(k_{\text{up}}, k_{\text{down}})$-Interpose PUFs for sizes $(1, k)$ and (k, k) for $k \leq 8$ and analyzed how the time to first success changes for different choices of security parameters n and k as well as training set size N. For performance reasons, choices different from $n = 64$ were only studied for the relatively small parameters of $k_{\text{up}}, k_{\text{down}} \leq 4$. Training set sizes were guessed using the figures of Tobisch and Becker [TB15] and optimized empirically. For results presented here, the choice of training set size which empirically resulted in lowest time to first success (cf. Chapter 4) was chosen. As all training times refer to wall-clock time, attack times across different CPUs are not comparable. We conducted all modeling attacks for Interpose PUFs with varying reliability between 70% and 100%.

In Figure 5.2.1, we summarize the required time until first success for smaller Interpose PUF sizes and different choices for the used challenge length. It can be seen that the required time increases approximately polynomial with the number of used challenge bits, which is in line with results reported both in the practical and theoretical realm of XOR Arbiter PUF attacks [Rüh+13b; GTS15].

For different choices of the number of employed arbiter chains k_{up} and k_{down}, we observed an exponential increase in the number of required challenge-response pairs and required attack time until first success, as shown in Figure 5.2.2. Note that shown training

[3] Note that for both PUFs, the re-training performance is much higher than the initial training performance.

set sizes produced the best result among several guessed choices, but *do not* constitute strict lower bounds. Careful optimization may lead to fewer required challenge-response pairs or shorter time to first success.

In all of our experiments we observed that lower reliability of the Interpose PUF does not have a big impact on the required training time.

For the choice of training set size and smaller choices of $k_{\text{up}}, k_{\text{down}}$ we observed a saturation threshold, beyond which adding more challenge-response pairs to the training set would increase training time instead of decreasing it. This may very well be related to implementation details of the Logistic Regression learner, including the fact that we did not use mini batches. For Interpose PUF sizes larger than $(6, 6)$, we were not able to confirm or refute this observation due to limitations in computational power.

While the attack as given in Algorithm 5.2 is using an infinite loop, practical experiments were limited to at most five iterations, after which the learning attempt was given up. For Interpose PUF sizes larger than $(7, 7)$, we empirically observed that this is barely of any use, and limited the number of iterations to two.

Finally, the memory footprint of the attacks is manageable and proportionate to the training set size. The storage of 100 million CRPs requires about 6GB of memory; our attack needs a peak memory of about two times the training set size. This implies that all attacks requiring 100 million CRPs or less can be carried out on an up-to-date laptop. Attacks on larger instances require up to 300 million CRPs and 750 million CRPs and thus allocate a total of about 36GB and 90GB of memory, respectively.

Details on memory consumption of our attack implementation can be found in Table 5.1. Also note that memory consumption depends on many implementation details. Our implementation currently does not swap out memory and, as a time-memory trade-off, uses 1 byte to store 1 challenge bit. We naively store both the upper and lower layer training set separately, which results in storage of heavily redundant data.

5.3. Neural Network Splitting Attack

Instead of basing the Splitting Attack on the Logistic Regression Attack, Neural Network Attacks (Section 3.6) can be used as a drop-in replacement for the modeling of upper and lower layer of the Interpose PUF under attack.

We found that, similar to the results we obtained on XOR Arbiter PUFs, the data complexity of the Splitting Attack can be significantly reduced. We were able to attack a 64-bit $(1, 7)$-Interpose PUF with 6 million CRPs in minutes, compared to 20 million CPRs and 20 hours required by the original implementation (Table 5.1).

We extrapolating our attack results (Table 3.4) and conclude that the Neural Network Attack is able to attack 64-bit (1,11) Interpose PUFs using 325M CRPs instead of the 750M required to attack a (1,9) Interpose PUF when using the Logistic Regression Attack, and 650M CRPs to attack a 64-bit (11,11)-Interpose PUF. While this has two orders of magnitude larger data complexity than the chosen message attacks by Tobisch, Aghaie, and Becker [TAB21], the Neural Network Attack provides better convergence rate, faster computation time, and uses the weaker known message attacker model.

$(k_{\text{up}}, k_{\text{down}})$	CRPs	rel.	Mem. (GB)	Time (# Threads)	Success Rate	Samples
(1, 5)	500k	0.8	<1	10.36min (1/⋆)	1.00	100
(1, 5)	500k	0.9	<1	8.70min (1/⋆)	1.00	100
(1, 5)	500k	1.0	<1	9.14min (1/⋆)	1.00	100
(1, 6)	2M	0.8	<1	1.62h (1/⋆)	1.00	57
(1, 6)	2M	1.0	<1	1.48h (1/⋆)	1.00	70
(1, 6)	5M	0.9	<1	1.42h (1/⋆)	1.00	55
(1, 7)	20M	0.8	2.5	17.54h (1/•)	0.97	39
(1, 7)	20M	0.9	2.5	16.17h (1/•)	1.00	33
(1, 7)	20M	1.0	2.5	20.07h (1/•)	1.00	31
(1, 9)	750M	1.0	91	approx. 8w (8/⋆)	0.26	23
(5, 5)	600k	0.8	<1	16.95min (1/⋆)	0.85	195
(5, 5)	600k	0.9	<1	16.13min (1/⋆)	0.88	191
(5, 5)	1M	1.0	<1	14.59min (1/⋆)	0.98	93
(6, 6)	5M	0.7	<1	3.79h (1/⋆)	0.63	54
(6, 6)	5M	0.8	<1	2.86h (1/⋆)	0.78	58
(6, 6)	5M	0.9	<1	2.62h (1/⋆)	0.83	58
(6, 6)	5M	1.0	<1	2.50h (1/⋆)	0.75	53
(7, 7)	40M	0.7	4.9	1.73d (10/•)	0.40	100
(7, 7)	40M	0.8	4.9	1.11d (10/•)	0.62	100
(7, 7)	40M	0.9	4.9	23.38h (10/•)	0.68	100
(7, 7)	40M	1.0	4.9	17.21h (10/•)	0.74	100
(8, 8)	150M	0.7	17.9	∞ (10/•)	0.00	43
(8, 8)	150M	0.8	17.9	2.07w (10/•)	0.25	48
(8, 8)	150M	0.9	17.9	1.59w (10/•)	0.33	55
(8, 8)	150M	1.0	17.9	1.54w (10/•)	0.35	49
(8, 8)	300M	0.7	35.8	18.96w (8/⋆)	0.04	26
(8, 8)	300M	0.8	35.8	2.73w (8/⋆)	0.30	10
(8, 8)	300M	0.9	35.8	1.64w (8/⋆)	0.42	26
(8, 8)	300M	1.0	35.8	2.53w (8/⋆)	0.28	99

Table 5.1.: Overview of the performance of the Splitting Attack on 64-bit $(k_{\text{up}}, k_{\text{down}})$-Interpose PUFs. For each size and reliability, the best-performing training set size is shown, defined as the setting that gave the shortest time to first success with our software; success is defined as final prediction accuracy above 95%. We used two different Intel® Xeon® CPU types, namely Gold 6130 at 2.1GHz (⋆) and E5-2630 v4 at 2.2GHz (•). Additionally to the number of CRPs shown in the table, the attacker was provided with a test set containing an additional 10^4 challenge-response pairs.

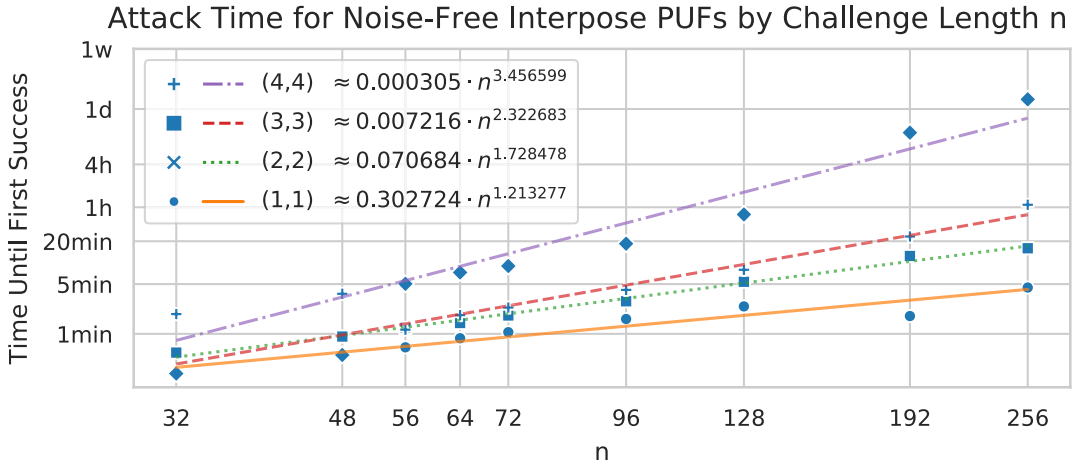


Figure 5.2.1.: Attack run time for different challenge lengths; times shown refer to time until first success in single-threaded runs. Every data point shows the best obtained time until first success for various choices of guessed amounts of challenge-response pairs. Interpolations given were computed using regression for a, b on $a \cdot n^b$.

5.4. Variants of the Interpose PUF

Given the successful attacks on the original Interpose PUF design, we ask if the design can be augmented to achieve better security. To facilitate a swift design process, and in contrast to the tailor-made attack presented in Section 5.1 and Section 5.2, this section uses a neural network modeling approach similar to the ones shown in Section 3.6.

Excluded from discussion in this section are intermediate calculations on interpose bits (we consider these in Chapter 8), as they may increase hardware attack surface, as well as interpose positions different from $n/2$. For different choice of interpose positions, we refer to the discussion of Nguyen et al. [Ngu+19], who conclude that interposing in the middle position has strongest security properties.

5.4.1. Design Details and Motivation

We study extensions of the Interpose PUF design in somewhat natural way by iterating the idea of interposition. Subsequently, weaknesses of this iteration are discussed and mitigated using novel ideas. Along this path, the five variants Domino Interpose PUF, XOR Interpose PUF, XOR Domino Interpose PUF, Tree Interpose PUF, and XOR Cascaded Interpose PUF are derived. While the first two variants are relatively straightforward extensions of the Interpose PUF, the latter three are more complex and are therefore explicitly depicted in Figure 5.4.1.

The Domino Interpose PUF reiterates the Interpose PUF's design such that the number of layers, i.e. the number of sequential interpositions, is increased. Thus, instead of two

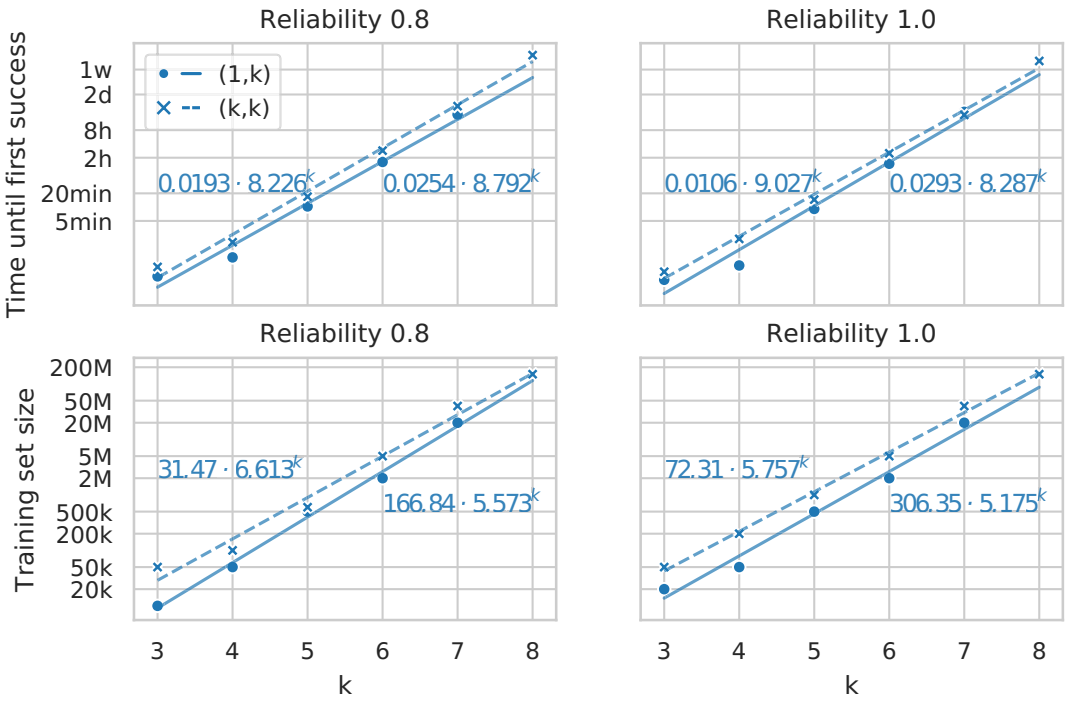


Figure 5.2.2.: Attack run time and best-performing number of CRPs for Interpose PUFs of different reliability and varying number of employed arbiter chains. Interpolations given were computed using regression for a, b on $a \cdot b^k$.

layers it consists of three layers, where the first one's output is interposed into the second one, whose output is in turn interposed into the third layer. Note that the Divide-and-Conquer attack could be extended to be applied to this variant. We refer to the number of arbiter chains employed in the upper, middle, and lower layer by $k_{\text{up}}, k_{\text{middle}}$, and k_{down}, respectively.

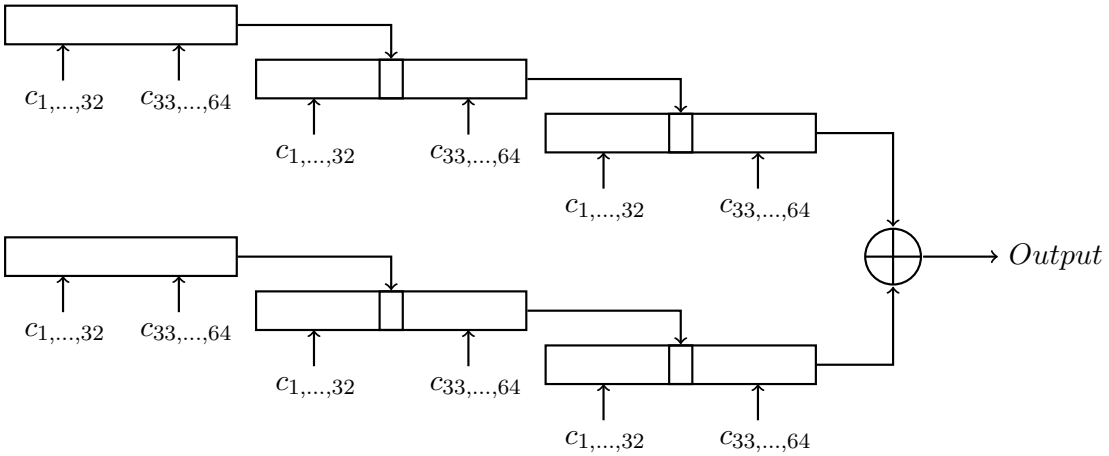
An important observation is that in this iterated design, the influence of every additional layer on the final response halves. Consequently, an attacker having the exact knowledge of the lower m layers achieves an expected prediction accuracy of at least $1 - \frac{1}{2^{(1+m)}}$ by guessing the interpose bit on the highest known layer. This accuracy will be achieved independently of the total number of layers. Hence, when maintaining an interpose position at $n/2$, layer numbers greater than three increase the design's security only marginally and can thus be disregarded.

The other straightforward extension, the XOR Interpose PUF, modifies original Interpose PUF with $k_{\text{up}} = k_{\text{down}}$ such that the Arbiter PUF chains' outputs in the upper layer are interposed separately, each into one corresponding chain in the lower layer, instead of being XORed and then interposed into every lower layer's chain. This modification is an effective mitigation against our Splitting Attack. (An alternative perspective on this is to consider this Interpose PUF variant as the XOR of k separate $(1, 1)$-Interpose PUF; hence the name XOR Interpose PUF.)

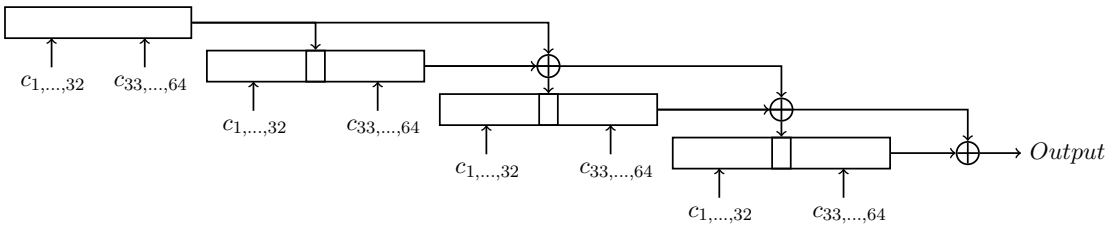
To increase the depth of the XOR Interpose PUF we contrived the XOR Domino Interpose PUF (see Figure 5.4.1a). It combines both of the previously described designs. Thus, it is the XOR of k separate $(1, 1, 1)$-Domino Interpose PUF`s`, where $k = k_{\text{up}} = k_{\text{middle}} = k_{\text{down}}$.

Nevertheless, the number of layers is still subject to the limitation due to sharply dropping influence on the response bit. This motivated the design of the Tree Interpose PUF shown in Figure 5.4.1c. As upper layers influence exponentially more leaves, the drop in influence is compensated. Compared to XOR Interpose PUF of same number of layers, it reduces the number of employed arbiter chains while being immune to the Splitting Attack. Being a binary tree, this design consists of $2^{d+1} - 1$ (XOR) Arbiter PUFs, where d is the depth of the tree, i.e. the distance between the first and the last layer. Except for the PUFs in the last layer, the output of every PUF is interposed to two corresponding PUFs in the subsequent layer. The output bits of the last layer are combined via XOR into the final response. All nodes in the tree are k-XOR Arbiter PUFs.

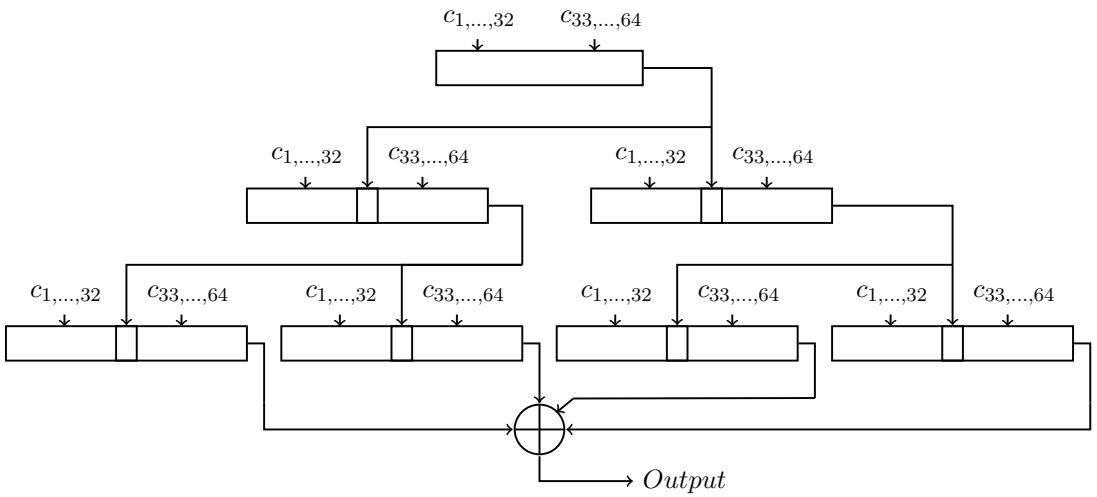
A yet different design that combines both the concept of interposition and the idea of more layers, while maintaining high influence of all building blocks, is the XOR Cascaded Interpose PUF (see Figure 5.4.1b). It is an iterated Interpose PUF design that addresses the above-mentioned problem of the influence loss by reusing each interpose bit: Each layer's output is used for interposition into the subsequent layer as well as for the final response which constitutes the parity of every layer's output bit (including the lowest layer's) and so is influenced by all layers equally. We refer to the number of layers by *length* l; each layer is comprised of a k-XOR Arbiter PUF.



(a) XOR Domino Interpose PUF: An iteration of the Interpose PUF design, but instead of having the same interpose bits everywhere, we evaluate interpose chains separately and return the parity of all responses. Each layer shown consists of an k-XOR Arbiter PUF.



(b) XOR Cascaded Interpose PUF: A variant of the Interpose PUF design, where intermediate results have full influence on the response. We refer to the number of layer by length l; each layer consists of a k-XOR Arbiter PUF.



(c) Tree Interpose PUF: A binary-tree of depth d, where each node is a k-XOR Arbiter PUF. Intermediate nodes receive one interpose bit from the layer above and insert their result into two "child" nodes in the layer below. The responses of the leafs are XORed into the final output bit.

Figure 5.4.1.: Variants of the Interpose PUF design.

5.4.2. Empirical Results of Deep Learning Modeling Attacks

We empirically tested the above introduced five Interpose PUF design derivatives to gain insight in their machine-learning resistance. The designs were parameterized using 64-bit challenge-length and a number of arbiter chains in between 9 and 16, corresponding to an (1,8)-Interpose PUF, and an (8,8)-Interpose PUF, respectively. As attack strategy, following Santikellur, Bhattacharyay, and Chakraborty [SBC19], a neural network modeling algorithm was chosen, as it requires little customization and no precise model of the concept class to be learned (Section 3.6).

The results in Table 5.2 show that none of the discussed designs showed increased machine-learning resistance, even when using a generic attack not specialized for the PUF under attack. While some designs may possess practical parameter choices for which modeling is hard, all designs studied in this section were easier to model than the original Interpose PUF of comparable size.

Our results show that the machine-learning hardness of the Interpose PUF at least cannot easily be increased by all too naive augmentation to the original design. Furthermore, the modeling based on neural networks has proven to be an easy-to-use and powerful tool for preliminary analysis of Strong PUF designs. We hence recommend the Neural Network Attack (Section 3.6) to be part of every Strong PUF security analysis.

PUF Variant	Size	Arb. Chains	CRPs	Rel.	Mem. (GB)	Time	Succ. Rate
Domino	$k = (3, 3, 3)$	9	2M	0.8	2.5	55.6min	1.00
Domino	$k = (4, 4, 4)$	12	20M	0.8	12.7	1.0d	0.88
XOR Interpose	$k = 4$	8	10M	0.8	9.7	4.3h	1.00
XOR Interpose	$k = 5$	10	40M	0.8	37.8	2.8d	1.00
XOR Dom. Interp.	$k = 3$	9	2M	0.8	2.2	15.6min	1.00
XOR Dom.Interp.	$k = 4$	12	40M	0.8	37.8	2.0d	1.00
Tree Interpose	$d = 2, k = 2$	14	5M	0.8	10.4	10.7h	1.00
Tree Interpose	$d = 3, k = 1$	15	5M	0.8	10.0	8.8h	1.00
XOR Casc. Interp.	$l = 2, k = 4$	8	5M	0.8	7.3	16.2h	1.00
XOR Casc. Interp.	$l = 3, k = 3$	9	10M	0.8	10.4	8.4h	1.00
XOR Casc. Interp.	$l = 2, k = 5$	10	20M	0.8	46.0	2.1d†	0.99
XOR Casc. Interp.	$l = 5, k = 2$	10	10M	0.9	9.7	2.7h	1.00

Table 5.2.: Overview over Deep-Learning-Attacks on derivatives of the Interpose PUF design. Each derivative has been tested for a selection of different parameters with the challenge-length fixed to 64 bit. For each type and parameter, 100 simulations and attacks were conducted (experiments marked with † have at least 75 samples). Success was defined as prediction accuracy above 90%, relative to the PUF's reliability. Note that attack times are all below comparable times for modeling the original Interpose PUF of similar sizes, even though a generic, non-specialized attack methodology was used.

6. Feed-Forward Arbiter PUF

6.1. Design

Feed-Forward Arbiter PUFs were first introduced by Gassend et al. [Gas+04] and Lee et al. [Lee+04] and are based on the idea of introducing non-linearity to the response behavior (as discussed in Section 3.1) by adding more arbiter elements that pick up the signal on the delay lines before they reach the last stage. These arbiter elements produce additional challenge bits which will be used in later stages. As such, the Feed-Forward Arbiter PUF can be thought of as a predecessor of the Interpose PUF of Chapter 5, using the same Arbiter PUF instead of an additional layer of PUFs to produce additional, attacker-unknown, challenge bits. We will refer to the additional arbiter elements and challenge bits as feed-forward *loops*. An extension of the Feed-Forward Arbiter PUF is the (homogeneous) XOR Feed-Forward Arbiter PUF [AZP20], where the result bit is – similar to the XOR Arbiter PUF – determined by a number of k individual Feed-Forward Arbiter PUFs with identical loop placements.

6.2. Evolution Strategies Attacks

The Feed-Forward Arbiter PUF shows much stronger modeling attack resistance than Arbiter PUFs of comparable size. Attacks in the literature are known message attacks. Rührmair et al. [Rüh+10] attack Feed-Forward Arbiter PUF whose loops are arranged in regular patterns. Kumar and Burleson [KB15] demonstrated attacks on silicon data of Feed-Forward Arbiter PUFs with up to 8 loops, using an attack based on evolution strategies. They report data complexity much lower than our attack allows, but did not report results on XOR Feed-Forward Arbiter PUFs.

6.3. Neural Network Attack

Prior to our work, Alkatheiri and Zhuang [AZ17] use a neural network approach for learning, however their attack shows declining accuracy for an increase in the number of loops. Furthermore, the attack also requires the loop pattern to be known to the attacker, a condition that will not easily hold since different Feed-Forward Arbiter PUFs can have different loop patterns [AZ17, personal communication]. While this deficiency could be alleviated by either an (computationally expensive) brute-force search or via a physical attack on the circuit, they do not report attack results on XOR Feed-Forward Arbiter PUFs.

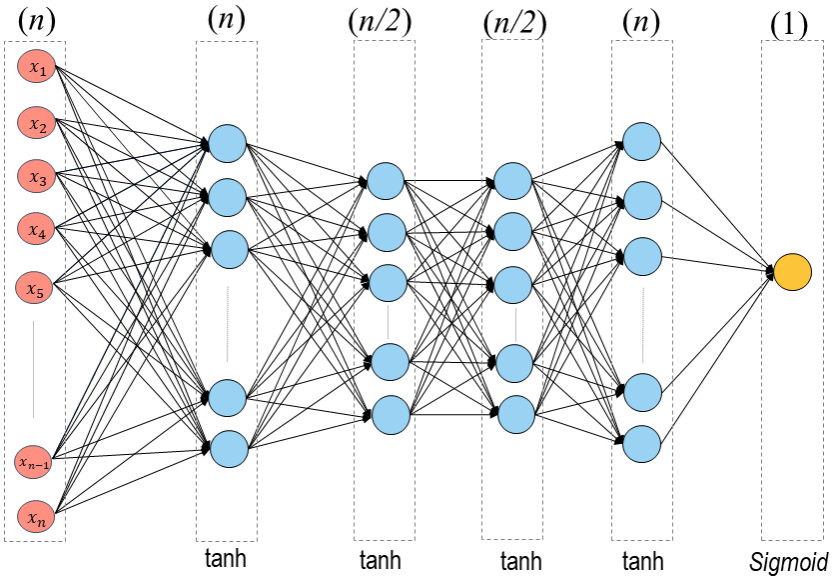


Figure 6.3.1.: The feed-forward neural network architecture for attacking n-bit Feed-Forward Arbiter PUFs and Feed-Forward XOR Arbiter PUFs. There are $n, n/2, n/2, n$ neurons used per layer for based; the hidden layer uses the tanh activation function.

We attack n-bit Feed-Forward Arbiter PUFs and XOR Feed-Forward Arbiter PUFs with a up to 10 loops. To run the attack, we use a Multilayer Perceptron (MLP) of fitted size. In contrast to the neural network network used to attack the XOR Arbiter PUF in Section 3.6, this network is composed of four hidden layers with $n, n/2, n/2, n$ neurons, respectively. The network shape and hyperparameters are chosen independently of the number of loops and their positioning, hence no knowledge of this information is required to run the attack. As argued for the XOR Arbiter PUF (see Section 3.6), we chose the tanh activation function over the common choice of ReLU. The attack network is displayed in Figure 6.3.1. We target both simulated PUFs, with the simulation based of an appropriate adaptation of the additive delay model (Theorem 8), and PUFs implemented on FPGAs.

We implemented 64-stage Feed-Forward Arbiter PUFs on three Artix-7 FPGAs using the Xilinx Vivado design suite that consists of an editable MicroBlaze CPU. VHSIC Hardware Description Language (VHDL) was used to build the Feed-Forward Arbiter PUFs designs. The placement of each Feed-Forward Arbiter PUF on the chip was carried out horizontally on the chips using the Tool Command Language (TCL). AXI Universal Asynchronous Receiver Transmitter (UART), with baud rate of 230K bits/second, was used to speed up the CRPs transformation between the Tera Term terminal and the FPGAs. The Xilinx SDK was utilized to program the input/output workflow of the CRPs generation from the chips. The implementation was done on three FPGA chips. We generated five million CRPs out of each implemented silicon PUF. The CRPs were

n	no. of loops	CRPs	success rate	duration	accuracy	memory	Alkatheiri et. al.[AZ17] CRPs	duration	acc.
64	4	135k	10/10	6 min	95%	<1 GiB	200K	1.5 min	89%
64	5	180k	10/10	8 min	93%	<1 GiB	200K	1.9 min	87%
64	6	315k	10/10	10 min	93%	<1 GiB	200K	6.2 min	87%
64	7	405k	9/10	16 min	92%	<1 GiB	-	-	-
64	8	540k	10/10	21 min	92%	1 GiB	-	-	-
64	9	585k	9/10	22 min	92%	1 GiB	-	-	-
64	10	630k	10/10	26 min	93%	1 GiB	-	-	-
128	1	36k	10/10	2 min	96%	<1 GiB	20K	<1 min	95%
128	2	72k	10/10	3 min	93%	<1 GiB	80K	<1 min	96%
128	3	180k	10/10	8 min	93%	<1 GiB	100K	<1 min	92%
128	4	225k	10/10	12 min	92%	<1 GiB	200K	<1 min	95%
128	5	405k	10/10	17 min	93%	1.6 GiB	200K	1.4 min	87%
128	6	540k	10/10	22 min	92%	1.9 GiB	200K	5.4 min	84%
128	7	900k	10/10	35 min	92%	3 GiB	-	-	-
128	8	900k	10/10	34 min	92%	3.1 GiB	-	-	-
128	9	1.2M	10/10	45 min	91%	4 GiB	-	-	-

Table 6.1.: Results of attacking 64-stage and 128-stage Simulated FF PUFs using our Keras based implementation of the multilayer perception compared with the multilayer perception attack by Alkatheiri et. al. [AZ17]. Alkatheiri et. al.'s method assumes the loop pattern is known to the attacker, the proposed method has no such assumption.

no. of loops	k	CRPs	success rate	duration	avg. success accuracy	memory
1	2	120K	10/10	0.5 min	98%	<1 GiB
	4	540K	10/10	2.8 min	98%	2.6 GiB
	6	900K	10/10	7 min	98%	4.3 GiB
	8	6M	7/10	6 hrs	96%	10 GiB
2	2	180K	10/10	5 min	97%	1 GiB
	4	720K	10/10	32 min	98%	3.5 GiB
	6	2.7M	9/10	1.4 hrs	97%	9.8 GiB
	8	9.5M	9/10	22 hrs	96%	18.8 GiB
3	2	360K	10/10	8 min	97%	1.8 GiB
	4	900K	10/10	2.3 hrs	96%	3.3 GiB
	6	3.15M	10/10	9.2 hrs	94%	8.1 GiB
	8	13.5M	6/10	40 hrs	91%	21 GiB
4	2	900K	10/10	1.4 hrs	97%	3.7 GiB
	4	2.7M	9/10	8.3 hrs	96%	6.9 GiB
	6	9M	6/10	26 hrs	94%	13 GiB
	8	18M	4/10	46 hrs	92%	27 GiB
5	2	1.8M	10/10	4.9 hrs	95%	6.4 GiB
	4	7.2M	7/10	19 hrs	93%	14 GiB
	6	11.7M	5/10	32 hrs	91%	22 GiB
	8	18M	4/10	24 hrs	90%	30 GiB

Table 6.2.: Results of attacking 64-stage simulated k-XOR Feed-Forward Arbiter PUFs.

generated at an ambient temperature of around 22°C, and core voltage set to 1.0V using the built-in chips resistor.

Our results show that our MLP-based method is able to model 64-stage Feed-Forward Arbiter PUFs with 10 loops and 64-stage 8-XOR Feed-Forward Arbiter PUFs when using 5 homogeneous loops per Arbiter PUF. Note that the challenge length of the PUF is reduced by the number of loops inserted, however our results also indicate no fundamental change in attack performance even when the challenge length is doubled to approx. 128 bit. The detailed results on simulated data is shown in Table 6.1 for Feed-Forward Arbiter PUFs and in Table 6.2 for XOR Feed-Forward Arbiter PUFs.

Our experiments with real-world data largely confirm the attacks on 64-bit Feed-Forward Arbiter PUFs, independently of the choice of loop pattern, with the attack on 10 loops requiring a lightly larger amount of 770,000 CPRs. As the XOR operation of the XOR Feed-Forward Arbiter PUF is done in Boolean logic and noise-free, we expect that our results in simulation also transfer to real-world data on XOR Feed-Forward Arbiter PUFs. A table showing the results is omitted for brevity.

A further extension of the k-XOR Feed-Forward Arbiter PUF can be made by introducing *heterogeneous* loops, i.e. by using individual loop placements on the k involved Feed-Forward Arbiter PUFs. We report that the network of our MLP attack as shown in Figure 6.3.1 is unable to attack heterogeneous XOR Feed-Forward Arbiter PUFs even for moderate parameter settings involving just one loop per Arbiter PUF and $k = 3$. Modeling accuracy was saturated at around 60%. We hence believe that the neural network structure will need major modifications to allow successful training for this extension and encourage further research in this direction.

Our results demonstrate that the Neural Network Attack can model variants of the homogeneous Feed-Forward Arbiter PUFs in the known message attack model that previously were out of reach for modeling attacks. This underlines our argument that neural network attacks should be part of the security analysis of future PUF designs and shows that the family of homogeneous Feed-Forward Arbiter PUFs needs to be considered insecure under EUF-KMA.

7. Beli PUF

The previous chapters demonstrated that none of the Arbiter-PUF-based designs can be considered EUF-CMA secure. A further exploration of the Arbiter PUF design space is necessary to see if secure PUFs based on Arbiter PUFs can be build. The Beli PUF explores variations of the Arbiter PUF itself, as opposed to merely combining multiple Arbiter PUFs into larger designs.

7.1. Design

We consider *Multiple Permuted Delay Line PUFs* (MPDL PUFs), a generalization of the Arbiter PUF to a circuit that uses more than two delay lines. An MPDL PUF that has m delay lines can have $\lceil \log_2 m \rceil$ output bits $o_0, \ldots o_{\lceil \log_2 m \rceil - 1}$, indicating the index of the fastest delay line. The fastest signal is detected using a circuit similar to the arbiter element of the Arbiter PUF. Alternatively, an MPDL PUF can have exactly one output bit o_s which is defined as the XOR of above output bits, i.e. $o_s = \prod_{l=0}^{\lceil \log_2 m \rceil - 1} o_l$. In an MPDL PUF, the delay lines can be switched by 2-to-2 multiplexer elements controlled by challenge bits as well as by a fixed permutation that is given by the design specification.

We first restrict out attention to a specific instance of the MPDL PUF, the *Beli PUF*: Beli PUF uses four delay lines, i.e. two output bits o_0, o_1 and one output bit $o_s = o_1 o_2$, respectively. To achieve a symmetric structure on n input bits, we arrange Beli PUF into $n/2$ blocks of four delay lines, using two challenge bits each. For filling in the permutations $\mathcal{P}_1, \ldots, \mathcal{P}_{n/2}$, we opt to use $\mathcal{P} = (i_0, i_1, i_2, i_3) \mapsto (i_0, i_2, i_1, i_3)$. By exchanging the top two lines with the bottom two lines, this permutation guarantees that our design does not degenerate into a design reminiscent of two Arbiter PUFs. Using the same permutation in all places simplifies the analysis in the remainder of this chapter. The detailed response behavior of the Beli PUF is shown in Table 7.1, where we denote the delays on the four delay lines with $d_0, \ldots, d_3$. The schematics of Beli PUF are shown in Figure 7.1.1.

As we will see, none of the specifics of Beli PUF are essential to out security analysis and results can be applied to MPDL PUFs in general with little restriction.

For Beli PUF, and for MPDL PUFs in general, XOR-versions similar to XOR Arbiter PUF are conceivable, where the same design is used in parallel in multiple instances and the outputs are defined as the XOR of output bits across the individual instances. We denote such Beli PUFs with k individual instances 1-bit k-XOR Beli PUF and 2-bit k-XOR Beli PUF, respectively.

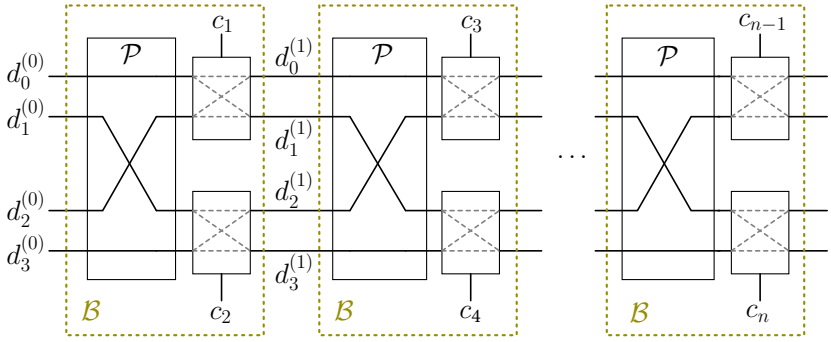


Figure 7.1.1.: Beli PUF structure

lowest signal delay	2-bit output		1-bit output	individual signals					
	o_0	o_1	$o_s = o_0 \cdot o_1$	$\theta_{0,1}$	$\theta_{0,2}$	$\theta_{0,3}$	$\theta_{1,2}$	$\theta_{1,3}$	$\theta_{2,3}$
d_0	1	1	1	-1	-1	-1	*	*	*
d_1	1	-1	-1	1	*	*	-1	-1	*
d_2	-1	1	-1	*	1	*	1	*	-1
d_3	-1	-1	1	*	*	1	*	1	1

Table 7.1.: Output of Beli PUF where $\theta_{i,j} = \operatorname{sgn}(d_i - d_j)$ and * denotes "any value", meaning that the output values in that row are independent of the value in the starred column.

7.2. Model Based on Additive Delay Model

For MPDL PUFs, models based on the additive delay model similar to the Arbiter PUF can be derived. We derive a model for the Beli PUF.

Let $d_l^{(j)}$ denote the delay after the j-th stage ($1 \leq j \leq n/2$) on line l. Let $c \in \{-1, 1\}^n$ be the challenge given to Beli PUF. In the j-th stage, we use c_{2j-1} for the top switch and c_{2j} for the bottom switch. We denote the internal delays of the switch box that receives challenge bit c_i with $d_i^{\mathrm{TT}}, d_i^{\mathrm{BT}}, d_i^{\mathrm{BB}}, d_i^{\mathrm{TB}}$ for the delay introduced by a signal traveling from top input to top output, bottom input to top output, bottom input to bottom output, and top input to bottom output, respectively.

As the top switch receives the input delays $d_0^{(j-1)}$ and $d_1^{(j-1)}$, the challenge bit c_{2j-1} and has the internal delays $d_{2j-1}^{\mathrm{TT}}, d_{2j-1}^{\mathrm{BT}}, d_{2j-1}^{\mathrm{BB}}, d_{2j-1}^{\mathrm{TB}}$, we obtain for the output delays of the top switch that

$$d_0^{(j)} = \begin{cases} d_0^{(j-1)} + d_{2j-1}^{\mathrm{TT}} & (c_{2j-1} = -1)\,, \\ d_1^{(j-1)} + d_{2j-1}^{\mathrm{BT}} & (c_{2j-1} = 1)\,, \end{cases} \quad d_1^{(j)} = \begin{cases} d_1^{(j-1)} + d_{2j-1}^{\mathrm{BB}} & (c_{2j-1} = -1)\,, \\ d_0^{(j-1)} + d_{2j-1}^{\mathrm{TB}} & (c_{2j-1} = 1)\,. \end{cases}$$

Similarly, for the bottom switch in the j-th stage we have

$$d_2^{(j)} = \begin{cases} d_2^{(j-1)} + d_{2j}^{\mathrm{TT}} & (c_{2j} = -1)\,, \\ d_3^{(j-1)} + d_{2j}^{\mathrm{BT}} & (c_{2j} = 1)\,, \end{cases} \quad d_3^{(j)} = \begin{cases} d_3^{(j-1)} + d_{2j}^{\mathrm{BB}} & (c_{2j} = -1)\,, \\ d_2^{(j-1)} + d_{2j}^{\mathrm{TB}} & (c_{2j} = 1)\,. \end{cases}$$

By using the fact that for $c_i \in \{-1, 1\}$, we have $c_i = -1 \iff 1/2 + 1/2 c_i = 0$ and $c_i = 1 \iff 1/2 - 1/2 c_i = 0$, we can write the output delays of the j-th stage, $d_0^{(j)}, d_1^{(j)}, d_2^{(j)}, d_3^{(j)}$ without case distinction as

$$\begin{aligned} d_0^{(j)} &= (1/2 - 1/2 c_i)\left(d_0^{(j-1)} + d_{2j-1}^{\mathrm{TT}}\right) + (1/2 + 1/2 c_i)\left(d_1^{(j-1)} + d_{2j-1}^{\mathrm{BT}}\right), \\ d_1^{(j)} &= (1/2 - 1/2 c_i)\left(d_1^{(j-1)} + d_{2j-1}^{\mathrm{BB}}\right) + (1/2 + 1/2 c_i)\left(d_0^{(j-1)} + d_{2j-1}^{\mathrm{TB}}\right), \\ d_2^{(j)} &= (1/2 - 1/2 c_i)\left(d_2^{(j-1)} + d_{2j}^{\mathrm{TT}}\right) + (1/2 + 1/2 c_i)\left(d_3^{(j-1)} + d_{2j}^{\mathrm{BT}}\right), \\ d_3^{(j)} &= (1/2 - 1/2 c_i)\left(d_3^{(j-1)} + d_{2j}^{\mathrm{BB}}\right) + (1/2 + 1/2 c_i)\left(d_2^{(j-1)} + d_{2j}^{\mathrm{TB}}\right). \end{aligned}$$

To obtain expressions for the delay in each line, we iteratively replace $d_i^{(j-1)}$ in $d_i^{(n)}$. However, in contrast to the similar procedure on the Arbiter PUF circuit, in the Beli PUF model this leads to expressions of exponential length in n for $d_i^{(n)} - d_j^{(n)}$. This behavior is caused by the introduction of the permutation $\mathcal{P}$. (If choosing $\mathcal{P}$ as the identity permutation, we obtain two Arbiter PUFs, and the Arbiter PUF additive delay model with a linear number of terms applies.)

To obtain a complete model of the Beli PUF from the expressions for d_0, d_1, d_2, d_3, we define the delay differences $\theta_{i,j} = \operatorname{sgn}(d_i - d_j)$ and observe that $\theta_{i,j} = 1$ if and only if $d_i > d_j$. The output bits o_0 and o_1 can be computed as a Boolean function of $\theta_{0,2}, \theta_{0,3}, \theta_{1,2}, \theta_{1,3}, \theta_{2,3}$ and $\theta_{0,1}, \theta_{0,3}, \theta_{1,2}, \theta_{1,3}, \theta_{2,3}$, respectively, based on the observations on $\theta_{i,j}$ given in Table 7.1. For output bit o_0, we find that d_2 has the lowest delay if and only if $\max\{-\theta_{0,2}, -\theta_{1,2}, \theta_{2,3}\} = -1$ ("2 faster than 0 and 2 faster than 1 and 2 faster than 3"). d_3 has the lowest delay if and only if $\max\{-\theta_{0,3}, -\theta_{1,3}, -\theta_{2,3}\} = -1$ ("3 faster than 0 and 3 faster than 1 and 3 faster than 2"). As o_0 is -1 if and only if either d_2 or d_3 has the lowest delay, we obtain

$$o_0 = \min\left\{\max\{-\theta_{0,2}, -\theta_{1,2}, \theta_{2,3}\}, \max\{-\theta_{0,3}, -\theta_{1,3}, -\theta_{2,3}\}\right\}.$$

In a similar way, we can derive

$$o_1 = \min\left\{\max\{-\theta_{0,1}, \theta_{1,2}, \theta_{1,3}\}, \max\{-\theta_{0,3}, -\theta_{1,3}, -\theta_{2,3}\}\right\}$$

as o_1 is -1 if and only if either d_1 or d_2 have the lowest delay. Finally, in the case of 1-bit Beli PUF, the output can be modeled as $o_s = o_0 \cdot o_1$.

We empirically evaluated the length of the model expressions using Sage for $n \leq 18$, generating expressions for Beli PUF's d_0, d_1, and $d_0 - d_1$. (By symmetry, these findings extend other delay lines in Beli PUF as well.) In Figure 7.2.1, a comparison of length of expressions with the Arbiter PUF model and the algebraic maximum length is given.

By the exponential size, it is infeasible for an attacker to recover all coefficients of this model and thus bars them from using the Beli PUF additive delay model for modeling attacks, which give rise to hope that Beli PUF could resist modeling attacks.

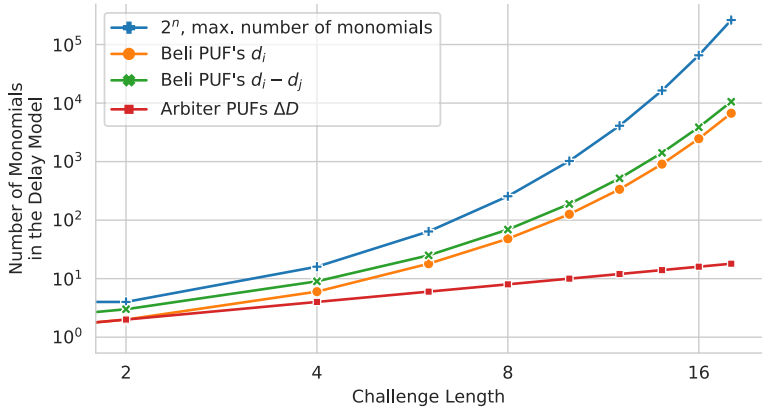


Figure 7.2.1.: Number of monomials for the Beli PUF's d_0, d_1 and their difference $d_0 - d_1$, as well as for the Arbiter PUF delay model, when written as multivariate polynomial of the challenge.

7.3. Implementation and Metrics

Beli PUF can be implemented on FPGA in a way similar to the Arbiter PUF. However, to obtain the final response bit(s) of Beli PUF, a variant of the arbiter element is needed. One option is to implement six arbiter elements, as motivated by the model equations for o_0 and o_1 given above, where each arbiter element determines the value of one of the involved $\theta_{i,j}$. Due to noise, it is possible that the determined $\theta_{i,j}$ values are contradicting, in which case the final response of Beli PUF also may be noisy.

We compare the bias and bit sensitivity of Beli PUF simulations with metrics obtained for the Arbiter PUF simulation.

As shown in Figure 7.3.1, the distribution of the bias of Beli PUF and Arbiter PUF is close to unbiased with only some variance. However, observe that the Arbiter PUF has a little smaller bias variance. For both designs, the bias can be further reduced by using an XOR-variant.

The bit sensitivity of Beli PUF is generally reduced and below the desirable value of $1/2$, which can be explained by the "minimum" operation on the four delay lines, as changed delay values in lines that do not involve the shortest delay do not change the PUF's output. However, the bit sensitivity shows less variance across the bit position on the challenge than it is the case for the Arbiter PUF. The distributions of bit sensitivities for different challenge bit positions is displayed in Figure 7.3.2.

To measure the reliability of Beli PUF, the Beli PUF FPGA-implementation with the lowest bias across our 100 test FPGA chips among several candidates has been chosen. The reliability has then been examined by generating 1 million uniform random challenges and querying each FPGA on this challenge set 11 times. The selected Beli PUF implementation enjoys a high reliability of 98% on average, with little variance.

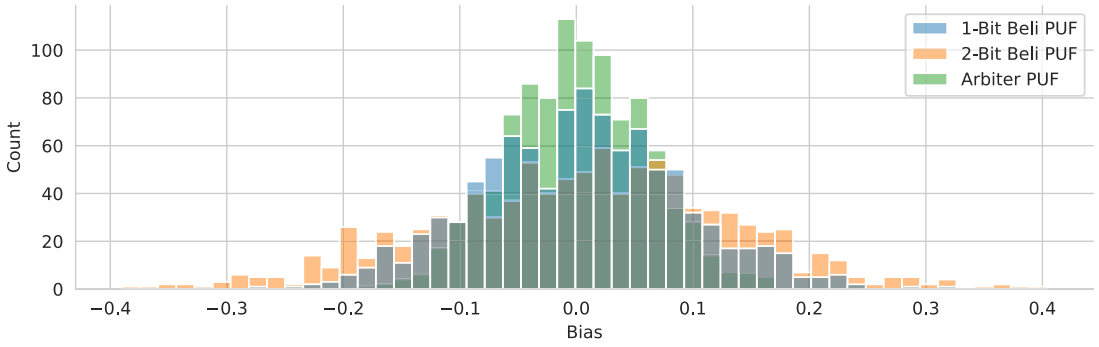


Figure 7.3.1.: Distribution of bias of delay-based PUF variants simulated for 1000 instances. Each experiment has been conducted on 1000 uniform random noise-free challenges.

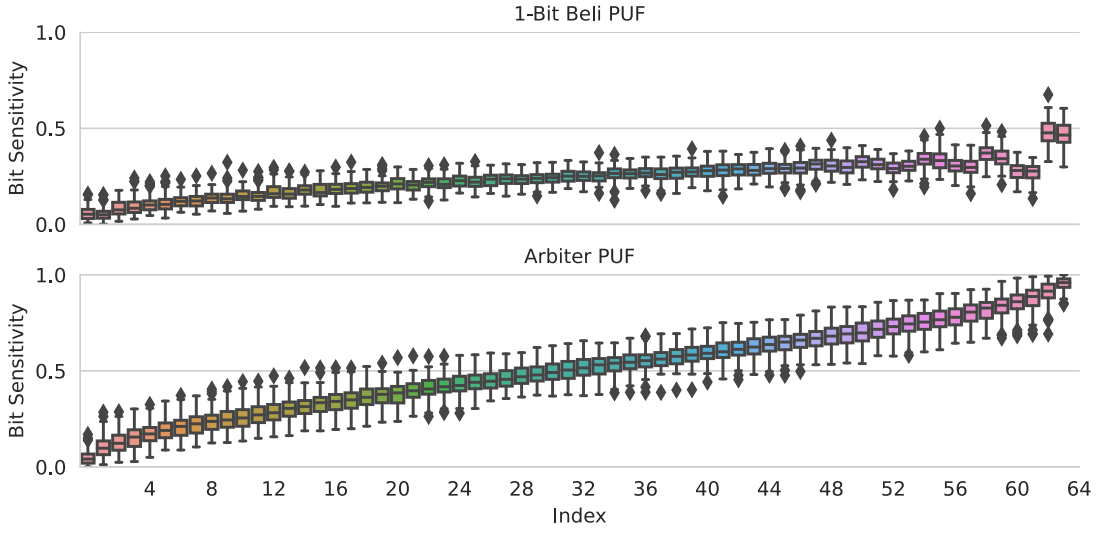


Figure 7.3.2.: Probability of flipping the response with respect to changing one challenge bit in delay-based PUF variants simulated for 100 instances. Each challenge index has been tested on 1000 uniform random noise-free challenges.

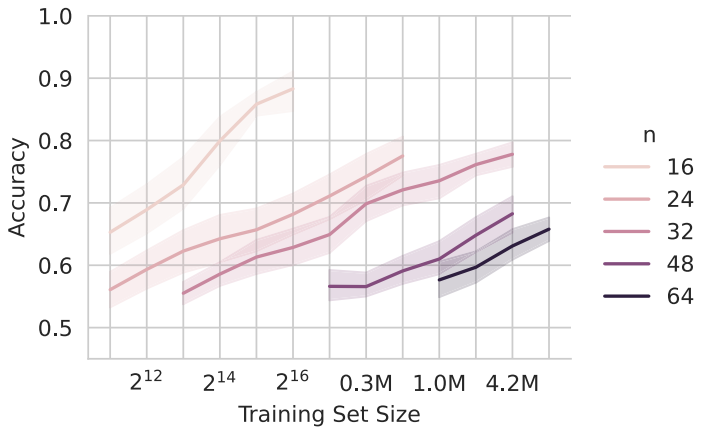


Figure 7.4.1.: Prediction accuracy of a general-purpose MLP modeling attack on 1-bit Beli PUF, simulated with various challenge sizes for 10 instances using noise-free CRP data sets of size N. The shaded area indicates a 95% confidence interval across attacked PUF instances.

7.4. Generic MLP Attack

One way to avoid the exponential number of parameters in the additive delay model of the Beli PUF is to use general-purpose neural networks for modeling. Such networks have been successfully used in the modeling attacks on XOR Arbiter PUFs and on the Interpose PUF [Wis+21a].

For attacking Beli PUF, we employed a Multilayer-Perceptron (MLP) Model which uses the ReLU activation function on its four hidden layers. Each hidden layer uses 256 neurons.

The resulting model accuracy indicates that, while Beli PUFs of smaller challenge lengths such as 16 bit can be modeled with high accuracy, the prediction accuracy of the modeling attack for 24 bit Beli PUFs stays below 80%, as shown in Figure 7.4.1. Increasing the training set size does not improve the result, which indicates that the used generic model is not suitable to model the Beli PUF accurately.

For comparison, similar neural network attacks on 4-XOR 64-bit Arbiter PUFs require merely 150,000 CRPs to achieve near-perfect prediction accuracy.

7.5. Specialized Neural Network Attack

In contrast to the generic attack, here we motivate a neural network attack based on a variant of the physically inspired Beli PUF model (Section 7.2). While we demonstrate the attack specifically for the Beli PUF design, the idea is applicable to all MPDL PUFs.

In any MPDL PUF, the paths the signals take through the circuit are fully defined by a given challenge. For a given path P_i, the total signal delay can be computed from the

physical parameters $d_j^{\mathrm{TT}}, d_j^{\mathrm{TB}}, d_j^{\mathrm{BT}}, d_j^{\mathrm{BB}}$ as

$$d_i = \sum_{(X,j)\in P} d_j^X.$$

This computation can be formalized as a dot product of a delay-indicator vector $x_i \in \{0,1\}^{4n}$, derived from the challenge c using the design specification of the circuit, as

$$d_i = \langle d, x_i \rangle ,$$

where d is a list of all $4n$ physical parameters, $d = \left(d_1^{\mathrm{TT}}, d_1^{\mathrm{TB}}, d_1^{\mathrm{BT}}, d_1^{\mathrm{BB}}, d_2^{\mathrm{TT}}, \ldots, d_n^{\mathrm{BB}}\right)$. This enables us to write all signal delays of the MPDL PUF as a function linear in the $4n$-dimensional space of the physical parameters. Using this observation, we can avoid the exponentially long model equations of Section 7.2 for the delay values.

We also observed that in the case of Beli PUF, the delay-indicator vectors x_i do not span across all $4n$ dimensions, but only use $2n+4$ dimensions. Hence, the dimensionality of the model could be reduce to $2n+4$ by using principal component analysis. However, to simplify notation, we stick to the analysis in $4n$ dimensions.

To model the Beli PUF response, instead of resorting to the relatively complex model equations of Section 7.2, we observe that o_0, o_1, and o_s can also be written as

$$\begin{aligned} o_0 &= \operatorname{sgn}\left(\min\{d_2, d_3\} - \min\{d_0, d_1\}\right), \\ o_1 &= \operatorname{sgn}\left(\min\{d_1, d_3\} - \min\{d_0, d_2\}\right), \quad \text{and} \\ o_s &= \operatorname{sgn}\left(\min\{d_1, d_2\} - \min\{d_0, d_3\}\right). \end{aligned}$$

For the modeling attack presented in this section, the usage of the full model equation is also possible. However, due to the nested nature of min, max, and sgn operations, it converges slower than the equations presented here.

These equations allow us to define a neural network that is able to model Beli PUF. Given the delay-indicator vectors derived from the challenges, this network can be trained to closely model Beli PUF responses.

Like done in attacks on the XOR Arbiter PUF, attacks on XOR Beli PUFs can be applied by adjusting the model to compute the product of the individual model output bits.

To confirm the validity of the derived Beli PUF model, we collected 100,000 CRPs from a 2-bit Beli PUF with 64 challenge bits and trained models as outlined above both for 2-bit Beli PUF and a 1-bit Beli PUF, using 99,000 CRPs. The resulting model showed correct prediction in roughly 88% of cases on a test set of 1,000 CRPs. This provides practical evidence that our model and implementation behave similarly.

Using Beli PUF simulations, we ran attacks on 1-bit and 2-bit Beli PUFs as well as their XORed variants. As a baseline for comparison, we run the state-of-the-art attacks on XOR Arbiter PUFs, implemented using the same software stack. All attacks were conducted for 32, 64, 128, and 256-bit challenge lengths. Our attack is implemented using a neural network and the Keras framework, but can – by the nature of the network

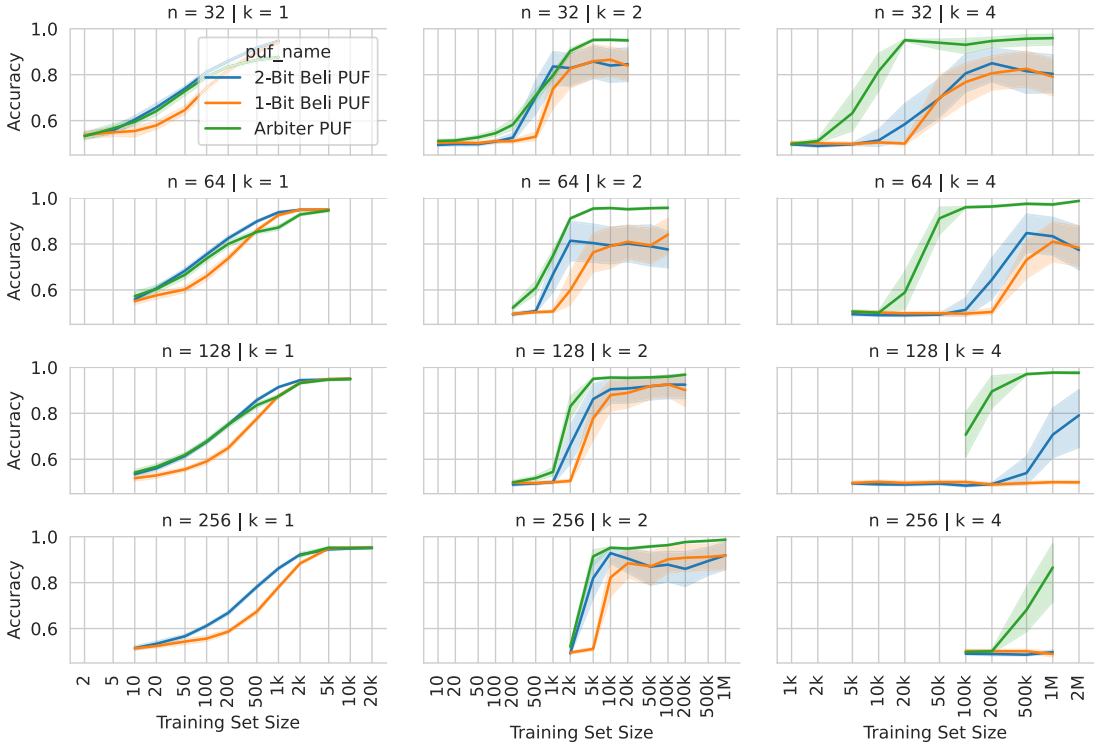


Figure 7.5.1.: Accuracy obtained when attacking the 1-bit Beli PUF, 2-bit Beli PUF, and XOR Arbiter PUF. For each combination of challenge length n, number of CRPs N, number of arbiter chains k, and PUF type, we ran at least 10 attacks with individual PUF simulation and attack initialization each. The accuracy shown is evaluated on an independent 1,000 CRPs test set. The shaded area indicates a 95% confidence interval across attacked PUF instances.

in use – also be thought of as an adaptation of the LR algorithm that is used to attack the XOR Arbiter PUF (Section 3.4).

The detailed results of our attacks are shown in Figure 7.5.1. Compared to the XOR Arbiter PUF baseline, our attacks on 1-bit and 2-bit Beli PUF generally show an increased data complexity. As expected, the results also indicate that slightly more data is required to train a model for the 1-bit Beli PUF, compared to its 2-bit version. Nevertheless, while differences in the data complexity exist, we summarize our results in saying that the Beli PUF cannot be expected to achieve a significant advantage over the attacker in terms of modeling attacks.

With the minimal feature set based on $2n + 4$ features instead of $4n$, we expect a slight decrease in data and time complexity of the attacks on Beli PUF. It thus remains unclear if the increase in data complexity shown in Figure 7.5.1 can be accounted to the structure of Beli PUF, or to the over-parameterized model, or to a combination of both. We also expect that the mediocre-performing attacks on Beli PUFs with accuracy around 80% will show better performance when operating on a minimal feature set. However, any modeling attack with prediction accuracy better than guessing should be taken as a precursor for a full break [Del19].

The difference in data complexity of 1-bit and 2-bit Beli PUF appears small when compared to the increase of data complexity by going from $k = 1$ to $k = 2$. This can be explained by the property of the Beli PUF model which requires an identical number of features to model 1-bit and 2-bit Beli PUFs. In contrast, adding another Beli PUF to the output via xoring, requires a substantial change to the modeling network.

Reviewing our modeling and attack methodology, there is no indication that this kind of modeling attack strategy cannot be applied to other MPDL PUFs. Our results thus show that PUFs based on multiple delay lines cannot be expected to provide significantly better EUF-CMA or EUF-KMA security than XOR Arbiter PUFs.

8. LP-PUF

Previous chapters show evidence that the XOR Arbiter PUF, Permutation PUF, Interpose PUF, XOR Feed-Forward PUF, and Beli PUF are not CMA-EUF secure.

Based on the Arbiter PUF and ideas discusses in previous chapters, this chapter introduces the LP-PUF and demonstrates its security against all attack strategies used above.

8.1. Design

The strong PUF circuit proposed for the LP-PUF is an advancement of the Interpose PUF design [Ngu+19] and an instance of the Composite PUF design [Sah+14]. The Composite PUF and Interpose PUF followed a design strategy similar to the Feed-Forward Arbiter PUF [Gas+04] by including challenge bits that have been generated internally, i.e. not been given as part of the challenge. This is in contrast to the MPDL PUFs introduced in Chapter 7. While this does not change the fact that the Arbiter PUF response can be effectively modeled by a linear threshold function, it is supposed to mitigate modeling attacks by depriving the attacker of the knowledge of all input bits.

We present the LP-PUF in the form of three layers, parameterized by a challenge length $n \in \mathbb{N}$ and an additional security parameter $m \in \mathbb{N}$ which must be a divisor of n.

1. In Layer 1, the LP-PUF generates m *private challenge* bits. To that end, the (*public*) challenge $c = (c_1, \ldots, c_n)$ to the PUF is split into m partitions of equal length by cutting it into m blocks $(c_1, \ldots, c_m)$,$(c_{m+1}, \ldots, c_{2m})$,$\ldots$,$(c_{n-m}, \ldots, c_n)$. Each block is fed into an individual Arbiter PUF of challenge length n/m, generating m response bits which are not part of the input, but an instance-specific function of the challenge. Note that the Arbiter PUFs in this layer are chosen deliberately short.

2. In Layer 2, the LP-PUF mixes the *private challenge* with the *public challenge* by computing a function $T : \{-1, 1\}^{n+m} \to \{-1, 1\}^n$, where each output bit of T is the parity (XOR) of exactly one of the public inputs and an individual subset of size $m/2$ of the m private inputs. We chose the involved subsets uniformly at random at design-time. This operation thus does not depend on the given PUF instance, but is a design-constant.

3. In Layer 3, the n-bit challenge computed in Layer 2 is fed into an ordinary n-bit m-XOR Arbiter PUF, which produces the final output bit of the LP-PUF.

Inspired by other Composite PUF designs such as the Feed-Forward Arbiter PUF and Interpose PUF, we designed the LP-PUF to use easy-to-model building blocks combined

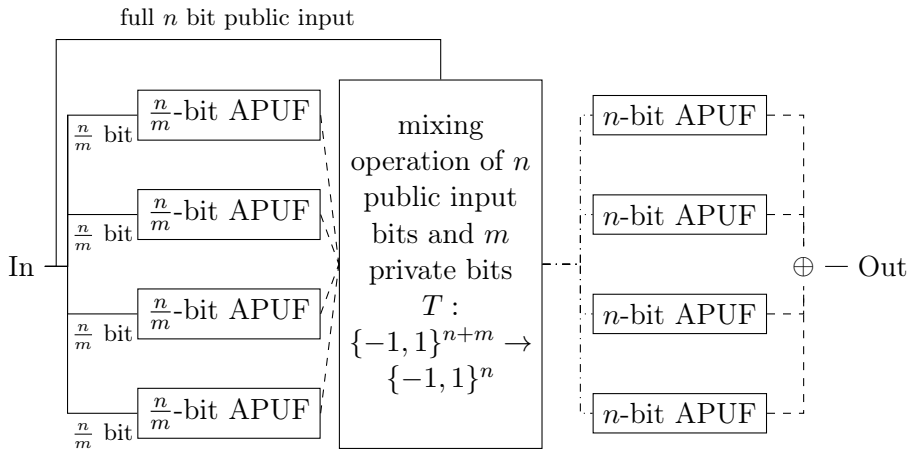


Figure 8.1.1.: The LP-PUF design, parameterized by the challenge length n and the additional security parameter m. Layers 1 through 3 are shown from left to right. Solid lines show attacker-known information, dashed lines attacker-unknown information. The result of the mixing operation is shown as dashed-dotted, as the attacker only has partial information. All Arbiter PUFs ("APUF") shown have the given number of input bits and a single output bit. Inputs to the mixing operation T are concatenated; $\oplus$ denotes the XOR operation.

with attacker-unknown outputs (in Layer 1) and attacker-unknown inputs (in Layer 3) to build a composite PUF which is resistant to known modeling attacks.

There are various motivations for the different aspects of our design. To mitigate the Splitting Attack (Section 5.1) and a generic attack on Composite PUFs [Sah+15], we introduced the use of more than one "interpose bit" as well as the mixing operation in Layer 2. This drastically reduces the chance of the attacker to guess the feature vector required to learn the XOR Arbiter PUF of Layer 3. We detail on the mitigation of the splitting attack in Section 8.3.

By reducing the attacker-knowledge about the input to Layer 3, the mixing operation of Layer 2 also mitigates reliability-based attacks [Bec15; TAB21] on Layer 3. This is detailed in Section 8.4.

The use of Arbiter PUFs in our design is to facilitate a CMOS-compatible design, which allows for fabrication of the LP-PUF using standard design processes [Gas+02]. It also benefits from literature available on implementation [such as DV13] and a well-studied model of its behavior (see Theorem 8).

The use of short Arbiter PUFs in Layer 1 is motivated by the hope that short Arbiter PUFs can be implemented such that they generate very reliable responses. In Section 8.4, we detail on this. In Section 8.6, we discuss potential problems with this choice with respect to chosen-challenge attacks.

This yields an overall structure that vaguely resembles a substitution-permutation-network, which are used in block cipher design. Specifically, and in contrast to proposals

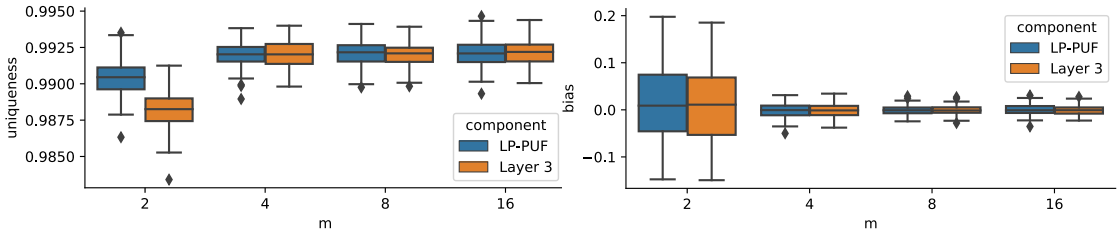


Figure 8.2.1.: Uniqueness and bias of the proposed LP-PUF, measured in noise-free simulations and for $n = 64$ challenge bits across different values of the m parameter. For comparison, the corresponding metrics are also shown for Layer 3 of the LP-PUF, which consists of a traditional n-bit m-XOR Arbiter PUF.

such as the Lightweight Secure PUF (Section 4.2) and Permutation PUF (Section 4.3), the LP-PUF employs a scheme where the attacker cannot compute the first or last operation in the network. Furthermore, in an advancement of the Interpose PUF design, by introducing the mixing operation in Layer 2, the LP-PUF combines operations of each low complexity, but from different "realms", albeit limited to only one and a half "rounds".

Alternatives and extensions of these design choices include to use several rounds, i.e. to introduce a second mixing layer, and/or to not use the original challenge input in deeper layers. However, we found that such variations exhibit poor reliability and are thus unpractical unless Arbiter PUFs with much better reliability can be build. Nevertheless, we believe that the security of the construction would greatly benefit from such modifications.

8.2. Metrics

The metrics of the LP-PUF shown here are computed based on the Arbiter PUF simulation and measurement implementations of pypuf [Wis+21b]. Values are based on 10,000 simulated PUF instances each. Reliability is measured by evaluating 1,000 challenges 5 times each on 10 instances per reliability setting of Layer 1 and Layer 3.

The results shown in this section justify the hope that the LP-PUF can fulfill requirements on uniqueness, bias, bit sensitivity, and reliability (see Section 2.1). The LP-PUF shows high uniqueness, low bias, bit sensitivity similar to that of an XOR Arbiter PUF, and fair reliability in our simulations.

In Figure 8.2.1, we show the uniqueness and bias values, compared to a baseline given by the XOR Arbiter PUF in Layer 3 of the LP PUF. In all studied cases, the LP-PUF shows the same or better uniqueness and bias distribution as the XOR Arbiter PUF.

In Figure 8.2.2, we show the bit sensitivity for LP-PUF and XOR Arbiter PUFs, which are very similar, and could be improved by adjusting the mixing operation in Layer 2.

The reliability of the LP-PUF must be studied in more detail, as it is crucial for the feasibility of LP-PUF implementations. (It is easy to come up with a PUF design that

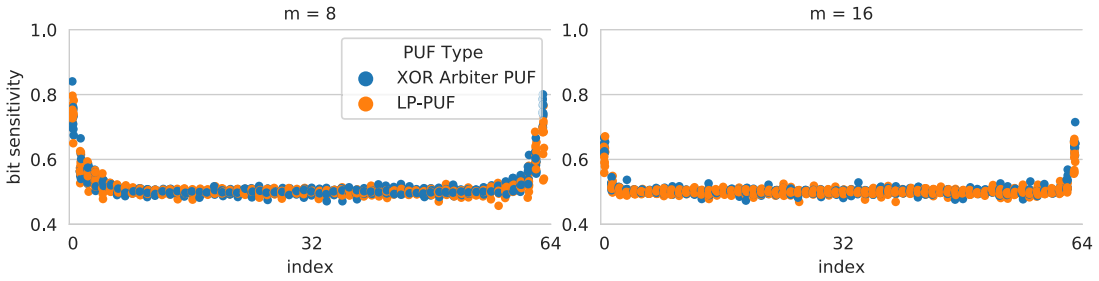


Figure 8.2.2.: Bit sensitivity values for the LP-PUF and XOR Arbiter PUF for both 64-bit challenges. Values of 1/2 are ideal.

is resilient to modeling attacks when reliability is not an issue.)

As a design composed of several building blocks, the reliability of the LP-PUF is a function of the reliability of the involved building blocks. Solely composed of Boolean logic, Layer 2 is assumed to be fully reliable. For the XOR Arbiter PUF in Layer 3, commonly believed reliability values can be found in the literature [DV13; Bec15; Wis+20b] if 64-bit challenges are employed. For the (short) Arbiter PUFs in Layer 1, however, to the extend of our knowledge, no reliability estimate is available in the literature. (There are some arguments to justify an increase in stability for very long Arbiter PUFs [Wis+20b].) The established noise model used for Arbiter PUFs [DV13] does not allow reliability predictions for longer or shorter challenges, as it remains unclear how much noise is introduced by the n stages used in the Arbiter PUF and how much noise is due to the one arbiter element. Other engineering factors which might change with increasing challenge length are also not taken into account by the commonly used Arbiter PUF noise model. We conclude that the reliability of Arbiter PUFs with challenge lengths other than the usual 64 and 128 bit remains an open research question. In lack of better options, we assume that the reliability of the LP-PUF Layer 1 will be in between 99.8% and 87.7%.

In Figure 8.2.3, we study the reliability of the LP-PUF based on simulations and as a function of the reliability of Layers 1 and 3. For Layer 1, we give the average reliability of the m Arbiter PUFs in use, but remark that there is little variance. For Layer 3, we give the reliability of the single output bit of Layer 3, as measured individually, i.e., with challenges directly applied to Layer 3, disregarding layers 1 and 2.

We conclude that assuming a 96.3% reliability for Layer 1 and a 79% reliability of Layer 3, the LP-PUF is conceivable for at least $m = 8$, as the total reliability in this case is estimated at 73%. While this reliability is within the acceptable range for a basic authentication protocol based on pre-recorded challenges, we remark that an even lower reliability could make the protocol inefficient or shrink the security margin against attackers using models with weak prediction accuracy. Hence, to obtain a definite answer on the feasibility of the LP-PUF design, a study of the reliability of real-world data will be necessary.

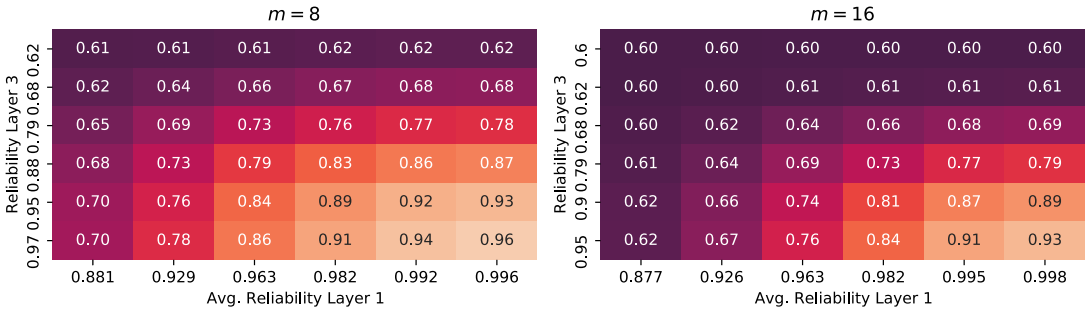


Figure 8.2.3.: Simulated reliability of the 64-bit LP-PUFs depending on reliability of the building blocks.

8.3. Splitting Attack

As the LP-PUF is an extension of the Interpose PUF [Ngu+19], we first consider an extension of the Splitting Attack as described in Section 5.1. It crucially relies on guessing the challenge vector applied to the lower layer of the Interpose PUF by guessing $c^{(-)}$ and $c^{(+)}$ in Algorithm 5.1. If the challenge vector has been guessed correctly, also the feature vector for the lower layer is uniquely identified. For the analysis of the Splitting Attack with regard to the LP-PUF, we hence analyze the probability that the attacker guesses the *feature vectors* x required for training correctly. In the Interpose PUF, the guessed challenge bit is $c_{n/2+1}$, and due to the nature of the feature vector required for training models of XOR Arbiter PUFs (see Theorem 8), this challenge bit appears in the first $n/2 + 1$ features x_i for training. So, while the attacker has to guess many features, they are perfectly correlated. The probability to guess an entire n-feature vector correctly is thus 50%. The probability to guess individual feature bits correctly is approx. 75% on average (50% for the feature bits including the interpose bit, 100% for the feature bits not including it).

We note that it is not sufficient to extend the Interpose PUF with a number of l interpose bits to mitigate this attack. One could expect that the guessing probability of the attacker in that scenario is degraded to 2^{-l}, but we show that this is not the case. If there were two interpose bits c_i and c'_i in the middle of the lower layer, then the first $n/2$ features of the lower layer all include the XOR of c_i and c'_i — a value that the attacker can still guess with probability 50%; so not much is gained in this setting. Similar arguments apply for any number of interpose bits. Distributing these interpose bits across the challenge of the lower layer, i.e. not only interposing in the middle, opens up other attack surfaces, as outlined by the original authors [Ngu+19].

In the LP-PUF design, the mixing operation in Layer 2 is aimed at removing correlations from the feature bits to minimize the guessing advantage. The goal is that the attacker can guess feature bits correctly only with probability 50%, and feature vectors only with probability approximately 2^{-m}. We confirmed this in our simulations. The measured guessing probabilities for feature *bits* were at 53% and 50% for $m = 4$ and

$m \geq 8$, respectively. The measured probabilities for guessing feature *vectors* correctly were 13% for $m = 4$ and 0.7% for $m = 8$ and $< 1/10{,}000$ for $m = 16$. In this way, the LP-PUF provides a way to introduce almost m bit of entropy in the challenges to Layer 3.

We did not study the guessing probability for each feature bit separately, but remark that if the attacker is able to guess single feature bits with higher probability, or finds correlations between the feature bits, then using this knowledge may enable the attacker to increase their guessing probability.

The reduced guessing probability for the features to the model of Layer 3 constitutes itself in a significant increase of required CRPs for successfully training a model. In the case of $m = 4$, the (adapted) splitting attack requires approx. 500,000 CPRs to train a high-accuracy model, compared to 60,000 CRPs for the (m, m)-Interpose PUF and 30,000 CRPs for the m-XOR Arbiter PUF. We believe that the reason that the training succeeds at all is that the probability to guess feature vectors correctly is still at 13%, which means that guessing errors can be averaged out over large sets of CRPs. However, as the guessing advantage of the attacker declines exponentially with m, we also expect an exponential increase of the required CRPs in m. Unfortunately, we also have seen in Section 8.4 that the reliability of the LP-PUF suffers greatly from an increase of m.

Nevertheless, there is hope that the LP-PUF could find a sweet spot that mitigates the Splitting Attack, while at the same time provides sufficiently reliable responses, e.g. for $m = 8$ or $m = 16$. Note that while the XOR Arbiter PUF also suffers from decreasing reliability in it's security parameter, known chosen message attacks based on response reliability [Bec15; TAB21], in contrast to LP-PUF, cannot be mitigated by increasing the XOR Arbiter PUF size.

8.4. Reliability Attack

In the past, chosen message attacks based on the individual challenge reliability have targeted Arbiter PUFs [DV13], XOR Arbiter PUFs (Section 3.8), and Interpose PUFs ([TAB21]). The observation fundamental to all of these attacks is that the reliability of an Arbiter PUFs response to a given challenge is a function of the delay difference corresponding to this challenge. The smaller the absolute value of the delay difference, the higher the unreliability (Theorem 22). This means that Arbiter PUFs can be identified not only by their response behavior, but also by their reliability. In the case of single Arbiter PUFs, it is sufficient to obtain an approximate solution to a system of linear equations. In the case of XOR Arbiter PUFs, evolution strategies or gradient-decent machine learning algorithms can be employed to find high accuracy models. These approaches are based on the Pearson correlation of the measured reliability of target PUF and model; the higher the correlation, the more accurate the model will be.

All Arbiter PUFs in a composite design can be target of a reliability-based attack if the attacker can correlate any measurable reliability to the reliability of the target Arbiter PUF. In case of the XOR Arbiter PUF, it was found that the XOR Arbiter PUF's output reliability is correlated with the reliability of the individual Arbiter PUFs as shown in

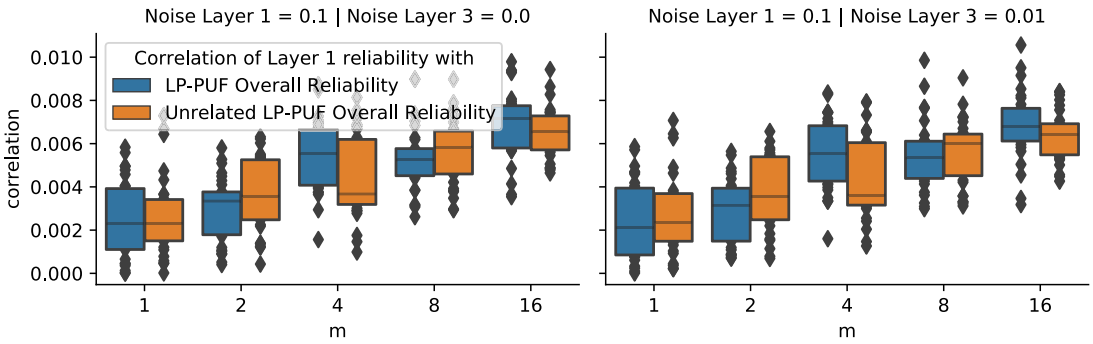


Figure 8.4.1.: The correlation of the attacker-observable overall reliability of a given $n = 64$ challenge bit LP-PUF with the reliability of Layer 1. To account for all response bits of Layer 1, the maximum correlation in each instance is taken. If a large correlation of Layer 1 reliability and LP-PUF reliability can be established, an attacker could attempt a reliability-based modeling attack on Layer 1.

Section 3.8. Similarly, the output of the Interpose PUF has reliability correlation with the lower layer [TAB21].

To analyze the vulnerability of the LP-PUF towards reliability-based attacks, we thus study the reliability correlation of Layer 1 and Layer 3 with the attacker-measurable reliability of responses at the LP-PUF output. Based on our simulations, we could not find significant correlations of output reliability and Layer 1, as shown in Fig. 8.4.1. Instead, the correlation shows values that are also measured when compared to an entirely unrelated PUF. (The increase with m can be explained as we show the *maximum* correlation to any of the m individual reliability vectors of Layer 1; it thus does not constitute a significant correlation.) This result is expected and applies similarly to the Interpose PUF.

The reliability correlation of Layer 3 with the output of the LP-PUF is high for small values of m and indicates that an attack for these values of m will be possible. However, as we increase m, the correlation vanishes, with $m = 8$ and $m = 16$ hardly showing any difference when compared to the correlation with an unrelated PUF instance (Figure 8.4.2) and hence constitute an improvement over the reliability behavior of the Interpose PUF. We conclude that increasing m will mitigate current versions of the reliability attack.

We note that the attack is *not* mitigated by removing unreliable challenges from Layer 3, or by improving the reliability of the implementation. (Using the reliability behavior as studied in Section 3.7 as evidence, we believe that this approach is not promising.) Instead, by decreasing attacker knowledge of the challenge applied to Layer 3, we remove the attackers ability to meaningfully correlate the measured reliability, which prevents an application of evolution-strategies or gradient descent machine learning algorithms. Still, the theoretical analysis of the reliability-based attacks is quite thin, and we are afraid that there could be a way to adapt the attack to account for the mixing operation

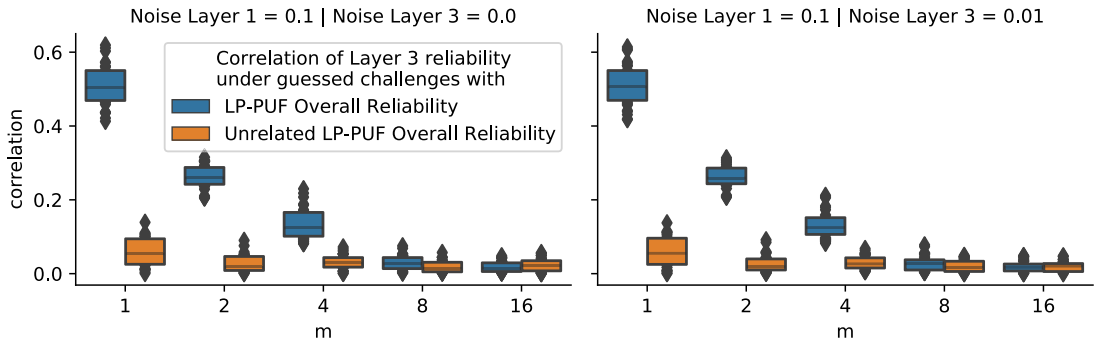


Figure 8.4.2.: The correlation of the attacker-observable overall reliability of a given $n = 64$ challenge bit m-LP-PUF with the reliability of it's Layer 3 under attacker-guessed challenges. High correlations pave the way to conduct a reliability-based attack on Layer 3 (as done on the Interpose PUF Tobisch, Aghaie, and Becker [TAB21]). The absence of high correlation for the LP-PUF is *not* caused by increasing the reliability of Layer 3, but by reducing the ability of the attacker to guess Layer 3 input bits.

in Layer 2, especially since the attacker can choose the individual inputs to the PUFs in Layer 1.

8.5. MLP Attack

In Section 3.6, we have shown that Neural Network Attacks on XOR Arbiter PUFs can work well; in Section 5.4 we demonstrated that they can also work against composite designs. Such attacks have the advantage over the LR attack that no exact model is required, while at the same time, the required CRPs for modeling XOR Arbiter PUF and Interpose PUF is decreased. On the downside, even a successful modeling attack will not allow much insight in the inner workings of the attack, as it cannot be expected that the trained weights of the model can be interpreted. As such, it is well-suited for quick and preliminary analysis of novel PUF designs such as the LP-PUF.

We first consider the MLP attack on the full 64-bit LP-PUF as done similarly in Section 5.4 on variants of the Interpose PUF, i.e. without applying any technique similar to the splitting attack discussed in Section 8.3.

While for $m = 2$, we were able to obtain models with an accuracy around 80% (reminiscent of the first step of the splitting attack), already for $m = 4$ we did not achieve any significant success, even when using 50 million CPRs. (In Section 3.6 we have shown that the 64-bit 4-XOR Arbiter PUF requires merely 150,000 CRPs.) We can conclude that either we chose inappropriate network parameters (we tried networks which have been shown to be able to attack 4-XOR Arbiter PUFs and 5-XOR Arbiter PUFs), or that the MLP attack might not be able to infer the features required to model Layer 3. As evidence for the latter case, it was reported that MLP is also unable to train a model

given the *challenges*, instead of the Arbiter PUF *features* [SBC19].

An MLP-based Splitting Attack on the LP-PUF is also conceivable, as it has been demonstrated against the Interpose PUF in Section 5.1. However, it faces the same difficulties in guessing the feature bits for the model of Layer 3, and is hence largely covered by our arguments in Section 8.3. Given that the MLP attack has been shown to reduce the number of required CRPs (Section 3.6), this may also apply to the splitting attack discussed in this paper.

8.6. Limitations

The design of the LP-PUF and the results presented on its metrics and security properties aim at making the case that the LP-PUF and related designs are secure against known modeling attacks; however the analysis presented here is by no means exhaustive and should be extended to go beyond the attacks that are known in the literature:

To conduct a more rigorous security analysis, a formal model of the LP-PUF, based on the additive delay model, should be derived. Due to the verbose definition of the mixing operation in Layer 2, the use of a computer algebra system such as Sage may be necessary. To the extend of our knowledge, no such analysis has ever been done on a PUF. Such a formal model will serve as a basis for a formalization of some of the arguments made above, e.g. for the decreasing chances of the attacker to guess the feature bits when m is increased. It could also allow for a more rigorous choice of the mixing operation, rather than just using (at design time) randomly chosen subsets and allow for an improvement in the bit sensitivity of the LP-PUF.

Likewise, a model for the reliability of the LP-PUF needs to be developed, to make sure that attacks based on the correlation of reliability behavior cannot be adjusted to somehow take the mixing operation in Layer 2 into account and recover an Arbiter PUF model (see Section 8.4). To the best of our knowledge, no such model has been published for comparable designs like the Interpose PUF.

To increase the trustworthiness of our failed modeling attempts using machine learning algorithms, it will also be necessary to revisit the chosen hyperparameters and argue in detail that also hyperparameter optimization will not enable the attacker to obtain a model of the LP-PUF or parts thereof.

Furthermore, it is uncertain if Arbiter PUFs of short length and high reliability can be build; the commonly used noise model of the Arbiter PUF (Equation (3.1.1)) is ill-suited to make a prediction. This can be clarified by studying the behavior of short Arbiter PUFs in real hardware or by replacing them with an alternative solution.

Adjusting the design of Layer 1 may also be indicated in the face of general chosen message attacks to defend against attackers that choose challenges to attack individual Arbiter PUFs of Layer 1. Alternatives to Layer 1 could try to limit the freedom of the attacker in choosing which challenges are applied to which Arbiter PUF; at the very least, they should remove known weaknesses in the bit sensitivity of the Arbiter PUFs in Layer 1. Hence, if technically feasible, the use of n-bit Arbiter PUFs in Layer 1 is preferred as it provides better protection against the Composite PUF crypt-analytic attack [Sah+15].

Also not included in this work is an analysis of the LP-PUF with respect to its PAC learnability. While here, we cannot expect to obtain a negative result, the known proofs of learnability should be applied to the new setting to verify that no known attack applies. As a first step, the PUF-G framework [CMH20] and the PUFMeter [GFS19] should be applied to the LP-PUF.

Finally, even though we argue that a PUF design needs to withstand all scrutiny in an idealized form, i.e. theoretically and in simulation, eventually also an implementation needs to be analyzed with the same precision. To that end, FPGA or ASIC data has to be collected. Due to the highly specific nature of the mixing operation in Layer 2 which generates the challenge to Layer 3, none of the publicly available Arbiter PUF measurement data is suited for this task. Nevertheless, by the composite design of the LP-PUF, we can already state that the consumed area will be on the order of the area utilized by two m-XOR n-bit Arbiter PUFs.

9. pypuf: Python Software Library for PUF Research

The research presented in this work crucially depends on implementations of modeling attacks. Unfortunately, it has not yet become standard to publish data along with working source code as artifacts along with articles on the topic. Consequently, results cited in this work are often difficult to replicate. This is especially true for adjusting the hyperparameters of machine learning algorithms, which can be decisive for the achieved learning result (as shown in Figure 3.6.1).

To address these issues, we started the pypuf project, which provides a Python software library [Wis+21b] containing PUF simulations, access to real-world PUF data sets, PUF metrics, as well as implementations of PUF attacks from the literature, including proof-of-concept examples. It is published as open-source software under a free license.

In detail, pypuf supports the simulation of Arbiter PUFs and XOR Arbiter PUFs based on the additive delay model (Chapter 3) as well as simulations of XOR Arbiter PUFs with Input Transformations (Chapter 4), both noise-free and with adjustable noise. Beyond that it supports the simulation of Feed-Forward Arbiter PUFs using a naive extension of the additive delay model. Simulations of the Bistable Ring PUF [Che+11] and integrated optical PUFs [Rüh+13a] are based on findings on modeling attacks in the literature and have not undergone the same scrutiny as the additive delay model in the case of Arbiter PUFs.

To complement PUF simulations, real-world data is also shipped for the Arbiter PUF and the Interpose PUF.

To assess PUF metrics as described in Section 2.1, pypuf implements functions to measure the bias, reliability, uniqueness, similarity, correlation, bit sensitivity, and total influence (the last one being important for analysis using PUFMeter [GFS19]) of challenge-response pairs.

Crucially, pypuf includes working examples of attacks on XOR Arbiter PUFs and variants both based on the LR attack (Section 3.4) as well as based on MLPs (Section 3.6). It also contains variants of these attacks on the Interpose PUF, Beli PUF, and LP-PUF.

pypuf can be used to replicate all results of this work and can thus help to spot software errors in past and future publications.

10. Conclusion

In this work, we explored various directions in the design space for Physical Unclonable Functions based on Arbiter PUF variants. Our efforts can roughly be divided into two categories. One studied composite designs, i.e. PUFs built from Arbiter PUF building blocks, such as the XOR Arbiter PUF (Chapter 3) or the Interpose PUF (Chapter 5). The second category involves modifying the Arbiter PUF design itself, such as it is the case in the Feed Forward Arbiter PUF (Chapter 6) and the Beli PUF (Chapter 7).

All variants and variations have been studied with respect to security with respect to existential unforgability (EUF) against modeling attacks based on the known message model (KMA) and chosen message model (CMA) as introduced in Section 2.1. We summarize our findings in Table 10.1.

In more detail, we find that the Arbiter PUF (Chapter 3) does not provide security against modeling attacks under either KMA or CMA. While data complexities under both premises is very low, the CMA in Section 3.8 on the Arbiter PUF require exceptionally few measurements of the PUF under attack.

The k-XOR Arbiter PUF (Chapter 3) provides an additional security parameter that allows defense against KMAs by increasing k. However, in real-world implementations this also results in loss of reliability and has thus only limited applicability. In the known message attack model, we demonstrate fast attacks based on the Logistic Regression attack (Section 3.4) and attacks with reduced data complexity based on neural networks (Section 3.6) that cover most realistic security parameter settings and leave little hope that KMA security can be achieved. There are also CMAs that cannot be mitigated by increasing k (Section 3.8). We must conclude that (real-world) XOR Arbiter PUFs are insecure against both known and chosen message attacks.

The XOR Majority Vote Arbiter PUF (Section 3.7) is an attempt to provide KMA security on the basis of XOR Arbiter PUFs, but can only do so with an decrease in hardware security as additional circuitry is needed (see also Section 2.4).

Adding an input transformation to the XOR Arbiter PUF, which modifies the given challenge before application to the PUF, as done in Chapter 4, can only slightly increase the data complexity of KMA and thus does not provide sufficient security against modeling attacks.

In Chapter 5, we demonstrated that also the Interpose PUF cannot provide security significantly better than XOR Arbiter PUFs under known message attacks. In concurrent work, the Interpose PUF has also been shown to be vulnerable to chosen message attacks [TAB21].

Also for the Feed Forward Arbiter PUF (Chapter 6), we demonstrated successful known message attacks based on neural networks. Similar to the XOR Arbiter PUF, increasing the security parameter on Feed Forward PUFs also will lead to instability, so that this

	PUF Design	Parameters	Successful EUF attacks KMA	CMA
Chapter 3	Arbiter PUF	n	LR, NN	algebraic
Chapter 3	XOR Arbiter PUF (bounded k)	n, k	LR, NN	reliability
Chapter 3	XOR Majority Vote Arbiter PUF (unbounded k)	n, k	—	reliability
Chapter 4	Lightweight Secure PUF	n, k	LR, NN	
Chapter 4	Permutation PUF	n, k	LR, NN	
Chapter 5	Interpose PUF	$n, k_{\text{up}}, k_{\text{down}}$	Splitting	reliability
Chapter 6	XOR Feed-Forward Arbiter PUF	n, l, k	NN	
Chapter 7	Beli PUF	n, k	NN	
Chapter 8	LP-PUF	n, m	—	—

Table 10.1.: Overview of PUF designs and modeling attacks in the known message and chosen message attacker model, as discussed in this work. All KMA attacks are also applicable in the CMA setting. "—" indicates that no attacks are known, however, existence of attacks can, by matter of principle, not ruled out. Attacks refer to Logistic Regression (LR, Section 3.4), Neural Networks (NN, Section 3.6, Section 7.5), Splitting Attack (Section 5.1), and algebraic and reliability attacks (Section 3.8).

class of PUFs also needs to be considered vulnerable to modeling attacks under the KMA model.

In Chapter 7, we propose a modification of the Arbiter PUF with more than two delay lines. While the resulting Beli PUF shows some promising metrics and interesting model properties, also for Beli PUF we demonstrate vulnerability under the known message attack model, with data complexity only slightly increased over the XOR Arbiter PUF.

Finally, we propose LP-PUF (Chapter 8), and study its security using all above-mentioned modeling attack strategies. We find that LP-PUF is not vulnerable to any of the known message attacks studied in this work. Furthermore, it remains unclear how existing chosen message attacks could be applicable to LP-PUF. We conclude that given the current state-of-the-art, LP-PUF can be considered EUF-CMA secure. While the hardware attack surface of LP-PUF is slightly increased compared to the Interpose PUF, intermediate values only need to be stored while the evaluation of the involved Arbiter PUF lasts. This indicates that the hardware attack surface of the LP-PUF remains below the attack surface of the XOR Majority Vote Arbiter PUF (Section 3.7).

The fact that we find LP-PUF to be secure against the state-of-the-art does not replace the investigation of novel attack strategies, as suggested in Section 8.6. To enable further study of the LP-PUF's modeling attack security, this work is accompanied by a large software library that facilitates the simulation, testing, and attacking of PUFs (Chapter 9).

Acknowledgments

I would like to thank the many people that supported me in many ways during the journey of my Ph.D. study.

First, I thank my advisors Marian Margraf and Jean-Pierre Seifert for their patient guidance for my thesis as well as for the expertise they shared with me.

For helpful comments and discussions on the matter of the XOR Majority Vote Arbiter PUF, I would like to thank Christoph Graebnitz, Manuel Oswald, Tudor A. A. Soroceanu, and Benjamin Zengin.

Also, I would like to thank Christopher Mühl and Niklas Pirnay for joining my research efforts and his contributions to the Splitting Attack on the Interpose PUF and the findings on the systematic bias of XOR Arbiter PUFs. For work on the Interpose PUF Splitting Attack, I'm also thankful for the support of Phuong Ha Nguyen, Marten van Dijk, and Ulrich Rührmair.

I also thank Pranesh Santikellur for helpful comments on the TRN-based attack and Ahmad O. Aseeri for providing the source code of their attack. With respect to the resulting work on neural network attacks on PUFs, I thank my co-authors Khalid T. Mursi, Bipana Thapaliya, and Yu Zhuang.

For fruitful collaborations, I would like to thank Anita Aghaie, Georg Becker, Amir Moradi, and Johannes Tobisch.

For inspiring further experiments on the noise resilience of the LR attack, which ultimately resulting in the LP-PUF, I thank Roel Maes.

I'm indebted to Peter Thomassen for the many conversations that we had, both on general concepts in the field of Physical Unclonable Functions and on very specific details of my research.

For contributions to pypuf, I thank Wolfgang Studier, Yannan Tuo, and Adomas Baliuka. I also thank Franziskus Kiefer, Matthias Krause, Maximilian Hünemörder, and Nanni Schüler for helpful discussions and suggestions.

I extend my thanks to the anonymous referees of my paper submissions, who provided many helpful comments and guided me to improve the quality of this work greatly.

For a substantial amount of computing time, I thank the high performance computing centers of Texas Tech University, Technische Universität Berlin, and Freie Universität Berlin.

Last, but not least, I thank my family for their endless support. Jenny, without you, this thesis would not have seen the light of day.

Bibliography

[Agh+22] Anita Aghaie et al. "Security Analysis of Delay-Based Strong PUFs with Multiple Delay Lines". Jan. 2022.

[AK96] Ross Anderson and Markus Kuhn. "Tamper Resistance - a Cautionary Note". In: *Proceedings of the 2nd Workshop On Electronic Commerce*. Nov. 1996. URL: https://www.usenix.org/legacy/publications/library/proceedings/ec96/full_papers/kuhn/index.html (visited on 01/24/2022).

[AM20] Anita Aghaie and Amir Moradi. "TI-PUF: Toward Side-Channel Resistant Physical Unclonable Functions". In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 3470–3481. ISSN: 1556-6013, 1556-6021. URL: https://ieeexplore.ieee.org/document/9063465/ (visited on 03/22/2021).

[AM21] Anita Aghaie and Amir Moradi. "Inconsistency of Simulation and Practice in Delay-based Strong PUFs". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (July 2021), pp. 520–551. ISSN: 2569-2925. URL: https://tches.iacr.org/index.php/TCHES/article/view/8985 (visited on 08/20/2021).

[Arm+09] Frederik Armknecht et al. "Physically Unclonable Pseudorandom Functions". In: *Poster at EUROCRYPT - 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2009), p. 18.

[Arm+11] Frederik Armknecht et al. "A Formalization of the Security Features of Physical Functions". In: *2011 IEEE Symposium on Security and Privacy*. Oakland, CA, USA: IEEE, May 2011, pp. 397–412. ISBN: 978-1-4577-0147-4. URL: http://ieeexplore.ieee.org/document/5958042/ (visited on 09/12/2018).

[Arm+16] Frederik Armknecht et al. "Towards a Unified Security Model for Physically Unclonable Functions". In: *Cryptographers' Track at the RSA Conference*. 2016. URL: http://eprint.iacr.org/2016/033 (visited on 09/12/2018).

[AZ17] Mohammed Saeed Alkatheiri and Yu Zhuang. "Towards Fast and Accurate Machine Learning Attacks of Feed-Forward Arbiter PUFs". In: *2017 IEEE Conference on Dependable and Secure Computing*. Aug. 2017, pp. 181–187.

[AZA18] A. O. Aseeri, Y. Zhuang, and M. S. Alkatheiri. "A Machine Learning-Based Security Vulnerability Study on XOR PUFs for Resource-Constraint Internet of Things". In: *2018 IEEE International Congress on Internet of Things (ICIOT)*. July 2018, pp. 49–56.

[AZP20] S. V. Sandeep Avvaru, Ziqing Zeng, and Keshab K. Parhi. "Homogeneous and Heterogeneous Feed-Forward XOR Physical Unclonable Functions". In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 2485–2498. ISSN: 1556-6021.

[Bec15] Georg T. Becker. "The Gap Between Promise and Reality: On the Insecurity of XOR Arbiter PUFs". In: *Cryptographic Hardware and Embedded Systems – CHES 2015*. Ed. by Tim Güneysu and Helena Handschuh. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2015, pp. 535–555. ISBN: 978-3-662-48324-4.

[Ber41] Andrew C. Berry. "The Accuracy of the Gaussian Approximation to the Sum of Independent Variates". In: *Transactions of the American Mathematical Society* 49.1 (1941), pp. 122–136. ISSN: 0002-9947, 1088-6850. URL: `https://www.ams.org/tran/1941-049-01/S0002-9947-1941-0003498-3/` (visited on 11/01/2021).

[Brz+11] Christina Brzuska et al. "Physically Uncloneable Functions in the Universal Composition Framework". In: *Advances in Cryptology – CRYPTO 2011*. Ed. by David Hutchison et al. Vol. 6841. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 51–70. ISBN: 978-3-642-22791-2 978-3-642-22792-9. URL: `http://link.springer.com/10.1007/978-3-642-22792-9_4` (visited on 09/12/2018).

[Che+11] Qingqing Chen et al. "The Bistable Ring PUF: A New Architecture for Strong Physical Unclonable Functions". In: *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*. San Diego, CA, USA: IEEE, June 2011, pp. 134–141. ISBN: 978-1-4577-1059-9. URL: `http://ieeexplore.ieee.org/document/5955011/` (visited on 09/11/2018).

[CMH20] D. Chatterjee, D. Mukhopadhyay, and A. Hazra. "PUF-G: A CAD Framework for Automated Assessment of Provable Learnability from Formal PUF Representations". In: *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. Nov. 2020, pp. 1–9.

[Del19] J. Delvaux. "Machine-Learning Attacks on PolyPUFs, OB-PUFs, RPUFs, LHS-PUFs, and PUF–FSMs". In: *IEEE Transactions on Information Forensics and Security* 14.8 (Aug. 2019), pp. 2043–2058.

[DV13] Jeroen Delvaux and Ingrid Verbauwhede. "Side Channel Modeling Attacks on 65nm Arbiter PUFs Exploiting CMOS Device Noise". In: *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium On*. IEEE, 2013, pp. 137–142.

[Gan+15] Fatemeh Ganji et al. "Lattice Basis Reduction Attack against Physically Unclonable Functions". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15*. Denver, Colorado, USA: ACM Press, 2015, pp. 1070–1080. ISBN: 978-1-4503-3832-5.

URL: http://dl.acm.org/citation.cfm?doid=2810103.2813723 (visited on 09/20/2019).

[Gan+16] Fatemeh Ganji et al. "Strong Machine Learning Attack Against PUFs with No Mathematical Model". In: *Cryptographic Hardware and Embedded Systems – CHES 2016*. Ed. by Benedikt Gierlichs and Axel Y. Poschmann. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2016, pp. 391–411. ISBN: 978-3-662-53140-2.

[Gas+02] Blaise Gassend et al. "Silicon Physical Random Functions". In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. CCS '02. New York, NY, USA: ACM, 2002, pp. 148–160. ISBN: 978-1-58113-612-8. URL: http://doi.acm.org/10.1145/586110.586132 (visited on 10/14/2019).

[Gas+04] Blaise Gassend et al. "Identification and Authentication of Integrated Circuits". In: *Concurrency and Computation: Practice and Experience* 16.11 (Sept. 2004), pp. 1077–1098. ISSN: 1532-0634. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.805 (visited on 09/19/2018).

[GFS19] Fatemeh Ganji, Domenic Forte, and Jean-Pierre Seifert. "PUFmeter a Property Testing Tool for Assessing the Robustness of Physically Unclonable Functions to Machine Learning Attacks". In: *IEEE Access* 7 (2019), pp. 122513–122521. ISSN: 2169-3536. URL: https://ieeexplore.ieee.org/document/8819883/ (visited on 09/20/2019).

[GTS15] Fatemeh Ganji, Shahin Tajik, and Jean-Pierre Seifert. "Why Attackers Win: On the Learnability of XOR Arbiter PUFs". In: *Trust and Trustworthy Computing*. Ed. by Mauro Conti, Matthias Schunter, and Ioannis Askoxylakis. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 22–39. ISBN: 978-3-319-22846-4.

[GTS16] Fatemeh Ganji, Shahin Tajik, and Jean-Pierre Seifert. "PAC Learning of Arbiter PUFs". In: *Journal of Cryptographic Engineering* 6.3 (Sept. 2016), pp. 249–258. ISSN: 2190-8516. URL: https://doi.org/10.1007/s13389-016-0119-4 (visited on 09/24/2018).

[GTS18] Fatemeh Ganji, Shahin Tajik, and Jean-Pierre Seifert. "A Fourier Analysis Based Attack Against Physically Unclonable Functions". In: *Financial Cryptography and Data Security*. Ed. by Sarah Meiklejohn and Kazue Sako. Vol. 10957. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018, pp. 310–328. ISBN: 978-3-662-58386-9 978-3-662-58387-6. URL: http://link.springer.com/10.1007/978-3-662-58387-6_17 (visited on 09/20/2019).

[Gua+07] Jorge Guajardo et al. "FPGA Intrinsic PUFs and Their Use for IP Protection". In: *Cryptographic Hardware and Embedded Systems - CHES 2007*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 63–80. ISBN: 978-3-540-74735-2.

[HBF07] Daniel E. Holcomb, Wayne P. Burleson, and Kevin Fu. "Initial SRAM State as a Fingerprint and Source of True Random Numbers for RFID Tags". In: *In Proceedings of the Conference on RFID Security*. 2007.

[Hel+13] C. Helfmeier et al. "Cloning Physically Unclonable Functions". In: *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. June 2013, pp. 1–6.

[HMV12] Gabriel Hospodar, Roel Maes, and Ingrid Verbauwhede. "Machine Learning Attacks on 65nm Arbiter PUFs: Accurate Modeling Poses Strict Bounds on Usability." In: *WIFS*. 2012, pp. 37–42.

[KB15] Raghavan Kumar and Wayne Burleson. "Side-Channel Assisted Modeling Attacks on Feed-Forward Arbiter PUFs Using Silicon Data". In: *Radio Frequency Identification*. Ed. by Stefan Mangard and Patrick Schaumont. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 53–67. ISBN: 978-3-319-24837-0.

[KB17] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv:1412.6980 [cs]* (Jan. 2017). arXiv: 1412.6980 [cs]. URL: http://arxiv.org/abs/1412.6980 (visited on 04/12/2021).

[KK99] Oliver Koemmerling and Markus G Kuhn. "Design Principles for Tamper-Resistant Smartcard Processors". In: (1999), p. 13.

[Lee+04] J. W. Lee et al. "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)*. June 2004, pp. 176–179.

[Lim+05] Daihyun Lim et al. "Extracting Secret Keys from Integrated Circuits". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 13.10 (Oct. 2005), pp. 1200–1205. ISSN: 1557-9999.

[LMN93] Nathan Linial, Yishay Mansour, and Noam Nisan. "Constant Depth Circuits, Fourier Transform, and Learnability". In: *J. ACM* 40.3 (July 1993), pp. 607–620. ISSN: 0004-5411. URL: http://doi.acm.org/10.1145/174130.174138 (visited on 09/23/2019).

[Loh+16] Heiko Lohrke et al. "No Place to Hide: Contactless Probing of Secret Data on FPGAs". In: *Cryptographic Hardware and Embedded Systems – CHES 2016*. Ed. by Benedikt Gierlichs and Axel Y. Poschmann. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2016, pp. 147–167. ISBN: 978-3-662-53140-2.

[Mae+12] Roel Maes et al. "Experimental Evaluation of Physically Unclonable Functions in 65 Nm CMOS". In: *2012 Proceedings of the ESSCIRC (ESSCIRC)*. Bordeaux, France: IEEE, Sept. 2012, pp. 486–489. ISBN: 978-1-4673-2213-3 978-1-4673-2212-6 978-1-4673-2211-9. URL: http://ieeexplore.ieee.org/document/6341361/ (visited on 09/11/2019).

[Mae13] Roel Maes. *Physically Unclonable Functions: Constructions, Properties and Applications*. Berlin Heidelberg: Springer-Verlag, 2013. ISBN: 978-3-642-41394-0. URL: https://www.springer.com/gp/book/9783642413940 (visited on 10/21/2021).

[Mah97] David P. Maher. "Fault Induction Attacks, Tamper Resistance, and Hostile Reverse Engineering in Perspective". In: *Financial Cryptography*. Ed. by Rafael Hirschfeld. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1997, pp. 109–121. ISBN: 978-3-540-69607-0.

[Maj+12] M. Majzoobi et al. "Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by Substring Matching". In: *2012 IEEE Symposium on Security and Privacy Workshops*. May 2012, pp. 33–44.

[MEK10] Mehrdad Majzoobi, Ahmed Elnably, and Farinaz Koushanfar. "FPGA Time-Bounded Unclonable Authentication". In: *Information Hiding*. Ed. by Rainer Böhme, Philip W. L. Fong, and Reihaneh Safavi-Naini. Vol. 6387. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–16. ISBN: 978-3-642-16434-7 978-3-642-16435-4. URL: http://link.springer.com/10.1007/978-3-642-16435-4_1 (visited on 09/11/2018).

[MKP08] Mehrdad Majzoobi, Farinaz Koushanfar, and Miodrag Potkonjak. "Lightweight Secure PUFs". In: *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*. ICCAD '08. Piscataway, NJ, USA: IEEE Press, 2008, pp. 670–673. ISBN: 978-1-4244-2820-5. URL: http://dl.acm.org/citation.cfm?id=1509456.1509603 (visited on 09/19/2018).

[MP69] Marvin Minsky and Seymour A. Papert. *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, Jan. 1969. ISBN: 978-0-262-13043-1.

[Mur+20] Khalid T. Mursi et al. "A Fast Deep Learning Method for Security Vulnerability Study of XOR PUFs". In: *Electronics* 9.10 (Oct. 2020), p. 1715. URL: https://www.mdpi.com/2079-9292/9/10/1715 (visited on 02/05/2021).

[Ned+13] D. Nedospasov et al. "Invasive PUF Analysis". In: *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. Aug. 2013, pp. 30–38.

[Ngu+16] Phuong Ha Nguyen et al. "Security Analysis of Arbiter PUF and Its Lightweight Compositions Under Predictability Test". In: *ACM Trans. Des. Autom. Electron. Syst.* 22.2 (Dec. 2016), 20:1–20:28. ISSN: 1084-4309. URL: http://doi.acm.org/10.1145/2940326 (visited on 04/03/2019).

[Ngu+19] Phuong Ha Nguyen et al. "The Interpose PUF: Secure PUF Design against State-of-the-art Machine Learning Attacks". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (Aug. 2019), pp. 243–290. ISSN: 2569-2925. URL: https://tches.iacr.org/index.php/TCHES/article/view/8351 (visited on 10/11/2019).

[ODo14] Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

[Pap+02] Ravikanth Pappu et al. “Physical One-Way Functions”. In: *Science* 297.5589 (Sept. 2002), pp. 2026–2030. ISSN: 0036-8075, 1095-9203. URL: http://science.sciencemag.org/content/297/5589/2026 (visited on 09/24/2018).

[RBK10] Ulrich Rührmair, Heike Busch, and Stefan Katzenbeisser. “Strong PUFs: Models, Constructions, and Security Proofs”. In: *Towards Hardware-Intrinsic Security: Foundations and Practice*. Ed. by Ahmad-Reza Sadeghi and David Naccache. Information Security and Cryptography. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 79–96. ISBN: 978-3-642-14452-3. URL: https://doi.org/10.1007/978-3-642-14452-3_4 (visited on 09/24/2018).

[Rüh+10] Ulrich Rührmair et al. “Modeling Attacks on Physical Unclonable Functions”. In: *Proceedings of the 17th ACM Conference on Computer and Communications Security*. CCS ’10. New York, NY, USA: ACM, 2010, pp. 237–249. ISBN: 978-1-4503-0245-6. URL: http://doi.acm.org/10.1145/1866307.1866335 (visited on 09/19/2018).

[Rüh+13a] Ulrich Rührmair et al. *Optical PUFs Reloaded*. Tech. rep. 215. 2013. URL: https://eprint.iacr.org/2013/215 (visited on 08/20/2021).

[Rüh+13b] Ulrich Rührmair et al. “PUF Modeling Attacks on Simulated and Silicon Data”. In: *IEEE Transactions on Information Forensics and Security* 8.11 (2013), pp. 1876–1891.

[Ruh20] Ulrich Ruhrmair. “SoK: Towards Secret-Free Security”. In: *Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security*. ASHES’20. New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 5–19. ISBN: 978-1-4503-8090-4. URL: https://doi.org/10.1145/3411504.3421220 (visited on 05/30/2021).

[Ruk+10] Andrew Rukhin et al. “A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications”. In: *NIST Special Publication 800-22* (2010), p. 131.

[Sah+14] Durga Prasad Sahoo et al. “Composite PUF: A New Design Paradigm for Physically Unclonable Functions on FPGA”. In: *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. Arlington, VA, USA: IEEE, May 2014, pp. 50–55. ISBN: 978-1-4799-4112-4 978-1-4799-4114-8. URL: http://ieeexplore.ieee.org/document/6855567/ (visited on 04/03/2019).

[Sah+15] Durga Prasad Sahoo et al. “A Case of Lightweight PUF Constructions: Cryptanalysis and Machine Learning Attacks”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.8 (Aug. 2015), pp. 1334–1343. ISSN: 0278-0070, 1937-4151. URL: http://ieeexplore.ieee.org/document/7131531/ (visited on 04/03/2019).

[Sah+16] Durga Prasad Sahoo et al. “Fault Tolerant Implementations of Delay-Based Physically Unclonable Functions on FPGA”. In: *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. Santa Barbara, CA, USA: IEEE, Aug. 2016, pp. 87–101. ISBN: 978-1-5090-1108-7. URL: http://ieeexplore.ieee.org/document/7774485/ (visited on 04/03/2019).

[San+19] Pranesh Santikellur et al. “A Computationally Efficient Tensor Regression Network Based Modeling Attack on XOR APUF”. In: *2019 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. Dec. 2019, pp. 1–6.

[SBC19] Pranesh Santikellur, Aritra Bhattacharyay, and Rajat Subhra Chakraborty. “Deep Learning Based Model Building Attacks on Arbiter PUF Compositions”. In: (2019), p. 10.

[SD07] G. Edward Suh and Srinivas Devadas. “Physical Unclonable Functions for Device Authentication and Secret Key Generation”. In: *Proceedings of the 44th Annual Design Automation Conference*. DAC ’07. New York, NY, USA: ACM, 2007, pp. 9–14. ISBN: 978-1-59593-627-1. URL: http://doi.acm.org/10.1145/1278480.1278484 (visited on 09/19/2018).

[Sid+19] Akhilesh Anilkumar Siddhanti et al. “Analysis of the Strict Avalanche Criterion in Variants of Arbiter-Based Physically Unclonable Functions”. In: *Progress in Cryptology – INDOCRYPT 2019*. Ed. by Feng Hao, Sushmita Ruj, and Sourav Sen Gupta. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 556–577. ISBN: 978-3-030-35423-7.

[Söl09] Jan Sölter. “Cryptanalysis of Electrical PUFs via Machine Learning Algorithms”. M.Sc. Thesis. München: Technische Universität München, 2009. URL: https://www.researchgate.net/profile/Jan_Soelter/publication/259580784_Cryptanalysis_of_electrical_PUFs_via_machine_learning_algorithms/links/00b4952cc03621836c000000.pdf.

[SS06] Eric Simpson and Patrick Schaumont. “Offline Hardware/Software Authentication for Reconfigurable Platforms”. In: *Cryptographic Hardware and Embedded Systems - CHES 2006*. Ed. by David Hutchison et al. Vol. 4249. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 311–323. ISBN: 978-3-540-46559-1 978-3-540-46561-4. URL: http://link.springer.com/10.1007/11894063_25 (visited on 10/22/2021).

[TAB21] Johannes Tobisch, Anita Aghaie, and Georg T. Becker. “Combining Optimization Objectives: New Modeling Attacks on Strong PUFs”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (Feb. 2021), pp. 357–389. ISSN: 2569-2925. URL: https://tches.iacr.org/index.php/TCHES/article/view/8798 (visited on 03/01/2021).

[Taj+14] Shahin Tajik et al. “Physical Characterization of Arbiter PUFs”. In: *Advanced Information Systems Engineering*. Ed. by David Hutchison et al. Vol. 7908. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 493–509. ISBN: 978-3-642-38708-1 978-3-642-38709-8. URL: http://link.springer.com/10.1007/978-3-662-44709-3_27 (visited on 04/11/2019).

[Taj+15] Shahin Tajik et al. “Laser Fault Attack on Physically Unclonable Functions”. In: *2015 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. Sept. 2015, pp. 85–96.

[TB15] Johannes Tobisch and Georg T. Becker. “On the Scaling of Machine Learning Attacks on PUFs with Application to Noise Bifurcation”. In: *International Workshop on Radio Frequency Identification: Security and Privacy Issues*. Springer, 2015, pp. 17–31.

[Tyu10] Il’ya S Tyurin. “Refinement of the Upper Bounds of the Constants in Lyapunov’s Theorem”. In: *Russian Mathematical Surveys* 65.3 (Sept. 2010), pp. 586–588. ISSN: 0036-0279, 1468-4829. URL: http://stacks.iop.org/0036-0279/65/i=3/a=L09?key=crossref.ec669802876a5dabacaa7709cadd97a8 (visited on 11/01/2021).

[UTB16] Christine Utz, Johannes Tobisch, and Georg T Becker. “Extended Abstract: Analysis of 1000 Arbiter PUF Based RFID Tags”. In: (2016), p. 5.

[Val84] L. G. Valiant. “A Theory of the Learnable”. In: *Communications of the ACM* 27.11 (Nov. 1984), pp. 1134–1142. ISSN: 0001-0782. URL: https://doi.org/10.1145/1968.1972 (visited on 07/02/2021).

[Wis+19] Nils Wisiol et al. *Breaking the Lightweight Secure PUF: Understanding the Relation of Input Transformations and Machine Learning Resistance*. Tech. rep. 799. 2019. URL: https://eprint.iacr.org/2019/799 (visited on 10/11/2019).

[Wis+20a] Nils Wisiol et al. “Breaking the Lightweight Secure PUF: Understanding the Relation of Input Transformations and Machine Learning Resistance”. In: *Smart Card Research and Advanced Applications*. Ed. by Sonia Belaïd and Tim Güneysu. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 40–54. ISBN: 978-3-030-42068-0.

[Wis+20b] Nils Wisiol et al. “Splitting the Interpose PUF: A Novel Modeling Attack Strategy”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (June 2020), pp. 97–120. ISSN: 2569-2925. URL: https://tches.iacr.org/index.php/TCHES/article/view/8584 (visited on 09/01/2020).

[Wis+21a] Nils Wisiol et al. *Neural-Network-Based Modeling Attacks on XOR Arbiter PUFs Revisited*. Tech. rep. 555. 2021. URL: https://eprint.iacr.org/2021/555 (visited on 04/28/2021).

[Wis+21b] Nils Wisiol et al. *Pypuf*. Zenodo. June 2021. URL: https://zenodo.org/record/3901410 (visited on 07/07/2021).

[Wis21] Nils Wisiol. *Towards Attack Resilient Arbiter PUF-Based Strong PUFs*. Tech. rep. 1004. 2021. URL: https://eprint.iacr.org/2021/1004 (visited on 08/19/2021).

[WM18] Nils Wisiol and Marian Margraf. *Attacking RO-PUFs with Enhanced Challenge-Response Pairs*. Tech. rep. 862. 2018. URL: https://eprint.iacr.org/2018/862 (visited on 10/11/2019).

[WM19] Nils Wisiol and Marian Margraf. "Why Attackers Lose: Design and Security Analysis of Arbitrarily Large XOR Arbiter PUFs". In: *Journal of Cryptographic Engineering* 9.3 (Sept. 2019), pp. 221–230. ISSN: 2190-8516. URL: https://doi.org/10.1007/s13389-019-00204-8 (visited on 10/11/2019).

[WP20] Nils Wisiol and Niklas Pirnay. "Short Paper: XOR Arbiter PUFs Have Systematic Response Bias". In: *Financial Cryptography and Data Security*. Ed. by Joseph Bonneau and Nadia Heninger. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 50–57. ISBN: 978-3-030-51280-4.

[Yam+14] Dai Yamamoto et al. "Security Evaluation of Bistable Ring PUFs on FPGAs Using Differential and Linear Analysis". In: *2014 Federated Conference on Computer Science and Information Systems*. Sept. 2014, pp. 911–918. URL: https://fedcsis.org/proceedings/2014/drp/122.html (visited on 09/20/2019).

[Yas+16] Risa Yashiro et al. "Deep-Learning-Based Security Evaluation on Authentication Systems Using Arbiter PUF and Its Variants". In: *Advances in Information and Computer Security*. Ed. by Kazuto Ogawa and Katsunari Yoshioka. Vol. 9836. Cham: Springer International Publishing, 2016, pp. 267–285. ISBN: 978-3-319-44523-6 978-3-319-44524-3. URL: http://link.springer.com/10.1007/978-3-319-44524-3_16 (visited on 10/01/2018).

[Yu+14] Meng-Day Yu et al. "A Noise Bifurcation Architecture for Linear Additive Physical Functions". In: *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. May 2014, pp. 124–129.

[Yu+16] Meng-Day Yu et al. "A Lockdown Technique to Prevent Machine Learning on PUFs for Lightweight Authentication". In: *IEEE Transactions on Multi-Scale Computing Systems* 2.3 (July 2016), pp. 146–159. ISSN: 2332-7766. URL: http://ieeexplore.ieee.org/document/7450665/ (visited on 04/09/2019).

[ZPK17] Chen Zhou, Keshab K. Parhi, and Chris H. Kim. "Secure and Reliable XOR Arbiter PUF Design: An Experimental Study Based on 1 Trillion Challenge Response Pair Measurements". In: *Proceedings of the 54th Annual Design Automation Conference 2017 on - DAC '17*. Austin, TX, USA: ACM Press, 2017, pp. 1–6. ISBN: 978-1-4503-4927-7. URL: http://dl.acm.org/citation.cfm?doid=3061639.3062315 (visited on 01/31/2019).

A. Arbiter PUF Additive Delay Model

Theorem. *For $n \in \mathbb{N}$ and given stage delay values $d_i^{\mathrm{TT}}, d_i^{\mathrm{TB}}, d_i^{\mathrm{BT}}, d_i^{\mathrm{BB}} \in \mathbb{R}^+$, with $1 \leq i \leq n$, there exists $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$ such that for all $c \in \{-1,1\}^n$,it holds that*

$$\Delta D_{\mathrm{Model}}(c) = \langle w, x \rangle + b,$$

where $x = (x_i)_i = \left(\prod_{j=i}^n c_j\right)_i$. For even n we have

$$w = \frac{1}{2}\begin{pmatrix} d_1^{\mathrm{TT}} - d_1^{\mathrm{BB}} - d_1^{\mathrm{BT}} + d_1^{\mathrm{TB}} \\ -d_2^{\mathrm{TT}} + d_2^{\mathrm{BB}} + d_2^{\mathrm{BT}} - d_2^{\mathrm{TB}} + d_1^{\mathrm{TT}} - d_1^{\mathrm{BB}} + d_1^{\mathrm{BT}} - d_1^{\mathrm{TB}} \\ d_3^{\mathrm{TT}} - d_3^{\mathrm{BB}} - d_3^{\mathrm{BT}} + d_3^{\mathrm{TB}} - d_2^{\mathrm{TT}} + d_2^{\mathrm{BB}} - d_2^{\mathrm{BT}} + d_2^{\mathrm{TB}} \\ \vdots \\ -d_{n-2}^{\mathrm{TT}} + d_{n-2}^{\mathrm{BB}} + d_{n-2}^{\mathrm{BT}} - d_{n-2}^{\mathrm{TB}} + d_{n-3}^{\mathrm{TT}} - d_{n-3}^{\mathrm{BB}} + d_{n-3}^{\mathrm{BT}} - d_{n-3}^{\mathrm{TB}} \\ d_{n-1}^{\mathrm{TT}} - d_{n-1}^{\mathrm{BB}} - d_{n-1}^{\mathrm{BT}} + d_{n-1}^{\mathrm{TB}} - d_{n-2}^{\mathrm{TT}} + d_{n-2}^{\mathrm{BB}} - d_{n-2}^{\mathrm{BT}} + d_{n-2}^{\mathrm{TB}} \\ -d_n^{\mathrm{TT}} + d_n^{\mathrm{BB}} + d_n^{\mathrm{BT}} - d_n^{\mathrm{TB}} + d_{n-1}^{\mathrm{TT}} - d_{n-1}^{\mathrm{BB}} + d_{n-1}^{\mathrm{BT}} - d_{n-1}^{\mathrm{TB}} \end{pmatrix},$$

$$b = 1/2\left(d_n^{\mathrm{TT}} - d_n^{\mathrm{BB}} + d_n^{\mathrm{BT}} - d_n^{\mathrm{TB}}\right);$$

Similar formulae exist for odd n.

Proof. The proof is by induction over i. For $i = 1$ we have

$$\Delta D_1 = d_1^{\mathrm{T}} - d_1^{\mathrm{B}} = \begin{cases} d_1^{\mathrm{TT}} - d_1^{\mathrm{BB}} & (c_1 = -1 \iff 1/2 - 1/2 c_1 = 1), \\ d_1^{\mathrm{BT}} - d_1^{\mathrm{TB}} & (c_1 = 1 \iff 1/2 - 1/2 c_1 = 0). \end{cases}$$

Using the fact that $1/2 - 1/2 c_1 \in \{0, 1\}$, we can write

$$\begin{aligned} \Delta D_1 &= (1/2 - 1/2 c_1)\left(d_1^{\mathrm{TT}} - d_1^{\mathrm{BB}}\right) + (1 - (1/2 - 1/2 c_1))\left(d_1^{\mathrm{BT}} - d_1^{\mathrm{TB}}\right) \\ &= c_1 \cdot \underbrace{1/2\left(d_1^{\mathrm{BB}} - d_1^{\mathrm{TT}} + d_1^{\mathrm{BT}} - d_1^{\mathrm{TB}}\right)}_{w_1^{(1)}} + \underbrace{1/2\left(d_1^{\mathrm{TT}} - d_1^{\mathrm{BB}} + d_1^{\mathrm{BT}} - d_1^{\mathrm{TB}}\right)}_{b^{(1)}} \\ &= \left\langle x^{(1)}, w^{(1)} \right\rangle + b^{(1)} \end{aligned}$$

where $c^{(1)} = \left(c_1^{(1)}\right) \in \{-1,1\}^1$, $x^{(1)} = \left(\prod_{j=i}^1 c_i^{(1)}\right)_i = \left(x_1^{(1)}\right) = (c_1)$ and $w^{(1)} = \left(w_1^{(1)}\right) \in \mathbb{R}^1$.

Assuming $\Delta D_{i-1} = \left\langle x^{(i-1)}, w^{(i-1)} \right\rangle + b^{(i-1)}$, we have

$$\begin{aligned} \Delta D_i = d_i^{\mathrm{T}} - d_i^{\mathrm{B}} &= \begin{cases} d_{i-1}^{\mathrm{T}} + d_i^{\mathrm{TT}} - d_{i-1}^{\mathrm{B}} - d_i^{\mathrm{BB}} & (c_i = -1 \iff 1/2 - 1/2 c_i = 1), \\ d_{i-1}^{\mathrm{B}} + d_i^{\mathrm{BT}} - d_{i-1}^{\mathrm{T}} - d_i^{\mathrm{TB}} & (c_i = 1 \iff 1/2 - 1/2 c_i = 0), \end{cases} \\ &= \begin{cases} \Delta D_{i-1} + d_i^{\mathrm{TT}} - d_i^{\mathrm{BB}} & (c_i = -1 \iff 1/2 - 1/2 c_i = 1), \\ -\Delta D_{i-1} + d_i^{\mathrm{BT}} - d_i^{\mathrm{TB}} & (c_i = 1 \iff 1/2 - 1/2 c_i = 0), \end{cases} \end{aligned}$$

Using the fact that $1/2 - 1/2 c_i \in \{0, 1\}$, we can write

$$\begin{aligned}
\Delta D_i &= (1/2 - 1/2 c_i)\left(\Delta D_{i-1} + d_i^{\mathrm{TT}} - d_i^{\mathrm{BB}}\right) + (1/2 + 1/2 c_i)\left(-\Delta D_{i-1} + d_i^{\mathrm{BT}} - d_i^{\mathrm{TB}}\right) \\
&= c_i \cdot 1/2 \left(\underbrace{-d_i^{\mathrm{TT}} + d_i^{\mathrm{BB}} + d_i^{\mathrm{BT}} - d_i^{\mathrm{TB}}}_{2\hat{w}^{(i)}} - 2\Delta D_{n-1} \right) - \underbrace{1/2\left(d_i^{\mathrm{TT}} - d_i^{\mathrm{BB}} + d_i^{\mathrm{BT}} - d_i^{\mathrm{TB}}\right)}_{\hat{b}^{(i)}} \\
&= c_i \cdot \left(\hat{w}^{(i)} - \Delta D_{i-1}\right) + \hat{b}^{(i)} \\
&= c_i \cdot \left(\hat{w}^{(i)} - \left\langle x^{(i-1)}, w^{(i-1)} \right\rangle - b^{(i-1)}\right) + \hat{b}^{(i)} \\
&= c_i \cdot \left(\hat{w}^{(i)} - \left(\sum_{l=1}^{i-1} w_l^{(i-1)} \prod_{j=l}^{i-1} c_j\right) - b^{(i-1)}\right) + \hat{b}^{(i)} \\
&= \left(\sum_{l=1}^{i-1} -w_l^{(i-1)} \prod_{j=l}^{i} c_j\right) + \left(\hat{w}^{(i)} - b^{(i-1)}\right) c_i + \hat{b}^{(i)}.
\end{aligned}$$

Setting $\mathbb{R}^i \ni w^{(i)} = \left(-w_1^{(i-1)}, \ldots, -w_{i-1}^{(i-1)}, \hat{w}^{(i)} - b^{(i-1)}\right)$ and $b^{(i)} = \hat{b}^{(i)}$ we continue

$$\Delta D_i = \sum_{l=1}^{i} w_l^{(i)} \prod_{j=l}^{i} c_j + \hat{b}^{(i)} = \left\langle w^{(i)}, x^{(i)} \right\rangle + b^{(i)}.$$

Finally, for $i = n$, we set $w = w^{(n)}, x = x^{(n)}$, and $b = b^{(n)}$ and conclude that

$$\Delta D_{\mathrm{Model}} = \Delta D_n = \langle w, x \rangle + b.$$

Hence, an Arbiter PUF with positive delays $d_i^{\mathrm{TT}}, d_i^{\mathrm{TB}}, d_i^{\mathrm{BT}}, d_i^{\mathrm{BB}}$ for each stage i can be modeled as claimed. Formulae for w and b can be obtained by recursively plugging $w^{(i-1)}$ into the definition of $w^{(i)}$. □

B. Permutation PUF

The permutations shown here are the first 10 permutations on vectors of length 64 among the ones generated by the numpy `RandomState` object using seeds starting from 0xbad, i.e.

- `RandomState(0xbad).permutation(64)`,
- `RandomState(0xbae).permutation(64)`,
- `RandomState(0xbaf).permutation(64)`,
- ...

that meet the criteria stated in Section 4.3, where `RandomState` is from numpy 1.3.13. The tables were generated with the code below.

```
from numpy.random import RandomState

seeds = [2989, 2992, 3038, 3084, 3457, 6200, 7089,
          18369, 21540, 44106]
parts = [(0, 16), (16,32), (32,48), (48, 64)]
print(r'\begin{tabular}{c|' + 'c'*64 + '}')
for a, b in parts:
    if a != 0:
        print(r'\hline\hline')
    print(
        r'  ' + r'\(i\)&' +
        '&'.join(
            [
                r'\(x^{(i)}_{%i}\)' % (i+1)
                for i in range(a, b)
            ]
        ) + r'\\ \hline')
    for l, seed in enumerate(seeds):
        print('  ' + '%i&' % (l+1) + '&'.join(
            [
                r'\(c_{%i}\)' % (i+1) for i in
                RandomState(seed).permutation(64)[a:b]
            ]) + r'\\')
print(r'\end{tabular}')
```

i	$c_1^{(i)}$	$c_2^{(i)}$	$c_3^{(i)}$	$c_4^{(i)}$	$c_5^{(i)}$	$c_6^{(i)}$	$c_7^{(i)}$	$c_8^{(i)}$	$c_9^{(i)}$	$c_{10}^{(i)}$	$c_{11}^{(i)}$	$c_{12}^{(i)}$	$c_{13}^{(i)}$	$c_{14}^{(i)}$	$c_{15}^{(i)}$	$c_{16}^{(i)}$
1	c_{45}	c_{12}	c_{54}	c_{31}	c_{11}	c_{40}	c_{47}	c_{58}	c_{22}	c_{24}	c_6	c_8	c_5	c_3	c_{41}	c_{53}
2	c_{29}	c_{56}	c_{28}	c_{32}	c_{53}	c_{34}	c_{25}	c_{27}	c_{61}	c_{26}	c_{33}	c_{57}	c_{50}	c_{30}	c_{35}	c_{44}
3	c_{26}	c_{55}	c_{44}	c_{28}	c_6	c_{39}	c_{20}	c_{30}	c_{23}	c_7	c_{63}	c_{56}	c_{27}	c_{31}	c_{11}	c_{37}
4	c_{59}	c_5	c_{63}	c_{61}	c_{29}	c_{12}	c_{37}	c_{32}	c_6	c_{44}	c_{34}	c_{42}	c_{45}	c_{38}	c_{14}	c_{31}
5	c_{52}	c_{29}	c_{16}	c_9	c_{10}	c_{18}	c_{64}	c_{33}	c_{38}	c_{43}	c_{22}	c_{19}	c_{25}	c_{62}	c_{53}	c_{50}
6	c_7	c_{28}	c_{35}	c_{53}	c_{43}	c_{45}	c_{34}	c_4	c_{19}	c_{50}	c_{54}	c_{11}	c_{15}	c_{29}	c_{20}	c_{12}
7	c_{61}	c_{31}	c_1	c_{22}	c_{63}	c_{14}	c_{49}	c_{47}	c_{55}	c_{51}	c_{58}	c_{48}	c_{62}	c_{39}	c_{25}	c_{21}
8	c_{16}	c_{19}	c_{22}	c_{41}	c_{27}	c_{28}	c_{21}	c_{23}	c_{11}	c_{25}	c_{17}	c_4	c_{44}	c_{26}	c_6	c_{35}
9	c_9	c_{49}	c_{51}	c_{52}	c_{42}	c_{46}	c_{19}	c_1	c_{29}	c_{18}	c_2	c_{10}	c_{20}	c_{55}	c_{47}	c_7
10	c_{43}	c_4	c_{45}	c_{38}	c_{51}	c_{19}	c_6	c_{35}	c_{46}	c_{54}	c_8	c_{47}	c_{34}	c_{52}	c_{61}	c_{63}

i	$c_{17}^{(i)}$	$c_{18}^{(i)}$	$c_{19}^{(i)}$	$c_{20}^{(i)}$	$c_{21}^{(i)}$	$c_{22}^{(i)}$	$c_{23}^{(i)}$	$c_{24}^{(i)}$	$c_{25}^{(i)}$	$c_{26}^{(i)}$	$c_{27}^{(i)}$	$c_{28}^{(i)}$	$c_{29}^{(i)}$	$c_{30}^{(i)}$	$c_{31}^{(i)}$	$c_{32}^{(i)}$
1	c_{16}	c_{62}	c_{57}	c_{27}	c_9	c_{52}	c_{18}	c_{51}	c_{50}	c_{14}	c_{32}	c_{15}	c_{56}	c_{46}	c_{44}	c_{23}
2	c_9	c_{54}	c_{39}	c_{12}	c_4	c_{10}	c_{15}	c_{60}	c_{42}	c_6	c_7	c_{21}	c_{31}	c_{19}	c_{18}	c_{38}
3	c_4	c_{13}	c_{34}	c_9	c_{40}	c_{48}	c_{54}	c_{62}	c_{60}	c_{18}	c_{24}	c_{10}	c_1	c_{25}	c_{35}	c_{36}
4	c_{49}	c_{16}	c_{43}	c_{19}	c_{22}	c_8	c_{40}	c_4	c_{41}	c_{10}	c_2	c_{51}	c_{11}	c_{33}	c_{58}	c_{20}
5	c_{35}	c_{57}	c_{41}	c_{42}	c_{17}	c_3	c_{56}	c_{27}	c_1	c_{47}	c_{49}	c_{14}	c_{30}	c_{58}	c_5	c_4
6	c_3	c_{10}	c_{60}	c_{62}	c_{47}	c_{31}	c_{64}	c_{33}	c_{37}	c_{23}	c_{39}	c_{30}	c_{25}	c_{41}	c_{49}	c_{51}
7	c_{23}	c_{26}	c_9	c_{44}	c_{12}	c_{27}	c_{50}	c_{10}	c_{19}	c_{40}	c_{59}	c_{18}	c_8	c_{42}	c_{43}	c_{54}
8	c_{56}	c_{40}	c_3	c_{49}	c_7	c_{60}	c_{36}	c_{57}	c_{62}	c_{42}	c_{52}	c_{45}	c_{32}	c_{55}	c_{20}	c_{47}
9	c_{28}	c_{59}	c_{62}	c_{13}	c_{48}	c_{23}	c_{14}	c_{38}	c_{53}	c_{60}	c_5	c_4	c_{34}	c_{40}	c_{50}	c_3
10	c_{31}	c_{17}	c_{30}	c_{18}	c_{39}	c_9	c_{42}	c_{40}	c_{13}	c_{15}	c_{14}	c_{48}	c_{12}	c_{23}	c_{10}	c_{59}

i	$c_{33}^{(i)}$	$c_{34}^{(i)}$	$c_{35}^{(i)}$	$c_{36}^{(i)}$	$c_{37}^{(i)}$	$c_{38}^{(i)}$	$c_{39}^{(i)}$	$c_{40}^{(i)}$	$c_{41}^{(i)}$	$c_{42}^{(i)}$	$c_{43}^{(i)}$	$c_{44}^{(i)}$	$c_{45}^{(i)}$	$c_{46}^{(i)}$	$c_{47}^{(i)}$	$c_{48}^{(i)}$
1	c_{39}	c_{49}	c_{34}	c_{63}	c_{61}	c_{60}	c_{64}	c_{55}	c_{29}	c_{30}	c_{19}	c_{25}	c_{43}	c_{33}	c_{37}	c_{26}
2	c_1	c_{51}	c_{23}	c_{64}	c_{63}	c_5	c_{37}	c_{14}	c_{62}	c_2	c_{48}	c_{17}	c_{55}	c_{41}	c_{11}	c_{22}
3	c_{61}	c_8	c_{41}	c_{49}	c_{16}	c_{15}	c_{53}	c_{33}	c_{57}	c_{45}	c_2	c_{12}	c_{50}	c_3	c_{59}	c_5
4	c_{47}	c_{30}	c_{52}	c_{13}	c_3	c_{54}	c_{48}	c_{18}	c_{26}	c_{25}	c_{62}	c_{35}	c_{60}	c_{21}	c_{55}	c_{46}
5	c_{60}	c_{45}	c_{11}	c_{26}	c_{44}	c_{40}	c_{36}	c_7	c_2	c_{28}	c_{32}	c_{21}	c_{24}	c_{48}	c_6	c_{13}
6	c_9	c_{52}	c_{48}	c_6	c_{59}	c_{63}	c_{55}	c_{61}	c_5	c_{32}	c_{14}	c_{58}	c_{46}	c_2	c_{56}	c_{36}
7	c_{57}	c_{13}	c_{60}	c_{37}	c_{34}	c_{64}	c_4	c_{46}	c_{52}	c_{15}	c_{17}	c_{20}	c_{38}	c_{45}	c_2	c_3
8	c_{48}	c_{14}	c_{39}	c_{24}	c_{31}	c_{50}	c_{12}	c_{29}	c_{37}	c_{53}	c_9	c_{43}	c_{10}	c_{58}	c_{63}	c_2
9	c_{24}	c_6	c_{56}	c_{54}	c_{33}	c_{21}	c_{17}	c_{43}	c_{35}	c_{58}	c_{15}	c_{27}	c_{31}	c_{12}	c_{16}	c_{63}
10	c_{58}	c_{21}	c_{57}	c_{25}	c_{27}	c_{53}	c_{26}	c_{36}	c_{32}	c_{33}	c_{60}	c_1	c_{11}	c_{62}	c_{50}	c_{20}

i	$c_{49}^{(i)}$	$c_{50}^{(i)}$	$c_{51}^{(i)}$	$c_{52}^{(i)}$	$c_{53}^{(i)}$	$c_{54}^{(i)}$	$c_{55}^{(i)}$	$c_{56}^{(i)}$	$c_{57}^{(i)}$	$c_{58}^{(i)}$	$c_{59}^{(i)}$	$c_{60}^{(i)}$	$c_{61}^{(i)}$	$c_{62}^{(i)}$	$c_{63}^{(i)}$	$c_{64}^{(i)}$
1	c_{28}	c_{38}	c_{59}	c_{17}	c_2	c_{20}	c_1	c_{42}	c_4	c_{48}	c_{13}	c_{10}	c_{21}	c_{35}	c_{36}	c_7
2	c_{46}	c_{47}	c_{24}	c_{40}	c_{13}	c_{43}	c_{58}	c_8	c_{49}	c_{20}	c_3	c_{52}	c_{36}	c_{16}	c_{59}	c_{45}
3	c_{42}	c_{14}	c_{21}	c_{19}	c_{29}	c_{17}	c_{47}	c_{64}	c_{51}	c_{52}	c_{43}	c_{46}	c_{22}	c_{58}	c_{32}	c_{38}
4	c_7	c_{28}	c_{50}	c_{57}	c_{64}	c_{36}	c_9	c_{23}	c_{24}	c_{27}	c_{15}	c_{53}	c_{56}	c_{17}	c_1	c_{39}
5	c_{59}	c_{37}	c_{12}	c_{55}	c_{39}	c_{23}	c_{34}	c_{63}	c_{20}	c_{54}	c_{61}	c_{31}	c_8	c_{51}	c_{46}	c_{15}
6	c_{38}	c_{17}	c_{27}	c_{16}	c_1	c_{42}	c_{24}	c_{18}	c_8	c_{57}	c_{44}	c_{13}	c_{26}	c_{40}	c_{22}	c_{21}
7	c_{24}	c_{35}	c_{33}	c_{11}	c_{36}	c_{56}	c_7	c_{28}	c_{41}	c_{32}	c_{29}	c_{30}	c_{16}	c_{53}	c_6	c_5
8	c_{33}	c_8	c_{18}	c_{13}	c_{51}	c_{59}	c_{64}	c_{46}	c_{30}	c_1	c_{34}	c_5	c_{54}	c_{38}	c_{15}	c_{61}
9	c_{61}	c_{30}	c_{57}	c_8	c_{26}	c_{11}	c_{39}	c_{22}	c_{44}	c_{36}	c_{45}	c_{37}	c_{32}	c_{41}	c_{64}	c_{25}
10	c_3	c_{64}	c_{41}	c_{22}	c_{37}	c_{28}	c_{29}	c_{44}	c_7	c_{56}	c_{49}	c_{16}	c_{24}	c_5	c_2	c_{55}

Table B.1.: Permutation as used in the 64-bit Permutation PUF.