J Cryptol (2023) 36:35
https://doi.org/10.1007/s00145-023-09475-1

Journal of
CRYPTOLOGY

Research Article

A Theoretical Framework for the Analysis of Physical Unclonable Function Interfaces and Its Relation to the Random Oracle Model*

Marten van Dijk
CWI, Amsterdam, The Netherlands
Department of Computer Science, Vrije Universiteit van Amsterdam, Amsterdam, The Netherlands
Electrical and Computer Engineering Department, University of Connecticut, Storrs, CT, USA
marten.van.dijk@cwi.nl

Chenglu Jin
CWI, Amsterdam, The Netherlands
chenglu.jin@cwi.nl

Communicated by Svetla Nikova.

Received 22 November 2022 / Revised 30 June 2023 / Accepted 30 June 2023

Abstract. Analysis of advanced physical unclonable function (PUF) applications and protocols relies on assuming that a PUF behaves like a random oracle; that is, upon receiving a challenge, a uniform random response with replacement is selected, measurement noise is added, and the resulting response is returned. In order to justify such an assumption, we need to rely on digital interface computation that to some extent remains confidential—otherwise, information about PUF challenge–response pairs leak with which the adversary can train a prediction model for the PUF. We introduce a theoretical framework that allows the adversary to have a prediction model (with a typical accuracy of 75% for predicting response bits for state-of-the-art silicon PUF designs). We do not require any confidential digital computing or digital secrets, while we can still prove rigorous statements about the bit security of a system that interfaces with the PUF. In particular, we prove the bit security of a PUF-based random oracle construction; this merges the PUF framework with fuzzy extractors.

Keywords. Physical unclonable function (PUF), Fuzzy extractor, Random oracle, Trusted computing base (TCB), PUF interfaces.

1. Introduction

A physical unclonable function (PUF) is a device that takes a challenge as input and measures a corresponding response bit as output [1,2]. Responses depend on manufacturing variations in the PUF that are practically unclonable with currently existing technology.

*This paper was reviewed by Frederik Armknecht.

Published online: 03 August 2023

Nevertheless, a PUF's behavior may be modeled by training a prediction model based on a set of challenge–response pairs (CRPs). For this reason, a PUF design can be broken if an attacker achieves a significant accuracy of a trained prediction model.[1]

Since physical unclonable functions have been introduced as a security primitive [1,2], a variety of applications have been proposed [5–7], including many advanced cryptographic protocols, e.g., key agreement, oblivious transfer, and bit commitment [8–10]. The security analysis of these advanced applications and protocols[2] relies on assuming that a PUF behaves like a random oracle; upon receiving a challenge, a uniform random response with replacement is selected, measurement noise is added, and the resulting response is returned. This assumption turns out to be too strong because (1) in practical implementations, the PUF returns biased response bits, and (2) classical ML and advanced ML attacks [11–17] demonstrate that a prediction model for response bits with accuracy typically up to 75% can be trained and this defeats the random oracle assumption. For example, FPGA implementations of the interpose PUF in [18] showed that the bias of individual Arbiter PUFs ranges from 50.2% to 61.6%. The highest prediction accuracy on interpose PUF entities under the best-known attacks by then was around 75% given 200,000 training challenge–response pairs. Although a follow-up work [16] proposed an attack that can improve the prediction accuracy on iPUF by means of an iterative approach, the prediction accuracy of the first iteration is still not higher than 75%.

To counter the response bit bias problem, the literature introduces a PUF interface that implements a fuzzy extractor (FE) [19–23]. Upon sufficient min-entropy in response vectors, random (unbiased) bit strings can be extracted using a FE. To counter the accurate training of a prediction model by the attacker, we eliminate access to challenge–response pairs by the attacker. In other words, we have a trusted computing base (TCB) that implements the PUF together with a FE interface isolated from the attacker—it assumes that the interface computes in a confidential digital computing environment (confidential TCB).

The above solution is satisfactory if we use a weak PUF that only has a few CRPs for masking/obfuscating a secret key based on a single response vector. (We want to re-measure responses whenever we want access to the de-obfuscated key—for this, we already need a confidential TCB.) The FE generates and publishes so-called helper information p, which is needed to extract a random bit string from the measured response vector with which the secret is masked. This helper information does leak some information about the response vector—after all, we use FE because the response vector does not have full min-entropy (i.e., it is not uniformly distributed over bit vectors). If we only publish one or a couple of p, then it is realistic to assume that this does not help the adversary gain sufficient information about challenge–response pairs for training an accurate prediction model.

[1] Public PUFs [3] and SIMPL systems [4] which base their security on the time differences between physical execution and model simulation is out of the scope of the paper and is not captured by our definitional framework and analysis. They do not provide similar security properties as conventional PUFs, so they should be treated as different types of security primitives.

[2] PUF identification and authentication only rely on hypothesis testing based on comparing collected CRPs with re-measured CRPs.

On the other hand, if, for other applications, a strong PUF is used with an 'exponentially large' challenge space, then many helper data p is published, and in theory, this can help the adversary in gathering statistical information about CRPs and train a prediction model (even though, in practice, we have no idea how to accomplish this). The strong PUF with FE interface still needs the confidential TCB in order to make it impossible for the adversary to observe processed CRPs directly. (Otherwise, just based on these observed CRPs, a prediction model can be trained.)

We notice that the computational FE based on the LPN problem in [24,25] also publishes helper data, but here it can be proven that this data does not reveal underlying information about CRPs.[3] (In fact, the computational FE is used to implement a random oracle based on a weak PUF with just one response vector.) But also here, the LPN interface is in a confidential TCB. (Its digital computation is not allowed to be observed by the adversary.)

This paper introduces a new framework for rigorously reasoning about the security of PUF interfaces. We get rid of the confidential TCB and allow the adversary access to a training set of challenge–response pairs. Only the way how these pairs can be adaptively selected is restricted. We take a pre-challenge as input to a cryptographic hash function to generate a challenge for the PUF[4]; this is the only way the PUF may get accessed by both legitimate users and adversaries, and no confidential digital computing is required. We construct and analyze the bit security of a PUF-based random oracle as a main example/demonstration of our theoretical framework.

Our main motivation for getting rid of the confidential TCB of a PUF interface is, first of all, of a more philosophical nature: In a more general context, we have an overarching system that queries a PUF and wants to base its security guarantees on (random) bit strings extracted from the PUF's responses. Some form of confidential computing of the system's functionality is required as its security guarantees will generally depend on keeping the PUF derived bit strings confidential. Since calling a bit-string a 'secret key' does not actually make it secret [26], such a system generally implements key renewal strategies for which the PUF is queried again. Here, the system relies on using the PUF with an interface to again generate fresh secret bit strings even though previous digital secrets have been leaked to the adversary. If the PUF interface itself relies on confidential digital computation in order to be able to keep on generating fresh secret bit strings, then the adversary will recognize the PUF interface as a weak link and an attractive point of attack. Rather than defending the confidentiality of computing of the PUF interface by means of a hardware design that isolates the PUF with the interface from the adversary so that no point of attack exists, is it possible to minimize the TCB and not require any form of confidential digital computing in the PUF interface and as a consequence not require any secret digital keys or digital information that needs to be kept secret from the adversary? This question of minimizing the TCB by instead relying

[3] Also, the LPN construction does not suffer a min-entropy loss due to the leftover hash lemma as in FE.

[4] We assume the hash function interface cannot be circumvented by the adversaries, and the hash function is correctly computed on pre-challenges. Note that this assumption is much weaker than the assumption of having any confidential TCB, as any information in the hash function interface is public. Also, it is not very hard to guarantee the integrity of the hash function interface in practice; we just need to implement it in hardware circuitry as long as the adversaries do not tamper with the circuitry or inject faults in the computation, which is usually costly and requires extensive physical access.

on certain computational hardness assumptions is at the heart of security research. This paper shows that this can be done (at least in theory) for a PUF interface that corrects measurement errors and extracts random bit strings. In order to accomplish this, we need to build a new theoretical framework (language) for capturing the exact computational assumptions that replace the assumption of a confidential TCB.

In future work, we will show how verifiable computing can be based on such a PUF interface (a first blueprint toward this goal is given in [27]): Here, a client outsources computing to a single compute node. (We do not consider outsourcing computing over multiple compute nodes in order to implement a Byzantine fault-tolerant scheme which allows a third of the used compute nodes to be adversarial.) Suppose that the compute node can be trusted to execute the compute job inside an environment that is protected from having an adversary tamper with its computing flow. That is, the adversary cannot violate the specified or expected behavior of the compute job. Even if the final computed result is correct, it needs to be communicated back to the client. This means that the compute node must engage in a remote attestation protocol with the client and be able to sign the computed result using a secret digital key. In [28], a one-time session key-based signature scheme (coined OTS-SKE) is presented, which in combination with our proposed PUF-based random oracle (used for masking all session keys) can offer remote attestation with the following property: Even of all but one session signing key is leaked, then a signature for the session of which the session key is not leaked cannot be impersonated, and other new signatures for older sessions can also not be impersonated. (The latter property is tricky and requires the features of the OTS-SKE scheme.) Based on the theory presented in this paper, we can show that to accomplish this security guarantee, no confidential TCB is needed for the PUF interface or signing. (Signing uses a session key extracted from memory whose content is masked by our PUF-based random oracle.) This shows that remote attestation, and by extension, verifiable computing, does not need to rely on confidential digital computing in that previous session keys and other digital information leaked to the adversary cannot be used to impersonate a signature in the current session or impersonate new signatures for older (observed) sessions. This will show for the first time how PUFs can be used to bootstrap such verifiable computation without confidential TCB.

The main problem that we solve is how to connect security definitions for PUFs to (computational) hardness problems on which PUF interfaces (such as FE) are based. Our framework aims at strong PUFs with an 'exponentially large' challenge space.

- We define a PUF device in Sect. 3 followed by an extended PUF interface GetResponse that first applies a cryptographic hash to a pre-challenge. We introduce the concept of (canonical) system-induced CRP distribution, where a system interfaces with the PUF and only uses CRPs of its 'liking,' i.e., have a 'nice distribution.'
- We define reliability and bias with respect to system-induced CRP distributions in Sect. 4. Conditioned on previously collected CRPs, the bias of a new CRP may change due to correlation. We characterize the amount of change by $\epsilon_{corbias}$ and show how $\epsilon_{\mathsf{corbias}}$ gets amplified due to post-processing of CRPs (Lemma 6).
- In Sect. 5, we show an interface that improves reliability by using repeated measurements, and we analyze $\epsilon_{\mathsf{corbias}}$ of the resulting system-induced CRP distribution. Similarly, in Sect. 6, we show an interface based on the von Neumann extractor

for reducing bias [29]. We show how resulting response bits behave as unbiased uniformly drawn bits in Lemma 11 and, as a consequence, explain a condition in (5) which allows us to replace the von Neumann system-induced CRP distribution by a 'uniform' one in a future reduction proof.

- We define PUF security with correlated CRPs in Sect. 7 and define the adversarial $\mathcal{A}^U$-model, which does not require a confidential TCB (i.e., we do not require any confidential digital computing or digital secrets), and only requires the adversary to access the PUF through GetResponse. We prove the 'Ber transformation lemma' (Lemma 14) which states that a (prediction) error-reducing oracle can be constructed, leading to error bits that are statistically independent and Bernoulli distributed. The bit error rate is essentially equal to one minus the accuracy of the best prediction model the adversary can construct (based on limited resources, the number of collected CRPs, and run time).
- Section 8 defines system security where the system interface has access to the PUF. We define a separation game and argue this is, at most, an exponential factor more difficult than the original system security game. We provide a number of definitions of properties of the underlying hardness problem. These definitions lead to the 'PUF separation theorem' in the $\mathcal{A}^U$-model (Theorem 22) where PUF assumptions and mathematical hardness assumptions are separated, still leading to a bit security of the overall system. We discuss a range of weaker adversarial models $\mathcal{A}^x \subseteq \mathcal{A}^U$ in Sect. 9.
- In order to merge the concept of fuzzy extractors with our framework, we introduce 'suitable' codes and discuss and prove properties about their related residual min-entropy in Sect. 10. This is used in Sect. 11 to construct a PUF-based random oracle (PRO). We characterize failure probabilities and analyze the security using Theorem 22. In order to prove some of the needed properties of the underlying hardness problem, we show how the von Neumann system-induced distribution can be replaced by a uniform one, how the Ber transformation lemma can be used to construct a problem instance without needing access to the PUF, and how the hardness of the resulting problem is related to residual min-entropy (as in secure sketches but now related to Bernoulli noise). This results in the final 'PUF-based random oracle theorem' in the $\mathcal{A}^U$-model (Theorem 28).

The final PRO primitive justifies how a PUF can be used to simulate a random oracle, as explained at the start of the introduction, even in the presence of an adversary who is able to achieve a typical accuracy of a prediction model of 75%, and even if no confidential TCB (i.e., no confidential digital computing and no digital secrets) is assumed. The latter allows PRO to execute in the presence of an adversary who can observe all digital computation and digital secrets. PRO only requires PUF access control through GetResponse. Our results can be easily plugged into the analysis of PUF-based protocols, like key exchange [8], oblivious transfer [8,9], bit commitment [30], and multi-party computation [31], where PUFs are all assumed to be random oracles. The presented work closes a major gap in the current PUF literature (Table 1).

Table 1. Index of all definitions, lemmas, and theorems.

PUF—Intrinsic properties	
Definition 1	CRPs and hardware unclonability of PUFs
Definition 2	System-induced CRP distributions
Definition 3	PUF reliability
Definition 4	PUF bias
Definition 5	PUF correlation ($\epsilon_{corbias}$), which can be assumed to be exponentially small for Arbiter-based PUF designs
Lemmas 6, 7, 8	Effect of composition of system-induced CRP distributions on correlation
Lemmas 9, 10	Characterization of bias and the improved reliability as a result of majority voting
Lemma 11	Characterization of the reduced bias as a result of applying the von Neumann trick
Figure 1	Diagram relating all concepts
PUF—security	
Definition 12	PUF security game with correlations inspired by [32]
Definition 13	Adversarial model
Lemma 14, 15	Ber transformation lemma
System security	
Definition 16	System security game where the system interfaces with and queries a PUF
Definition 17	Separation game where the adversary first predicts responses and next solves the system's instance of a computational hard problem
Definitions 18, 19, 20, 21	Error-based reduction; Bit security; Error-based equivalent; Effect of an error-reducing oracle
Theorem 22	PUF separation theorem
Figure 2	Diagram explaining the flow of the security reduction leading to the PUF separation theorem
PUF-based Random Oracle (PRO)	
Definitions 23, 24	Secure sketch; Suitable codes
Lemma 25, 26	Upper bound on the residual min-entropy
Definition 27	PRO correctness and bit security
Theorem 28, Lemma 29	Construction

2. Related Work

Existing PUF definitional frameworks. Since the introduction of PUFs, many attempts have been made to formally define PUFs. Most of the existing PUF definitional frameworks oversimplified the reality and omitted the fact that real PUFs produce errors in their responses due to environmental/measurement noises [32–34]. Rührmair et al. [33] first partitioned PUFs into weak PUFs and strong PUFs based on the sizes of their challenge spaces, and then, they defined strong PUFs as physical random functions that produce perfectly reliable outcomes and cannot be physically or mathematically cloned within a certain amount of time given to the adversary. Jin et al. [32,34] extended the framework to include stateful erasable PUFs and stateful programmable access-controlled PUFs, where the stateful PUFs can keep an internal state and alter CRPs based on its internal state and certain policies. However, in the above definitions, PUFs are always assumed to be noise-free with help from some error-correcting mechanisms. Our framework takes

noises into account and precisely discusses how the noises/biases will affect the security of the PUFs.

Noisy PUF behaviors are modeled in [8,35,36]. Brzuska et al. defined PUFs as a noisy random function whose error rate for any given challenge–response pair is within a noise bound [8]. However, the definition did not capture the bias presented in PUF responses. Armknecht et al. briefly discussed a PUF definition in [35] and further extended it in [36]. The definitions in [35] and [36] captured both physical properties and algorithmic properties of PUFs, including the reliability of PUFs. However, the definitions also oversimplified the reality and assumed no bias or correlations in PUF responses.

Fuzzy extractors. Fuzzy extractors are used to extract a reliable and uniformly distributed output from a noisy and biased output, e.g., biometrics [19] and PUF responses [20,21,23]. All existing fuzzy extractor studies focus on improving their capability of error correction and the min-entropy left in the final output of the fuzzy extractors, assuming the distribution/bias of the PUF responses are known to the adversary and that there is no or only spatial correlation between responses. These assumptions effectively constrained the fuzzy extractor theory to be applied to only weak PUFs rigorously. In our work, we consider a much stronger adversary who has a prediction model with a meaningful prediction accuracy, e.g., 75% accuracy of a one-bit PUF output. This is a realistic issue for strong PUFs under modeling attacks; even though some strong PUFs are claimed to be secure against certain attacks, the adversary can still build a prediction model of the PUF with a meaningful accuracy better than random guessing using the concerned attacks [11,18]. Our work effectively closes the gap and provides a solid foundation for using fuzzy extractors on strong PUFs securely.

Existence of strong PUFs. In this work, we are mainly interested in conventional strong PUFs whose challenge space is exponentially large with respect to the physical size of the PUFs [33]. However, if one wants to use a weak PUF in our interface, one needs to assume a confidential computing environment, where no leakage is allowed directly from the weak PUF to the adversary. This security assumption is not ideal when we want to eliminate (or minimize) the confidential computing environment for stronger security.

Given the recent development in the lightweight strong PUF area, the existence of strong PUFs may be deemed unclear. For example, XOR Arbiter PUFs [6] have been considered as a standard lightweight strong PUF design, until they were broken by reliability-based attacks [15]. The introduction of interpose PUF (iPUF) showed new hope for realizing a practical lightweight strong PUF that is secure against both classical modeling attacks and reliability-based attacks [18]. However, the security of iPUFs has been proven to be weaker than the authors originally thought in novel attacks [16,17,37]. Although the existence of a secure lightweight strong PUF design is still unclear, our framework is still needed as soon as (just like in designs for symmetric key encryption) a strong PUF design survives for a significant number of years. Indeed, strong PUF design is still an active research area and many new designs show great potential in defending against known attacks [38].

We notice that Sect. 9 describes a taxonomy of various adversarial models. One setting is about a system executing a series of ‘sessions’ where the adversary observes all but one session and where the security guarantee is about the unobserved session. This fits the remote attestation protocol example explained in the introduction where a re-

mote adversary can obtain a footprint observing digital computation and digitally stored values. Proper implementation will not allow the adversary to enforce repeated measurements and therefore the adversary cannot obtain reliability information. This adversarial model, denoted by $\mathcal{A}^{NR}$ in Sect. 9, restricts the adversary to 'classical' CRP-based ML attacks rather than 'advanced' challenge–reliability pair-based ML attacks. This allows us to still be able to use the XOR Arbiter PUF design in the remote attestation example. However, as discussed in Sect. 9, the amount of training data dictates the effectiveness of the used ML attack and in the remote attestation setting this forces the use of a XOR Arbiter PUF with a number of Arbiter PUFs that degrades reliability too much. It is clear that different adversarial models allow different types of strong PUF designs, and there is the ongoing research question of finding strong PUF designs with better security reliability trade-offs.

3. Physical Unclonable Functions

In this section, we formally define a PUF and introduce an extended PUF functionality (which is a PUF with a small interface). In the next sections, we define reliability, bias, and security.

Definition 1. (*Physical Unclonable Functions* [32]) A PUF $\mathcal{P}$ is a physical system that can be stimulated with so-called challenges c_i from a challenge set $C_{\mathcal{P}} = \{0,1\}^{\lambda}$, upon which it reacts by producing corresponding responses r_i from a response set $R_{\mathcal{P}} \subseteq \{0,1\}^{m}$. Each response r_i shall depend on the applied challenge, but also on manufacturing variations in $\mathcal{P}$ that are practically unclonable with currently existing technology. The tuples (c_i, r_i) are called the challenge–response pairs (CRPs) of $\mathcal{P}$. We often refer to λ as the security parameter of the PUF. □

This definition explicitly mentions that a hardware copy or clone of a PUF $\mathcal{P}$ cannot be manufactured due to uncontrollable manufacturing variations which provide the randomness from which responses are extracted. This leaves in question whether, rather than hardware cloning $\mathcal{P}$, a software simulator, which sufficiently accurately predicts responses, can be constructed and learned. Here, we assume that the adversary has access to $\mathcal{P}$ and can use $\mathcal{P}$ as an oracle to query a list of challenge–response pairs which are used to train a simulator in the form of a machine learning model which predicts responses given input challenges.

The querying of $\mathcal{P}$ can be adaptive and this can possibly be exploited by the adversary. For example, comparing responses of neighboring challenges that have a small Hamming distance may reveal detailed information about small subsets of manufacturing variations in the PUF design. In order to eliminate this possibility in practice, we process challenges by applying a one-way function before giving it as input to the PUF circuitry where the manufacturing variations are used to extract response bits. This leads to the extended PUF discussed next.

Extended PUF. We consider an extended PUF design, called GetResponse in Algorithm 1, which first applies a cryptographic hash function (which implies one-wayness,

see Appendix A) to an input c_{pre} which we call a pre-challenge. The output of the hash function serves as the input challenge c to a PUF $\mathcal{P}$. The extended PUF functionality returns $\mathcal{P}$'s response. The input–output pair (c_{pre}, r) of GetResponse can be used to compute a CRP

$$(c = \mathsf{Hash}(c_{\mathsf{pre}}), r).$$

The extended PUF design describes a small PUF interface that cannot be circumvented by the adversary; *we assume the interface is immutable with respect to the adversary*; hence, the adversary cannot freely choose the processed challenges that are input to $\mathcal{P}$. It can observe all intermediate digital computations of the PUF interface and see (or compute itself) the processed challenges that lead to CRPs for training a machine learning model.

The adversary cannot exert 'fine-grained' control over the output of the hash function. Therefore, from a practical perspective, the adversary cannot adaptively choose challenges whose hashes are designed to be equal or even close in Hamming distance. This property is not formally implied by requiring the hash to be collision-resistant, but it is used in related literature like Controlled PUFs [39]. We will formalize the security of GetResponse (a combination of Hash and PUF $\mathcal{P}$) as a whole in later sections—and argue that from a practical perspective, the best-known methods for predicting a silicon PUF's behavior is by using machine learning techniques based on CRPs that were generated without searching for pre-challenges that lead to hash evaluations that are close[5] in Hamming distance, but just use randomly[6] chosen pre-challenges that lead to random challenges.

Algorithm 1 Extended PUF interface

```
1: procedure GetResponse(c_pre)
2:     c = Hash(c_pre)
3:     r ← P(c)
4:     return r
5: end procedure
```

CRP distributions from the legitimate user/system perspective. We will analyze the security of a system that calls PUF $\mathcal{P}$ through GetResponse. We will assume that such a system will always call GetResponse for either (1) a new true random pre-challenge c_{pre} or (2) a previously selected pre-challenge c_{pre}. (This yields multiple measurements of the same response bit and, as we will explain, can be used to enhance reliability.) Selecting new pre-challenges according to a uniform distribution turns out to be important in our

[5] We can enforce a large Hamming distance between challenges if we encode $\mathsf{Hash}(c_{\mathsf{pre}})$ into a code word c by using a binary error-correcting code with large enough minimum Hamming distance. Code word c serves as the challenge for PUF $\mathcal{P}$.

[6] We can enforce this strategy by simply modeling Hash as a hash function in the random oracle model. But we wish to avoid such a strong assumption since it is generally not true in theory (even though it is usually considered true in practice).

security analysis (for proving a reduction step). For this reason we define the canonical distribution of GetResponse that corresponds to a system calling GetResponse for new true random pre-challenges:

Definition 2. (*System-induced canonical CRP distribution*) By $\mathcal{Y}^*$ we define the distribution of CRPs generated by GetResponse when called by a system $\mathcal{S}$ for new pre-challenges:

$$\mathsf{Pr}((c, r) \leftarrow \mathcal{Y}^*) = \mathsf{Pr}(c \leftarrow \mathcal{Y})\mathsf{Pr}(r \leftarrow \mathcal{P}(c) \mid c).$$

We call $\mathcal{Y}^*$ the *system-induced canonical CRP distribution* and $\mathcal{Y}$ the *system-induced canonical challenge distribution*. □

We notice that $\mathcal{Y}$ describes the distribution of challenges as a result of applying Hash to pre-challenges that are generated by system $\mathcal{S}$ according to some probabilistic algorithm. In practice we consider systems $\mathcal{S}$ that either call GetResponse for new pre-challenges c_{pre} that are uniform random or repeat GetResponse for a previously selected pre-challenge. In this context $\mathcal{Y}$ is the distribution of the output of Hash over uniform distributed input.

In practice, a system $\mathcal{S}$ does not want to generate a true random pre-challenge for every new GetResponse call. Instead, pre-challenges will be generated in batches using a formula of the kind

$$c_{\mathsf{pre},u} = \mathsf{Hash}(seed \| u), \tag{1}$$

where $seed$ is a true random bit string and u is simply an n_u-bit index representing some integer in $\{0, \ldots, 2^{n_u} - 1\}$. Since different batches of pre-challenges use their own truly random $seed$, these batches are statistically independent. Within a batch, since Hash is collision-resistant and therefore one-way, we may assume in practice that the different $c_{\mathsf{pre},u}$ are 'independent' and 'random.' The set of corresponding challenges generated by GetResponse,

$$\{\mathsf{Hash}(c_{\mathsf{pre},u})\}_{u=0}^{2^{n_u}-1},$$

defines some 'distribution' $\mathcal{Y}$.

We notice that from a cryptographic perspective it is possible to provide a solid provable pre-challenge generation scheme that leads to uniform $\mathcal{Y}$. A cryptographically secure solution is to use a pseudorandom generator (PRG) based on, e.g., the subset iterate construction using a hash from a hash function family that has cryptographic hashes [40]. Based on a true random bit string $seed$, $\mathsf{PRG}(seed)$ can be efficiently computed and the resulting bit sequence can be split into a sequence of 2^{n_u} pre-challenges. The resulting distribution of challenges is computationally indistinguishable from a uniform distribution over $\{0, 1\}^\lambda$ with respect to some security parameter and related advantage. By choosing appropriate parameters, the resulting distribution over challenges is ϵ-close in statistical distance to the uniform distribution for some ϵ exponentially small in the security parameter. By using the method in [41] we can replace the resulting distribution

over challenges by the uniform distribution in our security analysis of the system's security guarantee.[7]

We do not detail the PRG construction here but simply assume that in practice the Hash-based solution uses a Hash chosen according to international accepted standard already yields a system-induced canonical challenge distribution $\mathcal{Y}$ at least somewhat close (will be detailed later) to a uniform distribution over $\{0,1\}^\lambda$ (in the context of, e.g., the statistical distance). For ease readability, the reader may interpret next lemmas and theory by simply assuming that $\mathcal{Y}$ is the uniform distribution over the challenge space $\{0,1\}^\lambda$, i.e., $\mathsf{Pr}(c \leftarrow \mathcal{Y}) = 1/2^\lambda$ for $C_\mathcal{P} = \{0,1\}^\lambda$. Nevertheless, we stress that Definition 2 is sufficiently general to capture arbitrary $\mathcal{Y}$. We will see that the security analysis of our main theorems will not explicitly depend on $\mathcal{Y}$ and hold for a wide range of $\mathcal{Y}$—not just the uniform distribution. In Definition 5 we treat the intrinsic correlation among CRPs generated by GetResponse as a combination of Hash and PUF $\mathcal{P}$ for system-induced challenge distributions.

Our framework in the remainder of the paper is formulated for general system-induced canonical challenge distributions $\mathcal{Y}$. We notice that a system $\mathcal{S}$ may use the system-induced canonical distribution $\mathcal{Y}^*$ over CRPs, to extract another distribution over CRPs. In the setting above, corresponding to each batch, system $\mathcal{S}$ may decide to only use a *subset* of challenges generated by GetResponse,

$$\{\mathsf{Hash}(c_{\mathsf{pre},u})\}_{u \in U},$$

for some $U \subseteq \{0,\ldots,2^{n_u}-1\}$. For example, by using repeated measurements, only 'reliable' CRPs corresponding to the challenges indexed by U are selected. This will correspond to another (non-canonical) system-induced distribution over CRPs since $r \leftarrow \mathcal{P}(c)$ for reliable challenges c is less vulnerable to measurement noise, hence, most of the time the same response r is output by $\mathcal{P}(c)$. For this reason, the next definitions will be for arbitrary CRP distributions $\mathcal{Y}^*$ and we always say *system-induced distribution* to indicate that in our context the definitions are from the system's perspective, i.e., from how the system uses GetResponse (and not how the adversary uses GetResponse to find an accurate prediction model for PUF $\mathcal{P}$).

Intrinsic PUF properties. With respect to system-induced probability distributions, we define in Sect. 4 intrinsic PUF properties: reliability, bias, and correlation as depicted in Fig. 1. In Sect. 5, we analyze how majority voting can improve reliability, that is, majority voting creates a system-induced CRP distribution $\mathcal{Y}^*$ which corresponds to a higher reliability $\mathbb{E}_{\mathcal{Y}^*}[p_c]$. In Sect. 6, we analyze how by adding the von Neumann trick we reduce bias, that is, the von Neumann trick creates a system-induced CRP distribution $\mathcal{Y}^*$ which corresponds to a bias $q_r^{\mathcal{Y}^*}$ that is closer to the uniform distribution over $r \in R_\mathcal{P}$. The reason for the resulting distribution to not be exactly the same as the uniform distribution is because of correlation among CRP pairs produced by GetResponse. In Arbiter PUF-based designs, the probability that the responses of such CRPs correlate depends on whether the corresponding challenges are close in Hamming distance for

[7] Notice that the PRG-based solution is more computational intensive compared to the simpler Hash-based solution since reconstructing $\mathsf{Hash}(c_{\mathsf{pre},u})$ needs reconstruction of the whole PRG sequence up to and including $c_{\mathsf{pre},u}$.

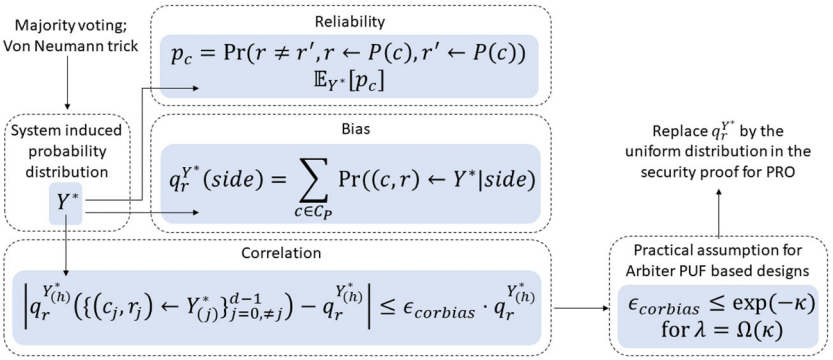

Fig. 1. Intrinsic PUF properties.

the system-induced CRP distribution. Since we use distributions that choose random pre-challenges, we will argue that the probability of having a close Hamming distance is exponentially small in λ. We will use this to show that the bias gets statistically close to uniform. We can prove that the distribution of vectors of response bits are close to uniform in the sense that $\epsilon_{\text{corbias}}$ is exponentially small in κ if $\lambda = \Omega(\kappa)$. In our security proof for the PUF-based random oracle construction we can use this to replace the actual distribution of vectors of response bits (that may not exactly come from a uniform distribution) by the uniform distribution. This will allow us to properly complete the security proof and show κ-bit security of the PRO construction.

4. Reliability and Bias

Reliability. The GetResponse interface does not fully eliminate an adaptive attack. Even though we argued that the adversary cannot adaptively choose distinct challenges whose hashes are designed to fit a priori defined properties, the adversary can still repeat the same challenge. Since PUF $\mathcal{P}$ is a physical device, it suffers from measurement noise (due to temperature and voltage variations, and aging). This means that repeating the same input challenge to $\mathcal{P}$ (the most basic form of an adaptive attack) can result in different response bits. This allows an adversary to measure the 'reliability' according to the next definition and construct challenge–reliability pairs rather than challenge–response pairs. Since the reliability of a response bit teaches more information compared to a single sample of a response bit, this can lead to more advanced and more efficient machine learning of a PUF simulator. So, reliability information helps the adversary.

Definition 3. (*PUF Reliability*) Due to measurement noise, the responses to the same challenge may not always be the same. We define the *failure probability* p_c of PUF $\mathcal{P}$ with respect to a challenge $c \in C_{\mathcal{P}}$ as

$$p_c = \mathsf{Pr}(r \neq r', r \leftarrow \mathcal{P}(c), r' \leftarrow \mathcal{P}(c)),$$

where $r, r' \in R_{\mathcal{P}}$ and the probability is over measurement noise.

From the legitimate user/system perspective, we define the *reliability* of $\mathcal{P}$ with respect to c as $1 - p_c$. Let $\mathcal{Y}$ be a system-induced challenge distribution. The *overall reliability* of $\mathcal{P}$ with respect to $\mathcal{Y}$ is defined as $1 - \delta$ with $\delta = \mathbb{E}_{\mathcal{Y}}[p_c]$ where the expectation is over $c \leftarrow \mathcal{Y}$. □

In Arbiter PUF [2] like designs such as the iPUF [18] or XOR Arbiter PUF [6], two stimuli race against each other following complementary paths indicated by a challenge c. Which stimulus arrives first determines the response bit $r \in \{0, 1\}$. The difference in arrival times of the stimuli is modeled as a difference in aggregated delays that characterize each of the two paths. Without measurement noise, this is a deterministic function. With measurement noise, the arrival times may vary, and as a result, the response bit flips. Let p_c be the probability that the response bit flips due to measurement noise, given a selected challenge c. Different challenges indicate different complementary paths over which stimuli race against each other. And for this reason, the p_c are generally different for different c. If we assume that challenge c is selected uniformly from the challenge space, denoted by $\{0, 1\}^\lambda$, then this gives rise to a distribution of p_c with respect to the uniform distribution of c over the challenge space $\{0, 1\}^\lambda$. So, if r is a first measurement, r' a second measurement, and $e = r + r'$ (XOR operation) represents the error between the two, then the probability $r \neq r'$ is equal to

$$\delta = \mathsf{Pr}[e = 1] = \frac{1}{2^\lambda} \sum_{c \in \{0,1\}^\lambda} p_c = \mathbb{E}_{\mathcal{Y}}[p_c], \tag{2}$$

where the probability is over uniformly selected c, which defines distribution $\mathcal{Y}$ (in this example). We denote this probability by δ and assume $\delta \leq 1/2$. In practice, we have $\delta \leq 10\%$. (In general, temperature variations, voltage variations, and aging keep δ below 10% [42].)

Bias. In practice, PUF produces biased responses, due to systematic design (architectural) biases [43,44] or manufactured biases [45]. This leads to system-induced distributions over CRPs that experience bias. The next definition formalizes this concept.

Definition 4. (*PUF Bias*) Let $\mathcal{Y}^*$ be a system-induced distribution over challenge–response pairs $C_{\mathcal{P}} \times R_{\mathcal{P}}$ defined as a ppt algorithm with oracle access to PUF $\mathcal{P}$. We may project $\mathcal{Y}^*$ to a distribution $\mathcal{Y}$ over the challenge space $C_{\mathcal{P}}$ by defining

$$\mathsf{Pr}(c \leftarrow \mathcal{Y}) = \sum_{r \in R_{\mathcal{P}}} \mathsf{Pr}((c, r) \leftarrow \mathcal{Y}^*).$$

We define *bias* $q_r^{\mathcal{Y}^*}$ of PUF $\mathcal{P}$ with respect to a response $r \in R_{\mathcal{P}}$ as

$$q_r^{\mathcal{Y}^*} = \sum_{c \in C_{\mathcal{P}}} \mathsf{Pr}((c, r) \leftarrow \mathcal{Y}^*),$$

where the probability is over measurement noise and $c \leftarrow \mathcal{Y}$. We may also introduce the knowledge of side information in the form of *other known CRPs* that affects the bias

(because it is correlated with $\mathcal{Y}$ and $\mathcal{Y}^*$):

$$q_r^{\mathcal{Y}^*}(side) = \sum_{c \in C_{\mathcal{P}}} \mathsf{Pr}((c, r) \leftarrow \mathcal{Y}^* \mid side).$$

The *bias* of $\mathcal{P}$ with respect to $\mathcal{Y}$ and side information $side$ is defined as

$$q^{\mathcal{Y}}(side) = \max_{r \in R_{\mathcal{P}}} q_r^{\mathcal{Y}}(side).$$

By $q^{\mathcal{Y}}$, we denote the bias for empty side information $side$. □

From a legitimate user/system's perspective, we want high overall reliability and a bias close to $1/|R_{\mathcal{P}}|$ such that responses (corresponding to $c \leftarrow \mathcal{Y}$) used by the system have the most information content. Notice that if the bias is close to 1, then the PUF always generates the same response regardless of the input challenge. Hence, the PUF becomes predictable. In order to use a PUF for the purpose of identification, authentication, key masking, etc., it needs to be unpredictable, that is, a bias sufficiently close to $1/|R_{\mathcal{P}}|$.

Correlation. In order to model correlation among CRPs, we need a definition that takes a distribution over multiple challenges into account and considers the correlation between their responses. As an example in Sect. 5, this allows us to reason about how the bias is affected if one uses only 'reliable' CRPs where reliable CRPs are extracted by a (simple) interface with access to GetResponse.

Definition 5. (*PUF Correlation*) Let $\Upsilon^{\mathcal{P}}$ be a set of system-induced distributions over challenge–response pairs. Let λ be the security parameter of PUF $\mathcal{P}$, that is, $C_{\mathcal{P}} = \{0,1\}^{\lambda}$. Suppose that there exists an $\epsilon_{corbias} \geq 0$ such that for all $d \geq 1$ with $d = poly(\lambda)$, for all distributions $\mathcal{Y}^*_{(j)} \in \Upsilon^{\mathcal{P}}$, $0 \leq j \leq d-1$, over challenge–response pairs $C_{\mathcal{P}} \times R_{\mathcal{P}}$, for all $h \in \{0, \ldots, d-1\}$,

$$\left| q_r^{\mathcal{Y}^*_{(h)}} \left(\{(c_j, r_j) \leftarrow \mathcal{Y}^*_{(j)}\}_{j=0, \neq h}^{d-1} \right) - q_r^{\mathcal{Y}^*_{(h)}} \right| \leq \epsilon_{\mathsf{corbias}} \cdot q_r^{\mathcal{Y}^*_{(h)}},$$

where $\{(c_j, r_j) \leftarrow \mathcal{Y}^*_{(j)}\}_{j=0, \neq h}^{d-1}$ reflects knowledge of explicit CRP values drawn from the other distributions $\mathcal{Y}^*_{(j)}$, $j \neq h$. Then we say that PUF $\mathcal{P}$ has correlation bias at most $\epsilon_{\mathsf{corbias}}$ over the set of distributions $\Upsilon^{\mathcal{P}}$. □

In this definition, we talk about using the PUF from a legitimate user's perspective. Here, a system may want to query the PUF multiple (d) times and want to use the outputted response to, e.g., extract some keys. It is important to know whether the outputted response bits are correlated or whether they can be assumed more or less statistically independent. The latter is often assumed even if, e.g., only 'reliable challenge–response pairs' are used by the system (corresponding to a specific distribution $\mathcal{Y}^*$).

The definition given above does not yet model security; that is, there is no adversarial algorithm trying to use 'intelligence' to predict responses. This will be discussed in Sect. 7. Here, we model the amount of correlation among CRPs due to the intrinsic

properties of the PUF itself. Consider the case where two CRPs are independently chosen according to the *canonical* system-induced $\mathcal{Y}^*$, that is, $(c_0, r_0) \leftarrow \mathcal{Y}^*$ and $(c_1, r_1) \leftarrow \mathcal{Y}^*$ where $\mathcal{Y}$ is the distribution of the output of Hash over uniform pre-challenges. It is well known that Arbiter PUF-based designs produce response bits that can be considered statistically independent for challenges that have sufficiently large Hamming distance between one another. That is, if c_0 and c_1 happen to be close in Hamming distance, then values r_0 and r_1 are correlated. If the Hamming distance between c_0 and c_1 is large, then r_0 and r_1 behave as statistically independent variables. Challenges $c_0 \leftarrow \mathcal{Y}$ and $c_1 \leftarrow \mathcal{Y}$ are chosen close to uniform from $C_{\mathcal{P}} = \{0,1\}^{\lambda}$ (see our previous discussion) and here we make this more precise: For large λ, a perfect uniform distribution generally leads to challenges c_0 and c_1 that have large Hamming distance and only with probability exponentially small in λ the challenges are close enough in Hamming distance to lead to observable correlation among their responses r_0 and r_1. Here, we assume that system-induced canonical challenge distribution $\mathcal{Y}$ is close enough to the uniform distribution in that it also has the property of an exponentially small probability in λ of generating two challenges c_0 and c_1 close enough in Hamming distance leading to observable correlation among their responses r_0 and r_1. We can extend this argument to d CRPs if $d = poly(\lambda)$. Therefore, we may assume that

> $\epsilon_{\mathsf{corbias}}$ decreases in parameter λ and is exponentially small in λ for $\Upsilon^{\mathcal{P}} = \{\mathcal{Y}^*\}$ where $\mathcal{Y}^*$ is the system-induced canonical CRP distribution.

The next sections will show that non-canonical system-induced CRP distributions (used in this paper) have an $\epsilon_{\mathsf{corbias}}$ that scales 'linearly' with the exponentially small $\epsilon_{\mathsf{corbias}}$ for the canonical system-induced CRP distribution. This allows us to conclude that $\epsilon_{\mathsf{corbias}}$ is exponentially small for the whole set $\Upsilon^{\mathcal{P}}$ of system-induced CRP distributions (used in this paper).

Before diving into properties that can be proved based on our definitions so far, let us discuss an example of a set $\Upsilon^{\mathcal{P}}$ of system-induced distributions over challenge–response pairs. Since we assume that system $\mathcal{S}$ can only access the PUF through GetResponse, this means that $\mathcal{S}$ can only query PUF $\mathcal{P}$ by challenges drawn from a canonical system-induced distribution $\mathcal{Y}$ with replacement. That is, $\mathcal{S}$ can repeatedly ask the PUF for responses for the same challenge c. And whenever the PUF is queried for a new challenge, then this challenge must be chosen according to $\mathcal{Y}$. This induces certain types of distributions over the challenge space: If $\mathcal{S}$ asks for a response of the same challenge c exactly h times and observes for some $i \leq h/2$ that $h - i$ have the same response bit r and the other i response bits are the complement $r + 1$ (xor), then it is as if the challenge c with response r is drawn from a distribution over the challenge space $C_{\mathcal{P}} = \{0,1\}^{\lambda}$ which causes either i or $h - i$ measurement errors (each with probability p_c). In other words, $c \leftarrow \mathcal{Y}_{h,i}$ with

$$\mathsf{Pr}(c \leftarrow \mathcal{Y}_{h,i}) = \frac{\binom{h}{i}(p_c^{h-i}(1-p_c)^i + p_c^i(1-p_c)^{h-i}) \cdot \mathsf{Pr}(c \leftarrow \mathcal{Y})}{\sum_{c' \in \{0,1\}^{\lambda}} \binom{h}{i}(p_{c'}^{h-i}(1-p_{c'})^i + p_{c'}^i(1-p_{c'})^{h-i}) \cdot \mathsf{Pr}(c' \leftarrow \mathcal{Y})}.$$

System $\mathcal{S}$ cannot a priori decide to choose $c \leftarrow \mathcal{Y}_{h,i}$, but after its repeated measurements, it will turn out that $c \leftarrow \mathcal{Y}_{h,i}$ for some h and i that match observation. We may

define the response of c to be the majority vote of the observed responses. (Or if there is no majority, 0 or 1 is selected as response with probability $1/2$.) Denote this majority vote by r. Then, this process defines $(c, r) \leftarrow \mathcal{Y}^*_{h,i}$. System $\mathcal{S}$ draws $\mathcal{Y}^*_{h,i}$ from the set

$$\Upsilon^{\mathcal{P}} = \{\mathcal{Y}^*_{h',i'}\}_{h' \geq 1, 0 \leq i' \leq h'/2}$$

according to a probabilistic process. Our security game **SecGameSys** of Definition 16 gives the adversary the exact knowledge of which $\mathcal{Y}^*_{(j)} \in \Upsilon^{\mathcal{P}}$ was selected.

The next lemma shows how Definition 5 can be used to analyze $\epsilon_{\mathsf{corbias}}$ for more general distributions. It is rather general, and we will provide proof in small steps, which themselves gain an understanding of how to use the lemma.

Lemma 6. *Suppose that PUF $\mathcal{P}$ has correlation bias at most $\epsilon_{\mathsf{corbias}}$ over a set of system-induced distributions $\Upsilon^{\mathcal{P}}$ and let λ be the security parameter of $\mathcal{P}$. Consider products $\bar{\mathcal{Y}}^*_{(i)} = \mathcal{Y}^*_{(i,0)} \times \ldots \times \mathcal{Y}^*_{(i,d-1)}$ with $\{\mathcal{Y}^*_{(i,j)}\} \subseteq \Upsilon^{\mathcal{P}}$ and $d = poly(\lambda)$. Let $\mathcal{X}$ be a distribution statistically independent of all $\bar{\mathcal{Y}}^*_{(i)}$. For each i, define a new distribution $(\hat{c}_i, \hat{r}_i) \leftarrow \hat{\mathcal{Y}}^*_{(i)}$ over the CRP space represented by some polynomial algorithm that takes a drawing $(\bar{c}_i, \bar{r}_i) \leftarrow \bar{\mathcal{Y}}^*_{(i)}$ and a drawing $x \leftarrow \mathcal{X}$ as input. Let $side$ be a random variable statistically independent of $x \leftarrow \mathcal{X}$ such that*

$$(\bar{r}_h, \{(\bar{c}_i, \bar{r}_i)\}^{\hat{d}-1}_{i=0,\neq h}) \rightarrow \{(\bar{c}_i, \bar{r}_i)\}^{\hat{d}-1}_{i=0,\neq h} \rightarrow \{(\hat{c}_i, \hat{r}_i)\}^{\hat{d}-1}_{i=0,\neq h} \rightarrow side$$

is a Markov chain with $\hat{d} = poly(\lambda)$. Then, for all $\hat{r}$, $q^{\hat{\mathcal{Y}}^*_{(h)}}_{\hat{r}}(side)$ satisfies[8]

$$\left| q^{\hat{\mathcal{Y}}^*_{(h)}}_{\hat{r}}(side) - q^{\hat{\mathcal{Y}}^*_{(h)}}_{\hat{r}} \right| \leq ((1 + \epsilon_{\mathsf{corbias}})^d - 1) \cdot q^{\hat{\mathcal{Y}}^*_{(h)}}_{\hat{r}}.$$

□

Definition 5 about the correlation among CRPs is sound in that we can prove natural properties. First, rather than conditioning on knowledge of explicit CRP values $\{(c_j, r_j)\}^{d-1}_{j=0,\neq h}$, we may have partial knowledge about the CRP values in the form of a random variable $side$ which is correlated to $\{(c_j, r_j)\}^{d-1}_{j=0,\neq h}$ but independent from (c_h, r_h) given $\{(c_j, r_j)\}^{d-1}_{j=0,\neq h}$, and we expect to still have the same $\epsilon_{\mathsf{corbias}}$. Second, we may define response vectors of multiple bits by calling GetResponse multiple times; we also want to know the bias of such response vectors, which we expect to be the product of the biases of each of the response bits separately corrected with $\epsilon_{\mathsf{corbias}}$. Third, we may define a new distribution that post-processes random drawings from distributions in $\Upsilon^{\mathcal{P}}$ and we want to characterize the resulting $\epsilon_{\mathsf{corbias}}$.

[8] It is possible to refine Lemma 6 and make d itself a random variable depending on drawings from distributions in $\Upsilon^{\mathcal{P}}$.

Lemma 7. *Suppose that PUF $\mathcal{P}$ has correlation bias at most $\epsilon_{\mathsf{corbias}}$ over a set of system-induced distributions $\Upsilon^{\mathcal{P}}$ and let λ be the security parameter of $\mathcal{P}$. Let $(c_j, r_j) \leftarrow \mathcal{Y}^*_{(j)} \in \Upsilon^{\mathcal{P}}$ be random variables*[9] *and let $side$ be a random variable such that*

$$(r_h, \{(c_j, r_j)\}_{j=0,\neq h}^{d-1}) \rightarrow \{(c_j, r_j)\}_{j=0,\neq h}^{d-1} \rightarrow side$$

is a Markov chain with $d = poly(\lambda)$. Then, in Definition 5 we have for all r,

$$\left| q_r^{\mathcal{Y}^*_{(h)}}(side) - q_r^{\mathcal{Y}^*_{(h)}} \right| \leq \epsilon_{\mathsf{corbias}} \cdot q_r^{\mathcal{Y}^*_{(h)}}.$$

□

Proof of Lemma 7. Let r represent the random variable $(\cdot, r) \leftarrow \mathcal{Y}^*_{(h)}$, that is, $\mathsf{Pr}(r) = \sum_{c \in C_{\mathcal{P}}} \mathsf{Pr}((c, r) \leftarrow \mathcal{Y}^*_{(h)})$. Let y be the random variable $y = \{(c_j, r_j)\}_{j=0,\neq h}^{d-1} \leftarrow \mathcal{Y}^*_{(0)} \times \ldots \times \mathcal{Y}^*_{(h-1)} \times \mathcal{Y}^*_{(h+1)} \times \ldots \times \mathcal{Y}^*_{(d-1)}$. We have $(r, y) \rightarrow y \rightarrow side$. By Definition 5 we have $\mathsf{Pr}(r|y) = \mathsf{Pr}(r) \cdot (1 \pm \epsilon_{\mathsf{corbias}})$. We use these properties to derive

$$\begin{aligned}
\mathsf{Pr}(r \mid side) &= \frac{\mathsf{Pr}(r, side)}{\mathsf{Pr}(side)} = \frac{\sum_y \mathsf{Pr}(r, y, side)}{\sum_y \mathsf{Pr}(y, side)} \\
&= \frac{\sum_y \mathsf{Pr}(y)\mathsf{Pr}(r|y)\mathsf{Pr}(side|r, y)}{\sum_y \mathsf{Pr}(y, side)} \\
&= \frac{\sum_y \mathsf{Pr}(y)\mathsf{Pr}(r)(1 \pm \epsilon_{\mathsf{corbias}})\mathsf{Pr}(side|y)}{\sum_y \mathsf{Pr}(y, side)} \\
&= \frac{\sum_y \mathsf{Pr}(y, side)\mathsf{Pr}(r)(1 \pm \epsilon_{\mathsf{corbias}})}{\sum_y \mathsf{Pr}(y, side)} \\
&= \mathsf{Pr}(r)(1 \pm \epsilon_{\mathsf{corbias}}).
\end{aligned}$$

Notice that the lemma follows from $q_r^{\mathcal{Y}^*_{(h)}}(side) = \mathsf{Pr}(r|side)$ and $q_r^{\mathcal{Y}^*_{(h)}} = \mathsf{Pr}(r)$. □

The next lemma generalizes Lemma 7 to system-induced distributions over vectors of CRPs:

Lemma 8. *Suppose that PUF $\mathcal{P}$ has correlation bias at most $\epsilon_{\mathsf{corbias}}$ over a set of system-induced distributions $\Upsilon^{\mathcal{P}}$ and let λ be the security parameter of $\mathcal{P}$. Consider the product $\bar{\mathcal{Y}}^*_{(i)} = \mathcal{Y}^*_{(i,0)} \times \ldots \times \mathcal{Y}^*_{(i,d-1)}$ with $\{\mathcal{Y}^*_{(i,j)}\} \subseteq \Upsilon^{\mathcal{P}}$ and $d = poly(\lambda)$ which outputs a vector of challenges $\bar{c}_i = (c_{i,0}, \ldots, c_{i,d-1})$ and a vector of responses $\bar{r}_i = (r_{i,0}, \ldots, r_{i,d-1})$. Let $side$ be a random variable such that*

$$(\bar{r}_h, \{(\bar{c}_i, \bar{r}_i)\}_{i=0,\neq h}^{\hat{d}-1}) \rightarrow \{(\bar{c}_i, \bar{r}_i)\}_{i=0,\neq h}^{\hat{d}-1} \rightarrow side$$

[9] By abuse of notation we mean by $\mathsf{Pr}((c, r))$ the probability that the random variables represented by (c, r) realize the values (c, r).

is a Markov chain with $\hat{d} = poly(\lambda)$. *Then,* $q_{\bar{r}}^{\bar{\mathcal{Y}}^*_{(h)}}(side)$ *with* $\bar{r} = (r_0, \ldots, r_{d-1})$ *satisfies*

$$\left| q_{\bar{r}}^{\bar{\mathcal{Y}}^*_{(h)}}(side) - \prod_{j=0}^{d-1} q_{r_j}^{\mathcal{Y}^*_{(h,j)}} \right| \leq ((1+\epsilon_{\mathsf{corbias}})^d - 1) \cdot \prod_{j=0}^{d-1} q_{r_j}^{\mathcal{Y}^*_{(h,j)}}.$$

□

Proof of Lemma 8. The proof of the lemma follows by generalizing the following argument: For $d = 2$, we have $\bar{r}_i = (r_0, r_1)$, and by Lemma 7 we have

$$\begin{aligned}
&q_{\bar{r}}^{\mathcal{Y}^*_{(h,0)} \times \mathcal{Y}^*_{(h,1)}}(side) \\
&= \sum_{c_0, c_1 \in C_{\mathcal{P}}} \mathsf{Pr}((c_0, r_0) \leftarrow \mathcal{Y}^*_{(h,0)}, (c_1, r_1) \leftarrow \mathcal{Y}^*_{(h,1)} \mid side) \\
&= \sum_{c_0 \in C_{\mathcal{P}}} \mathsf{Pr}((c_0, r_0) \leftarrow \mathcal{Y}^*_{(h,0)} \mid side) \\
&\quad \cdot \sum_{c_1 \in C_{\mathcal{P}}} \mathsf{Pr}((c_1, r_1) \leftarrow \mathcal{Y}^*_{(h,1)} \mid (c_0, r_0), side) \\
&= \sum_{c_0 \in C_{\mathcal{P}}} \mathsf{Pr}((c_0, r_0) \leftarrow \mathcal{Y}^*_{(h,0)} \mid side) \cdot q_{r_1}^{\mathcal{Y}^*_{(h,1)}}(1 \pm \epsilon_{\mathsf{corbias}}) \\
&= q_{r_0}^{\mathcal{Y}^*_{(h,0)}}(1 \pm \epsilon_{\mathsf{corbias}}) \cdot q_{r_1}^{\mathcal{Y}^*_{(h,1)}}(1 \pm \epsilon_{\mathsf{corbias}}).
\end{aligned}$$

Generalizing this argument to $d > 2$ yields

$$q_{\bar{r}}^{\bar{\mathcal{Y}}^*_{(h)}}(side) = \prod_{j=0}^{d-1} q_{r_j}^{\mathcal{Y}^*_{(h,j)}} \cdot (1 \pm \epsilon_{\mathsf{corbias}})^d.$$

The lemma follows from

$$1 - (1 - \epsilon_{\mathsf{corbias}})^d \leq (1 + \epsilon_{\mathsf{corbias}})^d - 1.$$

□

Lemma 6 generalizes the previous lemma to system-induced distributions over post-processed vectors of CRPs:

Proof of Lemma 6. We may characterize $\mathsf{Pr}(\hat{r} \leftarrow \hat{\mathcal{Y}}^*_{(h)})$ as a distribution

$$\mathsf{Pr}(\hat{r} \leftarrow \mathcal{G}(r_{h,0}, \ldots, r_{h,d-1}, x), \{r_{h,j} \leftarrow \mathcal{Y}^*_{(h,j)}\}_{j=0}^{d-1}, x \leftarrow \mathcal{X}),$$

where $\mathcal{G}$ is a polynomial time algorithm that represents distribution $\hat{\mathcal{Y}}^*_{(h)}$. In a way $\mathcal{G}$ defines a set of tuples $(r_{h,0}, \ldots, r_{h,d-1}, x)$ that lead to outputting $\hat{r}$. ($\mathcal{G}$ is not probabilistic,

and it uses randomness x in a deterministic way.) Let $\mathcal{G}(x, \hat{r})$ be the set of response bit vectors $(r_{h,0}, \ldots, r_{h,d-1})$ for which $\mathcal{G}$ together with input x outputs $\hat{r}$. Let $p_x = \mathsf{Pr}(x \leftarrow \mathcal{X})$. Then, by using the statistical independence of x and by using the notation of Lemma 8,

$$
\begin{aligned}
& q_{\hat{r}}^{\hat{\mathcal{Y}}^*_{(h)}}(side) \\
&= \mathsf{Pr}(\hat{r} \leftarrow \hat{\mathcal{Y}}^*_{(h)} \mid side) \\
&= \mathsf{Pr}\left(\begin{matrix} (r_{h,0}, \ldots, r_{h,d-1}) \in \mathcal{G}(x, \hat{r}), \\ \{r_{h,j} \leftarrow \mathcal{Y}^*_{(h,j)}\}_{j=0}^{d-1}, x \leftarrow \mathcal{X} \end{matrix} \mid side \right) \\
&= \sum_x p_x \cdot \mathsf{Pr}\left(\begin{matrix} (r_{h,0}, \ldots, r_{h,d-1}) \in \mathcal{G}(x, \hat{r}), \\ \{r_{h,j} \leftarrow \mathcal{Y}^*_{(h,j)}\}_{j=0}^{d-1} \end{matrix} \mid \begin{matrix} x, \\ side \end{matrix} \right) \\
&= \sum_x p_x \cdot \mathsf{Pr}\left(\begin{matrix} \bar{r}_h \in \mathcal{G}(x, \hat{r}), \\ \bar{r}_h \leftarrow \bar{\mathcal{Y}}^*_{(h)} \end{matrix} \mid \begin{matrix} x, \\ side \end{matrix} \right) \\
&= \sum_x p_x \cdot \sum_{\bar{r} \in \mathcal{G}(x,\hat{r})} \mathsf{Pr}\left(\bar{r} \leftarrow \bar{\mathcal{Y}}^*_{(h)} \mid side \right) \\
&= \sum_x p_x \cdot \sum_{\bar{r} \in \mathcal{G}(x,\hat{r})} q_{\bar{r}}^{\bar{\mathcal{Y}}^*_{(h)}}(side).
\end{aligned}
$$

This derivation holds for $side$ and also for the special case where $side$ is empty. By using Lemma 8, this shows that

$$
\begin{aligned}
& \left| q_{\hat{r}}^{\hat{\mathcal{Y}}^*_{(h)}}(side) - q_{\hat{r}}^{\hat{\mathcal{Y}}^*_{(h)}} \right| \\
&\leq \sum_x p_x \cdot \sum_{\bar{r} \in \mathcal{G}(x,\hat{r})} |q_{\bar{r}}^{\bar{\mathcal{Y}}^*_{(h)}}(side) - q_{\bar{r}}^{\bar{\mathcal{Y}}^*_{(h)}}| \\
&\leq ((1 + \epsilon_{\mathsf{corbias}})^d - 1) \cdot \sum_x p_x \cdot \sum_{\bar{r} \in \mathcal{G}(x,\hat{r})} q_{\bar{r}}^{\bar{\mathcal{Y}}^*_{(h)}} \\
&= ((1 + \epsilon_{\mathsf{corbias}})^d - 1) \cdot q_{\hat{r}}^{\hat{\mathcal{Y}}^*_{(h)}}.
\end{aligned}
$$

□

5. Improving Reliability

In Algorithm 2, we present a simple interface that improves reliability. The interface changes a uniform selection from challenges to a selection among challenges that lead to 'reliable responses.'

For $h > 1$, GetReliableCRP$_h$ reduces probability $\delta = \mathbb{E}_{\mathcal{Y}}[p_c]$ as defined in Definition 3. The same measurement $r_j = \textsc{GetResponse}(c_{\mathsf{pre}})$ is repeated h times, and only if

Algorithm 2 Get reliable CRPs

```
1: procedure GetReliableCRP_h
2:     Found =false
3:     while Found =false do
4:         c_pre ←_R C_P
5:         for j ∈ {1, ..., h} do
6:             r_j = GetResponse(c_pre)
7:         end for
8:         if all r_j are equal then
9:             c = Hash(c_pre); r = r_1
10:            Found =true
11:        end if
12:    end while
13:    return (c_pre, r)
14: end procedure
```

all measured responses are equal, the agreed upon response r is returned. The probability that all h measurements agree is equal to $p_c^h + (1 - p_c)^h$ where $c = \mathsf{Hash}(c_{\mathsf{pre}})$. This shows that the while loop will take $(\mathbb{E}_{\mathcal{Y}}[p_c^h + (1 - p_c)^h])^{-1}$ iterations in expectation over the canonical system distribution $c \leftarrow \mathcal{Y}$. For sufficiently small h, this is a small enough number, and the GetReliableCRP$_h$ interface can be used in practice.

In order to avoid using a True Random Number Generator (TRNG) for selecting $c_{\mathsf{pre}} \leftarrow_R C_{\mathcal{P}} = \{0, 1\}^{\lambda}$, we will generate a sequence of pre-challenges $c_{\mathsf{pre},a}$ as defined in (1) (for $u = a$), where a is the iteration count of the while loop. In order to do this, we need to know how to a priori represent a, i.e., we select a fixed n_a-bit representation of a for use in (1). This limits GetReliableCRP$_h$ to at most 2^{n_a} loop iterations. As argued above, we may choose n_a small and still have a low failing probability (the probability that no reliable CRP is found after 2^{n_a} loop iterations).

Seed $seed$ in (1) can be given as input to GetReliableCRP$_h$ and selected at random by the system calling GetReliableCRP$_h$. Since the cryptographic hash function is collision-resistant and one-way, an adversary who observes the outputted c_{pre} cannot extract $seed$ and a. However, knowledge about $seed$ and a would in addition teach the adversary $a - 1$ unreliable CRPs corresponding to $c_{\mathsf{pre},u}$, $0 \leq u \leq a - 1$. If this needs to be avoided (in order to limit the adversary in applicable attack techniques), then the legitimate user/system should discard $seed$ as soon as GetReliableCRP$()_h(seed)$ has been called. (Otherwise $seed$ can be leaked to an adversary, and the adversary can extract a from $seed$ and $c_{\mathsf{pre}} = \mathsf{Hash}(seed \| a)$ by using repeated hash evaluations.)

The outputted r corresponds to a reliable challenge c, that is, a challenge that has demonstrated to give rise to repeated consistent measurements of the response. This means that such challenges lead to increased reliability: If GetResponse(c_{pre}) measures a response r' for $c = \mathsf{Hash}(c_{\mathsf{pre}})$ at a later time, then with probability p_c we have $r \neq r'$ (i.e., $e = 1$). However, for $h > 1$, the challenges are picked from the 'subset of reliable challenges'; that is, c is selected with probability

$$\mathsf{Pr}(c \leftarrow \mathcal{Y}_h) = \frac{(p_c^h + (1 - p_c)^h) \cdot \mathsf{Pr}(c \leftarrow \mathcal{Y})}{\sum_{c' \in \{0,1\}^{\lambda}} (p_{c'}^h + (1 - p_{c'})^h) \cdot \mathsf{Pr}(c' \leftarrow \mathcal{Y})},$$

which defines a new distribution $\mathcal{Y}_h$, where $\mathcal{Y}_1 = \mathcal{Y}$ denotes the canonical system-induced distribution. This implies:

Lemma 9. *Let $\mathcal{Y}_h^*$ be the system-induced distribution that generates CRPs according to* GetReliableCRP$_h$. *Then, the overall reliability with respect to $\mathcal{Y}_h$ is equal to $1-\delta_h$ with*

$$\delta_h = \mathbb{E}_{\mathcal{Y}_h}[p_c] = \frac{\mathbb{E}_{\mathcal{Y}_1}[p_c^{h+1}] + \mathbb{E}_{\mathcal{Y}_1}[(1-p_c)^h p_c]}{\mathbb{E}_{\mathcal{Y}_1}[p_c^h] + \mathbb{E}_{\mathcal{Y}_1}[(1-p_c)^h]}. \tag{3}$$

□

δ_h reduces the original $\delta_1 = \delta$ since the smaller p_c are counted more in the sum because of the larger $(1-p_c)^h$ term. $\delta_1 = \delta$ has a typical value of 10% [42], and we may assume that for relatively small h, GetReliableCRP$_h$ achieves a couple percentage points smaller δ_h [45].

Our argument shows how the legitimate user may want to use an interface that selects reliable CRPs. The adversary can still use the GetResponse interface, which covers the whole CRP space uniformly with respect to $\mathcal{Y}_1 = \mathcal{Y}$. The task of the adversary is to use access to GetResponse (possibly emulating GetReliableCRP$_h$) to learn a model with which she/he can predict the reliable response used by the legitimate user.

The new distribution $\mathcal{Y}_h^*$ yields a new bias $q^{\mathcal{Y}_h^*}$. From a mathematics perspective, we cannot conclude $q^{\mathcal{Y}_h^*} = q^{\mathcal{Y}^*}$ for $\mathcal{Y}^* = \mathcal{Y}_1^*$. However, in practice, there is no reason to assume that reliable challenges according to $\mathcal{Y}_h^*$ will have a different bias.

Lemma 10. *Let $\mathcal{Y}_h^*$ be the system-induced distribution that generates CRPs according to* GetReliableCRP$_h$ *where the while loop iterates at most 2^{n_a} times. (And a failure is returned if no reliable CRP is found after 2^{n_a} iterations.) Suppose that PUF $\mathcal{P}$ has correlation bias at most $\epsilon_{\mathsf{corbias}}$ over distribution $\mathcal{Y}_1^*$. Then, $\mathcal{P}$ has correlation bias at most*

$$\begin{aligned}\epsilon_{\mathsf{corbias},h} &= (1+\epsilon_{\mathsf{corbias}})^{h\cdot 2^{n_a}} - 1 \\ &= h2^{n_a}\epsilon_{\mathsf{corbias}} + O((h2^{n_a}\epsilon_{\mathsf{corbias}})^2)\end{aligned}$$

over distribution $\mathcal{Y}_h^*$. □

The proof follows from Lemma 6 by noting that GetReliableCRP$_h$ uses at most 2^{n_a} iterations and within each iteration $\mathcal{Y}_1^* = \mathcal{Y}^*$ corresponding to GetResponse is sampled h times. As discussed before, we expect $\epsilon_{\mathsf{corbias}} = negl(\lambda)$ where λ is the security parameter of PUF $\mathcal{P}$. Then, for $2^{n_a} = poly(\lambda)$, we also have a correlation bias $negl(\lambda)$ over distribution $\mathcal{Y}_h^*$.

Algorithm 3 Creating CRPs with reduced bias

```
1: procedure NEUMANN-GETRELIABLECRP_h
2:     Found =false
3:     while Found =false do
4:         (c_pre, r) ←GETRELIABLECRP_h
5:         (c'_pre, r') ←GETRELIABLECRP_h
6:         if r ≠ r' then Found =true
7:         end if
8:     end while
9:     return (c_pre, r)
10: end procedure
```

6. Reducing Bias

In order to reduce bias significantly, we can use the von Neumann trick which we also use in TRNG designs [29]. Algorithm 3 lists the pseudocode of a simple von Neumann interface where we use GETRELIABLECRP$_h$ twice in each while loop iteration until the two responses r and r' are different. We only output the first generated pre-challenge–response pair. The second application of GETRELIABLECRP$_h$ is used to simulate a probability distribution of a coin that tells the algorithm when to accept the first generated pre-challenge–response pair. Given the first generated pre-challenge–response pair (c_{pre}, r), the algorithm accepts and outputs this pair only if the second generated pre-challenge–response pair is of the form $(c'_{\mathsf{pre}}, r' = r + 1)$. The probability that this happens is equal to

$$\sum_{c' \in C_{\mathsf{P}}} \mathsf{Pr}((c', r+1) \leftarrow \mathcal{Y}_h^* \mid (c, r)), \tag{4}$$

where $\mathcal{Y}_h^*$ is the system-induced distribution of challenge–response pairs generated by GETRELIABLECRP$_h$ (here, $c' = \mathsf{Hash}(c'_{\mathsf{pre}})$ and $c = \mathsf{Hash}(c_{\mathsf{pre}})$ are selected according to distribution $\mathcal{Y}_h$, and $(c, r) \leftarrow \mathcal{Y}_h^*$). The probability that (c, r) is generated in the first pre-challenge–response pair is equal to $\mathsf{Pr}((c, r) \leftarrow \mathcal{Y}_h^*)$. Together with (4), this can be used to characterize $\mathcal{Y}_{\mathsf{neu},h}^*$, the probability that NEUMANN-GETRELIABLECRP$_h$ leads to CRP (c, r).

In NEUMANN-GETRELIABLECRP$_h$, we will use an iteration count b for the while loop, an index $i \in \{0, 1\}$ for c_{pre} and c'_{pre} in a loop iteration, and an iteration count a for the while loop in GETRELIABLECRP$_h$. This allows us to use a seed $seed$ as input and generate a sequence of pre-challenges $\mathsf{Hash}(seed \| a \| i \| b)$ as in (1). The proof of the lemma below is presented at the end of this section.

Lemma 11. (Bias von Neumann Trick) *Let* $(\mathcal{Y}_{\mathsf{neu},h}^*)^{\times d}$ *denote the system-induced distribution that generates challenge response vectors* $c = (c_0, \ldots, c_{d-1})$ *and* $r = (r_0, \ldots, r_{d-1})$ *according to* NEUMANN-GETRELIABLECRP$_h$ *where the while loop is at most called* 2^{n_b} *times. Suppose that PUF* $\mathcal{P}$ *has correlation bias at most* $\epsilon_{\mathsf{corbias}}$ *over a*

canonical system-induced distribution $\mathcal{Y}^*$. *Then,*

$$\left| q_r^{(\mathcal{Y}^*_{\mathsf{neu},h})^{\times d}} - 2^{-d} \right| \leq dh2^{n_a+1-d}(1+2^{n_b})\epsilon_{\mathsf{corbias}} + O((dh2^{n_a+n_b+1}\epsilon_{\mathsf{corbias}})^2 2^{-d})$$

and

$$q_r^{\mathcal{Y}^*_{\mathsf{neu},h}} \leq \frac{1}{2} + h2^{n_a}\epsilon_{\mathsf{corbias}} + O((h2^{n_a}\epsilon_{\mathsf{corbias}})^2).$$

Finally, since NEUMANN-GETRELIABLECRP$_h$ *outputs reliable CRPs, Lemma 9 shows that the overall reliability with respect to* $\mathcal{Y}_{\mathsf{neu},h}$ *is equal to* $1-\delta_h$ *with* $\delta_h = \mathbb{E}_{\mathcal{Y}_h}[p_c]$ *defined by* (3). □

An interesting notion is the so-called Hellinger distance between distribution $q_r^{(\mathcal{Y}^*_{\mathsf{neu},h})^{\times d}}$ over response vectors in $\{0,1\}^d$ and the uniform distribution over $\{0,1\}^d$, see [41]. Application of Lemma 11 shows that this is bounded by

$$\begin{aligned}
&\sqrt{1 - \sum_{r\in\{0,1\}^d} \sqrt{q_r^{(\mathcal{Y}^*_{\mathsf{neu},h})^{\times d}} \cdot 2^{-d}}} \\
&\leq \sqrt{1 - \sum_{r\in\{0,1\}^d} 2^{-d}\sqrt{1 - dh2^{n_a+1}(1+2^{n_b})\epsilon_{\mathsf{corbias}}}} \\
&= \sqrt{1 - \sqrt{1 - dh2^{n_a+1}(1+2^{n_b})\epsilon_{\mathsf{corbias}}}} \\
&\leq \sqrt{dh2^{n_a+1}(1+2^{n_b})\epsilon_{\mathsf{corbias}}},
\end{aligned}$$

if the latter is ≤ 1. In Sect. 11, we will want to have this Hellinger distance at most $2^{-(\kappa+5.946)/2}$, where κ represents the number of secure bits extracted from the PUF, and apply the theory of [41]; we will use this to show that we can replace distribution $q_r^{(\mathcal{Y}^*_{\mathsf{neu},h})^{\times d}}$ by the uniform distribution in our setting. This translates to the condition

$$\epsilon_{\mathsf{corbias}} \leq \frac{2^{-(\kappa+5.946)}}{dh2^{n_a+1}(1+2^{n_b})}. \tag{5}$$

As a final remark, we notice that the adversary can still use the biased responses from GETRESPONSE to train a machine learning model for predicting responses. The von Neumann trick only helps the legitimate user/system to get close to uniformly generated response bits.

Proof of Lemma 11. See Lemma 10, according to Definition 5, probability (4) is equal to $q_{r+1}^{\mathcal{Y}_h} \cdot (1 \pm \epsilon_{\mathsf{corbias},h})$. The probability that r is generated in the first pre-challenge–response pair is equal to $\sum_{c\in C_{\mathsf{P}}} \mathsf{Pr}((c,r) \leftarrow \mathcal{Y}_h^*) = q_r^{\mathcal{Y}_h}$. We conclude that the proba-

bility of an iteration producing the final output r is equal to

$$q_r^{\mathcal{Y}_h} q_{r+1}^{\mathcal{Y}_h} \cdot (1 \pm \epsilon_{\mathsf{corbias},h}). \tag{6}$$

This is the same for $r = 0$ and $r = 1$. The probability that an iteration is not yet producing the final output is therefore equal to $1 - 2q_0^{\mathcal{Y}_h} q_1^{\mathcal{Y}_h}(1 \pm \epsilon_{\mathsf{corbias},h})$. This teaches that it takes the while loop at most $(1 - 2q_0^{\mathcal{Y}_h} q_1^{\mathcal{Y}_h}(1 + \epsilon_{\mathsf{corbias},h}))^{-1}$ iterations in expectation to finish.

Let $\mathcal{Y}^*_{\mathsf{neu},h}$ indicate the distribution of a single CRP outputted by NEUMANN-GETRELIABLECRP$_h$. From (6), we infer

$$\begin{aligned}
&\sum_{c \in C_P} \mathsf{Pr}((c,r) \leftarrow \mathcal{Y}^*_{\mathsf{neu},h}) \\
&= \frac{q_r^{\mathcal{Y}_h} q_{r+1}^{\mathcal{Y}_h}(1 \pm \epsilon_{\mathsf{corbias},h})}{q_r^{\mathcal{Y}_h} q_{r+1}^{\mathcal{Y}_h}(1 \pm \epsilon_{\mathsf{corbias},h}) + q_{r+1}^{\mathcal{Y}_h} q_r^{\mathcal{Y}_h}(1 \pm \epsilon_{\mathsf{corbias},h})} \\
&= \frac{1}{2}\frac{1 \pm \epsilon_{\mathsf{corbias},h}}{1 \pm \epsilon_{\mathsf{corbias},h}} = \frac{1}{2} \cdot (1 \pm 2\epsilon_{\mathsf{corbias},h}/(1 - \epsilon_{\mathsf{corbias},h})).
\end{aligned}$$

This shows how the von Neumann trick reduces the bias $q^{\mathcal{Y}_h}$ down to only $\epsilon_{\mathsf{corbias},h}/(1 - \epsilon_{\mathsf{corbias},h})$ above $1/2$. This trick helps the legitimate user/system generate close to unbiased bits.

Our derivation above proves

$$\begin{aligned}
q_r^{\mathcal{Y}^*_{\mathsf{neu},h}} &\leq \frac{1}{2} + \frac{\epsilon_{\mathsf{corbias},h}}{1 - \epsilon_{\mathsf{corbias},h}} \\
&= \frac{1}{2} + h2^{n_a}\epsilon_{\mathsf{corbias}} + O((h2^{n_a}\epsilon_{\mathsf{corbias}})^2).
\end{aligned}$$

NEUMANN-GETRELIABLECRP$_h$ has at most 2^{n_b} loop iterations where in each iteration GETRELIABLECRP$_h$ is called twice. Applying Lemma 6 (see also the resemblance to Lemma 10) teaches us that the correlation bias over distribution $\mathcal{Y}^*_{\mathsf{neu},h}$ is at most

$$\begin{aligned}
\epsilon_{\mathsf{corbiasneu},h} &= (1 + \epsilon_{\mathsf{corbias},h})^{2 \cdot 2^{n_b}} - 1 \\
&= 2^{n_b+1}\epsilon_{\mathsf{corbias},h} + O((2^{n_b+1}\epsilon_{\mathsf{corbias},h})^2) \\
&= h2^{n_a+n_b+1}\epsilon_{\mathsf{corbias}} + O((h2^{n_a+n_b+1}\epsilon_{\mathsf{corbias}})^2)
\end{aligned}$$

Let $c = (c_0, \ldots, c_{d-1})$ and $r = (r_0, \ldots, r_{d-1})$ be vectors of challenges and responses generated by NEUMANN-GETRELIABLECRP$_h$. Let $(\mathcal{Y}^*_{\mathsf{neu},h})^{\times d}$ denote the distribution that generates such challenge response vectors. By the triangle inequality,

$$\left| q_r^{(\mathcal{Y}^*_{\mathsf{neu},h})^{\times d}} - 2^{-d} \right| \leq \left| q_r^{(\mathcal{Y}^*_{\mathsf{neu},h})^{\times d}} - \prod_{j=0}^{d-1} q_{r_j}^{\mathcal{Y}^*_{\mathsf{neu}}} \right| + \left| \prod_{j=0}^{d-1} q_{r_j}^{\mathcal{Y}^*_{\mathsf{neu}}} - 2^{-d} \right|.$$

By Lemma 8,

$$\begin{aligned}
&\left| q_r^{(\mathcal{Y}^*_{\mathsf{neu},h})^{\times d}} - \prod_{j=0}^{d-1} q_{r_j}^{\mathcal{Y}^*_{\mathsf{neu}}} \right| \\
&\leq ((1+\epsilon_{\mathsf{corbiasneu},h})^d - 1) \cdot \prod_{j=0}^{d-1} q_{r_j}^{\mathcal{Y}^*_{\mathsf{neu}}} \\
&\leq ((1+\epsilon_{\mathsf{corbiasneu},h})^d - 1) \cdot \left(\frac{1}{2} + \frac{\epsilon_{\mathsf{corbias},h}}{1-\epsilon_{\mathsf{corbias},h}} \right)^d \\
&= (d\epsilon_{\mathsf{corbiasneu},h} + O((d\epsilon_{\mathsf{corbiasneu},h})^2)) \\
&\quad \cdot 2^{-d}(1 + O(d\epsilon_{\mathsf{corbias},h})) \\
&= (dh2^{n_a+n_b+1}\epsilon_{\mathsf{corbias}} + O((dh2^{n_a+n_b+1}\epsilon_{\mathsf{corbias}})^2)) \\
&\quad \cdot 2^{-d}(1 + O(dh2^{n_a}\epsilon_{\mathsf{corbias}})) \\
&\leq dh2^{n_a+n_b+1-d}\epsilon_{\mathsf{corbias}} + O((dh2^{n_a+n_b+1}\epsilon_{\mathsf{corbias}})^2 2^{-d}).
\end{aligned}$$

Our previous analysis proves

$$\begin{aligned}
&\left| \prod_{j=0}^{d-1} q_{r_j}^{\mathcal{Y}^*_{\mathsf{neu}}} - 2^{-d} \right| \\
&\leq \left(\frac{1}{2} + \frac{\epsilon_{\mathsf{corbias},h}}{1-\epsilon_{\mathsf{corbias},h}} \right)^d - 2^{-d} \\
&= \left(\left(1 + 2\frac{\epsilon_{\mathsf{corbias},h}}{1-\epsilon_{\mathsf{corbias},h}} \right)^d - 1 \right) \cdot 2^{-d} \\
&= 2d\epsilon_{\mathsf{corbias},h}2^{-d} + O((2d\epsilon_{\mathsf{corbias},h})^2 2^{-d}) \\
&= dh2^{n_a+1-d}\epsilon_{\mathsf{corbias}} + O((dh2^{n_a+1}\epsilon_{\mathsf{corbias}})^2 2^{-d}).
\end{aligned}$$

Combining derivations proves Lemma 11. □

7. PUF Security

The following definition inspired by [32] defines the hardness of being able to software clone a PUF, i.e., the hardness of training an adversarial algorithm with oracle access to the PUF such that it can reliably predict responses for new randomly chosen challenges. Here, we only consider adversaries who can learn/observe digital information.

Definition 12. (*PUF Security with Correlated CRPs (inspired by* [32]*)*) We define a security game **SecGamePUFCor**$(\mathcal{P}, \Upsilon^{\mathcal{P}}, \mathcal{A}, k, t)$ for PUF $\mathcal{P}$, where $\Upsilon^{\mathcal{P}}$ is a set of system-induced distributions $\mathcal{Y}^*$ over challenge–response pairs in $C_{\mathcal{P}} \times R_{\mathcal{P}}$ (each $\mathcal{Y}^*$ is

represented as a ppt algorithm with oracle access to $\mathcal{P}$), $\mathcal{A}$ is a ppt $poly(t)$ algorithm,[10] and k represents the number of queries to $\mathcal{P}$ by $\mathcal{A}$:

1. For $j = 1, \ldots, k$, $\mathcal{A}$ adaptively selects a challenge $\hat{c}_j \in C_{\mathcal{P}}$ and receives[11] $\hat{r}_j \leftarrow \mathcal{P}(\hat{c}_j)$. Note that $\mathcal{A}$ is not able to get more responses beyond $\hat{r}_1, \ldots, \hat{r}_k$, due to the hardware unclonability of PUF $\mathcal{P}$.
2. For $0 \leq j \leq d-1$, let CRPs $(c_j, r_j) \leftarrow \mathcal{Y}^*_{(j)}$ for distributions selected from $\Upsilon^{\mathcal{P}}$. $\mathcal{A}$ is given set $\mathcal{D} = \{(c_j, \mathcal{Y}^*_{(j)})\}$.
3. $\mathcal{A}$ outputs guesses $\{r'_j\}_{j=0}^{d-1}$, and selects an index $0 \leq h \leq d-1$. Notice that $\mathcal{A}$ knows distributions $\mathcal{Y}^*_{(j)}$; that is, $\mathcal{A}$ knows the ppt algorithm which simulates $\mathcal{Y}^*_{(j)}$ with oracle access to $\mathcal{P}$. Even though $\mathcal{A}$ has no access to $\mathcal{P}$ in this step, $\mathcal{A}$ can make use of the knowledge that $c_j \leftarrow \mathcal{Y}_{(j)}$ (where $\mathcal{Y}_{(j)}$ is the projection of $\mathcal{Y}^*_{(j)}$ on the challenge space).

$\mathcal{A}$ *'wins'* the game if $r'_h = r_h$. Notice that the XOR value $r'_j + r_j$ is equal to the prediction error of the guess r'_j outputted by $\mathcal{A}$. We want to model how the probability of winning is conditioned on prediction errors, in other words, how are prediction errors for $j \neq h$ *correlated* with not making a prediction error for h (i.e., $r'_h = r_h$).

$\mathcal{P}$ is called a $(k, t, \epsilon_{\mathsf{corpred}})$-*secure PUF* for *correlations* with respect to $\mathcal{A}$ and set of distributions $\Upsilon^{\mathcal{P}}$ if $\mathcal{A}$ has an $\epsilon_{\mathsf{corpred}}$ advantage in predicting (any) r_h: Let λ be the security parameter of PUF $\mathcal{P}$. For all $d = poly(\lambda)$, advantage

$$\begin{aligned} &\mathsf{Adv}^{\mathsf{PUFCor}}_{\mathcal{A}} \\ &= \Pr\left(r'_h = r_h \mid \{r'_j + r_j\}_{j=0,\neq h}^{d-1}\right) - q^{\mathcal{Y}^*_{(h)}} \\ &\leq \epsilon_{\mathsf{corpred}}, \end{aligned}$$

where $\epsilon_{\mathsf{corpred}}$ represents the security,[12] and $q^{\mathcal{Y}^*_{(h)}}$ is the bias with respect to $\mathcal{Y}^*_{(h)}$ of $\mathcal{P}$. The probabilities are taken over $(c_j, r_j) \leftarrow \mathcal{Y}^*_{(j)}$, over measurement noise, and over all random procedures that $\mathcal{A}$ employs in the security game. $\mathcal{A}$'s advantage defines the software unclonability of PUF $\mathcal{P}$ when $\mathcal{P}$'s responses for challenges from distribution $\mathcal{Y}_{(j)}$ need to be predicted. □

Notice that the security game captures the scenario where the adversary has access to the PUF and uses this time to gather CRPs with which a machine learning model is trained. At a later moment access to the PUF is lost and the adversary uses its machine learning model to predict responses in order to impersonate the PUF or learn sensitive information whose confidentiality depends on private PUF responses. The predictive advantage $\epsilon_{\mathsf{corpred}}$ mainly depends on the amount of training data represented by k. The

[10] The cumulative number T of computational steps needed by $\mathcal{A}$ to play the security game (T represents the running time of $\mathcal{A}$) is $T = poly(t)$.

[11] Additional side-channel leakage of the PUF itself can be modeled by having $\mathcal{P}(c_j)$ output $side_j$ in addition to response r_j. However, we do not consider this type of attack in our adversarial model where the adversary can only learn digitally processed information.

[12] A larger challenge size λ generally corresponds to lower $\epsilon_{\mathsf{corpred}}$. From this perspective, λ can be considered a security parameter of PUF $\mathcal{P}$.

amount of computing time for training the 'best' model is a less important resource. For this reason, we do not specify an explicit bound on the running time T of adversary $\mathcal{A}$ in our definition, other than it being $poly(t)$ large enough for covering all practical adversaries.

Soundness. The special case $d = 1$ defines advantage

$$\mathsf{Pr}(r'_0 = r_0) - q^{\mathcal{Y}^*_{(0)}}. \tag{7}$$

This advantage does not take into account the additional advantage of correlations among different responses corresponding to randomly chosen challenges.

Our definition of $\epsilon_{\mathsf{corpred}}$ is sound in that we can always realize $\epsilon_{\mathsf{corpred}} = 0$ by defining the following simple adversary: The adversary always outputs the response r that maximizes $q_r^{\mathcal{Y}^*_{(0)}}$, that is, $q_r^{\mathcal{Y}^*_{(0)}} = q^{\mathcal{Y}^*_{(0)}}$. Then, $\mathsf{Pr}(r'_0 = r_0) = \mathsf{Pr}(r = r_0) = \sum_{c_0 \in C_\mathcal{P}} \mathsf{Pr}((c_0, r_0) \leftarrow \mathcal{Y}^*, r_0 = r) = q_r^{\mathcal{Y}^*_{(0)}} = q^{\mathcal{Y}^*_{(0)}}$ in (7). This achieves[13] $\epsilon_{\mathsf{corpred}} = 0$.

Adversary $\mathcal{A}^U$. For both the legitimate user as well as the adversary, we consider the hash-based interface GetResponse as the sole means for accessing PUF $\mathcal{P}$:

> In order to access PUF $\mathcal{P}$, the adversary cannot circumvent GetResponse; in addition GetResponse is immutable with respect to the adversary.

This assumption can be realized by means of 'hardware isolation' where PUF $\mathcal{P}$ is only accessible through the hardware interface defined by GetResponse. For example, in a secure processor architecture like Intel SGX [46] this access control[14] can be implemented by micro code which represents GetResponse and only allows access to the PUF by this micro-code.

Given that adversary $\mathcal{A}$ can only access $\mathcal{P}$ through the GetResponse interface in step 1, we essentially restrict $\mathcal{A}$'s adaptive strategy by forcing access to $\mathcal{P}$ through the cryptographic hash Hash. We will denote this type of adversary by $\mathcal{A}^U$. Notice that because of the hash function, it is not clear how the adversary can search for good challenges that help best in training an accurate prediction model. The current state-of-the-art analysis seems to indicate that the adversary may simply consider the hash function as an obstacle limiting selection of challenges $c_j \in C_\mathcal{P}$ according to a uniform distribution (just like the canonical system-induced distribution) with the possibility to repeat challenges.

Definition 13. An adversarial model for a PUF $\mathcal{P}$ defines how adversarial ppt algorithms $\mathcal{A}$ can have access to $\mathcal{P}$. By $\mathcal{A}^x$*-model for* $\mathcal{P}$, we denote a specific adversarial model where superscript x is a commonly understood abbreviation/name of the access restrictions to $\mathcal{P}$ imposed on adversaries $\mathcal{A}$ that are within the $\mathcal{A}^x$-model.

[13] We notice that the security game captures $\epsilon_{\mathsf{corbias}}$: See Definition 5 and Lemma 7, the intrinsic correlation present in the PUF can contribute at most $\epsilon_{\mathsf{corbias}} \cdot q^{\mathcal{Y}^*_{(h)}}$ to $\epsilon_{\mathsf{corpred}}$.

[14] Notice that this should not be confused with access control used by interactive protocols where users provide credentials and passwords. We mean that the hardware itself restricts under what circumstances resources and hardware modules can be accessed.

Adversarial ppt algorithms $\mathcal{A}$ within the $\mathcal{A}^U$-model for $\mathcal{P}$ cannot circumvent GetResponse in order to access $\mathcal{P}$. (This includes the implicit assumption that GetResponse is immutable with respect to the adversary.)

We write $\mathcal{A}^x \subseteq \mathcal{A}^U$ to mean that the $\mathcal{A}^x$-model is weaker than the $\mathcal{A}^U$-model, i.e., when compared to the $\mathcal{A}^U$-model, the $\mathcal{A}^x$-model imposes more restrictions on how adversaries can access $\mathcal{P}$. □

In practice $\mathcal{A}^U$ is powerful enough to train a machine learning model for $\mathcal{P}$ with a typical accurate prediction probability at most 75% (as in the iPUF paper [18]) for practical values of k (the amount of training data) and any practical run time T. This means that $\epsilon_{\mathsf{corpred}}$ is at most 25% in the worst case. Notice that this is quite different from the intrinsic PUF correlation modeled by $\epsilon_{\mathsf{corbias}}$. Given such a large $\epsilon_{\mathsf{corpred}}$ in practice, we need to be careful when designing a secure system that relies on a PUF.

We refer to Sect. 9 for an extensive discussion on weaker adversarial models $\mathcal{A}^x \subseteq \mathcal{A}^U$. In particular, we discuss an adversarial model which allows an XOR Arbiter PUF in a system that only queries each challenge at most once in an initialization mode and at most once in normal operation; the adversary turns out to be restricted to classical non-reliability-based ML attacks and the XOR Arbiter PUF is able to sufficiently withstand classical ML attacks for training based on reasonably sized CRP sets. Also here, we do not assume a confidential TCB for the PUF interface.

Ber Transformation Lemma. We are ready to define a special oracle $\mathcal{O}$ with knowledge of all prediction errors $e_j = r'_j + r_j$, $0 \leq j \leq d-1$, and who uses this knowledge to correct errors $e_j = 1$ as follows:

Lemma 14. (Ber transformation lemma) *Let $\mathcal{P}$ be a $(k, t, \epsilon_{\mathsf{corpred}})$-secure PUF for correlations with respect to $\mathcal{A}$ and set of system-induced distributions $\Upsilon^{\mathcal{P}}$. Let us consider* **SecGamePUFCor**$(\mathcal{P}, \Upsilon^{\mathcal{P}}, \mathcal{A}, k, t)$ *where $\mathcal{A}$ outputs predictions $\{r'_j\}_{j=0}^{d-1}$ of responses $\{r_j\}_{j=0}^{d-1}$ corresponding to distributions $\mathcal{Y}^*_{(j)} \in \Upsilon^{\mathcal{P}}$. Then, there exists an oracle $\mathcal{O}$ which takes predictions $\{r'_j\}_{j=0}^{d-1}$ as input and outputs partially corrected predictions $\{r"_j\}_{j=0}^{d-1}$ such that*

- *if r'_j is without error, then also $r"_j$ is without error, that is, $r"_j = r_j$ and*
- *each $r"_j = r_j + \hat{e}_j$ where $\hat{e}_j$ cannot be distinguished from a Bernoulli distribution*

$$Ber(1 - (q^{\mathcal{Y}^*_{(j)}} + \epsilon_{\mathsf{corpred}}))$$

with all $\hat{e}_j$ statistically independent.

□

The Ber transformation lemma is the main tool for proving our main theorem on the hardness of solving problem instances related to system security guarantees for a system with oracle access to $\mathcal{P}$.

Proof of Lemma 14. By using the same proof technique that shows Lemma 7, we can prove the next lemma. □

Lemma 15. *Let $\mathcal{P}$ be $(k, t, \epsilon_{\mathsf{corpred}})$-secure for correlations. Then,*

$$Pr\left(r'_h = r_h \mid side\right) - q^{\mathcal{Y}^*_{(h)}} \leq \epsilon_{\mathsf{corpred}}$$

for any random variable $side$ such that

$$\{r'_j + r_j\}_{j=0}^{d-1} \rightarrow \{r'_j + r_j\}_{j=0,\neq h}^{d-1} \rightarrow side$$

is a Markov chain. □

This lemma shows that the advantage holds for all side information $side$ that correlates with $\{e_j\}_{j=1,\neq h}^{d-1}$ but is statistically independent of e_h given $\{e_j\}_{j=1,\neq h}^{d-1}$, where $e_j = r_j + r'_j$ for $j \neq h$ are the prediction/guess errors (addition here is XOR): The adversary can select its own challenges and request corresponding measurements of responses in step 1 of the game in order to train a prediction model. A priori the adversary cannot indicate a predicate (in the form of side information) of its own choice that should be satisfied by the measured responses in step 2 and corresponding prediction errors in step 3. So, one cannot merge in step 1 the conditional knowledge of such a predicate ($side$).

The security game states that guess r'_h for challenge c_h cannot be improved by adversary $\mathcal{A}$ beyond advantage $\epsilon_{\mathsf{corpred}}$ even if a combination of challenges (excluding c_h) is known that corresponds to a joint statistical distribution of responses that satisfies a certain specified relation (or predicate), coded by $side$, with respect to guesses/predictions of these responses by $\mathcal{A}$.

We are ready to define a special oracle $\mathcal{O}$ with knowledge of all errors $e_j = r'_j + r_j$, $0 \leq j \leq d-1$, and who uses this knowledge to correct errors $e_j = 1$ as follows: Suppose e_j for $0 \leq j \leq h-1$ have already been corrected to $e_j + cor_j$ using the next randomized process implemented by $\mathcal{O}$. The oracle considers

$$side_h = \{e_j + cor_j\}_{j=0}^{h-1}$$

($side_h$ is empty for $h = 0$) and computes

$$\tau_h = q^{\mathcal{Y}^*_{(h)}} + \epsilon_{\mathsf{corpred}} \leq 1$$

and[15]

$$\alpha_h = \mathsf{Pr}(e_h = 0 \mid side_h).$$

$\mathcal{O}$ uses τ_h and α_h to define

$$b_h = \max\{0, (\tau_h - \alpha_h)/(1 - \alpha_h)\} \in [0, 1].$$

Oracle $\mathcal{O}$ computes

[15] Here, we mean random variable e_h takes on value 0, while in $side_h$ we mean that the random variable corresponding to $e_j + cor_j$ takes on the values $e_j + cor_j$.

- $cor_h = 0$ if $e_h = 0$, or
- $cor_h \leftarrow Ber(b_h)$ if $e_h = 1$, that is,

$$\mathsf{Pr}(cor_h = 1 \mid e_h = 1) = b_h.$$

The corrected error vector $\{e_j + cor_j\}_{j=0}^{d-1}$ has the property that only errors $e_h = 1$ are corrected (with probability b_h) and no new errors that did not exist before are introduced.

Notice that this correction procedure has the property $(e_h, side_h) \rightarrow cor_h$. In particular, $(e_{h-1}, side_{h-1}) \rightarrow cor_{h-1}$ for $h \geq 1$. Hence, for $h \geq 1$ we have the Markov chain

$$\begin{aligned} side_h &= (e_{h-1} + cor_{h-1}, side_{h-1}) \\ &\leftarrow (e_{h-1}, cor_{h-1}, side_{h-1}) \leftarrow (e_{h-1}, side_{h-1}) \end{aligned}$$

and, by using induction in h, we have

$$\begin{aligned} side_h &\leftarrow (e_{h-1}, side_{h-1}) \\ &\leftarrow (e_{h-1}, e_{h-2}, side_{h-2}) \leftarrow \ldots \leftarrow \{e_j\}_{j=0}^{h-1}. \end{aligned}$$

This shows that Lemma 15 applies and we conclude $\alpha_h \leq \tau_h$, hence,

$$b_h = (\tau_h - \alpha_h)/(1 - \alpha_h).$$

Now, we are ready to derive

$$\begin{aligned} &\mathsf{Pr}(\{e_j + cor_j\}_{j=0}^{d-1}) \\ &= \prod_{h=0}^{d-1} \mathsf{Pr}(e_h + cor_h \mid \{e_j + cor_j\}_{j=0}^{h-1}) \\ &= \prod_{h=0}^{d-1} \mathsf{Pr}(e_h + cor_h \mid side_h) \end{aligned}$$

with

$$\begin{aligned} &\mathsf{Pr}(e_h + cor_h = 0 \mid side_h) \\ &= \mathsf{Pr}(e_h = 0 \mid side_h)\mathsf{Pr}(e_h + cor_h = 0 \mid side_h, e_h = 0) \\ &\quad + \mathsf{Pr}(e_h = 1 \mid side_h)\mathsf{Pr}(e_h + cor_h = 0 \mid side_h, e_h = 1), \end{aligned}$$

where

$$\begin{aligned} &\mathsf{Pr}(e_h = 0 \mid side_h) = \alpha_h, \\ &\mathsf{Pr}(e_h = 1 \mid side_h) = 1 - \alpha_h, \\ &\mathsf{Pr}(e_h + cor_h = 0 \mid side_h, e_h = 0) \end{aligned}$$

$$
\begin{aligned}
&= \mathsf{Pr}(cor_h = 0 \mid side_h, e_h = 0) \\
&= \mathsf{Pr}(cor_h = 0 \mid e_h = 0) = 1, \text{ and} \\
\mathsf{Pr}(e_h + cor_h = 0 \mid side_h, e_h = 1) \\
&= \mathsf{Pr}(cor_h = 1 \mid side_h, e_h = 1) \\
&= \mathsf{Pr}(cor_h = 1 \mid e_h = 1) = b_h.
\end{aligned}
$$

Combining all equations yields

$$\mathsf{Pr}(e_h + cor_h = 0 \mid side_h) = \alpha_h + (1 - \alpha_h)b_h = \tau_h$$

and, for the partially corrected errors $\hat{e}_j = e_j + cor_j$,

$$\mathsf{Pr}(\{\hat{e}_j\}_{j=0}^{d-1}) = \prod_{j=0}^{d-1} \tau_j^{1-\hat{e}_j}(1 - \tau_j)^{\hat{e}_j}.$$

This shows that $\{\hat{e}_j\}_{j=0}^{d-1}$ cannot be distinguished from a distribution where $\hat{e}_j \leftarrow Ber(1 - \tau_j)$ and the $\hat{e}_j$ are statistically independent from one another. This proves the Ber transformation Lemma 14.

8. Interface Security

Imagine a system $\mathcal{S}$ with oracle access to a PUF $\mathcal{P}$. Suppose that proving a security guarantee of $\mathcal{S}$ entails proving the hardness of solving an instance of some class of problems. Explicitly, we want to prove that any problem instance generated by some problem distribution Q is hard. The difficulty of showing this hardness comes from the fact that system $\mathcal{S}$ has oracle access to $\mathcal{P}$; hence, also Q uses $\mathcal{P}$ to generate the parameters of a problem instance that corresponds to the security guarantee of $\mathcal{S}$. We denote this by $Q^{\mathcal{P}}$.

Figure 2 outlines our approach: We first define a system security game (referred to as the original game) where the adversary receives a problem instance g, which representation is a function of response bits $\{r_j\}$, and the adversary receives the corresponding challenges $\{c_j\}$ and the system-induced challenge distributions from which these are selected. Next we separate the prediction of response bits from the problem instance: We mean that the adversary can first create a prediction model of the PUF with which it predicts response bits $\{r'_j\}$ based on $\{c_j\}$ and their system-induced challenge distributions. Second, the adversary discards the challenges and their distributions and considers the problem instance $g(\{r_j\})$ with the predicted $\{r'_j\}$. We will argue that the $2^{\alpha\lambda}$ separation factor between the original and separated game is a mild assumption. And we will show in our final PUF separation theorem that the adversarial winning probability of the original game is still exponentially small in λ. The new problem instance $g(\{r_j\})$ with the predicted $\{r'_j\}$ can be reduced to a new formulation where the new problem instance g' is only a function of the prediction errors $\{r_j + r'_j\}$. (Addition is XOR.) For our PRO primitive, we will prove that there is only a $poly(\lambda)$ reduction factor. Now, we

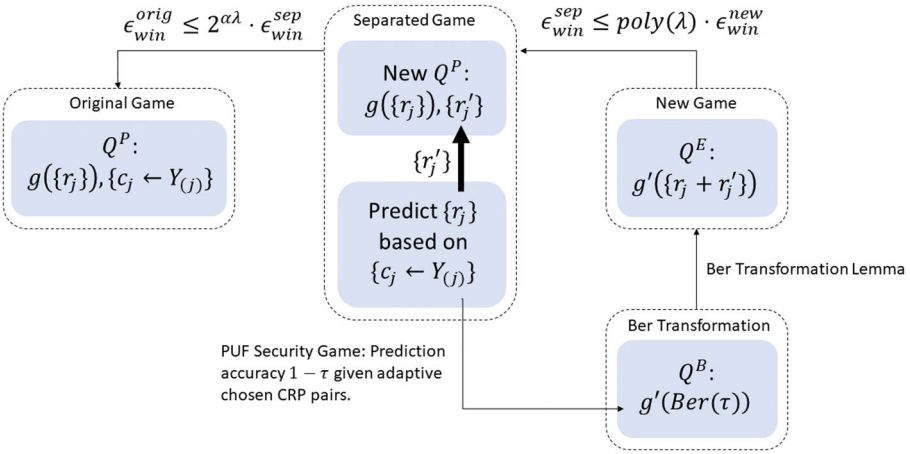

Fig. 2. Security reduction to a computational hard problem.

can use the Ber transformation lemma and obtain a computational hardness problem $Q^{\mathcal{B}}$ from which the PUF is completely separated out. What remains is a pure mathematical description which can now be properly analyzed. $Q^{\mathcal{B}}$ replaces $\{r_j + r_j'\}$ by bits produced by a Bernoulli distribution with parameter τ. This is related to the PUF security game which achieves a prediction accuracy $1 - \tau$. Now, we see how the hardness of $Q^{\mathcal{B}}$ is affected by the security of the PUF.

Definition 16. (*System Security*) Let $Q^{\mathcal{P}}$ be a ppt algorithm with oracle access to a PUF $\mathcal{P}$ that generates 'problem instances.' Let $\textsc{Ver}(q, s)$ be an algorithm that takes a problem $q \leftarrow Q^{\mathcal{P}}$ and a solution s as input and outputs whether s is a correct solution to q.

We define **SecGameSys**$(Q, \mathcal{P}, \Upsilon^{\mathcal{P}}, \mathcal{A}, k, T)$, where $\Upsilon^{\mathcal{P}}$ is a set of system-induced distributions $\mathcal{Y}^*$ over challenge–response pairs in $C_{\mathcal{P}} \times R_{\mathcal{P}}$ (each $\mathcal{Y}^*$ is represented as a ppt algorithm with oracle access to $\mathcal{P}$), $\mathcal{A}$ is a ppt algorithm, k represents the number of queries to $\mathcal{P}$ by $\mathcal{A}$, and T indicates the cumulative number of computational steps needed by $\mathcal{A}$ to play the security game:

1. For $j = 1, \ldots, k$, $\mathcal{A}$ adaptively selects a challenge $\hat{c}_j \in C_{\mathcal{P}}$ and receives $\hat{r}_j \leftarrow \mathcal{P}(\hat{c}_j)$. (This is step 1 of security game **SecGamePUF**.)
2. A problem instance $g \leftarrow Q^{\mathcal{P}}$ is generated. Let $\{r_0, \ldots, r_{d-1}\}$ be the responses on which g depends. Let $\{c_0, \ldots, c_{d-1}\}$ be the set of distinct challenges queried to $\mathcal{P}$ by Q that correspond to $\{r_0, \ldots, r_{d-1}\}$.
   Let $\mathcal{Y}_{(j)}^* \in \Upsilon^{\mathcal{P}}$ be the distributions implemented by Q for generating CRPs $(c_j, r_j) \leftarrow \mathcal{Y}_{(j)}^*$.
3. $\mathcal{A}$ is given set $\mathcal{D} = \{(c_j, \mathcal{Y}_{(j)}^*)\}$; here, $\mathcal{Y}_{(j)}^*$ indicates the ppt algorithm that uses oracle access to $\mathcal{P}$ which Q uses to draw challenges c_j and collect a corresponding response. $\mathcal{A}$ also receives problem instance g (which parameters depend on the CRPs collected by Q) and computes a solution s.

$\mathcal{A}$ *'wins'* the game if $\textsc{Ver}(g, s)$ returns true.

$Q^{\mathcal{P}}$ is called $(k, T, \epsilon_{\mathsf{win}})$-*system secure* with respect to $\mathcal{A}$ and set of probability distributions $\Upsilon^{\mathcal{P}}$ if the probability of winning is at most

$$\mathsf{Pr}(\text{true} \leftarrow \textsc{Ver}(g, s)) \leq \epsilon_{\mathsf{win}}.$$

□

Even though the adversary may have $\epsilon_{\mathsf{corpred}}$ up to 25% in **SecGamePUFCor**, we want ϵ_{win} to be exponentially small in some security parameter that defines Q.

Separating the PUF. On the one hand, we want Q to generate instances of a hard problem if $\mathcal{P}$ cannot be simulated by an adversary in probabilistic polynomial time. On the other hand, this requires us to show that such a simulation cannot be accurate enough even given the generated problem instance by $Q^{\mathcal{P}}$. But this may reveal through the generated problem instance information about the responses generated by $\mathcal{P}$ that was used by Q, and as a result prediction of these responses can be more accurate.

We have cyclic reasoning: In a sense, we want to assume that the problem instance itself is hard so that it cannot be used in the prediction of the used responses in the formulation of the problem instance. And if this is true, then the adversary can discard the problem instance when predicting the used responses, which leads to a $\epsilon_{\mathsf{corpred}}$. This can in turn be used to prove that the problem instance of Q is indeed hard to begin with. To break this cyclic reasoning, we need a separability assumption as defined below where we reformulate the steps of **SecGameSys** in an equivalent new way and a slightly modified 'separated' way as follows:

Definition 17. (*Separation Game*) We reformulate $\textbf{SecGameSys}(Q, \mathcal{P}, \Upsilon^{\mathcal{P}}, \mathcal{A}, k, T)$ by splitting $\mathcal{A}$ in two algorithms $\mathcal{A}_0$ and $\mathcal{A}_1$ where $\mathcal{A}_0$ is in charge of step 1 and step 3 is split into

- $\mathcal{A}_0$ is given set $\mathcal{D}$ and problem g and outputs a prediction $\{r'_j\}$ of the responses $\{r_j\}$ measured in step 2 by Q.
- $\mathcal{A}_1$ is given set $\mathcal{D}$, problem g, and $\{r'_j\}$ and computes a solution s of g.

Let $\mathcal{A}^x$-model be some adversarial model for $\mathcal{P}$. Clearly, $\mathcal{A}_1$ can play the role of the original $\mathcal{A}$ all by itself; hence, the probability of winning is still (equivalently) at most ϵ_{win} if $Q^{\mathcal{P}}$ is $(k, T, \epsilon_{\mathsf{win}})$-system secure with respect to all $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ in the $\mathcal{A}^x$-model and set of probability distributions $\Upsilon^{\mathcal{P}}$.

The separated game $\textbf{SecGameSysSep}(Q, \mathcal{P}, \Upsilon^{\mathcal{P}}, \mathcal{A}, k, T)$ for a pair $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ is defined as the game above where

- $\mathcal{A}_0$ does not have access to problem g, and
- $\mathcal{A}_1$ does not have access to set $\mathcal{D}$.

Let $\epsilon_{\mathsf{winsep}}$ be an upper bound on the probability of winning **SecGameSysSep** over all $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ within the $\mathcal{A}^x$-model.

If there exists a constant c_{sep} such that

$$\epsilon_{\mathsf{win}} \leq c_{\mathsf{sep}} \cdot \epsilon_{\mathsf{winsep}},$$

then we call $Q^{\mathcal{P}}$ c_{sep}-separable within the $\mathcal{A}^x$-model. □

Our ultimate goal is to separate PUF $\mathcal{P}$ out of the equation, that is, we want to reduce $Q^{\mathcal{P}}$ to $Q^{\mathcal{B}}$ where $\mathcal{B}$ is some predefined probability distribution over CRPs. This allows us to consider the hardness of $Q^{\mathcal{B}}$ which is a purely mathematical problem that does not involve a concrete physical device.

Let λ be a security parameter related to the class of problems $Q^{\mathcal{B}}$. We will see that in order to prove $\alpha\lambda$-bit security of $Q^{\mathcal{P}}$ it suffices to have $c_{\mathsf{sep}} = O(2^{(1-\alpha)\cdot\lambda})$. Hence, c_{sep} can be as large as exponential in λ. Notice that c_{sep} measures the multiplication factor increase in winning the system security game compared to winning the 'separated' system security game. As we have no idea how to design an adversarial strategy that combines all information (CRP pairs and problem instance g) in a more intelligent way than what is proposed in the separation game, it is unlikely that an attacker will be able to realize a multiplicative factor gain that is exponential in λ. Setting $c_{\mathsf{sep}} = 2^{\lambda/2}$ in the worst case seems large enough.

Hardness of Q. In order to separate the problem of predicting $\mathcal{P}$ from solving problems $g \leftarrow Q^{\mathcal{P}}$, we introduce the next definition. It defines under what circumstances solving g, given predictions of responses from $\mathcal{P}$ that were sampled by $Q^{\mathcal{P}}$ to formulate g, is equivalent to solving a similar problem as a function of only the prediction errors.

This is a reasonable assumption since in practice the system that measures responses of the PUF will, for example, use these to mask a key or store it as secret information for later use in an authentication or identification protocol. When demasking the key or when the identity of a PUF is verified, the PUF is measured a second time. The difference between the first and the second measurements should have a small number of measurement errors allowing demasking and identity verification. The adversary cannot measure the PUF a second time but tries to predict/estimate the used responses. Its success in demasking and extracting the key or impersonating the identity of the PUF depends on the number of prediction errors. This is exactly what we formalize in the next definitions.

Definition 18. (*Error-based reduction*) In step 4 of **SecGameSep** adversary $\mathcal{A}_1$ solves problem instance $g \leftarrow Q^{\mathcal{P}}$ given guesses $\{r'_j\}_{j=0}^{d-1}$ of the responses $\{r_j\}_{j=0}^{d-1}$ queried by Q to $\mathcal{P}$ in order to generate g. In this sense, g can be thought of as a function of $\{r_j\}_{j=0}^{d-1}$, denoted by

$$g(\{r_j\}_{j=0}^{d-1}),$$

and $\mathcal{A}_1$ can be thought of as solving an extended problem instance

$$g^{ext} = (g(\{r_j\}_{j=0}^{d-1}), \{r'_j\}_{j=0}^{d-1}).$$

Let $\mathcal{R}$ be the distribution that generates the vector pair $(\{r_j\}_{j=0}^{d-1}, \{r'_j\}_{j=0}^{d-1})$. We define $\mathcal{Q}^{\mathcal{R}}$ as the distribution that generates problems g^{ext}.

Suppose that there exists another formulation of a problem instance g' such that solving g^{ext} can be *reduced to solving*

$$g'(\{r'_j + r_j\}_{j=0}^{d-1}),$$

where g' has no other dependency on $\{r'_j\}_{j=0}^{d-1}$ or $\{r_j\}_{j=0}^{d-1}$ except through the 'errors' $\{e_j = r'_j + r_j\}_{j=0}^{d-1}$. That is,

- there exists a ppt algorithm $s \leftarrow \textsc{Tran}(s', \{r'_j\}_{j=0}^{d-1})$ and an algorithm $\textsc{VerE}(g', s')$ which returns true if and only if $\textsc{Ver}(g, \textsc{Tran}(s', \{r'_j\}_{j=0}^{d-1}))$ returns true.

If the above property holds, then we call g' an *error-based reduction* of g. Let $\mathcal{E}$ be the distribution that generates the error vector $\{e_j = r'_j + r_j\}_{j=0}^{d-1}$. We define $\mathcal{Q}^{\mathcal{E}}$ as the distribution that generates problems g' (by replaying **SecGameSep** as explained above). We call $Q^{\mathcal{E}}$ with (Trans, VerE) the *error-based reduction* of $Q^{\mathcal{R}}$ with Ver in **SecGameSep**. If such $Q^{\mathcal{E}}$ exists, then we say that $Q^{\mathcal{R}}$ *has an error-based reduction*. □

Reasoning about the hardness of $\hat{Q} = Q^{\mathcal{E}}$ for a *known representation* of an error vector distribution $\mathcal{E}$ is a purely mathematical problem that does not depend on a concrete physical PUF device $\mathcal{P}$—we have not yet accomplished this as the error vector distribution is implicitly defined by the exact functioning of $\mathcal{P}$. The next definition formalizes the hardness of $\hat{Q} = Q^{\mathcal{E}}$ or $\hat{Q} = Q^{\mathcal{R}}$.

Definition 19. (*Mathematical hardness assumption for* $\hat{Q}$) Define **HardnessQError** $(\hat{Q}, \textsc{VerH}, \mathcal{A}, T)$ as a game where $\mathcal{A}$ is a ppt algorithm that takes a problem instance $\hat{g} \leftarrow \hat{Q}$ as input and runs at most T computational steps after which it outputs a guess $\hat{s}$. Adversary $\mathcal{A}$ wins the game if $\textsc{VerH}(\hat{g}, \hat{s})$ returns true, that is, $\hat{s}$ is a solution of problem $\hat{g}$. We call $\hat{Q}$ $(T, \epsilon_{\mathsf{hard}})$-*hard* if the probability of winning **HardnessQError** is at most ϵ_{hard}.

We may consider ϵ_{hard} to be a function of T. If for all T,

$$\epsilon_{\mathsf{hard}}(T) \leq T \cdot 2^{-\lambda},$$

then we say that $\hat{Q}$ is λ-*bit secure*. □

In **HardnessQError** $\mathcal{A}$ plays the role of $\mathcal{A}_1$ in **SecGameSep** without access to set $\mathcal{D}$. In other words, $\mathcal{A}$ only knows that parameters have been generated using distributions $\mathcal{R}$ and $\mathcal{E}$, respectively. $\mathcal{A}$ does not know to which challenges these correspond to and as a consequence does not know a distribution $\mathcal{R}$ or $\mathcal{E}$ conditioned on the used challenges, which could have given $\mathcal{A}$ more information for finding a solution $\hat{s}$. In **SecGameSep**, we model the increased probability of winning, if such knowledge were available, by the multiplicative factor c_{sep}.

We next define how the bit securities of $\hat{Q} = Q^{\mathcal{E}}$ and $\hat{Q} = Q^{\mathcal{R}}$ are related.

Definition 20. (*Error-based equivalent of $Q^{\mathcal{P}}$*) Let $Q^{\mathcal{E}}$ be an error-based reduction of $Q^{\mathcal{R}}$. We say $Q^{\mathcal{E}}$ is an *error-based equivalent* of $Q^{\mathcal{R}}$ up to factor e_{equiv} if

$$[Q^{\mathcal{E}} \text{ is } \kappa\text{-bit secure}] \Rightarrow [Q^{\mathcal{R}} \text{ is } (\kappa - e_{\mathsf{equiv}})\text{-bit secure}].$$

In the remainder of the paper, rather than referring to $\mathcal{R}$, we say $Q^{\mathcal{E}}$ is an *error-based equivalent* of $Q^{\mathcal{P}}$ up to factor e_{equiv} and if such $Q^{\mathcal{E}}$ exists, then we say that $Q^{\mathcal{P}}$ *has an error-based equivalent* up to factor e_{equiv}. □

The next definition defines what it means when problems $Q^{\mathcal{E}}$ become easier to solve if some of the errors produced by $\mathcal{E}$ are corrected.

Definition 21. (*Q in the presence of an error-reducing oracle*) Let $\mathcal{E}$ be a distribution that generates error vector $\{e_j\}_{j=0}^{d-1}$. Let $\mathcal{O}$ be an oracle that takes as input an error vector $\{e_j\}_{j=0}^{d-1}$ and corrects some of the errors $e_j = 1$ by computing a new error vector $\{\hat{e}_j\}_{j=0}^{d-1}$ with $\hat{e}_j = 0$ if $e_j = 0$ is already without error, and $\hat{e}_j \in \{0, 1\}$ if $e_j = 1$. (Depending on $\mathcal{O}$'s strategy this may correct the error to $\hat{e}_j = 0$.) We call such oracle $\mathcal{O}$ an *error-reducing oracle*.

We say that $Q^{\mathcal{O}\circ\mathcal{E}}$ is *easier to solve* than $Q^{\mathcal{E}}$ if

$$[Q^{\mathcal{O}\circ\mathcal{E}} \text{ is } \kappa\text{-bit secure}] \Rightarrow [Q^{\mathcal{E}} \text{ is } \kappa\text{-bit secure}].$$

If $Q^{\mathcal{O}\circ\mathcal{E}}$ is easier to solve than $Q^{\mathcal{E}}$ for all error-reducing oracles $\mathcal{O}$, then we say that $Q^{\mathcal{E}}$ *becomes simpler in the presence of an error-reducing oracle*. □

This assumption seems to be a natural one since fewer prediction errors for the adversary should make it easier for the adversary to, for example, demask a key or impersonate a PUF. The above framework for reasoning about the hardness of $Q^{\mathcal{P}}$ will allow us to prove precise security guarantees. Our framework allows the following main separation theorem, which we apply in the next section (where we prove the validity of some of the conditions in the theorem):

Theorem 22. (PUF separation theorem) *Consider the following setting:*

- *Let $\mathcal{A}^x$-model be some adversarial model with $\mathcal{A}^x \subseteq \mathcal{A}^U$.*
- *Suppose that $\mathcal{P}$ is a $(k, t, \epsilon_{\mathsf{corpred}})$-secure PUF for correlations with respect to all $\mathcal{A}$ that are within the $\mathcal{A}^x$-model and with respect to a set of system-induced CRP distributions $\Upsilon^{\mathcal{P}}$. Let $q^{\Upsilon} = \sup_{\mathcal{Y}^* \in \Upsilon} q^{\mathcal{Y}^*}$ and define $\tau = 1 - (q^{\Upsilon} + \epsilon_{\mathsf{corpred}})$.*
- *Define $\mathcal{B}$ to be the distribution that generates statistically independent errors $\hat{e}_j \leftarrow Ber(\tau)$. Assume that $Q^{\mathcal{B}}$ is λ-bit secure.*
- *Assume that $Q^{\mathcal{P}}$ has an error-based equivalent $Q^{\mathcal{E}}$ up to factor $e_{\mathsf{equiv}} = O(\log_2 \lambda)$. And assume that $Q^{\mathcal{E}}$ becomes simpler in the presence of an error-reducing oracle.*
- *Assume that $Q^{\mathcal{P}}$ is c_{sep}-separable within the $\mathcal{A}^x$-model with $c_{\mathsf{sep}} = 2^{\alpha\lambda}$ for some $\alpha \in [0, 1)$.*

Then, for $T = poly(t)$, $Q^{\mathcal{P}}$ is $(k, T, \epsilon_{\mathsf{win}})$-system secure with respect to all $\mathcal{A}$ within the $\mathcal{A}^x$-model and set of system-induced distributions $\Upsilon^{\mathcal{P}}$ with

$$\epsilon_{\mathsf{win}} \leq c_{\mathsf{sep}} \cdot T 2^{e_{\mathsf{equiv}}} 2^{-\lambda},$$

which is $negl(\lambda)$. □

Interpretation. The list of assumptions made by the separation theorem starts by assuming an adversarial model that is at least restricted by $\mathcal{A}^U$ meaning that the PUF must be accessed through the GetResponse functionality. There may be further restrictions for the adversary. We further assume that $\mathcal{P}$ is a secure PUF for correlations; see our earlier discussion, an adversary who uses ML for constructing a response prediction model for the interpose PUF generally will see prediction errors with probability $\tau \leq 25\%$. This covers the best-known methods for creating a prediction model. The theorem shows how such a large τ can still lead to a secure scheme.

For the scheme, we assume an underlying hardness problem Q that uses responses as input. In practice, the legitimate user re-measures responses and is able to perform some 'decoding' operation which mathematically means that the re-measured responses can be xored with the responses in the instance of the hardness problem. This will naturally transform the problem into an equivalent one that only depends on the measurement errors. The adversary can do the same for its predicted responses and his problem will become equivalent to one that only depends on prediction errors (predicted responses can be discarded if the actual responses are nearly uniform, e.g., due to the application of the von Neumann trick). Section 11 demonstrates our main example. Generally, the smaller the expected number of errors, the more efficient algorithms for finding solutions become. From this perspective, it is natural to assume that the equivalent problem becomes simpler in the presence of an error-reducing oracle.

The assumption that states that $Q^{\mathcal{P}}$ is separable simply stems from the observation that known attacks and their analysis in the literature always perform or assume this two-step attack approach where first PUF responses are predicted (separate from the actual problem instance), after which the problem instance is solved using the predicted responses. By only requiring a large $c_{\mathsf{sep}} = 2^{\alpha\lambda}$, this assumption is likely satisfied.

All of the above reduces the analysis of the system's security to a mathematical problem: By proving that $Q^{\mathcal{B}}$ is hard, we can use the theorem to show that $Q^{\mathcal{P}}$ is system secure. Here, we generally assume $T = poly(\lambda)$.

Security protocols that use PUFs. SecGameSys models a general system $\mathcal{S}$ whose security guarantee reduces to the hardness of some problem statement $Q^{\mathcal{P}}$. Here, $\mathcal{S}$ does not need to be limited to the non-interactive setting. $\mathcal{S}$ can be an interactive protocol between colluding or non-colluding participants that each own or transmit PUFs from one party to another. When using multiple PUFs, we need to generalize oracle access to multiple PUFs in **SecGameSys**. When modeling multiple participants in a protocol, $\mathcal{S}$ replays protocol executions. Typically, different hardness problems $Q^{\mathcal{P}}$ model the different honest-but-curious or malicious parties. And oracle access to an underlying PUF $\mathcal{P}$ by such a hardness problem may be further constrained by the protocol implemented by $\mathcal{S}$. This would imply that in our game(s) adversary $\mathcal{A}^U$ is further restricted by assuming a larger trusted computing base (TCB) for the PUF interface; the TCB implements some

extended immutable interface (with or without including confidential processing) which restricts the adversary in its access to CRPs $(\hat{c}_i, \hat{r}_i)$ (in step 1 of **SecGameSys**).

Proof of Theorem 22. Suppose that $Q^{\mathcal{P}}$ is $(k, T, \epsilon_{\mathsf{win}})$-system secure with respect to all $\mathcal{A}$ within the $\mathcal{A}^x$-model and set of distributions $\Upsilon^{\mathcal{P}}$. We first apply Definition 17 which states that

$$\epsilon_{\mathsf{win}} \leq c_{\mathsf{sep}} \cdot \epsilon_{\mathsf{winsep}} \tag{8}$$

where $\epsilon_{\mathsf{winsep}}$ is an upper bound on the probability of winning **SecGameSysSep** $(Q, \mathcal{P}, \Upsilon^{\mathcal{P}}, \mathcal{A}, k, T)$ over all $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ within the $\mathcal{A}^x$-model.

In separation game **SecGameSysSep** adversary $\mathcal{A}_0$ disregards the knowledge of the problem instance $g \leftarrow Q^{\mathcal{P}}$. This means that $\mathcal{A}_0$ plays the steps of **SecGamePUFCor**$(\mathcal{P}, \Upsilon^{\mathcal{P}}, \mathcal{A}_0, k, T)$, see Definition 12. The goal of $\mathcal{A}_0$ is to provide estimates $\{r'_j\}$ which are most useful for $\mathcal{A}_1$.

In separation game **SecGameSysSep** adversary $\mathcal{A}_1$ disregards knowledge $\mathcal{D}$ and solves problem instance g as a function of $\{r_j\}$ with knowledge of estimates $\{r'_j\}$. This is equivalent to solving the extended problem g^{ext} from $Q^{\mathcal{R}}$, see Definition 18. The theorem states that $Q^{\mathcal{P}}$ has an error-based equivalent $Q^{\mathcal{E}}$ up to factor e_{equiv}, where, see Definition 18, $\mathcal{E}$ is the distribution that generates error vector $\{e_j = r'_j + r_j\}$. According to Definition 20, if $Q^{\mathcal{E}}$ is λ'-bit secure, then $Q^{\mathcal{R}}$ is $(\lambda' - e_{\mathsf{equiv}})$-bit secure. That is, adversary $\mathcal{A}_1$ wins **SecGameSysSep** with probability at most

$$\epsilon_{\mathsf{winsep}} \leq T 2^{-(\lambda' - e_{\mathsf{equiv}})}. \tag{9}$$

See Definition 21, since $Q^{\mathcal{E}}$ is assumed to become simpler in the presence of an error-reducing oracle, $\mathcal{A}_0$ will help $\mathcal{A}_1$ in achieving the largest upper bound on the winning probability in (9) by minimizing the number of prediction errors. We conclude that $\mathcal{A}_0$ plays **SecGamePUFCor** where $\mathcal{A}_0$'s goal is to maximize $\mathsf{Adv}^{\mathsf{PUFCor}}_{\mathcal{A}_0}$.

The theorem states that $\mathcal{P}$ is a $(k, t, \epsilon_{\mathsf{corpred}})$-secure PUF for correlations with respect to $\mathcal{A}_0$ (which is within the $\mathcal{A}^x$-model) and with respect to a set of CRP distributions $\Upsilon^{\mathcal{P}}$. (Notice that $T = poly(t)$ and indeed $\epsilon_{\mathsf{corpred}}$ applies to $\mathcal{A}_0$ which runs at most T accumulative computation steps.) Now, we apply the Ber transformation lemma (Lemma 14): There exists an error-reducing oracle $\mathcal{O}$, see Definition 21, with the additional property that $\mathcal{O} \circ \mathcal{E}$ outputs an error vector $\{\hat{e}_j\}$ where each entry $\hat{e}_j$ cannot be distinguished from a Bernoulli distribution $Ber(1 - (q^{\mathcal{Y}^*_{(j)}} + \epsilon_{\mathsf{corpred}}))$ with all $\hat{e}_j$ statistically independent. This in turn implies the existence of an error-reducing oracle $\mathcal{O}$ which reduces errors even more, such that $\mathcal{O} \circ \mathcal{E}$ outputs an error vector $\{\hat{e}_j\}$ where each entry $\hat{e}_j$ cannot be distinguished from a Bernoulli distribution $Ber(\tau)$ (where τ is defined in the theorem statement) and all $\hat{e}_j$ statistically independent. We have $Q^{\mathcal{O} \circ \mathcal{E}} = Q^{\mathcal{B}}$.

The theorem assumes that $Q^{\mathcal{B}}$ is λ-bit secure, see Definition 19. Since $Q^{\mathcal{E}}$ is assumed to become simpler in the presence of an error-reducing oracle, if $Q^{\mathcal{O} \circ \mathcal{E}} = Q^{\mathcal{B}}$ is λ-bit secure, then $Q^{\mathcal{E}}$ is λ-bit secure, see Definition 21. This means that we can substitute $\lambda' = \lambda$ in (9). Together with (8), $e_{\mathsf{equiv}} = O(\log \lambda)$, and $c_{\mathsf{sep}} = 2^{\alpha\lambda}$ for some $\alpha < 1$, the theorem follows. □

9. Adversarial PUF Models

We have already provided a detailed discussion on the $\mathcal{A}^U$-model (representing adversaries who cannot circumvent GetResponse for accessing PUF $\mathcal{P}$). In this section, we discuss other more restricted adversarial models, explain the role of k in our security games, and compare Definition 12 with that of [32].

Adversary $\mathcal{A}^N$. This adversary has no access to the PUF at all and cannot learn any CRPs in step 1 of **SecGamePUFCor** or **SecGameSys**; we have $k = 0$. In particular, this means that its best strategy is predicting response bits according to the bias of the PUF, that is, $\epsilon_{\mathsf{corpred}} = 0$.

For example, in the LPN-PUF [24,25], a PUF with only one CRP[16] is used, and the corresponding response is assumed to remain private with respect to an adversary. The whole LPN-PUF interface is assumed to be in a TCB while powered on. In power-off mode, the LPN-PUF interface circuitry can be observed, and this allows the adversary to learn fused-in keys. The LPN-PUF solves this problem by not depending on fused-in keys.

A second example is a simple masked memory which essentially obfuscates one single key key by using hardware circuitry that has (1) a pre-challenge fused in with which a vector r of response bits corresponding to a sequence of different challenges can be queried from a PUF, and (2) has fused in the vector $key + r$ from which key can be extracted given[17] r. In this setting, the TCB disallows the adversary to observe r when the key masking interface is powered on. When powered off, the adversary can read the fused-in challenges and $key + r$. This is the same model as for the LPN-PUF (with the difference that the LPN-PUF can generate many input–output pairs).

Adversary $\mathcal{A}^{NR}$. If we disallow repeated measurements in step 1 of **SecGamePUFCor** or **SecGameSys**, then we have adversary $\mathcal{A}^{NR} \subseteq \mathcal{A}^U$. Here, we assume that whenever the adversary is present, it cannot freely query PUF $\mathcal{P}$, and it can only query or observe queries of CRPs that are new to the adversary; they have distinct challenges, and hence, challenges and CRPs are not repeated in the adversary's observation.

The imposed restriction on the adversary may be due to a system implementation that has a secure initialization phase where we assume no presence of an adversary at all and a normal mode of operation that never repeats queries to the PUF and which resulting CRPs can be observed by the adversary. Here, we make the adversary weaker since rather than being able to freely call GetResponse, only the digital computations (including CRPs) of past normal mode operation can be observed by the adversary. This restriction can be enforced by hardware isolation mechanisms for access control[18] where, for example, only the system 'enclave' is allowed access to GetResponse.

Such a system may execute a series of 'sessions' and the considered adversary may observe all but one session. The security guarantee is about the unobserved session for which we want to prove that the adversary cannot impersonate the session or extract sensitive information from the session. All the other observed sessions give the adversary

[16] The used PUF is a ring oscillator PUF which can be used to measure the same response again and again.

[17] Circuitry for correcting measurement noise may also be included.

[18] Not for implementing confidential computing.

a number of CRPs with non-repeating challenges (in step 1 of **SecGamePUFCor** or **SecGameSys**). We find ourselves in the $\mathcal{A}^{NR}$ model. To guarantee this model, normal operation of the system may include a mechanism for checking whether challenges have not yet been queried in normal operation before. This can be done by using a Merkle tree like the logical erasable PUF interface of [32].

The weaker $\mathcal{A}^{NR}$ can only learn challenge–response pairs but cannot learn challenge–reliability pairs for which a response for a challenge needs to be repeatedly measured. This means that the adversary can only use classical machine learning to train a prediction model for the PUF. This reduces $\epsilon_{\mathsf{corpred}}$. In fact, silicon PUF designs, such as the XOR Arbiter PUF, that may have $\epsilon_{\mathsf{corpred}}$ closer to 40–50% (i.e., the adversary has very accurate predictions) for advanced (reliability-based) machine learning, can have a much lower $\epsilon_{\mathsf{corpred}}$ around 25% for classical machine learning. We conclude that the XOR Arbiter PUF can possibly be used in the $\mathcal{A}^{NR}$ model, while we may consider it broken in the $\mathcal{A}^{U}$ model.

If we are allowed to use the XOR Arbiter PUF, then we prefer this design over the iPUF since it has a smaller area size and is more reliable, but more importantly, we prefer the XOR Arbiter PUF for the following reason: $\mathcal{A}^{U}$ corresponds to an adversary who can use CRPs for repeated challenges. This means that advanced ML attacks based on challenge–reliability pairs can be utilized. In this context, the current state-of-the-art strong silicon PUF design is the Interpose PUF (iPUF)[18]. The iPUF is a combination of two XOR Arbiter PUFs that are connected through a kind of forwarding mechanism (where the response bit of the 'top' XOR Arbiter PUF is inserted in the middle of the challenge for the 'lower' XOR Arbiter PUF). The iPUF has seen recent ML-based attacks [16,17,37] and calls the existence of a strong PUF into question. Even though the published attacks still seem to leave concrete parameter settings for which the iPUF attacks have not yet been demonstrated, it remains an open problem for how long such settings remain secure. (And parameter settings cannot be chosen too 'large' as this hurts the reliability of the iPUF.) It may very well be that the iPUF design itself will need to be adapted by replacing Arbiter PUFs with Feed Forward Arbiter PUFs together with an extended new ML-based security analysis. For this reason, rather than assuming the security of the iPUF will not be broken, i.e., $\mathsf{Adv}^{\mathsf{PUFCor}}_{\mathcal{A}^R} \leq \epsilon_{\mathsf{corpred}} \approx 25\%$ for the iPUF in Definition 12 for all practical parameter settings, we assume the $\mathcal{A}^{NR}$ model where we can rely on the XOR Arbiter PUF. The security under classical machine learning attacks of the XOR Arbiter PUF has been well studied. The state-of-the-art work is in [37], which we discuss in more detail at the end of this section when talking about the role of k.

Notice that in an XOR Arbiter PUF design with $\lambda = 64$ challenge bits per Arbiter PUF and say $x = 10$ Arbiter PUFs in total, we can use a pre-challenge as input to a hash function or PRG that outputs the concatenation of x different challenges for each of the x Arbiter PUFs. This means that the Hamming distance argument for $\epsilon_{\mathsf{corbias}}$ in Sect. 3 is over $x\lambda = 640$ bits, and we expect $\epsilon_{\mathsf{corbias}}$ to be exponentially small in $x\lambda$ even though we only use challenges of size λ for the individual Arbiter PUFs. We notice that the XOR Arbiter PUF and iPUF are defined by copying the same challenge to the different component Arbiter PUFs. We strongly suggest using a hash function or PRG as described in Sect. 3 in order to compute different challenges for each of the Arbiter

PUFs as this will allow us to argue a 'negligibly' small $\epsilon_{\mathsf{corbias}}$ for practical parameter settings.[19]

As a final remark, suppose that only GetResponse is in the TCB together with a Merkle tree interface used for checking that challenges have not yet been queried in normal operation before. The secure initialization phase produces reliable CRPs based on GetReliableCRP$_h$ that will be consumed in normal operation in the presence of an adversary, who, if $h > 1$ and if GetReliableCRP$_h$ is implemented using a *seed* which is published by the system (we describe such an example in Sect. 11, see (12)), can extract information about repeated response measurements. (The adversary can reconstruct challenges that were found not to produce reliable responses as well as observe the used reliable challenge; this allows a form of advanced ML that uses reliability information.) Therefore, in the $\mathcal{A}^{NR}$ model the initialization phase can only use GetReliableCRP$_h$ for $h = 1$.

Adversary $\mathcal{A}^{R-x}$. In the $\mathcal{A}^{U}$-model adversaries can repeat measurements of CRPs as they wish. $\mathcal{A}^{R-x}$ defines a slight restriction where the number of repeated measurements is controlled/restricted to x. Again, this can be implemented by using an authenticated search tree, very much like the programmable access-controlled PUF interface of [32].

For example, we may consider a system with an initialization phase and normal operation as discussed above, where we now allow the adversary to also be *sometimes* present to and observe the digital computations done in the initialization phase. (We still want to prove a security guarantee for normal system operation based on CRPs generated during the initialization phase when no adversary was present.) If the initialization phase is such that CRPs are only measured at most once during initialization, then in total, the adversary may obtain two measurements for each challenge: One coming from the initialization phase and the other coming from normal operation. In this example, we have $x = 2$. This provides only limited reliability information to the adversary, and it is an open problem whether $\epsilon_{\mathsf{corpred}}$ remains around 25% (or slightly higher) for the XOR Arbiter PUF.

If we use GetReliableCRP$_h$ in initialization mode and if we do not assume its computations are confidential, then $x = h + 1$ reflecting h measurements in GetReliableCRP$_h$ during the initialization phase and one repeated measurement using GetResponse during normal operation. (If in normal mode a measurement is repeated as in GetReliableCRP$_{h'}$ in order to use the majority vote, which is more reliable, then $x = h + h'$.)

Role of k. Our definitions explicitly include the parameter k in the first steps of our security games. It indicates the maximum number of responses an adversary is allowed to measure. Upper bound k is of crucial importance as this indicates the amount of CRPs an adversary can use to build a prediction model.

For example, when using ML to train a prediction model for a silicon PUF, then the amount of training data indicates the accuracy of the resulting prediction model. That is, the more training data (the larger k), the higher the prediction advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{PUFCor}}$ and the higher $\epsilon_{\mathsf{corpred}}$ as a result. Parameter k indicates how often the adversary can have

[19] This will also likely slow down or make classical and advanced ML attacks on the XOR Arbiter PUF or iPUF more difficult.

access to the PUF. See [11,18,37,47] for studies on the trade-off between the amount of training data and prediction accuracy.

The most effective implemented attack[20] on the iPUF turns out to be a classical ML attack [37] (by combining the multilayer perceptron attack [48] with the splitting attack [16]). This is in line with the original iPUF security analysis, where it was shown that state-of-the-art reliability-based attacks at that moment could not be applied, and therefore the security of the iPUF is reduced to the best-known classical ML attack on the XOR Arbiter PUF. In [37], it is shown that the 9,10,11-XOR Arbiter PUFs can be learned with 98% accuracy by using 45 M, 119 M, and 325 M CRPs, respectively. This leads to breaking the (11,11)-iPUF using $2 \cdot 325 = 750$M CRPs. Extrapolating shows that to accurately learn a (20,20)-iPUF, one needs roughly 1 week CRP data collection at 1MHz CRP frequency for a total of about 605B CRPs. As an example, suppose that a system implements and uses the initialization and normal operation for at most 150B CRPs in a 10-year time frame (e.g., a system uses GetOutput-RO of our PRO construction for the concrete parameters setting in Sect. 11 every two minutes for 10 years). Then, if the legitimate user plans to query the iPUF at most twice (for initialization and normal operation) for 150B CRPs, then additional access control can be added in order to limit the querying of the iPUF to 300B CRPs (after which it locks). This means that the attacker can only collect a smaller portion of the needed 605B CRPs for 98% accuracy, hence, the prediction accuracy can be significantly lower. This fits our framework.

However, due to the many component Arbiter PUFs, the (20,20)-iPUF will suffer overall reliability. Formula (26) in [18] shows that the measurement noise rate of the (20,20)-iPUF is equal to $\beta_{20} + \frac{\beta_{20}}{2}(1 - 2\beta_{20}) = \frac{3}{2}\beta_{20} - \beta_{20}^2$ where $\beta_{20} = (1 - (1 - 2\beta)^{20})/2$ with β the noise rate of a single Arbiter PUF. If we want to achieve a 10% noise rate for the (20,20)-iPUF, then we need to be able to implement very reliable Arbiter PUFs with measurement noise rate $\beta = 0.36\%$. This is an engineering problem that needs to be solved unless a new follow-up PUF design avoids the XOR operation (which amplifies measurement noise) and, as a result, is much more reliable by design.

In the $\mathcal{A}^{NR}$ model, we saw that we could use a 20-XOR Arbiter PUF. This has a reliability equal to $\beta_{20} = 6.7\%$. If we only need to achieve $\beta_{20} = 10\%$, then the required β can be increased to 0.55%, still an engineering problem. If we only restrict ourselves to the $\mathcal{A}^{NR}$ model, which only allows classical ML attacks, then the multiplexer-based Arbiter PUF composition of [49] and its follow-up work is of interest. In this design, the security is claimed not to decrease while the reliability only corresponds to that of a single Arbiter PUF. This means that our framework with the PUF-based random oracle primitive of Sect. 11 can be applied in practice for the $\mathcal{A}^{NR}$ model for the (very) large k

[20] The reliability-based attack presented in [17] needs careful iterative tuning of parameters and objectives in order to be able to converge to a component Arbiter PUF that has low reliability information leakage at the output (of the iPUF), in particular, the Arbiter PUFs in the upper layer of the iPUF which determine the interpose bit value. The iPUF design relies on a reliability attack to almost never converge to an Arbiter PUF that has low reliability information leakage compared with other Arbiter PUFs (in the lower layer of the iPUF). [17] shows the potential of breaking the iPUF with a reliability-based attack. As it seems to require a lot of manual parameter tuning, it depends on some luck in having a successful attack. It is also unclear how the attack scales to larger parameters, e.g., the (11,11)-iPUF discussed here. The feasibility of this has not yet been demonstrated.

used in this example. We notice that if the adversary can be restricted to much smaller k, then the (well-studied) XOR Arbiter PUF is still useful.

As another example, when using a so-called weak PUF with one CRP, we assume the adversary cannot access the PUF at all, and we are in the $\mathcal{A}^N$-model. For a weak PUF that has a 'polynomial' number of CRPs that can possibly be read out one by one, we may explicitly mention the number k of CRPs an adversary is able to query. In practice, this may be established by introducing a throttling time added to each PUF measurement in normal operation; assuming an adversary can at most have access to the PUF for a small amount of time implies a small enough k. Another option is to implement a logical erasable PUF like interface that uses an authenticated search tree to keep track of how many times a challenge has been queried.

Since, for weak PUFs, we are in the $\mathcal{A}^N$-model or have restricted access by the adversary to a small k (lots smaller than the number of CRPs of the weak PUF), a crypto protocol or primitive that relies on the weak PUF must make use of the fact that the adversary can gather such little knowledge about CRPs that no accurate prediction model can be constructed at all.[21] That is, we assume $\epsilon_{\mathsf{corpred}}$ is much closer to a couple of percentage points (rather than 25% for Arbiter PUF-based designs where machine learning can be used to train on a large number of CRPs).

Comparison to [32]. The security definition of [32] merges steps 1 and 2 of **SecGamePUFCor** of Definition 12 in multiple rounds. In our definition, this is equivalent to having d rounds, where each round i implements part of step 1 for an adaptive selection of k_i challenges and at the end generates $(c_i, r_i) \leftarrow \mathcal{Y}^*_{(i)}$ of step 2 and gives $(c_i, \mathcal{Y}^*_{(i)})$ to the adversary. Each step 1 of a round is restricted in that the adversary may not query $\mathcal{P}(c_i)$ for any i. At the end of step 3, the adversary picks one of the c_h and predicts the corresponding response. In our $\mathcal{A}^U$ model, we may combine all the rounds together into a single step 1 and single step 2 as is done in our definition: Due to the statistically independent selection of new challenges by $\mathcal{A}^U$ in step 1 of each round, $\mathcal{A}^U$ cannot use the received information (challenges c_i) in step 2 of each round for improving an adaptive strategy for querying the PUF in step 1 for subsequent rounds. Since the adversary is in this way limited in its adaptive strategy, the steps of each round may as well be reordered by bundling all steps 1 into one step 1 as in **SecGamePUFCor** and bundling all steps 2 into one step 2. This gives the exact formulation of **SecGamePUF** (with the caveat that the definition in [32] does explicitly talk about information gained from physical side channels, which we do not explicitly cover). Rigorous modeling of how challenge–response pairs (c_j, r_j) are distributed according to some $\mathcal{Y}^*_{(j)}$ due to how a system interfaces with the PUF lacks in [32]. Also, the correlation among CRPs is not explicitly considered. The definition in [32] corresponds to using empty side information $side$ in Lemma 15.

[21] A weak PUF which can (partially) reconfigure its polynomial sized CRP space at regular intervals can possibly allow a larger k like a strong PUF.

10. Secure Sketches and Suitable Codes

By adding an appropriate interface between a PUF and the system using the PUF, PUFs can be used by the system to extract random secret bit strings. The interface is designed to correct measurement errors and, at the same time, amplify the privacy of the corrected responses with respect to adversaries. The main tool for accomplishing this goal is called a secure sketch [19]:

Definition 23. A sketch is a pair $(\mathsf{SSGen}, \mathsf{SSRep})$ of efficient ppt algorithms. For $x \leftarrow \mathcal{X}$, $p \leftarrow \mathsf{SSGen}(x)$ computes helper data $p \in \mathcal{P}$. We assume $x \in \{0,1\}^n$. Let $\tilde{x}$ be a noisy measurement of x, i.e., $\tilde{x} = x + e$ for some error vector $e \leftarrow \mathcal{M}$. On input $\tilde{x}$ and p, $\hat{x} \leftarrow \mathsf{SSRep}(\tilde{x}, p)$ computes an error corrected version of x. We have the following properties:

- Correctness: If the Hamming weight of error vector e is $\mathsf{wt}(e) \leq t$, then $\hat{x} = x$ is correctly reconstructed. If the Hamming weight of e is $\mathsf{wt}(e) > t$, no guarantee can be made; hence, the *failure probability* is at most $Pr(\mathsf{wt}(e) > t)$ where $e \leftarrow \mathcal{M}$ and $\mathcal{M}$ is a probability distribution representing measurement noise.
- Security: The sketch is L-secure if the min-entropy loss

$$\mathsf{H}_\infty(x) - \tilde{\mathsf{H}}_\infty(x \mid p) \leq L.$$

Here,

$$\mathsf{H}_\infty(x) = -\log_2 \max_x \mathsf{Pr}(x \leftarrow \mathcal{X}), \text{ and}$$
$$\tilde{\mathsf{H}}_\infty(x \mid p) = -\log_2 \mathbb{E}_p[\max_x \mathsf{Pr}(x \mid p)], \tag{10}$$

where x and p are jointly distributed according to $x \leftarrow \mathcal{X}$ and $p \leftarrow \mathsf{SSGen}(x)$. $\tilde{\mathsf{H}}_\infty(x \mid p)$ is called the residual min-entropy.

□

A secure sketch can be used to construct a fuzzy extractor that outputs a string that is nearly uniform and can therefore be used as a secret key. The main idea is to use a so-called (strong) randomness extractor to output a nearly uniform string on input x from the secure sketch. Universal hash functions lead to good randomness extractors but add a significant min-entropy loss (due to the leftover hash lemma). By using a cryptographic hash function in the random oracle model, which we (initially) do in this paper (for simplicity), the added min-entropy loss is zero. The hash function reduces the reconstructed $\hat{x}$ down to the number of bits given by (10).

The code-offset secure sketch [19] uses a binary code C with decoding algorithm Dec (e.g., a BCH code or the more recent Polar code achieves small decoding failure probability in practice [50]). It defines

- $p \leftarrow \mathsf{SSGen}(x)$ as $p = x + w$ for a uniformly random code word $w \in C$, and

- $\hat{x} \leftarrow \mathsf{SSRep}(\tilde{x}, p)$ as $\hat{x} = p + \mathsf{Dec}(\tilde{x} + p)$. Here, $\tilde{x} + p = \tilde{x} + x + w = e + w$ and decoding $e + w$ returns w if $\mathsf{wt}(e)$ is small enough. If w is returned, then $\hat{x} = p + w = x$.

The failure probability of the sketch is equal to the decoding failure probability of the decoding algorithm for code C under noise $e \leftarrow \mathcal{M}$.

We want to apply the concept of fuzzy extractors to our framework. In order to do this, we need to work with a slightly more general definition of secure sketches. The code-offset secure sketch construction can be combined with a PUF if we require a code C with the properties listed in the next definition. These properties will allow us to prove a stronger security guarantee than what is offered by a secure sketch. A secure sketch only assumes that the adversary knows the helper data p, while in our case, the adversary can also predict responses that were used by the sketch in the first place.

Definition 24. (*Suitable Codes*) Let C be a set of M binary code words of length n, i.e., $C \subseteq \{0, 1\}^n$ and $|C| = M$. Let Dec be a decoding algorithm for C.

We define the *decoding failure probability* of C with respect to measurement noise $\mathcal{M}$ as

$$\rho = \mathsf{Pr}(w \neq \mathsf{Dec}(w + e), w \leftarrow_R C, e \leftarrow \mathcal{M}).$$

We define the *residual min-entropy* of C with respect to $\mathcal{B}$ as

$$\kappa = \tilde{\mathsf{H}}_\infty(e \mid p) = -\log_2 \mathbb{E}_p[\max_e \mathsf{Pr}(e \mid p)],$$

where e and p are jointly distributed according to $e \leftarrow \mathcal{B}$, $p = w + e$ with $w \leftarrow_R C$.

We define the *min-entropy loss of C due to coset imbalance* with respect to $\mathcal{B}$ and subset size T as $\theta = \log_2(1 + \theta')$ for

$$\theta' = \max_{l \in \{0,1\}^n} \max_{\mathcal{T} \subseteq C + l, |\mathcal{T}| = T} \frac{\mathsf{Pr}(e \in \mathcal{T})}{\mathsf{Pr}(e \in C + l) - \mathsf{Pr}(e \in \mathcal{T})},$$

where $e \leftarrow \mathcal{B}$; θ is a function of subset size T. □

For the PUF-based random oracle construction in Sect. 11, we assume that the decoding failure probability ρ of C with respect to measurement noise $\mathcal{M}$ can be sufficiently accurately simulated (for practical purposes) by modeling $\mathcal{M}$ as a $Ber(\delta_h)$ Bernoulli distribution with δ_h defined in (3) that generates error vectors $e \leftarrow Ber(\delta_h)$ where each error entry has probability δ_h to be equal to 1. This makes the decoding failure probability ρ a function of δ_h.

Also, in our security analysis, the PUF-based random oracle construction, $\mathcal{B}$ will be defined as a $Ber(\tau)$ Bernoulli distribution that generates error vectors $e \leftarrow \mathcal{B}$ where each error entry has probability $\tau > \delta$ to be equal to 1. Hence, the residual min-entropy κ is a function of τ. Below we prove a lower bound on the residual min-entropy for a binary linear code and distribution $Ber(\tau)$.

Lower bound on the residual min-entropy. Consider a $[n, k]$ binary linear code C. For each $x \in \{0,1\}^n$, the set $x + C$ defines a coset of C. Let $l \in x + C$ be of minimal weight; we call this the coset leader. Let L be the set of all coset leaders. Since C is linear, we can write $C + L = \{0,1\}^n$. Hence, we can write $p = c + l$ for $c \in C$ and $l \in L$ in a unique way. For $\mathcal{T} \subseteq \{0,1\}^n$ with cardinality $|\mathcal{T}| = T$, this allows us to derive

$$\begin{aligned}
&\mathbb{E}_p[\max_e \mathsf{Pr}(e \mid p, e \notin \mathcal{T})] \\
&= \sum_p \mathsf{Pr}(p) \left[\max_{e \notin \mathcal{T}} \frac{\mathsf{Pr}(e, p)}{\mathsf{Pr}(p, e \notin \mathcal{T})}\right] \\
&\leq \sum_p \frac{\mathsf{Pr}(p)}{\mathsf{Pr}(p, e \notin \mathcal{T})} [\max_e \mathsf{Pr}(e, p)] \\
&= \sum_{c \in C} \sum_{l \in L} \frac{\mathsf{Pr}(p = c + l)}{\mathsf{Pr}(p = c + l, e \notin \mathcal{T})} [\max_e \mathsf{Pr}(e, p = c + l)] \\
&= \sum_{c \in C} \sum_{l \in L} \frac{\mathsf{Pr}(p = c + l)}{\mathsf{Pr}(p = c + l, e \notin \mathcal{T})} \mathsf{Pr}(e = l, p = c + l) \\
&= \sum_{c \in C} \sum_{l \in L} \frac{\mathsf{Pr}(e \in C + l)/2^k}{\mathsf{Pr}(e \in C + l, e \notin \mathcal{T})/2^k} \mathsf{Pr}(e = l, c) \\
&= \sum_{l \in L} \frac{\mathsf{Pr}(e \in C + l)}{\mathsf{Pr}(e \in C + l, e \notin \mathcal{T})} \mathsf{Pr}(e = l).
\end{aligned}$$

By assuming a min-entropy loss of $\theta = \log_2(1 + \theta')$ of C due to coset imbalance with respect to $Ber(\tau)$ and subset size T, we have

$$\begin{aligned}
&\max_{\mathcal{T}} \frac{\mathsf{Pr}(e \in C + l)}{\mathsf{Pr}(e \in C + l, e \notin \mathcal{T})} \\
&\leq \max_{\mathcal{T} \subseteq C + l} \frac{\mathsf{Pr}(e \in C + l)}{\mathsf{Pr}(e \in C + l, e \notin \mathcal{T})} \\
&= \max_{\mathcal{T} \subseteq C + l} \frac{\mathsf{Pr}(e \in C + l)}{\mathsf{Pr}(e \in C + l) - \mathsf{Pr}(e \in \mathcal{T})} \\
&= 1 + \max_{\mathcal{T} \subseteq C + l} \frac{\mathsf{Pr}(e \in \mathcal{T})}{\mathsf{Pr}(e \in C + l) - \mathsf{Pr}(e \in \mathcal{T})} \\
&\leq 1 + \theta' = 2^{\theta}.
\end{aligned}$$

This allows us to continue the previous derivation and obtain

$$\begin{aligned}
&\mathbb{E}_p[\max_e \mathsf{Pr}(e \mid p, e \notin \mathcal{T})] \\
&\leq \sum_{l \in L} 2^{\theta} \cdot \mathsf{Pr}(e = l) \\
&= 2^{\theta} \cdot \mathsf{Pr}(e \in L).
\end{aligned}$$

We notice that, for $\mathcal{T}$ equal to the empty set, we have equalities in the derivations above for $\theta = 0$. This proves

$$\tilde{\mathsf{H}}_\infty(e \mid p, e \notin \mathcal{T}) \geq -\theta + \tilde{\mathsf{H}}_\infty(e \mid p), \text{ and}$$
$$\tilde{\mathsf{H}}_\infty(e \mid p) = -\log_2 \mathsf{Pr}(e \in L).$$

Notice that the cardinality $|L| = 2^{n-k}$ and for $r/n \leq 1/2$,

$$\frac{2^{h(r/n)n}}{\sqrt{2n}} \leq \frac{2^{h(r/n)n}}{\sqrt{8r(1-r/n)}} \leq \sum_{i=0}^{r} \binom{n}{i} \leq 2^{h(r/n)n},$$

where $h(.)$ is the binary entropy function. By choosing

$$h(r/n) = 1 - (k - \log_2 \sqrt{2n})/n,$$

we have that L covers at most as many vectors as the Hamming sphere around the all-zero vector of radius r. In truth, L represents a Voronoi region and may not be spherical in shape; nevertheless, for upper bounding $\mathsf{Pr}(e \in L)$, this is a reasonable approximation (in order to get an idea of how to set parameters). By morphing the actual Voronoi region L into a sphere, we exchange less likely error probabilities by more likely error probabilities. For $r/n < \tau < 1/2$, this implies

$$\begin{aligned}\mathsf{Pr}(e \in L) &\leq \sum_{i=0}^{r} \binom{n}{i} \tau^i (1-\tau)^{n-i} \\ &\leq e^{-n(\tau - r/n)^2 \cdot (1-2\tau)^{-1} \ln((1-\tau)/\tau)} \\ &\leq e^{-2n(\tau - r/n)^2},\end{aligned}$$

where the right-hand side upper bounds follows from Hoeffding's inequality.[22] We have

$$\begin{aligned}\tilde{\mathsf{H}}_\infty(e \mid p) &\geq -\log_2 e^{-2n(\tau - r/n)^2} \\ &= 2n(\tau - r/n)^2 / \ln 2 \\ &= 2n(\tau - h^{-1}(1 - (k - \log_2 \sqrt{2n})/n))^2 / \ln 2.\end{aligned}$$

In order to have a small enough decoding failure probability, we cannot exceed the capacity of the Bernoulli distribution with bit error rate δ, i.e., we must have $k \leq$

[22] Hoeffding [51] proved a stronger upper bound: The binomial sum is upper bounded by

$$\begin{aligned}&\left(\frac{\tau}{r/n}\right)^{(r/n)n} \left(\frac{1-\tau}{1-r/n}\right)^{(1-r/n)n} \\ &\leq 2^{-n(h(\tau) - h(r/n) - (\tau - r/n)\log_2((1-\tau)/\tau))} \\ &\leq e^{-n(\tau - r/n)^2 \cdot (1-2\tau)^{-1} \ln((1-\tau)/\tau)},\end{aligned}$$

where $h(.)$ is the binary entropy function.

$(1 - h(\delta))n$. Let

$$k = (1 - h(\delta'))n + \log_2 \sqrt{2n} \text{ with } \delta < \delta' \leq \tau.$$

This yields the following lemma.

Lemma 25. *Let C be a binary linear $[n, k]$ code. Suppose that dimension $k = (1 - h(\delta'))n + \log_2 \sqrt{2n}$, where $h(.)$ is the binary entropy function. Let $\mathcal{B}$ be the Bernoulli distribution with bit error rate $1/2 > \tau \geq \delta'$. Then, the residual min-entropy of C with respect to $e \leftarrow \mathcal{B}$ has lower bound*

$$\begin{aligned}\tilde{\mathsf{H}}_\infty(e \mid p) &\geq (\tau - \delta')^2 n \cdot \frac{1}{1 - 2\tau} \log_2 \left(\frac{1 - \tau}{\tau} \right) \\ &\geq \frac{2}{\ln 2} (\tau - \delta')^2 n.\end{aligned}$$

Suppose that C has a min-entropy loss of θ due to coset imbalance with respect to $\mathcal{B}$ and subset size $T = |\mathcal{T}|$. Then,

$$\tilde{\mathsf{H}}_\infty(e \mid p, e \notin \mathcal{T}) \geq -\theta + \tilde{\mathsf{H}}_\infty(e \mid p).$$

□

Coarse upper bound on the min-entropy loss due to coset imbalance. By the triangle inequality, we have $wt(c + l) \leq wt(c) + wt(l)$. This allows us to derive

$$\begin{aligned}\mathsf{Pr}(e \in C + l) &= \sum_{c \in C} \tau^{wt(c+l)} (1 - \tau)^{n - wt(c+l)} \\ &\geq \tau^{wt(l)} (1 - \tau)^{n - wt(l)} \cdot \sum_{c \in C} (\frac{\tau}{1 - \tau})^{wt(c)}.\end{aligned}$$

Since l is the coset leader in $C + l$, if $\mathcal{T} \subseteq C + l$, then

$$\mathsf{Pr}(e \in \mathcal{T}) \leq T \cdot \tau^{wt(l)} (1 - \tau)^{n - wt(l)}.$$

We define the weight enumerator of code C as the polynomial

$$W(x) = \sum_{c \in C} x^{wt(c)}.$$

This proves

$$\frac{\mathsf{Pr}(e \in \mathcal{T})}{\mathsf{Pr}(e \in C + l)} \leq \frac{T}{W(\frac{\tau}{1 - \tau})}.$$

From this, we conclude

$$\frac{\mathsf{Pr}(e \in C + l)}{\mathsf{Pr}(e \in C + l) - \mathsf{Pr}(e \in \mathcal{T})} \leq \frac{1}{1 - \frac{T}{W(\frac{\tau}{1-\tau})}}$$

for all l with $\mathcal{T} \subseteq C + l$. This proves

$$1 + \theta' \leq \frac{1}{1 - T/W(\frac{\tau}{1-\tau})}.$$

Let us assume that C looks like a Binomial distribution. That is, for small x, the weight enumerator of code C is approximately a scaled version of the polynomial corresponding to the whole space $C = \{0, 1\}^n$,

$$W(x) \approx \frac{1}{2^{n-k}} \sum_{i=0}^{n} \binom{n}{i} x^i.$$

Assume, therefore, that there exists a constant γ such that

$$W(x) \geq \frac{\gamma}{2^{n-k}} \sum_{i=0}^{n} \binom{n}{i} x^i.$$

This assumption gives

$$W\left(\frac{\tau}{1-\tau}\right) \geq \frac{\gamma}{2^{n-k}} \left(1 + \frac{\tau}{1-\tau}\right)^n = \gamma 2^{-n \log_2(1-\tau) - (n-k)}.$$

For $k = (1 - h(\delta'))n + \log_2 \sqrt{2n}$, we have

$$1 + \theta' \leq \frac{1}{1 - \gamma^{-1} T \cdot 2^{-(-\log_2(1-\tau) - h(\delta'))n - \log_2 \sqrt{2n}}}.$$

For $\delta' = 0.074$ and $\tau = 0.25$, this yields

$$1 + \theta' \leq 1/(1 - \gamma^{-1} T \cdot 2^{-0.0344 \cdot n - \log_2 \sqrt{2n}}).$$

For $n = 2^{13}$, we have that if $T \leq \gamma \cdot 2^{0.0344 \cdot n + 7 - 1} = \gamma \cdot 2^{287.5}$, then $1 + \theta' \leq 2$; hence, $\theta \leq 1$. This holds for $T \leq 2^{\kappa}$ and $\gamma \approx 2^{-31.5}$ for $\kappa = 256$.

Lemma 26. *Let C be a binary linear $[n, k]$ code. Suppose that dimension $k = (1 - h(\delta'))n + \log_2 \sqrt{2n}$, where $h(.)$ is the binary entropy function. Let $\mathcal{B}$ be the Bernoulli distribution with bit error rate $1/2 > \tau \geq \delta'$. Suppose that the weight enumerator*

polynomial $W(x)$ of C behaves like a scaled version of the Binomial distribution

$$W(x) \geq \frac{\gamma}{2^{n-k}} \sum_{i=0}^{n} \binom{n}{i} x^i$$

for some constant γ. Then, for

$$T \leq \gamma \cdot 2^{(-\log_2(1-\tau)-h(\delta'))n+(\log_2\sqrt{2n})-1},$$

the min-entropy loss of C due to coset imbalance with respect to $\mathcal{B}$ and subset size T is at most $\theta \leq 1$. □

Secrecy capacity. We notice that a polar code can be used to achieve the so-called secrecy capacity of the wire-tap channel asymptotically [52]. Here, the legitimate user receives messages over the main channel, which produces noise according to $\mathcal{M}$ (in our notation). The adversary receives the same messages over a wire-tap channel which adds noise from $\mathcal{B}$ (in our notation). Since $\delta < \tau$, the secrecy capacity is equal to $h(\tau) - h(\delta)$, where $h(.)$ is the binary entropy function. This means that messages of length $(h(\tau) - h(\delta)) \cdot n$ bits can be encoded in n-bit polar code words such that the legitimate receiver can reconstruct the message bits after receiving the noisy message over the main channel, and the adversary receives a noisy version of the message which has close to zero mutual (Shannon) information with the reconstructed message. In our definition, we work with min-entropy as this has shown to model the best possible prediction of adversaries. (For more understanding, see the literature on fuzzy extractors and secure sketches.)

11. PUF-Based Random Oracle

As a powerful example of the application of our theoretical framework, we want to realize a PUF-based random oracle (PRO) as defined next. The PRO primitive has besides GetResponse two other algorithms: GetIO generates input–output pairs (aux, s), where the output s represents a random bit string about which the adversary can only learn negligible information given knowledge of the input aux and a prediction model of the underlying PUF—the security is defined within the language of our framework. GetOutput takes the input aux of a pair and is able to reconstruct the output s by calling GetResponse. Correctness defines the failure probabilities of both algorithms and defines that GetOutput correctly recovers the secret random bit string s if it does not fail.

In essence, the PRO primitive behaves exactly like a PUF, but now without significant measurement noise (we have small failure probabilities) and without the adversary being able to model the output of the PRO primitive, so, no prediction model with significant accuracy. So, both the reliability and software unclonability properties of the PUF are amplified.

The PRO primitive itself is useful in larger systems/protocols that use PUFs as a basis for their security. In fact, other systems/protocols often assume an idealized picture where

the PUF is equated to pseudorandom function. The PRO primitive, which is based on the realistic assumption that an adversary has a software model of the underlying PUF with significant accuracy, provides justification to the security of these systems/protocols.

Definition 27. (*PUF-based Random Oracle (PRO)*) We define a PUF-based random oracle as a triple

$$\mathsf{PRO} = (\textsc{GetResponse}, \textsc{GetIO}, \textsc{GetOutput})$$

with the following properties

- **Functionality.** Algorithms (GetIO, GetOutput) have access to a PUF $\mathcal{P}$ through GetResponse; let $\Upsilon^{\mathcal{P}}$ be the set of corresponding system-induced distributions over CRPs of $\mathcal{P}$. Upon input $seed$, GetIO either generates a pair $(\mathsf{aux}, s) \leftarrow \textsc{GetIO}(seed)$ or fails. Upon input aux, GetOutput either generates $\hat{s} \leftarrow \textsc{GetOutput}(\mathsf{aux})$ or fails.
- **Correctness.** We call $(F_{\mathsf{io\text{-}ro}}, F_{\mathsf{out\text{-}ro}})$ a pair of failure probabilities for PRO if
  - the probability $\textsc{GetIO}(seed)$ fails over a random uniformly chosen $seed$ is at most $F_{\mathsf{io\text{-}ro}}$,
  - the probability that $\textsc{GetOutput}(\mathsf{aux})$ fails over $(\mathsf{aux}, \cdot) \leftarrow \textsc{GetIO}(seed)$ is at most $F_{\mathsf{out\text{-}ro}}$, and
  - if $(\mathsf{aux}, s) \leftarrow \textsc{GetIO}(seed)$ and $\hat{s} \leftarrow \textsc{GetOutput}(\mathsf{aux})$ (both do not fail), then $\hat{s} = s$ with probability at least $1 - negl(\kappa)$, where κ represents the bit security defined next.
- **Security.** Let $Q^{\mathcal{P}}$ output problem instances aux by calling $(\mathsf{aux}, s) \leftarrow \textsc{GetIO-RO}_h$; the associated problem for an adversary $\mathcal{A}$ is to guess the correct solution s (when $\textsc{GetIO-RO}_h$ does not fail) by playing $\mathbf{SecGameSys}(Q, \mathcal{P}, \Upsilon^{\mathcal{P}}, \mathcal{A}, k, T)$, where $\Upsilon^{\mathcal{P}}$ is the set of system-induced CRP distributions used by PRO. Suppose that the probability that the adversary wins is at most $\epsilon_{\mathsf{win}}(k, T)$ as a function of k (number of PUF queries by $\mathcal{A}$) and T (run time of $\mathcal{A}$). We say that PRO has κ-bit security for k PUF queries with respect to $\mathcal{A}$ with run time $poly(t)$ if, for $T = poly(t)$,

$$\epsilon_{\mathsf{win}} \leq T 2^{-\kappa}.$$

□

In the introduction, we sketched (for future work) how PRO can be used in combination with an OTS-SKE signature scheme to construct a remote attestation (RA) protocol which does not rely on confidential digital computing. The failure probabilities of PRO will directly translate into failure of the RA protocol. Typical probabilities of 0.001 or 0.0001 will require the remote verifier to repeat the RA protocol once every 1000 of 10.000 times, acceptable in practice.

Algorithms 4 and 5 code the interface of an PRO primitive based on a PUF $\mathcal{P}$. To get an input–output pair, algorithm 4 calls GetResponse multiple times through $\textsc{GetReliableCRP}_h$ which is, in turn, called multiple times in $\textsc{Neumann-GetReliableCRP}_h$.

This leads to a pre-challenge vector c_{prevec} of λn bits. (We will discuss how to compress this in the next remark.)

We follow the code-offset sketch construction [19] based on a suitable code, see Definition 24, to compute helper data p, and we hash the corresponding response vector r_{vec} down to κ bits, where κ is the residual min-entropy of code C with respect to an appropriate Bernoulli distribution (discussed later).

To get an output from an input, algorithm 5 first calls GetResponse multiple times to get an estimate $\tilde{r}_{\mathsf{vec}}$ of r_{vec}. We use the code-offset sketch construction to first decode $p + \tilde{r}_{\mathsf{vec}}$ to a code word $\tilde{w}$ which is then used to recover $\hat{r}_{vec}$. The result is hashed down to $\hat{s}$. The failure probability $\mathsf{Pr}(\hat{s} \neq s) = \mathsf{Pr}(\hat{r}_{\mathsf{vec}} \neq r_{\mathsf{vec}}) = \mathsf{Pr}(\hat{w} \neq w)$ is equal to the decoding failure probability ρ of C with respect to a distribution $\mathcal{M}$ representing measurement noise: Since only reliable CRPs from GetReliableCRP$_h$ are used, we may assume that $\mathcal{M}$ behaves like $Ber(\delta_h)$ and $\rho(\delta_h)$ is a function of δ_h.

The PRO interface represented by GetIO-RO$_h$ and GetOutput-RO should have a small (decoding) failure probability ρ for typical PUF measurement noise and should produce a secret s that has κ bits, where κ is at least the security parameter (bit security) of the system using secret s (as a result of interfacing with RO).

In Algorithms 4 and 5, we use a hash Hash_κ which extracts a random κ-bit string s from an n-bit response vector. For simplicity, we assume Hash_κ in the random oracle model, but note that this can be replaced by a strong randomness extractor based on a universal family of hash functions [53], see our security analysis below.

Algorithm 4 Generating PRO input–output pairs

procedure GetIO-RO$_h$
  for $j \in \{0, \ldots, n-1\}$ **do**
    $(c_{\mathsf{pre},j}, r_j) \leftarrow$ Neumann-GetReliableCRP$_h$
  end for
  $c_{\mathsf{prevec}} = \{c_{\mathsf{pre},j}\}_{j=0}^{n-1}$; $r_{\mathsf{vec}} = \{r_j\}_{j=0}^{n-1}$
  $w \leftarrow_R C$; $p = r_{\mathsf{vec}} + w$
  $s = \mathsf{Hash}_\kappa(r_{\mathsf{vec}})$
  return $((c_{\mathsf{prevec}}, p), s)$
end procedure

Algorithm 5 Recovering PRO output from input

procedure GetOutput-RO(c_{prevec}, p)
  $\{c_{\mathsf{pre},j}\}_{j=0}^{n-1} = c_{\mathsf{prevec}}$
  for $j \in \{0, \ldots, n-1\}$ **do**
    $\tilde{r}_j \leftarrow$ GetResponse($c_{\mathsf{pre},j}$)
  end for
  $\tilde{r}_{\mathsf{vec}} = \{\tilde{r}_j\}_{j=0}^{n-1}$
  $\tilde{w} \leftarrow \mathsf{Dec}(p + \tilde{r}_{\mathsf{vec}})$; $\hat{r}_{\mathsf{vec}} = p + \tilde{w}$; $\hat{s} = \mathsf{Hash}_\kappa(\hat{r}_{\mathsf{vec}})$
  return $\hat{s}$
end procedure

Pre-challenge vector compression. We discussed how GetReliableCRP$_h$ can use a random input *seed* which which pre-challenges are computed. We argued that this avoids the use of a TRNG. Here, we may also extend GetIO-RO$_h$ with an input *seed* as a goal to not only avoid the use of a TRNG in the underlying calls to GetReliableCRP$_h$ but also to provide a compressed representation of the generated vector of pre-challenges c_{prevec}. The main idea is to keep the loop count number j for the loop that calls Neumann-GetReliableCRP$_h$, the loop count number b for the loop that defines Neumann-GetReliableCRP$_h$, the index $i = 0$ or $i = 1$ for the two calls to GetReliableCRP$_h$ in a single loop in Neumann-GetReliableCRP$_h$, and a loop count number a in GetReliableCRP$_h$. In GetReliableCRP$_h$, we call GetResponse(c_{pre}) for

$$c_{\mathsf{pre}} = \mathsf{Hash}(seed \| a \| i \| b \| j). \tag{11}$$

We remember the loop counts a and b for each j of the associated $c_{\mathsf{pre}} = \mathsf{Hash}(seed \| a \| 0 \| b \| j)$ returned by Neumann-GetReliableCRP$_h$ in GetIO-RO$_h$. We denote these loop counts by a_j and b_j. The sequence of pre-challenges returned by GetIO-RO$_h$ are now defined by

$$c_{\mathsf{pre},j} = \mathsf{Hash}(seed \| a_j \| 0 \| b_j \| j).$$

Hence, rather than storing c_{prevec}, we store

$$(seed, \{(a_j, b_j)\}_{j=1}^{n}). \tag{12}$$

Notice that we can also use *seed* as input to a hash function for selecting a random code word $w \leftarrow_R C$; also, here, we can avoid using a TRNG.

In order to avoid collisions in the hash function, we need all a_j to be of fixed length/size, say n_a bits, and all b_j to be of fixed size, say n_b bits. This means that the respective loops may run out before a 'good' pre-challenge is found, and this leads to a failure probability for GetIO-RO$_h$. We will analyze this in the next theorem. As a remark, we note that we have compressed the λn bits of c_{prevec} down to $(n_a + n_b) \cdot n$ bits.

Failure detection. We notice that GetIO-RO may also output another hash $\mathsf{Hash}'(s)$ of s. GetOutput-RO can use this hash of s to verify against the same hash of $\hat{s}$. This allows it to detect whether there is a failure to produce $\hat{s} = s$. If detected, then GetOutput-RO outputs a fail.

Theorem 28. (PUF-based Random Oracle theorem) GetResponse *with* GetIO-RO *and* GetOutput-RO *of Algorithms 4 and 5 with pre-challenge vector compression and a collision-resistant hash* Hash' *with security parameter*[23] κ *for failure detection define a PUF-based random oracle* PRO*:*

- *With respect to security,*

[23] See Definition 30.

– *Suppose that $\mathcal{P}$ has challenge space $C_{\mathcal{P}} = \{0,1\}^{\lambda}$ with correlation bias at most*

$$\epsilon_{\mathsf{corbias}} \leq \frac{2^{-(\kappa+5.946)}}{nh2^{n_a+1}(1+2^{n_b})}.$$

over the canonical system-induced CRP distribution $\mathcal{Y}_1^*$, where n is the length of the code words in C and $\lambda = \Omega(\kappa)$.
– *Let $\mathcal{A}^x$ be an adversarial model with $\mathcal{A}^x \subseteq \mathcal{A}^U$.*
– *Suppose that $\mathcal{P}$ is a $(k, t, \epsilon_{\mathsf{corpred}})$-secure PUF for correlations with respect to $\mathcal{A}$ that are within the $\mathcal{A}^x$-model and with respect to system-induced CRP distribution $\mathcal{Y}^*_{\mathsf{neu},h}$. Define*

$$\tau = 1/2 - (h2^{n_a}\epsilon_{\mathsf{corbias}} + \epsilon_{\mathsf{corpred}})$$

and let $\mathcal{B}$ be the distribution that generates statistically independent errors $\hat{e}_j \leftarrow Ber(\tau)$.
– *Assume that $Q^{\mathcal{P}}$ is c_{sep}-separable within the $\mathcal{A}^x$-model for $c_{\mathsf{sep}} = 2^{\kappa-5.946}$.*
– *Suppose that C is a binary linear code and has a min-entropy loss of at most θ due to coset imbalance with respect to $\mathcal{B}$ and subset size 2^{κ}. (For θ to be a small constant, this requires the dimension of C to be $\Omega(\kappa)$.) Suppose that the residual min-entropy of C with respect to $\mathcal{B}$ is at least $2\kappa + \theta$.*

Then, for $T = poly(t)$ we have $Q^{\mathcal{P}}$ is $(k, T, \epsilon_{\mathsf{win}})$-system secure with respect to all $\mathcal{A}$ within the $\mathcal{A}^x$-model for system-induced distribution $\mathcal{Y}^*_{\mathsf{neu},h}$ for

$$\epsilon_{\mathsf{win}} \leq T2^{-\kappa}.$$

We may replace hash Hash_{κ} in the random oracle model by a strong randomness extractor; this requires $c_{\mathsf{sep}} = 2^{\kappa-10.892}$ and residual min-entropy at least $3\kappa + \theta + 10.892$ (an additional min-entropy loss due to the leftover hash lemma).
– *With respect to correctness, we have a pair of failure probabilities $(F_{\mathsf{io\text{-}ro},h}, F_{\mathsf{out\text{-}ro},h})$ defined by*

$$\begin{aligned}
F_{\mathsf{rel},h} &= (1 - \mathbb{E}_{\mathcal{Y}_1}[p_c^h + (1-p_c)^h])^{2^{n_a}},\\
F_{\mathsf{neu},h} &\leq (1 - 2q_0^{\mathcal{Y}_h}q_1^{\mathcal{Y}_h} \cdot (1 - h2^{n_a}\epsilon_{\mathsf{corbias}})(1 - 2F_{\mathsf{rel},h}) + 2F_{\mathsf{rel},h})^{2^{n_b}},\\
F_{\mathsf{io\text{-}ro},h} &\leq n \cdot F_{\mathsf{neu},h},\\
F_{\mathsf{out\text{-}ro},h} &\approx \rho(\delta_h).
\end{aligned}$$

□

Interpretation—A TCB without confidential digital computing or digital secrets. Assuming that current-state-of-the-art advanced ML attacks on the iPUF train prediction models with accuracy at most a concrete number of percentage points less than 100%, then concrete parameter settings exist for meeting the conditions of Theorem 28

with $\mathcal{A}^x = \mathcal{A}^U$. (An example is given below.) The parameter setting gives rise to a more efficient PUF-based random oracle PRO (in computation and number of calls to GETRESPONSE) if the prediction accuracy is lower (e.g., more in the range of 75%).

The adversarial $\mathcal{A}^U$-model is very strong: The TCB does not include any confidential digital computing or digital secrets. For example, using a fuzzy extractor (that is, a combination of a secure sketch with a randomness extractor) without considering the adversarial capability of being able to train a prediction model for the underlying PUF means that no CRPs should be revealed to the adversary—the very weak $\mathcal{A}^N$-model. This means that the fuzzy extraction computation must be done in a confidential computing environment, e.g., implemented by specialized isolated hardware or a general purpose secure processor architecture. But even in this weak model, helper data can reveal information about the bits in the response vector; after all, we only require a residual min-entropy of a much smaller number of bits compared to the length of the vector out of which a secret bit string is extracted. And information about response bits may allow an adversary to train some sort of prediction model—at least in theory, this may be possible[24] (although we do not know how to do this in practice). Our framework clarifies the situation. Our analysis shows that a PUF-based random oracle can be constructed and implemented, and without a TCB that requires some form of confidential digital computing.

We notice that we may use hardware isolation to further restrict access to the PUF. For example, the PRO primitive may execute in its own 'enclave' and allow only the PRO enclave access to GETRESPONSE. The PRO enclave may use an authenticated search tree approach like the one for programmable erasable PUFs in [32] to limit the GETRESPONSE usage per challenge. This enforces the $\mathcal{A}^{R-x}$-model (with x at least $h+1$). Other system applications may connect to the PRO-primitive through local attestation (which can be made secret-free by using Sanctum's approach that implements a physically isolated channel between enclaves using the concept of mailboxes [54]).

The $\mathcal{A}^R$-model ($\mathcal{A}^R = \mathcal{A}^{R-1}$) can only be realized if $h = 1$ and GETIO-RO$_h$ computations cannot be observed. This either assumes PRO is part of a system enclave where adversaries can only be present during 'normal operation' (during which only GETOUTPUT-RO is called) or we have a TCB that computes GETIO-RO$_1$ in a confidential executing environment.

Concrete parameter setting. The various conditions of Theorem 28 are realistic: We already discussed why $\epsilon_{\mathsf{corbias}}$ is expected to be $negl(\lambda)$ for $\mathcal{Y}_1$ close to the uniform distribution. (And for this reason we assume $\lambda = \Omega(\kappa)$ so that the upper bound (5) on $\epsilon_{\mathsf{corbias}}$ can hold.) We are in the $\mathcal{A}^U$ model where the adversary has access to GETRESPONSE. (And may have additional access restrictions.) We expect $\tau \geq 25\%$ for reasonable training data set sizes k and large t; hence, the condition $T = poly(t)$ just translates into assuming feasible computing times for a real adversary in practice. The separability assumption states in this context that we do not know how to find a much more efficient method that learns how to predict the response vector r'_{vec} from training data together with the helper data aux. (The winning probability cannot be improved beyond an exponentially large multiplicative factor of order 2^κ.)

[24] The computational fuzzy extractor, called LPN-PUF, does show that its published helper data cannot be used to extract underlying response information. But it still needs confidential computing in its TCB.

Let us consider $\delta = 0.1$ and $\tau = 0.25$ with $2q_0^{\mathcal{Y}_1}q_1^{\mathcal{Y}_1} \approx 2 \cdot 0.55 \cdot 0.45 = 0.495$. Suppose that a small value for h, say $h = 4$, pushes δ down to $\delta_h = 0.06$, a few percentage points less.

Suppose that $q_r^{\mathcal{Y}_h} \approx q_r^{\mathcal{Y}_1}$, then $2q_0^{\mathcal{Y}_h}q_1^{\mathcal{Y}_h} \approx 0.495$. By choosing $n_a = \log\kappa$ and $n_b = \log\kappa$ and by aiming for $\kappa = 256$, we achieve a small failure probability $F_{\mathsf{io\text{-}ro},h}$ (exponentially small in κ).

Let $\delta' = 0.074$. We choose a polar code of length $n = 2^{13}$ and dimension $k = (1-h(\delta'))n + \log_2\sqrt{2n} = 0.619 \cdot n + 7 = 5080$. The polar code is known for achieving capacity $(1 - h(\delta))n = 0.673 \cdot n$ for large n. In our case, $0.673 \cdot n = 5509 \gg 5080$ and we can expect that the decoding failure probability $\rho(\delta_h)$ is small; hence, $F_{\mathsf{out\text{-}ro},h}$ is small.[25]

Notice that k is 20 times larger than $\kappa = 256$ making $\theta \leq 1$ a realistic assumption on the min-entropy loss due to coset imbalance. (The size of a coset is about $2^{20\cdot\kappa}$, where $T = poly(t) \ll 2^{\kappa}$.) If the weight enumerator polynomial of C looks like a scaled version of the Binomial distribution (i.e., C looks like a random code), then a coarse upper bound follows from Lemma 26 with $T \leq \gamma \cdot 2^{287.5}$; hence, $\gamma \approx 2^{-31.5}$ allows $T \leq 2^{\kappa}$ with $\theta = 1$.

We use a strong randomness extractor, and this requires a residual min-entropy of at least $3\kappa + \theta + 10.982 = 3\kappa + 11.982$. We apply the best lower bound on the residual min-entropy of Lemma 25 and obtain 804.4. This restricts κ to ≤ 264, and we can indeed realize $\kappa = 256$.

By substituting $n_a = n_b = \log\kappa = 8$, we require $\epsilon_{\mathsf{corbias}} \leq 2^{-(\kappa+5.946)}/(2nh\kappa(1+\kappa)) \approx 2^{-294}$. We may assume that each component Arbiter PUF in, e.g., the iPUF design gets its own challenge of 64 bits generated out of a pre-challenge by means of a PRG or hash-based scheme. By using multiple numbers $a \geq 5$ of component Arbiter PUFs, we have $64 \cdot a \geq 294$ and our earlier argument holds and it is reasonable to assume that $\epsilon_{\mathsf{corbias}}$ is indeed small enough and satisfies the condition.

Proof of Theorem 28: Correctness. See Definition 3, the probability that all h measurements for a $c = \mathsf{Hash}(c_{\mathsf{pre}})$ in GetReliableCRP$_h$ agree is equal to $p_c^h + (1-p_c)^h$. Since we assume Hash in the random oracle model with respect to $\mathcal{Y}_1$ (corresponding to GetResponse), the different challenges $c = \mathsf{Hash}(c_{\mathsf{pre}})$ for c_{pre} defined by (11) for $0 \leq j \leq n-1$, $0 \leq a \leq 2^{n_a}-1$, $0 \leq b \leq 2^{n_b}-1$, and $i \in \{0,1\}$ are randomly drawn from $\mathcal{Y}_1$. This shows that each while loop in GetReliableCRP$_h$ called during the execution of GetIO-RO$_h$ will take $(\mathbb{E}_{\mathcal{Y}_1}[p_c^h + (1-p_c)^h])^{-1}$ iterations in expectation. For small h, this is a small number, and for this reason, we can set n_a to a small number. If GetReliableCRP$_h$ cannot find a reliable CRP after 2^{n_a} loop iterations, then GetReliableCRP$_h$ fails. This leads to *failing probability*

$$F_{\mathsf{rel},h} = (1 - \mathbb{E}_{\mathcal{Y}_1}[p_c^h + (1-p_c)^h])^{2^{n_a}}, \tag{13}$$

which we can design to be very small for n_a large enough. (Notice that $F_{\mathsf{rel},1} = 0$.)

[25] Precise simulation for a polar code is beyond the scope of this paper; this parameter setting only serves as an example since polar codes only achieve capacity for large n, and a realistic parameter setting may need a larger n in order to achieve small $\rho(\delta_h)$.

Our analysis of NEUMANN-GETRELIABLECRP$_h$ shows in (6) that, *conditioned on* GETRELIABLECRP$_h$ *not failing*, the probability that an iteration in NEUMANN-GETRELIABLECRP$_h$ produces a final output is equal to

$$2q_0^{\mathcal{Y}_h} q_1^{\mathcal{Y}_h} \cdot (1 \pm \epsilon_{\mathsf{corbias},h}),$$

where $\epsilon_{\mathsf{corbias},h} \approx h2^{n_a}\epsilon_{\mathsf{corbias}}$. Hence, the *failing probability* of NEUMANN-GETRELIABLECRP$_h$ not being able to find distinct responses $r_0 \neq r_1$ (that lead to the final output) in 2^{n_b} loop iterations is equal to

$$\begin{aligned} & F_{\mathsf{neu},h} \qquad (14)\\ & = \begin{pmatrix} (1 - 2q_0^{\mathcal{Y}_h} q_1^{\mathcal{Y}_h} \cdot (1 \pm \epsilon_{\mathsf{corbias},h}))(1 - F_{\mathsf{rel},h})^2 \\ +(1 - (1 - F_{\mathsf{rel},h})^2) \end{pmatrix}^{2^{n_b}} \\ & \leq (1 - 2q_0^{\mathcal{Y}_h} q_1^{\mathcal{Y}_h} \cdot (1 - \epsilon_{\mathsf{corbias},h})(1 - 2F_{\mathsf{rel},h}) + 2F_{\mathsf{rel},h})^{2^{n_b}}, \end{aligned}$$

which can be made small for large enough n_b.

This leads in turn to the failing probability $F_{\mathsf{io\text{-}ro},h}$ of GETIO-RO$_h$. Since GETIO-RO$_h$ does not fail only if none of the n calls to NEUMANN-GETRELIABLECRP$_h$ fail, we have

$$F_{\mathsf{io\text{-}ro},h} = 1 - (1 - F_{\mathsf{neu},h})^n \leq n \cdot F_{\mathsf{neu},h}. \qquad (15)$$

For appropriately chosen n_a and n_b in relation to n, this can be made small in practice.

We already discussed the failure probability $F_{\mathsf{out\text{-}ro},h}$ of GETOUTPUT-RO which is equal to the decoding failure probability ρ of C with respect to distribution $\mathcal{M}$ representing measurement noise. We may assume that ρ is approximately equal to the failure probability of C with respect to distribution $Ber(\delta_h)$:

$$F_{\mathsf{out\text{-}ro},h} \approx \rho(\delta_h), \qquad (16)$$

where δ_h indicates the dependency of ρ on δ_h for the Bernoulli distribution.

Larger h for a given code C implies smaller $F_{\mathsf{out\text{-}ro},h}$. One needs to design a good combination of h with a suitable code C such that $F_{\mathsf{out\text{-}ro},h}$ is small enough and the residual min-entropy κ is large enough[26] for generating secrets. □

Proof of Theorem 28: Security. We notice that **HardnessQError** fits the definition of a security game, see [41]: 'An n-bit security game is a game played by an adversary interacting with a challenger. At the beginning of the game, the challenger chooses a uniformly random secret $x \in \{0, 1\}^n$, represented by the random variable X. At the end of the game, the adversary outputs some value v, represented by the random variable V. The goal of the adversary is to output v such that $R(x, v) = 1$, where R is a Boolean function. The adversary may output a special symbol $\perp$ such that $R(x, \perp) = 0$ for any x. During the game, the adversary or challenger may obtain a sample from a distribution

[26] Even if κ is small, we can use code C multiple, say m, times to scale κ up to $m\kappa$. This does mean that $F_{\mathsf{out\text{-}ro},h}$ will be multiplied with m as well.

Q. The success probability of the adversary is $Pr(R(X, V) = 1, V \neq \perp)$, where the probability is taken over the randomness of the entire security game, including the randomness of the adversary.' In our case, the challenger obtains a sample $\hat{g} \leftarrow \hat{Q}$ (the role of X) and the adversary guesses $\hat{s} \neq \perp$ (the role of V) given $\hat{g}$. The success probability of the adversary is $\mathsf{Pr}(\textsc{VerH}(\hat{g}, \hat{s})) \leq \epsilon_{\mathsf{hard}}$ ($\textsc{VerH}$ plays the role of R). (The $\perp$ symbol can capture the special case in which $\hat{Q}$ fails to produce a proper problem instance $\hat{g}$ or the adversary declares a failure of the attack.) The definition of bit security is consistent with the so-called bit security for a primitive based on a security game.

$\hat{Q}$ may query some probability distribution (other than the $\mathcal{E}$ or $\mathcal{R}$ mentioned above). For example, $\hat{Q}$ may query $q_r^{(\mathcal{Y}^*_{\mathsf{neu},h})^{\times d}}$ as a distribution over response vectors r of length d. Now, we can apply Theorem 1 in [41] together with (5) and noting that the derivation of Lemma 11 proves that $q_r^{(\mathcal{Y}^*_{\mathsf{neu},h})^{\times d}}$ and the uniform distribution $\mathcal{U}$ over $\{0,1\}^d$ form a so-called $2^{-(\kappa+5.946)/2}$-Hellinger close pair:

Lemma 29. *Let $\hat{Q}$ with query access to the uniform distribution $\mathcal{U}$ over $\{0,1\}^d$ be $(\kappa + 5.946)$-bit secure. Suppose that PUF $\mathcal{P}$ has correlation bias at most $\epsilon_{\mathsf{corbias}}$ over the distribution $\mathcal{Y}_1^*$ of CRPs generated by* $\textsc{GetResponse}$. *Consider $q_r^{(\mathcal{Y}^*_{\mathsf{neu},h})^{\times d}}$ with n_a and n_b satisfying (5). Then, $\hat{Q}$ with query access to $q_r^{(\mathcal{Y}^*_{\mathsf{neu},h})^{\times d}}$ is κ-bit secure.* □

In our security proof, we will see how $Q^{\mathcal{E}}$ also queries a uniform distribution $\mathcal{U}$ and how $Q^{\mathcal{R}}$ can be seen as querying $q_r^{(\mathcal{Y}^*_{\mathsf{neu},h})^{\times d}}$ instead.

See Definition 18, in order to win $Q^{\mathcal{R}}$, adversary $\mathcal{A}_1$ needs to guess the correct extracted secret s given knowledge of $\mathsf{aux} = (c_{\mathsf{prevec}}, p)$, where $p = r_{\mathsf{vec}} + w$ with $w \leftarrow_R C$, and given a predicted response vector r'_{vec}. Since s is a result of a hash evaluation in the random oracle model, the adversary either needs to guess the correct r_{vec} (by computing/solving information about r_{vec}) or guess s according to a uniform distribution. (And verify against the hash of s used for detecting failures.) The latter has a success probability of $q2^{-\kappa}$, where $q \leq T$ is the total number of guesses and T is the running time of the adversary. (Hence, the adversary can at most guess and verify T values for s.)

The remaining time $T - q$ can be used for computing information about r_{vec}. Since s is a result of the random oracle model, the adversary can at best use s in combination with its hash for detecting failures to exclude at most T vector values r_{vec}. (If there is a successful attempt, then this is covered by the $q2^{-\kappa}$ probability above.) In other words, knowledge about the hash of s translates in the best case to knowing a set $\mathcal{T}$ of cardinality $|\mathcal{T}| = T$ for which $r_{\mathsf{vec}} \notin \mathcal{T}$. This is used in the next derivation (instead of knowledge of the hash of s).

Let $e_{\mathsf{vec}} = r'_{\mathsf{vec}} + r_{\mathsf{vec}}$. We derive

$$\begin{aligned}
&\mathsf{Pr}(r_{\mathsf{vec}} \mid c_{\mathsf{prevec}}, r_{\mathsf{vec}} + w, r'_{\mathsf{vec}}, r_{\mathsf{vec}} \notin \mathcal{T}) \\
&= \mathsf{Pr}(r_{\mathsf{vec}} \mid c_{\mathsf{prevec}}, r_{\mathsf{vec}} + w, r_{\mathsf{vec}} + e_{\mathsf{vec}}, r_{\mathsf{vec}} \notin \mathcal{T}) \\
&= \mathsf{Pr}(e_{\mathsf{vec}} \mid c_{\mathsf{prevec}}, r_{\mathsf{vec}} + w, r_{\mathsf{vec}} + e_{\mathsf{vec}}, r_{\mathsf{vec}} \notin \mathcal{T}) \\
&= \mathsf{Pr}(e_{\mathsf{vec}} \mid c_{\mathsf{prevec}}, e_{\mathsf{vec}} + w, r_{\mathsf{vec}} + e_{\mathsf{vec}}, r_{\mathsf{vec}} \notin \mathcal{T}).
\end{aligned}$$

We may equivalently cast problem instances of $Q^{\mathcal{R}}$ as the task to guess the correct e_{vec} given c_{prevec}, $e_{\mathsf{vec}} + w$, $r_{\mathsf{vec}} + e_{\mathsf{vec}}$, and $r_{\mathsf{vec}} \notin \mathcal{T}$. In order to create such a problem instance, $Q^{\mathcal{R}}$ can be simulated by (1) querying $r_{\mathsf{vec}} = r$ according to probability $q_r^{(\mathcal{Y}^*_{\mathsf{neu},h})^{\times n}}$ conditioned on $r \notin \mathcal{T}$, where n is the length of the response vector, and (2) querying e_{vec} according to distribution $\mathcal{E}$ as defined in Definition 18. (And draw $w \leftarrow_R C$.)

Now, consider $Q_u^{\mathcal{R}}$ where distribution $q_r^{(\mathcal{Y}^*_{\mathsf{neu},h})^{\times n}}$ is replaced by the uniform distribution $\mathcal{U}$. By Lemma 29 for $d = n$ and the assumed upper bound on $\epsilon_{\mathsf{corbias}}$, if $Q_u^{\mathcal{R}}$ is $(\kappa' + 5.946)$-bit secure, then $Q^{\mathcal{R}}$ is κ'-bit secure. Let $\mathcal{T}' = \{(r_{\mathsf{vec}} + e_{\mathsf{vec}}) + r \mid r \in \mathcal{T}\}$. Notice that for $Q_u^{\mathcal{R}}$,

$$\begin{aligned}
&\mathsf{Pr}(e_{\mathsf{vec}} \mid c_{\mathsf{prevec}}, e_{\mathsf{vec}} + w, r_{\mathsf{vec}} + e_{\mathsf{vec}}, r_{\mathsf{vec}} \notin \mathcal{T}) \\
&= \mathsf{Pr}(e_{\mathsf{vec}} \mid c_{\mathsf{prevec}}, e_{\mathsf{vec}} + w, e_{\mathsf{vec}} \notin \mathcal{T}') \\
&= \mathsf{Pr}(e_{\mathsf{vec}} \mid e_{\mathsf{vec}} + w, e_{\mathsf{vec}} \notin \mathcal{T}').
\end{aligned}$$

In other words, given $e_{\mathsf{vec}} + w$ and $e_{\mathsf{vec}} \notin \mathcal{T}'$, the adversary needs to guess e_{vec}. We redefine this problem as $Q^{\mathcal{E}}$ and conclude that $Q^{\mathcal{E}}$ is an error-based equivalent of $Q^{\mathcal{R}}$ up to factor $e_{\mathsf{equiv}} = 5.946$, see Definition 18. Also, clearly, $Q^{\mathcal{E}}$ becomes simpler in the presence of an error-reducing oracle. These properties satisfy one of the conditions of Theorem 22.

Lemma 11 shows that $q_r^{\mathcal{Y}^*_{\mathsf{neu},h}} \leq \frac{1}{2} + h2^{n_a}\epsilon_{\mathsf{corbias}} + O((h2^{n_a}\epsilon_{\mathsf{corbias}})^2)$; hence, τ has the same form as the one defined in Theorem 22.

For $\mathcal{B} = Ber(\tau)$, we define $Q^{\mathcal{B}}$ as a distribution that generates problem instances where the adversary needs to guess e_{vec} given $e_{\mathsf{vec}} + w$, $e_{\mathsf{vec}} \notin \mathcal{T}'$, and $e_{\mathsf{vec}} \leftarrow \mathcal{B}$.

The residual min-entropy of code C with respect to $\mathcal{B}$ is assumed to be $2\kappa + \theta$ bits. The min-entropy loss of C due to coset imbalance with respect to $\mathcal{B}$ and subset size T is at most θ. By Lemma 25, we have

$$\tilde{\mathsf{H}}_\infty(e \mid p, e \notin \mathcal{T}) \geq -\theta + \tilde{\mathsf{H}}_\infty(e \mid p) \geq 2\kappa,$$

where e and p are jointly distributed according to $e \leftarrow \mathcal{B}$, $p = w + e$ with $w \leftarrow_R C$. We may conclude that $Q^{\mathcal{B}}$ is 2κ-bit secure. (Notice that this means that we use $\kappa' = 2\kappa$ for λ in Theorem 22.)

We satisfy all conditions of Theorem 22 and conclude that for running time $T - q$, the probability of winning

$$\epsilon_{\mathsf{win}} \leq c_{\mathsf{sep}} \cdot (T - q)2^{5.946}2^{-2\kappa} = (T - q)2^{-\kappa}.$$

Adding the guessing probability $q2^{-\kappa}$ proves the security property for Hash_κ in the random oracle model.

Proof of Theorem 28: Strong randomness extractor. The construction depends on the hash function Hash_κ used for extracting secret s in GetIO-RO$_h$ and GetOutput$_h$. Hash_κ can be replaced by a strong randomness extractor, and this eliminates assuming Hash_κ in the random oracle model.

The main idea is to use a universal family $\mathcal{H}$ of hash functions $\in \{0,1\}^n \to \{0,1\}^m$ of size $|\mathcal{H}| = 2^d$. Universal means that for all $x, x' \in \{0,1\}^n$ with $x \neq x'$, the probability that $H(x) = H(x')$ is at most 2^{-m} where the probability is taken over uniform random $H \in \mathcal{H}$. The randomness extractor is defined as $\mathsf{Ext}(x, H) = H(x)$. If $m = k + 1 - 2\log(1/\epsilon)$, then Ext is a (k, ϵ)-Hellinger extractor according to the leftover hash Lemma for Hellinger, see Theorem 3 in [41]. This means that for every random variable X over $\{0,1\}^n$ with min-entropy at least k, it holds that the Hellinger distance between distribution $(\mathsf{Ext}(X, U_d), U_d)$ and U_{m+d} is at most ϵ. Here, U_d is the random variable representing a uniform drawing from $\{0,1\}^d$ indicating the used hash function $H \in \mathcal{H}$ and U_{m+d} can be regarded as an $(m+d)$-bit output of a function evaluated in X that cannot be distinguished from a random oracle; hence, U_{m+d} draws uniform $(m+d)$-bit strings. In GETIO-RO$_h$, we select $H \leftarrow_R \mathcal{H}$ (this can also be given as input to GETIO-RO$_h$ together with *seed* as discussed earlier in this section) and evaluate $s = \mathsf{Ext}(r_{\mathsf{vec}}, H) = H(r_{\mathsf{vec}})$ instead of $s = \mathsf{Hash}_\kappa(r_{\mathsf{vec}})$ and return $((c_{\mathsf{prevec}}, p, H), s)$. GETOUTPUT-RO receives the additional input H and computes $\hat{s} = H(\hat{r}_{\mathsf{vec}})$. By setting $\epsilon = 2^{-(\kappa"+5.946)/2}$, we require $m = k - \kappa" - 4.946$, i.e., $k = m + \kappa" + 4.946$. We want to output $m = \kappa$ random bits (coded in s); hence, $k = \kappa" + \kappa + 4.946$. This means that we require a residual min-entropy of C with respect to $\mathcal{B}$ of at least k.

By setting $\epsilon = 2^{-(\kappa"+5.946)/2}$, we may conclude from Theorem 1 in [41] that if our RO construction with distribution $(\mathsf{Ext}(X, U_d), U_d)$ replaced by U_{m+d} leads to a $Q^{\mathcal{R}}$ which is $(\kappa" + 5.946)$-bit secure, then $Q^{\mathcal{R}}$ with randomness extractor Ext is $\kappa"$-bit secure. In our proof above, this implies that $Q^{\mathcal{R}}$ with randomness extractor has error-based equivalent $Q^{\mathcal{E}}$ up to a factor $e_{\mathsf{equiv}} = 2 \cdot 5.946$; we now apply Theorem 1 in [41] twice and set $\kappa" = \kappa' + 5.946$ in the proof of our security lemma. We compensate for this by requiring $c_{\mathsf{sep}} = 2^{\kappa - 2 \cdot 5.946}$. Also, notice that in the proof of our security lemma, we set $\kappa' = 2\kappa + \theta$; hence, the residual min-entropy of C with respect to $\mathcal{B}$ must be at least $k = \kappa" + \kappa + 4.946 = \kappa' + \kappa + 10.892 = 3\kappa + \theta + 10.892$ (rather than $2\kappa + \theta$ in our security lemma for Hash_κ in the random oracle model). This shows how the leftover hash lemma introduces an extra (significant) min-entropy loss. □

12. Conclusion

In the literature, we have seen how a FE interface on top of a weak PUF uses CRPs for obfuscating a single key; this requires confidential computing. Computational FE based on the LPN problem still requires confidential computing but can extract many random bit strings out of a weak PUF. We may also apply FE to a strong PUF in order to generate many random bit strings; besides that this again requires and uses confidential computing, also the generated helper data can in theory be used to learn information about CRPs for training a prediction model—no rigorous security analysis capturing this attack possibility exists. Our framework allows rigorous security proofs for PUF interfaces. In particular, the PRO design leads to a random oracle with a small failure probability for which the bit security is precisely characterized. PRO does not rely on any confidential digital computing or digital secrets, and the adversary is allowed to train and use a PUF prediction model with accuracy typically up to 75%. This closes a major gap in PUF literature. Our framework, lemmas, and theorems can be used to analyze the

security of other PUF interfaces as well and motivate the search for strong PUF designs with better security reliability trade-offs for various adversarial models.

Appendix A: Cryptographic hash function

Definition 30. (*Cryptographic Hash Functions*) A cryptographic hash function $y = \mathsf{Hash}(x)$ with a security parameter λ is a cryptographic primitive that takes as input a message $x \in \{0,1\}^{l_x}$ and generates (in $poly(l_x)$ time) a digest $y \in \{0,1\}^{l_y}$, where $l_x \geq l_y$. A family of cryptographic hash functions with parameters l_x and l_y is a set of cryptographic hash functions $\{\mathsf{Hash}_s(x)\}_{s\in\{0,1\}^{l_x}}$, where s plays the role of 'seed.' The family of cryptographic hash functions is secure if $l_x = O(\lambda)$ and satisfies the following three properties:

1. **Pre-image Resistance.** Given $y = \mathsf{Hash}_s(x) : x \leftarrow_R \{0,1\}^{l_x}, s \leftarrow_R \{0,1\}^{l_x}$, the probability $\mathsf{Adv}^{\mathsf{HashPre}}$ that there exists a ppt adversary $\mathcal{A}(1^{l_x}, s)$ who can output $x' \in \{0,1\}^{l_x} : y = \mathsf{Hash}_s(x')$ is negligible in λ. A pre-image-resistant hash function is also called a one-way function.
2. **Second-Pre-image Resistance.** Given $x \leftarrow_R \{0,1\}^{l_x}$ and $s \leftarrow_R \{0,1\}^{l_x}$, the probability $\mathsf{Adv}^{\mathsf{HashSec}}$ that there exists a ppt adversary $\mathcal{A}(1^{l_x}, s)$ who can output $x' \in \{0,1\}^{l_x} : \mathsf{Hash}_s(x) = \mathsf{Hash}_s(x'), x \neq x'$ is negligible in λ.
3. **Collision Resistance.** Given the hash function $\mathsf{Hash}_s() : s \leftarrow_R \{0,1\}^{l_x}$, the probability $\mathsf{Adv}^{\mathsf{HashCol}}$ that there exists a ppt adversary $\mathcal{A}(1^{l_x}, s)$ who can output $x, x' \in \{0,1\}^{l_x} : \mathsf{Hash}_s(x) = \mathsf{Hash}_s(x'), x \neq x'$ is negligible in λ.

Based on [55], we know

$$\begin{aligned}\mathsf{Adv}^{\mathsf{HashPre}} &\leq 2\mathsf{Adv}^{\mathsf{HashSec}} + 2^{l_y - l_x} \\ &\leq 2\mathsf{Adv}^{\mathsf{HashCol}} + 2^{l_y - l_x}.\end{aligned}$$

□

Definition 31. (*Hash Function in the Random Oracle Model*) We define a random oracle with respect to the uniform distribution over $\{0,1\}^{l_y}$ as a function $\mathsf{RO}(x)$ that takes as input a message $x \in \{0,1\}^{l_x}$ and outputs a message y that is uniformly selected from $\{0,1\}^{l_y}$ with replacement (meaning that if an input x is queried twice, then RO returns the same output value y for both queries)—let $\mathcal{R}(l_x, l_y)$ be the set of all such random oracle functions.
We may define a hash function in the random oracle model as follows:

$$\mathsf{Hash}^{\mathsf{RO}(\cdot)}(x) = \mathsf{RO}(x).$$

This defines the standard random oracle model, and this definition of the hash function satisfies $\mathsf{Adv}^{\mathsf{HashCol}} \leq T^2/2^{l_x}$ where $T = poly(l_x)$ is the number of queries by adversary $\mathcal{A}(1^{l_x})$ to $\mathsf{Hash}^{\mathsf{RO}(\cdot)}(x) = \mathsf{RO}(x)$ [56]. If we consider a family of hash functions with parameters $l_x = O(\lambda)$ and l_y in the random oracle model, then we have the property that for all ppt distinguishers $\mathcal{D}$ the following distinguishing advantage is negligible in l_x:

$$\left| \begin{array}{l} \mathsf{Pr}_{s\leftarrow_R\{0,1\}^{l_x}}(\mathcal{D}^{\mathsf{Hash}_s(\cdot)}(1^{l_x}) = 1) \\ -\mathsf{Pr}_{\mathsf{RO}\leftarrow_R\mathcal{R}(l_x,l_y)}(\mathcal{D}^{\mathsf{RO}(\cdot)}(1^{l_x}) = 1) \end{array} \right|.$$

□

References

[1] R. Pappu, B. Recht, J. Taylor, N. Gershenfeld, Physical one-way functions. *Science* **297**(5589), 2026–2030 (2002)

[2] B. Gassend, D. Clarke, M. Van Dijk, S. Devadas, Silicon physical random functions, in *Proceedings of the 9th ACM Conference on Computer and Communications Security* (2002), pp. 148–160

[3] M. Potkonjak, V. Goudar, Public physical unclonable functions. *Proceedings of the IEEE* **102**(8), 1142–1156 (2014)

[4] U. Rührmair, Q. Chen, M. Stutzmann, P. Lugli, U. Schlichtmann, G. Csaba, Towards electrical, integrated implementations of simpl systems, in *IFIP International Workshop on Information Security Theory and Practices* (Springer, 2010), pp. 277–292

[5] D. Lim, J.W. Lee, B. Gassend, G.E. Suh, M. Van Dijk, S. Devadas, Extracting secret keys from integrated circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **13**(10), 1200–1205 (2005)

[6] G.E. Suh, S. Devadas, Physical unclonable functions for device authentication and secret key generation, in *2007 44th ACM/IEEE Design Automation Conference* (IEEE, 2007), pp. 9–14

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

[8] C. Brzuska, M. Fischlin, H. Schröder, S. Katzenbeisser, Physically uncloneable functions in the universal composition framework, in *Annual Cryptology Conference* (Springer, 2011), pp. 51–70

[9] U. Rührmair, Oblivious transfer based on physical unclonable functions, in *International Conference on Trust and Trustworthy Computing* (Springer, 2010), pp. 430–440

[10] R. Ostrovsky, A. Scafuro, I. Visconti, A. Wadia, Universally composable secure computation with (malicious) physically uncloneable functions, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (Springer, 2013), pp. 702–718

[11] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, J. Schmidhuber, Modeling attacks on physical unclonable functions, in *Proceedings of the 17th ACM Conference on Computer and Communications Security* (2010), pp. 237–249

[12] U. Rührmair, J. Sölter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, S. Devadas, Puf modeling attacks on simulated and silicon data. *IEEE transactions on information forensics and security* **8**(11), 1876–1891 (2013)

[13] U. Rührmair, X. Xu, J. Sölter, A. Mahmoud, M. Majzoobi, F. Koushanfar, W. Burleson, Efficient power and timing side channels for physical unclonable functions, in *International Workshop on Cryptographic Hardware and Embedded Systems* (Springer, 2014), pp. 476–492

[14] F. Ganji, S. Tajik, F. Fäßler, J.-P. Seifert, Strong machine learning attack against pufs with no mathematical model, in *International Conference on Cryptographic Hardware and Embedded Systems* (Springer, 2016), pp. 391–411

[15] G.T. Becker, The gap between promise and reality: On the insecurity of xor arbiter pufs, in *International Workshop on Cryptographic Hardware and Embedded Systems* (Springer, 2015), pp. 535–555

[16] N. Wisiol, C. Mühl, N. Pirnay, P.H. Nguyen, M. Margraf, J.-P. Seifert, M. van Dijk, U. Rührmair, Splitting the interpose puf: A novel modeling attack strategy. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 97–120 (2020)

[17] J. Tobisch, A. Aghaie, G.T. Becker, Combining optimization objectives: New modeling attacks on strong pufs. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 357–389 (2021)

[18] P.H. Nguyen, D.P. Sahoo, C. Jin, K. Mahmood, U. Rührmair, M. van Dijk, The interpose puf: Secure puf design against state-of-the-art machine learning attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 243–290 (2019)

[19] Y. Dodis, L. Reyzin, A. Smith, Fuzzy extractors: How to generate strong keys from biometrics and other noisy data, in *International Conference on the Theory and Applications of Cryptographic Techniques* (Springer, 2004), pp. 523–540

[20] H. Kang, Y. Hori, T. Katashita, M. Hagiwara, K. Iwamura, Cryptographie key generation from puf data using efficient fuzzy extractors, in *16th International Conference on Advanced Communication Technology* (IEEE, 2014), pp. 23–26

[21] J. Delvaux, D. Gu, I. Verbauwhede, M. Hiller, M.-D.M. Yu, Efficient fuzzy extraction of puf-induced secrets: Theory and applications, in *International Conference on Cryptographic Hardware and Embedded Systems* (Springer, 2016), pp. 412–431
[22] B. Fuller, X. Meng, L. Reyzin, Computational fuzzy extractors. *Information and Computation* **275**, 104602 (2020)
[23] R. Ueno, K. Kazumori, N. Homma, Rejection sampling schemes for extracting uniform distribution from biased pufs. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 86–128 (2020)
[24] C. Herder, L. Ren, M. Van Dijk, M. Yu, S. Devadas, Trapdoor computational fuzzy extractors and stateless cryptographically-secure physical unclonable functions. *IEEE Transactions on Dependable and Secure Computing* **14**(1), 65–82 (2016)
[25] C. Jin, C. Herder, L. Ren, P.H. Nguyen, B. Fuller, S. Devadas, M. Van Dijk, Fpga implementation of a cryptographically-secure puf based on learning parity with noise. *Cryptography* **1**(3), 23 (2017)
[26] R. Rivest, Illegitimi non carborundum. Invited keynote talk given at CRYPTO (2011)
[27] M. van Dijk, D. Gurevin, C. Jin, O. Khan, P.H. Nguyen, Autonomous secure remote attestation even when all used and to be used digital keys leak. Cryptology ePrint Archive (2021)
[28] D. Gurevin, C. Jin, P.H. Nguyen, O. Khan, M. van Dijk, Secure remote attestation with strong key insulation guarantees. *IEEE Transactions on Computers* (2023). https://doi.org/10.1109/TC.2023.3290870
[29] J. Von Neumann, 13. various techniques used in connection with random digits. *Appl. Math Ser* **12**(36-38), 3 (1951)
[30] I. Damgård, A. Scafuro, Unconditionally secure and universally composable commitments from physical assumptions, in *International Conference on the Theory and Application of Cryptology and Information Security* (Springer, 2013), pp. 100–119
[31] S. Badrinarayanan, D. Khurana, R. Ostrovsky, I. Visconti, Unconditional uc-secure computation with (stronger-malicious) pufs, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (Springer, 2017), pp. 382–411
[32] C. Jin, W. Burleson, M. van Dijk, U. Rührmair, Programmable access-controlled and generic erasable puf design and its applications. *Journal of Cryptographic Engineering* **12**, 413–432 (2022)
[33] U. Rührmair, J. Sölter, F. Sehnke, On the foundations of physical unclonable functions. Cryptology ePrint Archive (2009)
[34] C. Jin, W. Burleson, M. van Dijk, U. Rührmair, Erasable pufs: Formal treatment and generic design, in *Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security* (2020), pp. 21–33
[35] F. Armknecht, R. Maes, A.-R. Sadeghi, B. Sunar, P. Tuyls, Memory leakage-resilient encryption based on physically unclonable functions, in Sadeghi, A.-R., Naccache, D. (eds.) *Towards Hardware-Intrinsic Security* (Springer, New York, NY, 2010), pp. 135–164
[36] F. Armknecht, R. Maes, A.-R. Sadeghi, F.-X. Standaert, C. Wachsmann, A formalization of the security features of physical functions, in *2011 IEEE Symposium on Security and Privacy* (IEEE, 2011), pp. 397–412
[37] N. Wisiol, B. Thapaliya, K.T. Mursi, J.-P. Seifert, Y. Zhuang, Neural network modeling attacks on arbiter-puf-based designs. *IEEE Transactions on Information Forensics and Security* (2022)
[38] N. Wisiol, Towards attack resilient arbiter puf-based strong pufs. Cryptology ePrint Archive (2021)
[39] B. Gassend, M.V. Dijk, D. Clarke, E. Torlak, S. Devadas, P. Tuyls, Controlled physical random functions and applications. *ACM Transactions on Information and System Security (TISSEC)* **10**(4), 1–22 (2008)
[40] A. Boldyreva, V. Kumar, A new pseudorandom generator from collision-resistant hash functions, in *Cryptographers' Track at the RSA Conference* (Springer, 2012), pp. 187–202
[41] K. Yasunaga, Replacing probability distributions in security games via hellinger distance. In: 2nd Conference on Information-Theoretic Cryptography (ITC 2021) (2021). Schloss Dagstuhl-Leibniz-Zentrum für Informatik
[42] C. Herder, M.-D. Yu, F. Koushanfar, S. Devadas, Physical unclonable functions and applications: A tutorial. *Proceedings of the IEEE* **102**(8), 1126–1141 (2014)
[43] N. Wisiol, N. Pirnay, Short paper: Xor arbiter pufs have systematic response bias, in *International Conference on Financial Cryptography and Data Security* (Springer, 2020), pp. 50–57
[44] D.P. Sahoo, P.H. Nguyen, R.S. Chakraborty, D. Mukhopadhya, On the architectural analysis of arbiter delay puf variants. Cryptology ePrint Archive (2016)

[45] C. Zhou, K.K. Parhi, C.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* (2017), pp. 1–6

[46] V. Costan, S. Devadas, Intel sgx explained. IACR Cryptol. *ePrint Arch.* **2016**, 86 (2016)

[47] J. Tobisch, G.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

[48] K.T. Mursi, Y. Zhuang, M.S. Alkatheiri, A.O. Aseeri, Extensive examination of xor arbiter pufs as security primitives for resource-constrained iot devices, in *2019 17th International Conference on Privacy, Security and Trust (PST)* (IEEE, 2019), pp. 1–9

[49] D.P. Sahoo, D. Mukhopadhyay, R.S. Chakraborty, P.H. Nguyen, A multiplexer-based arbiter puf composition with enhanced reliability and security. *IEEE Transactions on Computers* **67**(3), 403–417 (2017)

[50] E. Arikan, E. Telatar, On the rate of channel polarization, in *2009 IEEE International Symposium on Information Theory* (IEEE, 2009), pp. 1493–1495

[51] W. Hoeffding, Probability inequalities for sums of bounded random variables, in Fisher, N.I., Sen, P.K. (eds.) *The Collected Works of Wassily Hoeffding* (Springer, Berlin, Heidelberg, 1994), pp. 409–426

[52] H. Mahdavifar, A. Vardy, Achieving the secrecy capacity of wiretap channels using polar codes. *IEEE Transactions on Information Theory* **57**(10), 6428–6443 (2011)

[53] D. Micciancio, M. Walter, On the bit security of cryptographic primitives, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (Springer, 2018), pp. 3–28

[54] V. Costan, I. Lebedev, S. Devadas, Sanctum: Minimal hardware extensions for strong software isolation, in *25th USENIX Security Symposium (USENIX Security 16)* (2016), pp. 857–874

[55] P. Rogaway, T. Shrimpton, Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance, in *International Workshop on Fast Software Encryption* (Springer, 2004) , pp. 371–388

[56] R. Canetti, O. Goldreich, S. Halevi, The random oracle methodology, revisited. *Journal of the ACM (JACM)* **51**(4), 557–594 (2004)