Physically Uncloneable Functions in the Universal Composition Framework

Christina Brzuska Marc Fischlin Heike Schröder Stefan Katzenbeisser

Darmstadt University of Technology
Center for Advanced Security Research Darmstadt

Abstract. Recently, there have been numerous works about hardware-assisted cryptographic protocols, either improving previous constructions in terms of efficiency, or in terms of security. In particular, many suggestions use Canetti's universal composition (UC) framework to model hardware tokens and to derive schemes with strong security guarantees in the UC framework. Here, we augment this approach by considering Physically Uncloneable Functions (PUFs) in the UC framework. Interestingly, when doing so, one encounters several peculiarities specific to PUFs, such as the intrinsic non-programmability of such functions. Using our UC notion of PUFs, we then devise efficient UC-secure protocols for basic tasks like oblivious transfer, commitments, and key exchange. It turns out that designing PUF-based protocols is fundamentally different than for other hardware tokens. For one part this is because of the non-programmability. But also, since the functional behavior is unpredictable even for the creator of the PUF, this causes an asymmetric situation in which only the party in possession of the PUF has full access to the secrets.

Keywords. Universal Composition, Physically Uncloneable Function, Oblivious Transfer, Commitment, Key Exchange

1 Introduction

Cryptographic protocols which simultaneously satisfy high efficiency demands as well as strong security requirements (like composable security), are scarce. One recent trend in this regard is to use the potential of hardware components like signature cards [20], one-time programs [14], standard smart cards [19], or even more complex tokens [22]. Most of these hardware-assisted protocols actually achieve security in Canetti's universal composition (UC) framework [4] and thus provide strong security guarantees.

1.1 Physically Uncloneable Functions

Here we consider another type of hardware component which recently gained a lot of attention because of the irresistible progress in their realization: Physically Uncloneable Functions (PUFs) [29, 28]. Basically, a PUF is a function derived through a complex physical manufacturing process such that the behavior of the PUF is hard to clone. The PUF itself can be evaluated by a physical stimulus (aka. challenge) on which it provides a noisy response.

Modeling PUFs appropriately is a highly non-trivial task. Most importantly, there are different types of PUFs with different (physical) properties. Furthermore, there does not seem to be a general agreement upon common security properties of PUFs even for a single type only (e.g., whether a PUF is one-way or not, or if the output is pseudorandom). We refer the reader to [31, 25] for a comprehensive overview. We thus consider a very minimalistic model which basically says that only the party in possession of a PUF can evaluate it by sending some stimulus to the PUF and observing the output, and where learning outputs for some stimuli does not facilitate the task of predicting the function's output for other stimuli.

There have been several approaches to define PUFs cryptographically [29, 17, 12, 2, 31, 11, 32]. However, these definitions usually are either rather informal, or follow the more stringent game-based approach, but stipulate uncloneability and tamper-resistance as an external property "outside of the game". A recent exception is the work of Armknecht et al. [1] which provides a game-based definition for uncloneability on a physical level. In the UC world, such features are more handy to specify. We hence follow previous approaches for other token-based protocols to model PUFs formally in the UC framework, exposing several peculiarities for this kind of hardware.

1.2 PUFs and the UC Framework

The UC Framework. The UC framework supports an easy modeling of tamper-proof hardware tokens via ideal functionalities. Roughly, the ideal functionality captures the abstract security properties of the token, and one considers a hybrid world in which real-world protocols and parties also have access to this ideal functionality (and thus the token). This is the approach which has been used extensively in the literature [22, 19, 26, 16, 15] and which we also use to model PUFs in the UC framework, in particular, to model restricted access depending on possession of the token or uncloneability.

In its original form the hybrid model supports the decomposition of cryptographic tasks into basic building blocks and to conclude security of protocols which are composed out of such building blocks. Loosely speaking, Canetti's composition theorem —or, actually a corollary of a more general statement— says that, if a protocol $\pi^{\mathcal{F}}$ UC-securely realizes some functionality $\mathcal{G}$ in the hybrid world with efficient functionality $\mathcal{F}$, and some protocol ρ UC-securely realizes $\mathcal{F}$, then the composed protocol π^{ρ} which invokes ρ whenever π would call $\mathcal{F}$, also UC-securely realizes $\mathcal{G}$.

PUFs in the UC Framework. Our ideal functionality for PUFs only allows the party in possession to stimulate it in order to retrieve a response, thus ensuring restricted access. Uncloneability is enforced through unpredictability. Parties can hand over the PUF to other parties, but then we allow the adversary temporary access before the PUF reaches the recipient. This models the classical example of PUF-augmented credit cards, sent via postal service, which are read out before getting delivered. As in other works about hardware based tokens, we assume some kind of tamper-evidence in the sense that the receiver can later verify the authenticity and integrity of the PUF. We note that this need not be ensured by the PUF technology itself. One may also consider reliable delivery (in which case the adversary may have read-only access during the manufacturing process).

Our ideal functionality covers different kinds of PUF technologies and comprises even PUFs with small input or output size (in which case unpredictability should be understood relative to the small output size). We note that for designing secure *protocols*, the intermediate access of the adversary also necessitates that the challenge space of the PUF is super-polynomial; else the adversary could clone the PUF easily. This domain requirement may currently not be true for all kinds of PUF technology; we comment on this at the end of the paper.

The usage of hardware components in the UC context, especially of PUFs, causes several unpleasant side effects, though. At foremost, PUFs are not known to be implementable by probabilistic polynomial-time (PPT) Turing machines; the manufacturing process seems to be inherently based on physical properties. Hence, while the claims in the hybrid model are technically sound, any realization in practice through actual PUFs leaves a gap in the security claim of the composed protocol, as, strictly speaking, the composition theorem only applies to *probabilistic polynomial-time computable* functionalities $\mathcal{F}$. Fortunately, Canetti [4] proves the composition theorem to hold for a broader class of interactive Turing machines, and we sketch in Appendix B that the same holds for PUFs.

The uninstantiability of PUFs through efficient algorithms causes another issue when it comes to complex cryptographic protocols. For any PUF-based protocol relying on further cryptographic assumptions like the hardness of computing discrete logarithms, the assumption would need to hold relative to the additional computational power given through PUFs. That is, the underlying problem must be hard to solve even for attackers with "more than probabilistic polynomial-time power". It is therefore advantageous to avoid additional cryptographic assumptions in protocols and provide solutions with statistical security.

Non-Programmability. For PUFs, another aspect is the intrinsic *non-programmability* of these tokens: Even the manufacturer usually has no control over the functional behavior. Hence, the ability of the ideal-world simulator to adapt the outcome of a PUF measurement adaptively, as guaranteed when modeling the PUF through an ideal functionality in the hybrid world, appears to be exceedingly optimistic. A similar observation has been made by Nielsen [27] about the (non-)programmability of random oracles in the UC framework.

Roughly, Nielsen takes away the ability of the simulator to program the random oracle by giving the environment direct access to the random oracle. To support the argument in favor of non-programmable PUFs we also note that for random oracles it is straightforward to program consistently given a partial view of the function for other values, namely, by providing independent random values; for PUFs this is less clear since one would need to take the (not necessarily efficiently computable) conditional distribution of the specific PUF type into account.

We adopt Nielsen's approach and augment the environment's ability by giving it also access to the concrete PUF instantiation used in a protocol. Unlike in the case of the publicly available random oracle, though, the environment can only access this PUF when it is in possession of the adversary, i.e., we assume that a PUF, once in possession of the user, can only be accessed by this user. This approach corresponds to the case that the user somewhat prevents further unauthorized access then. In a stronger version one could also allow further "uncontrolled" interaction between the environment, i.e., other protocols, and the PUF even then. This would somehow correspond to a permanently shared PUF functionality in the GUC model [5]. However, many advantages of deploying PUFs for designing efficient protocols would then disappear. With the restriction on temporary access we can still devise efficient solutions, e.g., circumventing impossibility results for UC commitment schemes in the plain model [6] and for GUC commitment schemes in the common reference string model [5].

1.3 PUF-based Protocols in the UC Framework

We finally exemplify the usability of our PUF modeling by presenting PUF-based protocols for three classical areas: oblivious transfer (OT), commitments, and key exchange. Our protocols are UC-secure in the hybrid world (where we grant the environment access to the PUF instantiation as described above), and typically require only a few operations besides PUF evaluations. In particular, all protocols require only sending one party a token in the first step. The protocols do not rely on additional cryptographic assumptions, except for authenticated channels.

Designing PUF-based protocols is not just a matter of adopting other token-based solutions. One reason is clearly the non-programmability property which is usually not stipulated for other tokens (cf. [22, 14]). In fact, most protocols take advantage of the ability to adapt the token's outputs on the fly. But more importantly, the main difference between PUFs and other tokens is that PUFs are by nature even unpredictable for the manufacturer. It follows that only the party in possession of the PUF has full access to the secrets; other parties may only draw from a small set of previously sampled values. In comparison, for the wrapper tokens [22], for example, the creator still knows the program placed inside the token, and the token holder can fully access this program in a black-box way. Hence, both parties somehow share a complete view of the secret. For PUFs the situation is rather "asymmetric".

Our oblivious transfer protocol bears some similarity to a PUF-based protocol of Rührmair

[30]. His protocol, however, has a high round complexity due to an interactive hashing step, and does not provide a formal security proof. Still, [30] points out that, using symmetry of oblivious transfer [40] in the sense that one can change the roles of sender and receiver, one obtains an oblivious transfer protocol in which the other party sends the PUF. We confirm that this symmetry also holds in the UC setting.

Designing a UC-secure commitment scheme with the help of our PUFs turns out to be quite challenging. The non-programmability of our PUFs inhibits equivocality, a property which allows to adapt committed values appropriately, and which is usually required for such commitments [6]. We therefore use our PUF-based oblivious transfer protocol to derive a UC-secure bit commitment scheme. Interestingly, while the standard construction of commitment schemes out of OT [9] uses cut-and-choose techniques with a linear number of oblivious transfers, our transformation does not add any significant overhead. It only needs a single execution of the OT protocol and one extra message. We were not able to trace this idea back to any previous work.

A noteworthy aspect is that, while our OT protocol only withstands static corruptions, our derived commitment scheme is secure in presence of adaptive corruptions. The reason is that for commitments, in contrast to OT, the receiver does not obtain any external input; the values used in the OT sub protocol are chosen internally. This facilitates the simulation of the receiver's side. Hence, if we use our transformation we derive an adaptively secure commitment protocol from a concrete statically secure OT protocol!

Finally, our key exchange protocol follows the folklore approach of using essentially the PUF to transport the key, only that our protocol is stated and formalized in the UC framework. That is, the sender samples some challenge-response pairs, sends the PUF, and later reveals a challenge to the receiver who recoves the image with the help of the PUF. Both parties use the images as the key, after applying a fuzzy extractor for error corretion and smoothing the output. It is clear that for a key exchange protocol where only one party sends a PUF, some additional, one-sided authentication mechanism is required. Else the adversary with temporary access to the PUF could impersonate the honest sender.

All our protocols allow to re-use the PUF for multiple executions. By the unpredictable nature of PUFs, however, it is clear that the number of executions must be fixed in advance and must be known to the parties: The sender, once having sent the PUF, cannot access the PUF anymore and must thus challenge the PUF before sufficiently often, unless the PUF is frequently exchanged or further PUF tokens are sent. An interesting feature of PUFs is that, unlike other hardware tokens (e.g., [22]), protocols using PUFs are automatically secure against reset attacks because they implement (noisy) functions.

2 Physically Uncloneable Functions

A Physically Uncloneable Function (PUF) is a source of randomness that is implemented by a physical system. Roughly speaking, the randomness of PUFs relies on uncontrollable manufacturing variations during their fabrication. For PUF evaluation, the physical system is queried with a stimulus, usually called *challenge*. The device then produces a physical output, which is usually referred to as *response*. A pair of a stimulus and an output is called a *challenge/response pair* (CRP). Furthermore, a PUF, being a physical system, might not necessarily implement a mathematical function, i.e., querying the PUF twice on the same challenge may yield distinct responses. However, we require such "noise" to be bounded so that the two responses are closely related in terms of distance.

2.1 Defining PUFs

A PUF-family $\mathcal{P}$ consists of two (not necessarily efficient) algorithms Sample and Eval. The index sampling algorithm Sample which obtains as input the security parameter and returns as output an index id of the PUF family corresponds to the PUF fabrication process. The evaluation algorithm Eval takes as input a challenge c, evaluates the PUF on c, and generates as output the corresponding response r.

Note that we require the challenge space to be equal to a full set of strings of a certain length. For some classes of PUFs, this is naturally satisfied, for example arbiter PUFs and SRAM PUFs (see [25]). For others types this can be achieved through appropriate encoding, as for angles in optical PUFs.

Definition 2.1 (Physically Uncloneable Functions) *Let rg be the dimension of the range of the PUF responses of the PUF family, and let d_{noise} be a bound on the PUF's noise. A pair $\mathcal{P} = (\mathsf{Sample}, \mathsf{Eval})$ is a family of (rg, d_{noise})-PUFs if it satisfies the following properties:*

Index Sampling. *Let $\mathcal{I}_\lambda$ be an index set. The sampling algorithm Sample outputs, on input the security parameter 1^λ, an index $\mathsf{id} \in \mathcal{I}_\lambda$. We do not require that the index sampling can be done efficiently. Each index $\mathsf{id} \in \mathcal{I}_\lambda$ corresponds to a set $\mathcal{D}_{\mathsf{id}}$ of distributions. For each challenge $c \in \{0,1\}^\lambda$, $\mathcal{D}_{\mathsf{id}}$ contains a distribution $\mathcal{D}_{\mathsf{id}}(c)$ on $\{0,1\}^{rg(\lambda)}$. We do* not *require that $\mathcal{D}_{\mathsf{id}}$ has a short description or an efficient sampling algorithm.*

Evaluation. *The evaluation algorithm Eval gets as input a tuple $(1^\lambda, \mathsf{id}, c)$, where $c \in \{0,1\}^\lambda$. It outputs a response $r \in \{0,1\}^{rg(\lambda)}$ according to distribution $\mathcal{D}_{\mathsf{id}}(c)$. It is* not *required that Eval is a PPT algorithm.*

Bounded Noise. *For all indices $\mathsf{id} \in \mathcal{I}$, for all challenges $c \in \{0,1\}^\lambda$, we have that when running $\mathsf{Eval}(1^\lambda, \mathsf{id}, c)$ twice, then the Hamming distance of any two outputs r_1, r_2 of the algorithm is smaller than $d_{\mathsf{noise}}(\lambda)$.*

Instead of $\mathcal{D}_{\mathsf{id}}(c)$, we usually write $\mathsf{PUF}_{\mathsf{id}}(c)$. Moreover, if misunderstandings are unlikely to occur, we write $\mathcal{D}(c)$ instead of $\mathcal{D}_{\mathsf{id}}(c)$ and PUF instead of $\mathrm{PUF}_{\mathsf{id}}$. Finally, we usually write rg instead of $rg(\lambda)$ and $\mathcal{I}$ instead of $\mathcal{I}_\lambda$.

2.2 Security of PUFs

Various security properties of PUFs have been introduced in the literature (see [25, 31] for overviews) such as unpredictability, uncloneability, bounded noise, uncorrelated outputs, one-wayness, and tamper-evidence. We give a detailed analysis of these properties in Appendix C as well as the relation to our security notions. The main security properties of PUFs are *uncloneability* and *unpredictability*. Unpredictability is covered via an entropy condition on the PUF distribution. This condition also implies mild forms of uncloneability as well as uncorrelated outputs, see Appendix C.2. Moreover, one usually requires that tampering with PUFs can be detected easily, the idea being that a user does not use the PUF anymore after detecting it has been tampered with. Our UC-functionality will cover this property implicitly, as we permit the adversary black-box access to the PUF and the choice of delivering the PUF or not. Tampering with the PUF is treated as not delivering it. For an explicit treatment of tamper-evidence, see Appendix C.5.

We will now turn to our main security definition of PUFs, namely the unpredictability. The behavior of the PUF on input a challenge c should be unpredictable, i.e., have some significant amount of intrinsic entropy, even if the PUF has been measured before on several challenge values. Here, (*conditional*) *min-entropy* is a main tool. It indicates the residual min-entropy on a response value for a challenge c, when one has already measured the PUF on (not necessarily different) challenges $c_1, ..., c_\ell$ before. Since the random responses are not under adversarial control we can look at the residual entropy for the answer to r by taking the (weighted) average over all possible response values $r_1, ..., r_\ell$. Demanding that a PUF has a certain *average* min-entropy [10] is weaker than asking for all possible responses $r_1, ..., r_\ell$, that the residual entropy remains above a certain level. This weaker requirement suffices for our purposes. However, the challenges c are chosen by the adversary such that we will ask the average min-entropy to be high for all challenges and defined by the maximal probability of a possible response r.

Definition 2.2 (Average Min-Entropy) *The average min-entropy of* $\mathsf{PUF}(c)$ *conditioned on the measurements of challenges* $\mathcal{C} = (c_1, \ldots, c_\ell)$ *is defined by*

$$\begin{aligned}
&\tilde{H}_\infty(\mathsf{PUF}(c)|\mathsf{PUF}(\mathcal{C})) \\
&:= -\log\left(\mathbb{E}_{r_i \leftarrow \mathsf{PUF}(c_i)}\left[\max_r \Pr[\mathsf{PUF}(c) = r | r_1 = \mathsf{PUF}(c_1), \ldots, r_\ell = \mathsf{PUF}(c_\ell)]\right]\right) \\
&:= -\log\left(\mathbb{E}_{r_i \leftarrow \mathsf{PUF}(c_i)}\left[2^{-H_\infty(\mathsf{PUF}(c)|r_1=\mathsf{PUF}(c_1),\ldots,r_\ell=\mathsf{PUF}(c_\ell))}\right]\right)
\end{aligned}$$

where the probability is taken over the choice of id *from* $\mathcal{I}$ *and the choice of possible PUF responses on challenge* c*. The term* $\mathsf{PUF}(\mathcal{C})$ *denotes a sequence of random variables* $\mathsf{PUF}(c_1), ...,$ $\mathsf{PUF}(c_\ell)$ *each corresponding to an evaluation of the PUF on challenge* c_k*.*

We also write $\tilde{H}_\infty(\mathsf{PUF}(c)|\mathcal{C})$ as an abbreviation for $\tilde{H}_\infty(\mathsf{PUF}(c)|\mathsf{PUF}(\mathcal{C}))$. We now turn to our definition of unpredictability, which is derived from the notion of unpredictability for random variables.

Definition 2.3 (Unpredictability) *We call a* (rg, d_{noise})*-PUF family* $\mathcal{P} = (\mathsf{Sample}, \mathsf{Eval})$ *for security parameter* 1^λ *is* $(d_{\mathsf{min}}(\lambda), m(\lambda))$*-unpredictable if for any* $c \in \{0,1\}^\lambda$ *and any challenge list* $\mathcal{C} = (c_1, \ldots, c_\ell)$*, one has that, if for all* $1 \leq k \leq \ell$ *the Hamming distance satisfies* $\mathsf{dis}_{\mathsf{ham}}(c, c_k) \geq d_{\mathsf{min}}(\lambda)$*, then the average min-entropy satisfies* $\tilde{H}_\infty(\mathsf{PUF}(c)|\mathsf{PUF}(\mathcal{C})) \geq m(\lambda)$*. Such a PUF-family is called a* $(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, m)$*-PUF family.*

Note that one could also define a computational version of unpredictability via computational min-entropy (aka. HILL entropy, named after [18]) where the entropy is defined via the entropy of computationally indistinguishable random variables. All proofs considered in this paper carry through when replacing statistical by computational min-entropy; we nonetheless use the statistical variant for sake of simplicity. As explained in the introduction, indistinguishability for defining computational min-entropy then needs to be considered with respect to distinguishers that have PUF power (see also Section B), and not with respect to mere PPT algorithms.

Also, one could define unpredictability in terms of a game where an efficient adversary, after seeing some challenge/response pairs, tries to predict the response for another challenge which is not within close distance to the previous queries (see Appendix C.2); the success probability should then be negligible. Clearly, the PUF would need super-logarithmic min-entropy in the above sense to make it unpredictable according to this game, but the lower bound on the entropy would vary with the adversary. We do not take this approach because the fuzzy extractors, which are necessary to eliminate the noise of a PUF, usually need a *fixed* lower bound on the min-entropy in order to be applicable. Also, while it is easy to incorporate a distributional property as above into the ideal functionality, using game-based properties to specify abstract and ideal security requirements appears to be very peculiar.

3 PUFs and Fuzzy Extractors

By nature, PUF evaluation is noisy, so that same stimuli results in closely related but different outputs. We use fuzzy extractors of Dodis et al. [10] to convert noisy, high-entropy measurements of PUFs into reproducible random values.

3.1 Fuzzy Extractors

A fuzzy extractor consists of a pair of algorithms $(\mathsf{Gen}, \mathsf{Rep})$. The generation algorithm Gen takes as input a noisy measurement w and generates as output a secret st together with helper data p. The helper data can be stored publicly, since it does not reveal information about the secret. It is later used to reproduce the same secret st from related measurements. That is, the reproduction algorithm Rep takes as input a noisy measurement w' and helper data p. If w and w' are sufficiently close, Rep gives the same reply st. The value st is distributed almost uniformly and thereby allows to be used for cryptographic purposes.

In the following we use U_ℓ to denote the uniform distribution on ℓ-bit binary strings. Furthermore, a set $\mathcal{M}$ with a distance function $\mathsf{dis} : \mathcal{M} \times \mathcal{M} \leftarrow \mathbb{R}^+ = [0, \infty)$ is called a metric space. The statistical distance SD of two probability distributions A and B is defined as $\mathrm{SD}(A, B) = \frac{1}{2} \sum_v |Pr(A = v) - Pr(B = v)|$.

Definition 3.1 *Let* dis *be a distance function for metric space* $\mathcal{M}$*. A* (m, ℓ, t, ϵ)*-fuzzy extractor consists of two efficient randomized algorithms* $(\mathsf{Gen}, \mathsf{Rep})$*:*

Gen: *The algorithm* Gen *outputs on input* $w \in \mathcal{M}$ *a secret string* $st \in \{0,1\}^\ell$ *and a helper data string* $p \in \{0,1\}^*$*.*

Rep: *The algorithm* Rep *takes an element* $w' \in \mathcal{M}$ *and a helper data string* $p \in \{0,1\}^*$ *and outputs a string* st*.*

Correctness: *If* $\mathsf{dis}(w, w') \leq t$ *and* $(st, p) \leftarrow \mathsf{Gen}(w)$*, then* $\mathsf{Rep}(w', p) = st$*.*

Security: *For any distribution* $\mathcal{W}$ *on the metric space* $\mathcal{M}$ *of min-entropy* m*, the first component of the random variable* (st, p)*, defined by drawing* w *according to* $\mathcal{W}$ *and then applying* Gen*, is distributed almost uniformly, even if* p *is observed, i.e.,* $SD((st, p), (U_\ell, p)) \leq \epsilon$*.*

3.2 Applying Fuzzy Extractors to PUFs

We now determine parameters to combine a PUF and the fuzzy extractor in order to achieve almost uniformly random values. Let λ be the security parameter. We let the parameters of the fuzzy extractor depend on the parameters of the PUF. Assume that we have a $(rg(\lambda), d_{\mathsf{noise}}(\lambda), d_{\mathsf{min}}(\lambda), m(\lambda))$-PUF family with d_{min} being in the order of $o(\lambda / \log \lambda)$. We now determine the corresponding parameters for the fuzzy extractor as follows. Let $\ell(\lambda) := \lambda$ be the length parameter for value st. Let $\epsilon(\lambda)$ be a negligible function and let $t(\lambda) = d_{\mathsf{noise}}(\lambda)$. For each λ, let $(\mathsf{Gen}, \mathsf{Rep})$ be a $(m(\lambda), \ell(\lambda), t(\lambda), \epsilon(\lambda))$-fuzzy extractor. The metric space $\mathcal{M}$ is $\{0,1\}^{rg(\lambda)}$ with Hamming distance $\mathsf{dis}_{\mathsf{ham}}$.[1]

[1] Note that such fuzzy extractors only exist if $rg(\lambda)$ and $m(\lambda)$ are sufficiently large. In order to achieve this, several PUFs can be combined. When combining two PUFs of the same family, rg gets doubled and so does m.

Definition 3.2 *If a PUF and a fuzzy extractor* $(\mathsf{Gen}, \mathsf{Rep})$ *satisfy the above requirements, then they are said to have* matching parameters.

If a PUF and a fuzzy extractor have matching parameters, then the properties well-spread domain, extraction independence and response consistency follow.

Well-Spread Domain: For all polynomials $p(\lambda)$ and all sets of challenges $c_1, ..., c_{p(\lambda)}$, the probability of a random challenge to be within distance smaller d_{min} of any of the c_k is negligible.

Extraction Independence: For all challenges $c_1, ..., c_{p(\lambda)}$, it holds that the PUF evaluation on a challenge c with $\mathsf{dis}(c_k, c) > d_{\mathsf{min}}$ for all $1 \leq k \leq p(\lambda)$ and subsequent application of Gen yields an almost uniform value st even for those who observe p.

Response Consistency: The fuzzy extractor helps to map two evaluations of the same PUF to the same random string, i.e., if PUF is measured on challenge c twice and returns r and r', then for $(st, p) \leftarrow \mathsf{Gen}(r)$, one has $st \leftarrow \mathsf{Rep}(r', p)$.

We now prove that a PUF and a fuzzy extractor with matching parameters have a well-spread domain and meet extraction independence as well as response consistency.

Consider the well-spread domain property: the number of challenges in $\{0,1\}^\lambda$ within distance smaller d_{min} of at least one of the c_k can be upper bounded by the number of elements in a ball of radius d_{min} multiplied with the number $p(\lambda)$ of challenges, i.e.,

$$p(\lambda) \sum_{i=1}^{d_{\mathsf{min}}(\lambda)} \binom{\lambda}{i} \leq p(\lambda)\lambda^{d_{\mathsf{min}}(\lambda)},$$

which is a negligible fraction of 2^λ, as the term

$$p(\lambda)\lambda^{d_{\mathsf{min}}(\lambda)}2^{-\lambda} = p(\lambda)2^{d_{\mathsf{min}}(\lambda)\log\lambda - \lambda}$$

is negligible if $d_{\mathsf{min}}(\lambda) = o(\lambda/\log\lambda)$.

For extraction independence, we observe that $H_\infty(\mathsf{PUF}(c)|\mathcal{C})$ is greater than $m(\lambda)$. Thus, evaluating a PUF on challenge c corresponds to drawing a random value r from a distribution on the metric space $\{0,1\}^{rg(\lambda)}$ which has entropy greater than $m(\lambda)$. Hence, the distribution of the random variable (st, p) defined by drawing a response r from $\mathsf{PUF}(c)$, conditioned on $(r_1, c_1), ..., (r_{p(\lambda)}, c_{p(\lambda)})$ and applying Gen to it, is statistically close to uniform, i.e., $\mathrm{SD}((st, p), (U_\lambda, p)) \leq \epsilon$ by definition of the fuzzy extractor.

Thus, if there are PUFs with $m(\lambda)$ being non-negligible they can be combined to a useful PUF-family — even if a PUF-family has *less than one bit* entropy, it still can be combined to obtain a good PUF-family with outputs which has high entropy of many bits. Thus, we may assume that the PUF has corresponding parameters.

Finally, for the response consistency, we notice that $d_{\mathsf{min}}(\lambda) = t(\lambda)$. Thus, the bounded noise property of the PUF assures that two PUF evaluations yield responses r, r' with distance smaller than $t(\lambda)$. Thus, the properties of the fuzzy extractor assure that for $(st, p) \leftarrow \mathsf{Gen}(r)$, one has that $\mathsf{Rep}(p, r')$ outputs st.

4 Universally Composable Security and PUFs

We model PUFs in the universal composition framework introduced by Canetti in [4]. Note that we use, among other things, well-studied UC basics, such as authenticated message transmissions, which we now shortly review.

4.1 Authenticated Communication in UC

The UC functionality $\mathcal{F}_{\mathsf{auth}}$ for authenticated message transmission [4] is presented in Figure 1. The functionality prevents tampering and/or injecting messages. That is, a party P_j will receive a message msg from a party P_i only if P_i has sent msg to P_j.

$\mathcal{F}_{\mathsf{auth}}$ runs with parties $P_1, ..., P_n$ and adversary $\mathcal{S}$.

- Whenever P_i writes $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, \mathsf{msg})$ on $\mathcal{F}_{\mathsf{auth}}$'s input tape then $\mathcal{F}_{\mathsf{auth}}$ proceeds with the following actions:
  - $\star$ Store the tuple $(\mathtt{sid}, \mathtt{ssid}, P_i, P_j, \mathsf{msg})$ and output $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, \mathsf{msg})$ to the adversary $\mathcal{S}$.
- Whenever $\mathcal{S}$ writes $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, \mathsf{msg})$ on $\mathcal{F}_{\mathsf{auth}}$'s input tape check if a tuple $(\mathtt{sid}, \mathtt{ssid}, P_i, P_j, \mathsf{msg})$ is stored. If so, write $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, \mathsf{msg})$ on P_j's communication input tape. Else, ignore the input.

Figure 1: The ideal functionality for message authentication adapted from [4].

4.2 Modeling PUFs in UC

In the following we propose an ideal functionality $\mathcal{F}_{\mathsf{PUF}}$ that will model PUFs. The functionality is presented in Figure 2 and handles the following operations: (1) a party P_i is allocated a PUF; (2) P_i can query the PUF; (3) P_i gives the PUF to another party P_j who can also query the device; (4) an adversary can query the PUF during transition.

The functionality $\mathcal{F}_{\mathsf{PUF}}$ maintains a list $\mathcal{L}$ of tuples $(\mathtt{sid}, P_i, \mathsf{id}, \tau)$ where $\mathtt{sid}$ is the (public) session identifier and id is the (internal) PUF-identifier, essentially describing the output distribution. Note that the PUF itself does not use $\mathtt{sid}$. The element $\tau \in \{\mathsf{trans}(P_j), \mathsf{notrans}\}$ denotes whether the PUF is in transition to P_j. For $\mathsf{trans}(P_j)$, indicating that the PUF is in transition to P_j, the adversary is able to query the PUF. In turn, if it is set to $\mathsf{notrans}$ then only the possessing party can query the PUF.

The PUF functionality $\mathcal{F}_{\mathsf{PUF}}$ is indexed by the PUF parameters $(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, m)$-PUF and gets the security parameter λ in unary encoding as additional input. It is required to satisfy the bounded noise property for $d_{\mathsf{noise}}(\lambda)$ and the unpredictability property for $(d_{\mathsf{min}}(\lambda), m(\lambda))$. This enforces that the outputs obey the basic entropic requirements of PUFs (analogously to the requirement for the random oracle functionality to produce random and independent outputs). We write $\mathcal{F}_{\mathsf{PUF}}$ and $\mathcal{F}_{\mathsf{PUF}}(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, m)$ interchangeably.

$\mathcal{F}_{\mathsf{PUF}}(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, m)$ receives as initial input a security parameter 1^λ and runs with parties $P_1, ..., P_n$ and adversary $\mathcal{S}$.

- Whenever a party P_i writes $(\mathsf{init}_{\mathsf{PUF}}, \mathtt{sid}, P_i)$ on the input tape of $\mathcal{F}_{\mathsf{PUF}}$ then $\mathcal{F}_{\mathsf{PUF}}$ checks whether $\mathcal{L}$ already contains a tuple $(\mathtt{sid}, *, *, *, *)$:
  - ⋆ If this is the case then turn into the waiting state.
  - ⋆ Else, draw $\mathsf{id} \leftarrow \mathsf{Sample}(1^\lambda)$ from the PUF-family. The functionality $\mathcal{F}_{\mathsf{PUF}}$ puts the following tuple in $\mathcal{L}$: $(\mathtt{sid}, \mathsf{id}, P_i, *, \mathsf{notrans})$ and writes $(\mathsf{initialized}_{\mathsf{PUF}}, \mathtt{sid})$ on the communication input tape of P_i.
- Whenever a party P_i writes $(\mathsf{eval}_{\mathsf{PUF}}, \mathtt{sid}, P_i, c)$ on $\mathcal{F}_{\mathsf{PUF}}$'s input tape then $\mathcal{F}_{\mathsf{PUF}}$ first checks, if there exists a tuple $(\mathtt{sid}, \mathsf{id}, P_i, \mathsf{notrans})$ in $\mathcal{L}$:
  - ⋆ If this is not the case then turn into the waiting state.
  - ⋆ Else, run $r \leftarrow \mathsf{Eval}(1^\lambda, \mathsf{id}, c)$ and write $(\mathsf{eval'ed}_{\mathsf{PUF}}, \mathtt{sid}, c, r)$ on P_i's communication input tape.
- Whenever a party P_i sends $(\mathsf{handover}_{\mathsf{PUF}}, \mathtt{sid}, P_i, P_j)$ to $\mathcal{F}_{\mathsf{PUF}}$ then $\mathcal{F}_{\mathsf{PUF}}$ first checks, if there exists a tuple $(\mathtt{sid}, *, P_i, \mathsf{notrans})$ in $\mathcal{L}$:
  - ⋆ If this is not the case then turn into the waiting state.
  - ⋆ Else, modify the tuple $(\mathtt{sid}, \mathsf{id}, P_i, \mathsf{notrans})$ to the updated tuple $(\mathtt{sid}, \mathsf{id}, \perp, \mathsf{trans}(P_j))$. Write $\mathsf{invoke}_{\mathsf{PUF}}(\mathtt{sid}, P_i, P_j)$ on $\mathcal{S}$'s communication input tape to indicate that a $\mathsf{handover}_{\mathsf{PUF}}$ occurs between P_i and P_j.
- Whenever the adversary writes $(\mathsf{eval}_{\mathsf{PUF}}, \mathtt{sid}, \mathcal{S}, c)$ on the input tape of $\mathcal{F}_{\mathsf{PUF}}$ then $\mathcal{F}_{\mathsf{PUF}}$ first checks, if $\mathcal{L}$ contains a tuple $(\mathtt{sid}, \mathsf{id}, \perp, \mathsf{trans}(*))$:
  - ⋆ If this is not the case then turn into the waiting state.
  - ⋆ Else, run $r \leftarrow \mathsf{Eval}(1^\lambda, \mathsf{id}, c)$ and return $(\mathsf{eval'ed}_{\mathsf{PUF}}, \mathtt{sid}, c, r)$ to $\mathcal{S}$.
- Whenever the adversary writes $(\mathsf{ready}_{\mathsf{PUF}}, \mathtt{sid}, \mathcal{S})$ on $\mathcal{F}_{\mathsf{PUF}}$'s input tape then $\mathcal{F}_{\mathsf{PUF}}$ searches for a tuple $(\mathtt{sid}, \mathsf{id}, \perp, \mathsf{trans}(P_j))$ in $\mathcal{L}$:
  - ⋆ If such a tuple does not exist then turn into the waiting state.
  - ⋆ Else, modify the tuple $(\mathtt{sid}, \mathsf{id}, \perp, \mathsf{trans}(P_j))$ to the updated tuple $(\mathtt{sid}, \mathsf{id}, P_i, \mathsf{notrans})$. Write the message $(\mathsf{handover}_{\mathsf{PUF}}, \mathtt{sid}, P_i)$ on P_j's communication input tape and store the tuple $(\mathsf{received}_{\mathsf{PUF}}, \mathtt{sid}, P_i)$.
- Whenever the adversary sends $(\mathsf{received}_{\mathsf{PUF}}, \mathtt{sid}, P_i)$ to $\mathcal{F}_{\mathsf{PUF}}$, $\mathcal{F}_{\mathsf{PUF}}$ checks if a tuple $(\mathsf{received}_{\mathsf{PUF}}, \mathtt{sid}, P_i)$ has been stored. If so, it writes this tuple to the communication input tape of P_i. Else, $\mathcal{F}_{\mathsf{PUF}}$ turns into the waiting state.

Figure 2: The ideal functionality $\mathcal{F}_{\mathsf{PUF}}$ for PUFs.

Also note that our definition requires that a PUF is somehow certified. That is, the adversary cannot replace a PUF sent to an honest party by a fake token including some

"software emulation"; the adversary can only measure the PUF when in transition. The receiver can verify the constitution and authenticity of the received hardware.

4.3 Non-Programmability

As explained in the introduction we envision a non-programmable version of PUFs. The functionality above, if used in the standard way within the hybrid model, would be programmable, though, because the environment would not have direct access (even if the PUF is in possession of the adversary). One way to enforce non-programmability is to switch to the extended UC (EUC) model [5] where all parties, including the environment, share the above functionality. The PUF could then also be evaluated by the environment in which case the simulator is informed about the challenge and response.

To simplify we linger within the basic UC framework and instead allow the environment to dispatch special PUF queries to the adversary/simulator. This query needs to be answered faithfully by forwarding it to a genuine PUF instance, and the response is handed back to the environment. Put differently, we put some restriction on the how the simulator behaves, formally giving a UC-security proof which would transfer to the EUC model.

5 Oblivious Transfer with PUFs

In a 1-out-of-2 oblivious transfer (OT) protocol the sender possesses two secrets $\mathsf{s}_0, \mathsf{s}_1$ and the receiver holds a selection bit $b \in \{0, 1\}$, thereby choosing one of the two secrets. A 1-out-of-2 OT-protocol assures that at the end of the protocol execution, the receiver learns the secret s_b, but nothing about s_{1-b}, and the sender does not learn anything about the selection bit b.

Oblivious Transfer is a widely used cryptographic primitive for many cryptographic applications [23, 9, 13]. However, in many of those applications a bottleneck of OT is the computational requirements since, for instance, several public key operations are necessary. We here show how to avoid the number of public key operations by adopting hardware. In the following, we recall the oblivious transfer ideal functionality and then provide a PUF-based oblivious transfer protocol.

As noted in the introduction, we envision a scenario in which the PUF is used multiple times. In the plain UC model, however, a fresh PUF would need be sent for each OT execution. An alternative would be to switch to the joint-state theorem (JUC) [8] for the UC framework. However, JUC applies a transformation to the original protocol, and if a single session of a PUF protocol requires to hand over a PUF once, the JUC transformation would also require a handover per session. Nothing would be gained. Thus, we define and analyze multi-session protocols instead of the more common one-session protocols.

5.1 The Oblivious Transfer Ideal Functionality

1-out-of-2 oblivious transfer is an interaction between a sender P_i and a receiver P_j where the environment $\mathcal{Z}$ provides P_i with two inputs $\mathsf{s}_0, \mathsf{s}_1$ and P_j with an input bit b. As soon as both parties provided their inputs (and the simulator $\mathcal{S}$ allows delivery), the ideal functionality returns the secret s_b to the receiver. The ideal functionality for oblivious transfer $\mathcal{F}_{\mathsf{OT}}$ is given in Figure 3. We stress that this functionality only supports static corruption and can be used a bounded number of times, and only by the parties which have exchanged the PUF. Each execution will be accompanied by a unique sub session identifier $\mathtt{ssid}$.

$\mathcal{F}_{\mathsf{OT}}$ is parameterized by an integer N and receives as input a security parameter 1^λ, and runs with parties $P_1, ..., P_n$ and adversary $\mathcal{S}$. The functionality initially sets $(n, S, R) = (1, \perp, \perp)$.

In the following, the functionality ignores any input if $n > N$, or if $n > 1$ and $(S, R) \neq (P_i, P_j)$ for the parties' identities (P_i, P_j) in the input. Else,

- Whenever P_i writes $(\mathsf{send}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, (\mathsf{s}_0, \mathsf{s}_1))$ with $\mathsf{s}_0, \mathsf{s}_1 \in \{0,1\}^\lambda \cup \{\perp\}$ on $\mathcal{F}_{\mathsf{OT}}$'s input tape, $\mathcal{F}_{\mathsf{OT}}$ stores $(\mathsf{send}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, (\mathsf{s}_0, \mathsf{s}_1))$ and writes $(\mathsf{send}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ to the communication input tape of $\mathcal{S}$. The functionality increments n to $n+1$ and stores $(S, R) = (P_i, P_j)$ if $n = 2$ now.
- Whenever P_j writes $(\mathsf{choose-secret}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, b)$ on the input tape of $\mathcal{F}_{\mathsf{OT}}$, the functionality $\mathcal{F}_{\mathsf{OT}}$ stores this tuple and writes $(\mathsf{choose-secret}_{\mathsf{OT}}, \mathtt{ssid}, \mathtt{sid}, P_i, P_j)$ on the input tape of $\mathcal{S}$.
- When $\mathcal{S}$ writes $(\mathsf{deliver}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ on $\mathcal{F}_{\mathsf{OT}}$'s input communication tape then $\mathcal{F}_{\mathsf{OT}}$ checks if tuples $(\mathsf{send}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, (\mathsf{s}_0, \mathsf{s}_1))$ and $(\mathsf{choose-secret}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, b)$ have been stored. If so, write $(\mathsf{deliver}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, \mathsf{s}_b)$ on the input communication tape of P_j.

Figure 3: The ideal functionality for oblivious transfer adapted from [4].

5.2 Oblivious Transfer Scheme

In Figure 4, we provide an oblivious transfer protocol. For simplicity of exposition, we use the following notation. For a possibly empty set $\mathcal{C}$ we let $\mathsf{dis}(c, \mathcal{C}) > d_{\mathsf{min}}$ denote the check that each element c_i in $\mathcal{C}$ satisfies the bound $\mathsf{dis}(c, c_i) > d_{\mathsf{min}}$. If not, we assume that the corresponding party aborts. Also, when interacting with the PUF (functionality), we simply write for example $r \leftarrow (\mathsf{eval}_{\mathsf{PUF}}, \mathtt{sid}_0, P_i, c)$ to denote the fact that, for a call $(\mathsf{eval}_{\mathsf{PUF}}, \mathtt{sid}_0, P_i, c)$ the functionality has replied with $(\mathsf{eval'ed}_{\mathsf{PUF}}, \mathtt{sid}_0, c, r)$. Here, $\mathtt{sid}_0$ is the session identifier for $\mathcal{F}_{\mathsf{PUF}}$, as opposed to $\mathtt{sid}$ and $\mathtt{ssid}$ for the oblivious transfer protocol.

We note that the protocol does not achieve perfect completeness in the sense that executions between honest parties may fail. The probability for this is negligible, though. This follows straightforwardly again from the fact that the domain is well-spread: All (at most

polynomial) challenges are independent random values such that one is within small distance of the others with negligible probability only. If all challenges are sufficiently far apart, the receiver always obtains the correct value.

Sender $\boldsymbol{P_i}$	session $\texttt{sid}$	Receiver $\boldsymbol{P_j}$
		$(\mathsf{init}_{\mathsf{PUF}}, \texttt{sid}_0, P_i, \lambda)$
		$k = 1, ..., N$: $c_k \leftarrow \{0,1\}^\lambda$
		$r_k \leftarrow (\mathsf{eval}_{\mathsf{PUF}}, \texttt{sid}_0, P_i, c_k)$
$\mathcal{C} := \emptyset$	$\xleftarrow{(\mathsf{handover}_{\mathsf{PUF}}, \texttt{sid}_0, P_i, P_j)}$	$\mathcal{L} := (c_1, r_1, ..., c_\ell, r_\ell)$
		$\mathcal{C} := \emptyset$
	Repeat at most N times with fresh $\texttt{ssid}$	
Input: $\mathsf{s}_0, \mathsf{s}_1 \in \{0,1\}^\lambda$, $\texttt{sid}$		Input: $b \in \{0,1\}$, $\texttt{sid}$
$x_0, x_1 \xleftarrow{\$} \{0,1\}^\lambda$	$\xrightarrow{(\mathsf{send}_{\mathsf{auth}}, \texttt{sid}, \texttt{ssid}, P_i, P_j, (x_0, x_1))}$	Draw $(c, r) \xleftarrow{\$} \mathcal{L}$
		$v := c \oplus x_b$, $c' := c \oplus x_0 \oplus x_1$
$\mathsf{dis}(v \oplus x_0, \mathcal{C}) > d_{\mathsf{min}}$?	$\xleftarrow{(\mathsf{send}_{\mathsf{auth}}, \texttt{sid}, \texttt{ssid}, P_i, P_j, v)}$	$\mathsf{dis}(c, \mathcal{C}) > d_{\mathsf{min}}$?
$\mathsf{dis}(v \oplus x_1, \mathcal{C}) > d_{\mathsf{min}}$?		$\mathsf{dis}(c', \mathcal{C}) > d_{\mathsf{min}}$?
Add $v \oplus x_0$, $v \oplus x_1$ to $\mathcal{C}$		Add c, c' to $\mathcal{C}$
$r'_0 \leftarrow (\mathsf{eval}_{\mathsf{PUF}}, \texttt{sid}_0, P_i, v \oplus x_0)$		Delete (c, r) in $\mathcal{L}$
$r'_1 \leftarrow (\mathsf{eval}_{\mathsf{PUF}}, \texttt{sid}_0, P_i, v \oplus x_1)$		
$(st_0, p_0) \leftarrow \mathsf{Gen}(r'_0)$		
$(st_1, p_0) \leftarrow \mathsf{Gen}(r'_1)$		
$\mathsf{S}_0 := \mathsf{s}_0 \oplus st_0$, $\mathsf{S}_1 := \mathsf{s}_1 \oplus st_1$	$\xrightarrow{(\mathsf{send}_{\mathsf{auth}}, \texttt{sid}, \texttt{ssid}, P_i, P_j, (\mathsf{S}_0, p_0, \mathsf{S}_1, p_1))}$	$st'_b \leftarrow \mathsf{Rep}(r, p_b)$
		$\mathsf{s}_b = \mathsf{S}_b \oplus st'_b$

Figure 4: Oblivious transfer Scheme with PUFs.

We now sketch the security arguments for the OT-protocol in Figure 4, i.e., at the end of the OT protocol (1) a malicious sender learns nothing about the bit b and (2) a malicious receiver learns only the secret s_b and remains oblivious about s_{1-b}. For case (1), the receiver chooses the challenge c at random. Thus, $v = c \oplus x_b$ hides x_b information-theoretically and thus also b. We now consider case (2). For simplicity, assume that $b = 0$. Then, the sender shall remain oblivious about any information about s_1. If st_1 looks uniform to the sender, then s_1 is information-theoretically hidden. If the fuzzy extractor and the PUF have matching parameters (see Definition 3.2), then with overwhelming probability this is the case, as — due to the well-spread domain property (see Subsection 3.2) — the probability that the receiver queried the PUF on values c_k with $\mathsf{dis}_{\mathsf{ham}}(c_k, v \oplus x_1) < d_{\mathsf{min}}$ is negligible, and the checks on the sender side about list $\mathcal{L}$ provided that the sender does not reveal PUF responses to critical

challenges.

Theorem 5.1 *Assuming that* $(\mathsf{Gen}, \mathsf{Rep})$ *is a* (m, ℓ, t, ϵ)*-fuzzy generator, and* $\mathsf{PUF} = (\mathsf{Sample}, \mathsf{Eval})$ *is a PUF-family with matching parameters (see Definition 3.2), then protocol* PUFOT *securely realizes the functionality* $\mathcal{F}_{\mathsf{OT}}$ *in the* $\mathcal{F}_{\mathsf{PUF}}$*-hybrid model.*

Security holds in a statistical sense, i.e., the environment's views in the two worlds are statistically close. This remains true for unbounded algorithms $\mathcal{A}, \mathcal{S}$, and $\mathcal{Z}$, as long as the number of PUF evaluations is polynomially bounded. The proof is delegated to Appendix A.

5.3 Oblivious Transfer with Sender-PUF

Our OT-protocol requires the receiver to send a PUF to the sender. Sometimes it may be desirable to have the sender prepare the PUF, though. This can be achieved by switching the roles of the sender and the receiver via the protocol by Wolf and Wullschleger [40], but at the expense of having to run linear many OT executions for strings of length λ. This is unavoidable since the receiver in an OT-protocol just enters a bit such that, when acting as a sender, it can only transmit a single bit. In this protocol the sender of the outer OT-protocol acts as a receiver in the inner OT-protocol, thus sending the PUF.

The protocol in [40] requires only a single round of additional communication. It is UC-secure in the $\mathcal{F}_{\mathsf{OT}}$-hybrid world and inherits the security properties (statistical vs. computational security, and adaptive vs. static corruptions). With a linear overhead [3] and another extra round of communication one can then get an OT-protocol for strings, which is also UC-secure in the $\mathcal{F}_{\mathsf{OT}}$-hybrid world for bit-functionality $\mathcal{F}_{\mathsf{OT}}$. The final protocol is now a UC-secure OT-protocol for strings with linear many calls to $\mathcal{F}_{\mathsf{OT}}$, a few extra rounds, and inheriting all security characteristics from $\mathcal{F}_{\mathsf{OT}}$.

6 PUF-based Commitment Scheme

A commitment scheme is a two-party protocol between a sender and a receiver where the sender (also called committer) first sends a disguised version of the value to the receiver such that, later, only this value can be revealed. More precisely, a commitment scheme allows the committer to compute to a value msg a pair $(\mathrm{com}, \mathrm{decom})$ such that com reveals nothing about the value msg but using the pair $(\mathrm{com}, \mathrm{decom})$ one can open msg. Moreover it should be infeasible to find a value decom' such that $(\mathrm{com}, \mathrm{decom}')$ reveals $\mathsf{msg}' \neq \mathsf{msg}$.

6.1 The Commitment Scheme Ideal Functionality

In the UC world, the commitment scheme is realized by the (bounded) functionality $\mathcal{F}_{\mathsf{com}}$ as follows: $\mathcal{F}_{\mathsf{com}}$ receives an input $(\mathsf{commit}, \mathtt{sid}, \mathtt{ssid}, \mathsf{msg})$ from some committer P_i where msg is the value committed to. After verifying the validity of the session identifier $\mathtt{sid}$, $\mathcal{F}_{\mathsf{com}}$ records

the value msg. Subsequently, the functionality lets both the receiver P_j and the adversary $\mathcal{S}$ know that the committer has committed to some value by computing a public delayed output $(\mathsf{receipt}, \mathtt{sid}, \mathtt{ssid})$ and sending it to P_j (this phase is called the commitment phase).

To initiate the decommitment phase, the committer P_i sends $(\mathsf{open}, \mathtt{sid}, \mathtt{ssid})$ to the functionality $\mathcal{F}_{\mathsf{com}}$. Thereupon, $\mathcal{F}_{\mathsf{com}}$ checks if there exists a value msg; if so, the functionality computes a public delayed output $(\mathsf{open}, \mathtt{sid}, \mathtt{ssid}, \mathsf{msg})$ and sends it to P_j. When the adversary corrupts the committer by sending $(\mathsf{corrupt} - \mathsf{committer}, \mathtt{sid}, \mathtt{ssid})$ to $\mathcal{F}_{\mathsf{com}}$, the functionality reveals the recorded value msg to the adversary $\mathcal{S}$. Furthermore, if the receipt value was not yet delivered to P_j, then $\mathcal{F}_{\mathsf{com}}$ allows the adversary to modify the committed value. This is in order to deal with adaptive corruptions. The ideal functionality for commitment schemes $\mathcal{F}_{\mathsf{com}}$ is given in Figure 5.

$\mathcal{F}_{\mathsf{com}}$ is parameterized by an integer N and runs with parties P_i, P_j, and adversary $\mathcal{S}$. It initially sets $(n, S, R) = (1, \perp, \perp)$.

The functionality ignores any commit-input if $n > N$, or if $n > 1$ and $(S, R) \neq (P_i, P_j)$ for the parties' identities in the input. Else,

- Upon receiving input $(\mathsf{commit}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, \mathsf{msg})$ from party P_i, $\mathcal{F}_{\mathsf{com}}$ proceeds as follows:
  - ⋆ Records msg, generate a public delayed output $(\mathsf{receipt}, \mathtt{sid}, \mathtt{ssid})$, and send the output to P_j. Increment n to $n + 1$ and store $(S, R) = (P_i, P_j)$ if now $n = 2$.
- Upon receiving input $(\mathsf{open}, \mathtt{sid}, \mathtt{ssid})$ from party P_i, $\mathcal{F}_{\mathsf{com}}$ proceeds as follows:
  - ⋆ If a value msg is recorded, generate a public delayed output $(\mathsf{open}, \mathtt{sid}, \mathtt{ssid}, \mathsf{msg})$ and send it to P_j.
  - ⋆ Otherwise, do nothing.
- Upon receiving the input $(\mathsf{corrupt} - \mathsf{committer}, \mathtt{sid}, \mathtt{ssid})$ from the adversary $\mathcal{S}$, $\mathcal{F}_{\mathsf{com}}$ proceeds as follows:
  - ⋆ Send the value msg to $\mathcal{S}$.
  - ⋆ If $\mathcal{S}$ provides a value msg′ and the receipt output was not yet written on P_j's tape, then $\mathcal{S}$ can change the recorded value to msg′.

Figure 5: The ideal functionality for commitment schemes adapted from [4].

6.2 PUF-based Commitment Scheme

We now provide a *universal* transformation from OT-protocols to bit commitment schemes which —to our knowledge— has not been considered so far. Previous transformations [23, 9] rely on cut-and-choose and require linear many executions of the OT-protocol. Our transformation only requires a single additional message to be sent after executing the OT-protocol. The main idea of the protocol in Figure 6 is to inverse the roles of the sender and the receiver.

The OT-protocol transfers two secrets, and the committer only learns one of them, namely the one corresponding to its secret bit b. This secret is then used to open the commitment.

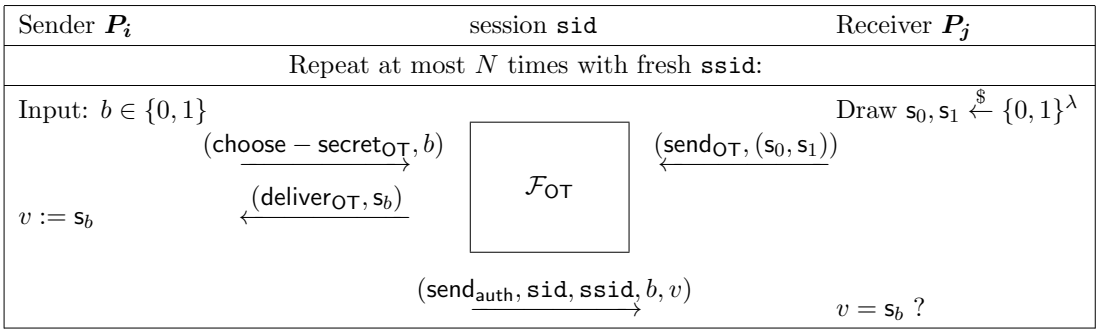

Figure 6: Commitment scheme with $\mathcal{F}_{\mathsf{OT}}$.

Theorem 6.1 *The commitment protocol in Figure 6 securely UC-realizes the functionality $\mathcal{F}_{\mathsf{com}}$ in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model.*

If functionality $\mathcal{F}_{\mathsf{OT}}$ is replaced by some OT-protocol, then the derived commitment protocol basically inherits the characteristics of the OT-protocol. That is, it is secure against adaptive corruptions if OT is, and it is statistically secure if OT is. Remarkably, we show in the next section that our PUF-based OT-protocol, while being only statically secure, makes the commitment scheme even adaptively secure.

We merely provide a proof sketch for Theorem 6.1 here. Note that in the case where both users are honest, only the modeling of the final message needs to be taken into consideration. The simulator learns the secret bit b from the commitment functionality $\mathcal{F}_{\mathsf{com}}$. It then draws a random string v from $\{0,1\}^\lambda$ and sends $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, b, v)$. If the receiver is dishonest, then it provides $\mathcal{F}_{\mathsf{OT}}$ with two secrets $\mathsf{s}_0, \mathsf{s}_1$. The simulator lets the sender provide a random bit b' to the simulated $\mathcal{F}_{\mathsf{OT}}$. It receives back the secret $\mathsf{s}_{b'}$. In the opening phase, $\mathcal{S}$ learns the (real) secret bit b from the commitment functionality and simulates the final protocol message as $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, b, \mathsf{s}_b)$. If the sender is corrupt then it provides the simulated $\mathcal{F}_{\mathsf{OT}}$ with a secret bit b. The simulator creates two random strings $\mathsf{s}_0, \mathsf{s}_1$ and passes them to the simulated $\mathcal{F}_{\mathsf{OT}}$ which passes s_b to the receiver. The simulator commits to the sender's bit b in the ideal world. If the sender sends a message $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, b, v)$ then $\mathcal{S}$ checks if $\mathsf{s}_b = v$. If so, it instructs $\mathcal{F}_{\mathsf{com}}$ to open the commitment.

All simulations are perfect.

6.3 Adaptively Secure Commitments

Consider the concrete commitment protocol where we plug in our OT-protocol from the previous section into the abstract scheme above (and work in the $\mathcal{F}_{\mathsf{PUF}}$-hybrid model instead of

the $\mathcal{F}_{\mathsf{OT}}$-hybrid model then), then we observe the following: In the commitment phase (i.e., the OT-phase), the message sent by the OT-receiver (=commitment sender) is statistically independent from its secret input: the OT-receiver merely sends a single uniformly random message.

For the OT-sender, this is not the case: When having access to the PUF, one can extract both secrets from the mere transcript of the protocol. This enables the simulator $\mathcal{S}$ to derive both secrets from the protocol, as it accesses the PUF. It can thus provide the simulated committer with open messages for both bit values. As the remaining part of the committer's state merely consists in challenge-response-pairs, the simulator can thus provide genuine internal state.

7 Key Exchange with PUFs

In a key exchange (ke) protocol two parties interact over an insecure network to establish a common secret key κ. This common secret key can then be used to build a secure channel or to ensure confidentiality of transmitted data.

7.1 The Key Exchange Ideal Functionality

The main idea of the key exchange ideal functionality $\mathcal{F}_{\mathsf{ke}}$ is the following: if both parties are honest, the functionality provides them with a common random value which is invisible to the adversary. If one of them is corrupted, though, the adversary determines the session key entirely thus modeling the participation of a corrupted party. The definition of the key exchange functionality $\mathcal{F}_{\mathsf{ke}}$ is depicted in Figure 7 adapted from [7].

7.2 Minimal Requirements

We present a key exchange protocol in Section 7.3 which sends a PUF in a setup phase. Afterwards, a single message per protocol execution is sent via a unidirectional authenticated channel. As mentioned in the introduction, it is desirable to circumvent the use of complexity-theoretic assumptions. However, for practical reasons, PUF transfers should also be minimized. If only a single PUF transfer occurs, then the assumption of a unidirectional authenticated channel cannot be dropped: The sender of the PUF measured the PUF several times and sent it to the receiver. The adversary can query the PUF during its transition. If the sender does not have any further secret information for authentication, then the adversary can can carry out the same computations as the sender. Thus, the protocol cannot be secure against impersonation attacks. In the following, we use the standard bidirectional $\mathcal{F}_{\mathsf{auth}}$ functionality. Deriving corresponding unidirectional definitions is straightforward.

$\mathcal{F}_{\mathsf{ke}}$ is parameterized by an integer N and receives as input a security parameter 1^λ, and runs with parties $P_1, ..., P_n$ and adversary $\mathcal{S}$. $\mathcal{F}_{\mathsf{ke}}$ obtains a list of corrupt parties. It initially sets $(n, S, R) = (1, \perp, \perp)$.

Ignore any $\mathsf{establish-session_{ke}}$-input if $n > N$, or if $n > 1$ and $(S, R) \neq (P_i, P_j)$ for the parties' identities in the input. Else,

- $(\mathsf{establish-session_{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ is written on $\mathcal{F}_{\mathsf{ke}}$'s input tape by a party P_i. Then $\mathcal{F}_{\mathsf{ke}}$ stores the tuple $(\mathsf{establish-session_{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ (and refuses if there already is a tuple $(\mathsf{establish-session_{ke}}, \mathtt{sid}, \mathtt{ssid}, P_j, P_i)$ or a tuple $(\mathsf{establish-session_{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$). $\mathcal{F}_{\mathsf{ke}}$ outputs $(\mathsf{establish-session_{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ to the adversary $\mathcal{S}$. If both users are honest then draw a random value κ from $\{0,1\}^\lambda$ and store the messages $(\mathsf{deliver_{ke}}, \mathtt{sid}, \mathtt{ssid}, \kappa, P_i)$ and $(\mathsf{deliver_{ke}}, \mathtt{sid}, \mathtt{ssid}, \kappa, P_j)$. Increment n to $n + 1$.
- When $\mathcal{S}$ writes $(\mathsf{choose-value_{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, \kappa)$ on $\mathcal{F}_{\mathsf{ke}}$'s input tape then check whether there is a message $(\mathsf{establish-session_{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ or a message $(\mathsf{establish-session_{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ and whether at least one of the users P_i and P_j is corrupt. If so, store the messages $(\mathsf{deliver_{ke}}, \mathtt{sid}, \mathtt{ssid}, \kappa, P_i)$ and $(\mathsf{deliver_{ke}}, \mathtt{sid}, \mathtt{ssid}, \kappa, P_j)$.
- $\mathcal{S}$ writes $(\mathsf{deliver_{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i)$ on $\mathcal{F}_{\mathsf{ke}}$'s input communication tape. Check if a tuple $(\mathsf{deliver_{ke}}, \mathtt{sid}, \mathtt{ssid}, \kappa, P_i)$ is stored. If so, write $(\mathsf{deliver_{ke}}, \mathtt{sid}, \mathtt{ssid}, \kappa, P_i)$ to P_i's input tape and delete $(\mathsf{deliver_{ke}}, \mathtt{sid}, \mathtt{ssid}, \kappa, P_i)$. Else, do nothing.

Figure 7: The key exchange ideal functionality adapted from [7].

7.3 PUF-based Key Exchange Scheme

Intuitively, our key exchange protocol proceeds as follows. In an enrollment phase, a server issues a PUF, measures for a set of randomly chosen challenges the corresponding responses, and finally ensures a noisy-free PUF measurement by generating for each response r a fuzzy extractor secret st from a set of random secrets as well as a corresponding helper data p. The server then sends the PUF to the client. Upon finishing the enrollment phase the server broadcasts a randomly chosen challenge c including its helper data p to the client and sets $\kappa = st$ to obtain the protocol key. The client evaluates the PUF on the challenge c, computes the corresponding fuzzy secret st due to the helper data p, and obtains the protocol key by setting $\kappa = st$. Consequently, both parties use the fuzzy extractor secret st as their common protocol key κ.

Theorem 7.1 *Protocol* PUFKE *securely realizes functionality* $\mathcal{F}_{\mathsf{ke}}$. *in the* $\mathcal{F}_{\mathsf{PUF}}$*-hybrid model.*

Proof. Throughout all simulations, queries to PUFs are genuinely relayed to the $\mathcal{S}$'s PUF functionality, and the PUF's answer is honestly delivered to the querying party.

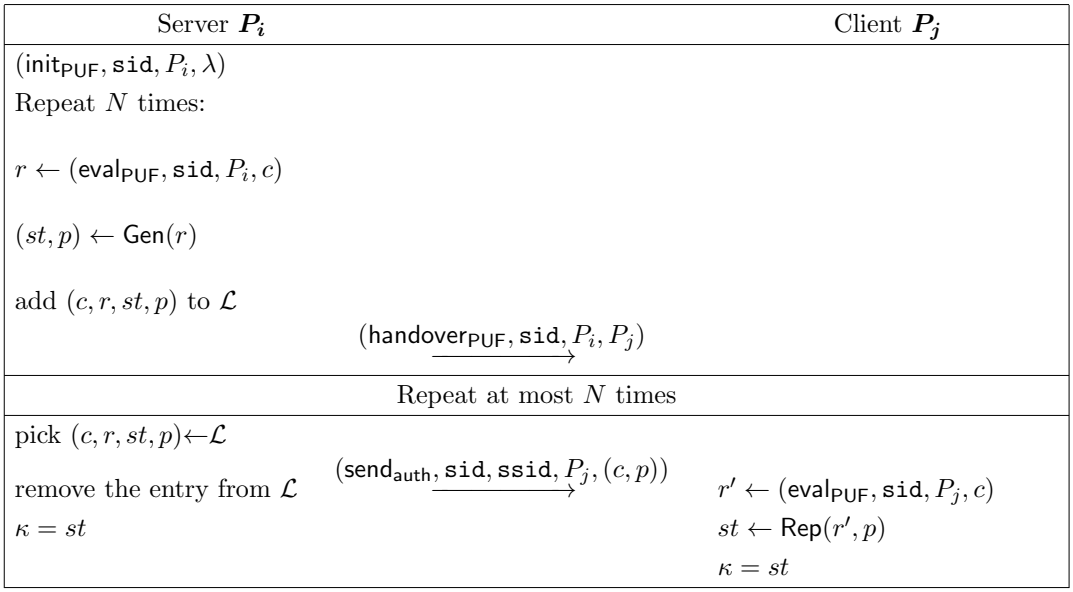

Figure 8: PUF-based key exchange scheme.

P_i and P_j are honest Whenever the functionality $\mathcal{F}_{\mathsf{ke}}$ sends message $(\mathsf{establish}-\mathsf{session}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ for the first time, the $\mathcal{S}$ initiates a PUF and queries it on N random challenges $c_1, ..., c_N$ to obtain responses $r_1, ..., r_N$. It then computes $(st_i, p_i) \leftarrow \mathsf{Gen}(r_i)$, and stores all tuples (c_i, r_i, st_i, p_i) in a list $\mathcal{L}$. The simulator $\mathcal{S}$ then simulates a $\mathsf{handover}_{\mathsf{PUF}}$ and lets the adversary query the PUF, until the adversary terminates the transition phase. The $\mathcal{S}$ sets the counter n to 1 and continues with the simulation of the first session. This concludes the simulation of the setup phase.

On receiving a message $(\mathsf{establish}-\mathsf{session}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$, the simulator $\mathcal{S}$ increases the counter n by one and sends $(\mathsf{deliver}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i)$ to $\mathcal{F}_{\mathsf{ke}}$. The party P_i then writes the key on its local output tape and $\mathcal{S}$ is activated again. It simulates that P_i sends the message $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, (c_n, p_n))$. When the adversary instructs to deliver the latter message to P_j, then the simulator $\mathcal{S}$ sends $(\mathsf{deliver}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ to $\mathcal{F}_{\mathsf{ke}}$.

Analysis of the simulation: Due to the well-spread domain property, with overwhelming probability, the adversary did not query the PUF on values closer than d_{min} to one of the challenges c_i. By response independence, the message (c_n, p_n) is then statistically independent from the value st. Hence, the simulation is sound.

The Receiver P_j is corrupt. The simulation of the setup phase is as in the honest case.

On receiving a message $(\mathsf{establish}-\mathsf{session}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$, the simulator $\mathcal{S}$ increases the counter n by one and writes $(\mathsf{choose}-\mathsf{value}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, st_n)$ on $\mathcal{F}_{\mathsf{ke}}$'s input tape. It

is activated again and writes $(\mathsf{deliver}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i)$ to $\mathcal{F}_{\mathsf{ke}}$. The party P_i then writes the key on its local output tape and $\mathcal{S}$ is activated again. It simulates that P_i sends $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, (c_n, p_n))$. When the adversary instructs to deliver the latter message to P_j, then the simulator $\mathcal{S}$ sends $(\mathsf{deliver}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ to $\mathcal{F}_{\mathsf{ke}}$. The simulation is perfect.

The Sender P_i is corrupt. The $\mathcal{S}$ allows the malicious user P_i to instantiate an arbitrary number of PUFs. The receiver only accepts a single $\mathsf{handover}_{\mathsf{PUF}}$. When the adversary instructs to deliver a message $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, (c, p))$ to P_j, then the $\mathcal{S}$ evaluates the PUF on c to obtain response r and computes $st \leftarrow \mathsf{Rep}(r, p)$. The $\mathcal{S}$ lets the corrupt dummy user P_i write the message $(\mathsf{establish} - \mathsf{session}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ on the input tape of $\mathcal{F}_{\mathsf{ke}}$ and subsequently passes the messages $(\mathsf{choose} - \mathsf{value}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, st)$ and $(\mathsf{deliver}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_j)$ to $\mathcal{F}_{\mathsf{ke}}$. The simulation is perfect.

□

8 Conclusion

While our modeling of PUFs is comprehensive enough to cover different PUF types, based on different technologies, our protocols withstanding temporary adversarial access requires that the challenge space of the PUF is super-polynomial. Else the adversary could perform a full read-out. From a technological point of view, currently only optical PUFs satisfy this (see [25] for an overview). To broaden the set of admissible PUFs one could develop methods to enlarge the challenge space for PUFs, either by technological means or algorithmically via domain extensions. In this case, our ideas can be applied as well, but at the cost of efficiency, essentially increasing the number of PUF tokens to be used.

Acknowledgments

We thank the anonymous reviewers for valuable comments.

The second author was supported by grants Fi 940/2-1 and Fi 940/3-1 of the German Research Foundation (DFG). This work was also supported by CASED (`www.cased.de`)

References

[1] Frederik Armknecht, Roel Maes, Ahmad-Reza Sadeghi, Francois-Xavier Standaert, and Christian Wachsmann. A formal foundation for the security features of physical functions. *To appear at IEEE S&P*, 2011.

[2] Frederik Armknecht, Roel Maes, Ahmad-Reza Sadeghi, Berk Sunar, and Pim Tuyls. Memory leakage-resilient encryption based on physically unclonable functions. In *ASI-*

ACRYPT, volume 5912 of *Lecture Notes in Computer Science*, pages 685–702. Springer, 2009.

[3] Gilles Brassard, Claude Crépeau, and Jean-Marc Robert. Information theoretic reductions among disclosure problems. In *FOCS*, pages 168–173. IEEE, 1986.

[4] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[5] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007.

[6] Ran Canetti and Marc Fischlin. Universally composable commitments. In *Advances in Cryptology - Crypto 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, 2001.

[7] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2002.

[8] Ran Canetti and Tal Rabin. Universal composition with joint state. In *Crypto*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 2003.

[9] Claude Crépeau. Equivalence between two flavours of oblivious transfers. In *CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 350–354. Springer, 1988.

[10] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[11] Keith B. Frikken, Marina Blanton, and Mikhail J. Atallah. Robust authentication using physically unclonable functions. In *ISC*, volume 5735 of *Lecture Notes in Computer Science*, pages 262–277. Springer, 2009.

[12] Blaise Gassend, Marten van Dijk, Dwaine E. Clarke, Emina Torlak, Srinivas Devadas, and Pim Tuyls. Controlled physical random functions and applications. *ACM Trans. Inf. Syst. Secur.*, 10(4), 2008.

[13] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.

[14] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2008.

[15] Vipul Goyal, Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. Interactive locking, zero-knowledge pcps, and unconditional cryptography. In *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 173–190. Springer, 2010.

[16] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 308–326. Springer, 2010.

[17] Jorge Guajardo, Sandeep S. Kumar, Geert-Jan Schrijen, and Pim Tuyls. Fpga intrinsic pufs and their use for ip protection. In *Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems*, pages 63–80. Springer-Verlag, 2007.

[18] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[19] Carmit Hazay and Yehuda Lindell. Constructions of truly practical secure protocols using standard smartcards. In *ACM Conference on Computer and Communications Security*, pages 491–500. ACM, 2008.

[20] Dennis Hofheinz, Dominique Unruh, and Jörn Müller-Quade. Universally composable zero-knowledge arguments and commitments from signature cards. In *In Proc. of the 5th Central European Conference on Cryptology MoraviaCrypt 2005*, 2005.

[21] Tanya Ignatenko, Geert-Jan Schrijen, Boris Skoric, Pim Tuyls, and Frans M. J. Willems. Estimating the secrecy rate of physical uncloneable functions with the context-tree weighting method. In *Proc. IEEE International Symposium on Information Theory 2006*, pages 499–503, Seattle, USA, July 2006.

[22] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 115–128. Springer, 2007.

[23] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31. ACM, 1988.

[24] Daihyun Lim. Extracting secret keys from integrated circuits. Master's thesis, MIT, 2004.

[25] Roel Maes and Ingrid Verbauwhede. *Physically Unclonable Functions: a Study on the State of the Art and Future Research Directions*, section 1. Towards Hardware-Intrinsic Security. Springer, 2010.

[26] Tal Moran and Gil Segev. David and goliath commitments: Uc computation for asymmetric parties using tamper-proof hardware. In *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 527–544. Springer, 2008.

[27] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *Crypto*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126. Springer, 2002.

[28] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld. Physical one-way functions. *Science*, 297:2026–2030, 2002.

[29] Ravikanth Srinivasa Pappu. *Physical One-Way Functions*. Phd thesis, Massachusetts Institut of Technology, Massachusetts Institut of Technology, March 2001.

[30] Ulrich Rührmair. Oblivious transfer based on physical unclonable functions. In *TRUST*, volume 6101 of *Lecture Notes in Computer Science*, pages 430–440. Springer, 2010.

[31] Ulrich Rührmair, Jan Sölter, and Frank Sehnke. On the foundations of physical unclonable functions, 2009.

[32] Ahmad-Reza Sadeghi, Ivan Visconti, and Christian Wachsmann. *Enhancing RFID Security and Privacy by Physically Unclonable Functions*. Towards Hardware-Intrinsic Security. Springer, 2010.

[33] B. Skoric, S. Maubach, T. Kevenaar, and P. Tuyls. Information-theoretic analysis of coating pufs. Technical report, Philips Research Laboratories, 2006.

[34] Boris Skoric, Stefan Maubach, Tom Kevenaar, and Pim Tuyls. Information-theoretic analysis of capacitive physical unclonable functions. *Journal of Applied physics*, 100, 2006.

[35] G. Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th annual Design Automation Conference*, DAC '07, pages 9–14, New York, NY, USA, 2007. ACM.

[36] P. Tuyls, B. Skoric, S. Stallinga, T. Akkermans, and W. Ophey. Information-theoretic security analysis of physical uncloneable functions. In Andrew S. Patrick and Moti Yung, editors, *Financial Cryptography and Data Security*, volume 3570. Springer Berlin / Heidelberg, 2005.

[37] Pim Tuyls, Geert-Jan Schrijen, Boris Skoric, Jan Geloven, Nynke Verhaegh, and Rob Wolters. Read-proof hardware from protective coatings. In *CHES*, pages 369–383, 2006.

[38] Pim Tuyls, Boris Skoric, Tanya Ignatenko, Frans Willems, and Geert-Jan Schrijen. Entropy estimation for optical pufs based on context-tree weighting methods. In Pim Tuyls, Boris Skoric, and Tom Kevenaar, editors, *Security with Noisy Data*, pages 217–233. Springer London, 2007.

[39] Pim Tuyls, Boris Skoric, and Thomas Andreas Maria Kevenaar. *Security with Noisy Data: Private Biometrics, Secure Key Storage and Anti-Counterfeiting*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[40] Stefan Wolf and Jürg Wullschleger. Oblivious transfer is symmetric. In *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 222–232. Springer, 2006.

Appendix

A Security Proof for the OT-Protocol of Section 5

As only static corruptions are considered, we can distinguish the simulation depending on the corrupted parties being involved. Our simulator $\mathcal{S}$ runs a black-box simulation of adversary $\mathcal{A}$, emulating the communication of honest parties with the limited information provided in the ideal model, and such that $\mathcal{S}$ can use the adversary to reply to the environment. As explained, for non-programmability we consider restricted simulators which faithfully initialize a PUF and allow the environment straight access when the PUF is in possession of the simulator. In the analysis below we analyze only a single execution; the argument extends straightforwardly to the at most N runs.

Simulating the case that both parties are honest.

- Whenever $\mathcal{F}_{\mathsf{OT}}$ writes $(\mathsf{send}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ to the communication input tape of $\mathcal{S}$ in the ideal world, then this message indicates in the real world that $\mathcal{Z}$ wrote two secrets $(\mathsf{s}_0, \mathsf{s}_1)$ to the communication input tape of P_i. $\mathcal{S}$ now draws two random values $(\mathsf{s}'_0, \mathsf{s}'_1)$ as fake secrets and writes them to the local input tape of the simulated P_i in the ideal world. The simulator $\mathcal{S}$ then chooses two random values $x_0, x_1 \in \{0,1\}^\lambda$ as the real P_i and runs the P_i algorithm which will write $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, (x_0, x_1))$ to the simulated copy of $\mathcal{F}_{\mathsf{auth}}$.
- If at some point, $\mathcal{F}_{\mathsf{OT}}$ outputs $(\mathsf{choose} - \mathsf{secret}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$, then $\mathcal{S}$ draws a random bit b and writes it to the local input tape of the simulated copy of the user P_j.
- All participants, i.e., the simulated parties, $\mathcal{F}_{\mathsf{PUF}}$, and $\mathcal{A}$ might write on the (simulated) input tapes of each other and are activated upon receiving input to run according to their program. As long as they produce no local output, $\mathcal{S}$ simply relays and activates its respective subroutines.

- If one of the simulated honest users P_j produces local output, then this indicates that the corresponding session $\mathtt{sid}$ finished running the protocol. Note that only the receiver P_j produces an output in this protocol and would output the secret s_b to the environment as the real P_j. In this case, $\mathcal{S}$ writes $(\mathsf{deliver}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ on the input tape of $\mathcal{F}_{\mathsf{OT}}$.

In the setup phase, $\mathcal{A}$ and $\mathcal{Z}$ had access to the PUF used in the protocol and may make polynomially many measurements. We show that with high probability, for the adversary $\mathcal{A}$, the sequence of strings $(x_0, x_1, c \oplus x_b, \mathsf{s}_0 \oplus st_0, \mathsf{s}_1 \oplus st_1)$ looks like a random 5-tuple of strings. Towards this goal, we are going to show that with high probability, the five random variables corresponding to each of the entries are almost uniformly and independently distributed. As dependence is a symmetric property, it suffices to show that no random variable depends on the previous ones: x_0 and x_1 are drawn independently at random; $c \oplus x_b$ is defined by drawing c at random and then Xoring it to x_b; the distribution of $c \oplus x_b$ is independent from x_0, x_1 and b. The well-spread domain property (see Subsection 3.2) assures that with overwhelming probability, the adversary did not query the PUF on challenges closer than d_{min} from $x_0 \oplus c \oplus x_b$ or $x_1 \oplus c \oplus x_b$. Assume from now on that this is indeed the case. Then, due to the well-spread domain property (see Subsection 3.2), the first outputs of $\mathsf{Gen}(r'_0)$ and $\mathsf{Gen}(r'_1)$ are distributed almost according to U_ℓ. Even when observing the helper data, their statistical distance from U_ℓ is at most $\epsilon(\lambda)$ and, thus, their statistical distance from distributions corresponding other challenges (and other values $(x_0, x_1, c \oplus x_b)$) is at most $2\epsilon(\lambda)$ and thereby negligible which concludes the analysis.

Receiver is corrupt. We next consider the case where the sender P_i is honest and the receiver P_j is dishonest. The simulator $\mathcal{S}$ observes the dishonest receivers' PUF queries (made by $\mathcal{A}$ and $\mathcal{Z}$) in the setup phase and stores the challenge-response pairs in a list $\mathcal{L}$. Those will later help the simulator to extract secret bits in the protocol execution. The simulator also keeps an initially empty list of challenge values $\mathcal{C}$.

Whenever $\mathcal{F}_{\mathsf{OT}}$ writes $(\mathsf{send}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ on the input tape of $\mathcal{S}$ then the simulator $\mathcal{S}$ draws a pair of random values (x_0, x_1) from $\{0,1\}^\lambda$ and sends $(\mathtt{sid}, P_i, P_j, (x_0, x_1))$ in the simulation. When $\mathcal{A}$ instructs the simulated P_j to send v to P_i as in the real world, simulator first checks if $\mathsf{dis}(v \oplus x_0, \mathcal{C}) > d_{\mathsf{min}}$ and $\mathsf{dis}(v \oplus x_1, \mathcal{C}) > d_{\mathsf{min}}$. If so, the simulator $\mathcal{S}$ checks for each $(c, r) \in \mathcal{L}$, if either the hamming distance of c and $v \oplus x_0$ is smaller than d_{min}, or if the hamming distance of c and $v \oplus x_1$ is smaller than d_{min}. With overwhelming probability, for a single challenge c, both cases cannot occur simultaneously, as $\mathsf{dis}(c, v \oplus x_0) < d_{\mathsf{min}}$ and $\mathsf{dis}(c, v \oplus x_1) < d_{\mathsf{min}}$ implies $\mathsf{dis}(v \oplus x_0, v \oplus x_1) < 2d_{\mathsf{min}}$ and $\mathsf{dis}(x_0, x_1) < 2d_{\mathsf{min}}$, but due to the well-spread domain property, the latter only occurs in negligibly many cases. We established that for a single challenge, only one of the statements can hold. We now show that with high probability, either there are no challenges in $\mathcal{L}$ close to $v \oplus x_0$, or there are none which are close to $v \oplus x_1$. Assume, the adversary had non-negligible probability that there are

c_0, c_1 in his list of previously made queries to the PUF, such that $\mathsf{dis}(c_0, v \oplus x_0) < d_{\mathsf{min}}$ and $\mathsf{dis}(c_1, v \oplus x_1) < d_{\mathsf{min}}$. It follows that $\mathsf{dis}(c_0 \oplus x_0 \oplus x_1, c_1) < 2d_{\mathsf{min}}$, or, equivalently, that $\mathsf{dis}(x_0 \oplus x_1, c_0 \oplus c_1) < 2d_{\mathsf{min}}$. Now, $x := x_0 \oplus x_1$ is a random value which, since $\mathcal{A}$ has at this point already made all queries, independent of any c-values which $\mathcal{A}$ has evaluated the PUF for. Thus, if $\mathcal{A}$ has non-negligible probability of having chosen such two challenges c_0 and c_1, then this must hold for a non-negilible fraction of values $x \in \{0,1\}^\lambda$. Let $p(\lambda)$ the number of challenges, the adversary queried. Then, the sums of two different challenges c_i and c_j are at most $\binom{p(\lambda)}{2} = \frac{1}{2}p(\lambda)(p(\lambda)-1)$ which is a negligible fraction of 2^λ. If we multiply this number by the number of elements in a ball of radius $2d_{\mathsf{min}}$, then the property still holds, as $2d_{\mathsf{min}}$ is still in $o(\lambda/\log\lambda)$. Hence, $\mathcal{A}$ cannot cover a non-negligible number of values x.

If there only exist challenges c with $\mathsf{dis}_{\mathsf{ham}}(x_0, v \oplus c) < d_{\mathsf{min}}$, then set $b := 0$. If there only exist challenges c with $\mathsf{dis}_{\mathsf{ham}}(x_1, v \oplus c) < d_{\mathsf{min}}$, then set $b := 1$. If no such challenge exists, then draw $b \leftarrow \{0,1\}$ at random. The simulator $\mathcal{S}$ now sends $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_j, P_i, v)$ in the simulation. Since P_j is dishonest, $\mathcal{S}$ can now write $(\mathsf{choose-secret}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, b)$ on the input tape of $\mathcal{F}_{\mathsf{OT}}$, and, upon receiving the message $(\mathsf{choose-secret}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$, $\mathcal{S}$ writes $(\mathsf{deliver}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ on the input tape of $\mathcal{F}_{\mathsf{OT}}$ which passes the message $(\mathsf{deliver}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, \mathsf{s}_b)$ to the corrupt user P_j and consequently to the simulator $\mathcal{S}$ which uses this value to compute a simulated output for P_j which is controlled by $\mathcal{A}$. The simulator $\mathcal{S}$ picks a random value $\mathsf{s}_{1-b} \leftarrow \{0,1\}^\lambda$. In response to the message $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, v)$, simulator uses the PUF and Gen to determine st_0, p_0, st_1 and p_1. The simulator $\mathcal{S}$ then passes $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, (\mathsf{s}_0 \oplus st_0, p_0, \mathsf{s}_1 \oplus st_1, p_1))$ to the simulated $\mathcal{F}_{\mathsf{auth}}$.

We now demonstrate that the joint view of $\mathcal{Z}$ and $\mathcal{A}$ in the protocol execution is indistinguishable from the joint view of $\mathcal{Z}$ and the simulated $\mathcal{A}$ with $\mathcal{S}$ in the ideal world model. The only difference between the two executions is the final message $(\mathsf{s}_0 \oplus st_0, p_0, \mathsf{s}_1 \oplus st_1, p_1)$ received by P_j. The checks performed assure that $\mathcal{A}$ did not query the PUF on any challenge with distance smaller than d_{min} from $v \oplus x_{1-b}$. Thus, by response independence (see Subsection 3.2), st_{1-b} is quasi uniform even in the presence of p_{1-b}. Thus, from the point of view of $\mathcal{A}$, the value $(\mathsf{s}_{1-b} \oplus st_{1-b}, p_{1-b})$ is identically distributed for all values s_{1-b}.

Sender is corrupt. We finally assume that the sender P_i is dishonest and the receiver P_j is honest. The $\mathcal{S}$ will use its permanent PUF access to extract the secrets $(\mathsf{s}_0, \mathsf{s}_1)$ from the dishonest sender. We now describe the simulation in detail: The simulator $\mathcal{S}$ waits for two conditions to be satisfied in arbitrary order:

- $\mathcal{A}$ instructs the malicious party P_i to send $(\mathtt{sid}, \mathtt{ssid}, P_i, P_j, (x_0, x_1))$ in the simulation.
- The ideal functionality $\mathcal{F}_{\mathsf{OT}}$ writes $(\mathsf{choose-secret}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ on the input tape of $\mathcal{S}$.

The simulator $\mathcal{S}$ then draws a random string $v \leftarrow \{0,1\}^\lambda$ and returns it to P_i. When $\mathcal{A}$ instructs P_i to send $(\mathtt{sid}, \mathtt{ssid}, P_i, P_j, a_0, p_0, a_1, p_1)$ to P_j, then $\mathcal{S}$ extracts the secrets s_0, s_1 as follows: The simulator $\mathcal{S}$ evaluates the PUF on $v \oplus x_0$ and $v \oplus x_1$ to get responses r_0 and r_1. It then uses the helper data p_0, p_1 and Rep to obtain st_0 and st_1. If one of the two Rep evaluation fails, then the corresponding value $\mathsf{s}_0/\mathsf{s}_1$ is set to be the empty value $\perp$. Else, they are set to be $\mathsf{s}_0 := st_0 \oplus a_0$ and $\mathsf{s}_1 := st_1 \oplus a_1$. The $\mathcal{S}$ then writes $(\mathsf{send}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, (\mathsf{s}_0, \mathsf{s}_1))$ on the input tape of the ideal functionality $\mathcal{F}_{\mathsf{OT}}$ and also instructs it with the message $(\mathsf{deliver}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$.

We now show that the simulation for the case that P_i is dishonest is indistinguishable from the real world execution. This is the case (1) since in both world the message received by P_i is a uniformly random string; and (2) since the output of P_j is the same in both world executions. We elaborate on the latter: If P_j has input bit $b = 0$ then it would proceed as the simulator $\mathcal{S}$ did to compute s_0. If P_j has input bit $b = 1$ then it would compute s_1 in the same way as the simulator $\mathcal{S}$.

B Complexity Theoretical Background on UC and PUFs

It is currently not known whether PUF distributions are simulatable by a probabilistic polynomial time (PPT) Turing machine. If so, then the UC framework allows to integrate PUFs easily. However, if they are not, then the UC theorem needs to be extended, because a PPT algorithm which has access to a PUF is computationally strictly stronger than a common PPT algorithm. It is thus important to investigate whether the environment $\mathcal{Z}$, the adversary $\mathcal{A}$, and the simulator $\mathcal{S}$ need to have access to a PUF.

When an algorithm is called a PUF-algorithm/-adversary/-simulator/-environment, then this means that it is a PPT algorithm which additionally has access to a PUF-functionality as described in Subsection 4.2 with the difference, that only $\mathsf{init}_{\mathsf{PUF}}$ and $\mathsf{eval}_{\mathsf{PUF}}$ queries may be asked. We sometimes also call this PUF-power or complexity PPT+PUF. This means, that the algorithm locally owns a PUF which it uses for computation. Moreover, these algorithms may "share" PUFs, and they may share PUFs with honest parties which use the PUFs during protocols. In the following, it is devoted to provide a clear and surprisingly simple view on the different possibilities to handle this issue as well as an analysis of compatibility with the universal composition theorem.

In the remaining part of this section, PUFs are considered not to be PPT simulatable. However, the observations do not become obsolete in case this claim turns out to be wrong, in particular because of programmability considerations. We will come back to this question in the end of this section.

B.1 PUF Access Models

We first argue that the real world adversary $\mathcal{A}$ and the simulator $\mathcal{S}$ both need to access PUFs and then investigate delicately on the different possibilities for PUF availability of the

environment.

We consider protocols that use PUFs. Thus, as a matter of fact, the real world adversary $\mathcal{A}$ may access PUFs through any of the corrupted parties taking part in the protocol. Thus, we consider the adversary $\mathcal{A}$ to have PUF-power. Consequently, the simulator $\mathcal{S}$ needs to be provided with PUF-power too. Else, any environment would trivially be able to distinguish the real world from the ideal world, as for $\mathcal{A}$ could send samples from its PUF to $\mathcal{Z}$. If $\mathcal{S}$ tries to simulate those, it fails because the PUF distribution is not PPT-simulatable so that the environment $\mathcal{Z}$ manages to distinguish the two adversaries easily irrespective of which protocol is considered.[2]

We now turn to the environment $\mathcal{Z}$. Figures 9, 10, and 11 illustrate different solutions to the problem. In Figure 9, the environment $\mathcal{Z}$ does not access any PUF, whereas in Figure 10, the environment has PUF-power. However, the adversary and the environment do not necessarily operate on the same PUF *instance*. This model merely takes care of the fact that the environment has the same computational power as the adversaries. In Figure 11, the environment as well as the adversary $\mathcal{A}$ and the honest protocol participants access the same PUF functionality and thus possibly the same instance. This setting corresponds to the extend UC framework with shared functionalities [5].

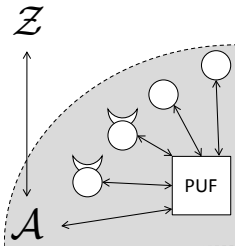

Figure 9: The environment $\mathcal{Z}$ does not access a PUF.

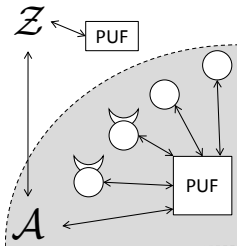

Figure 10: The environment $\mathcal{Z}$ and the adversary $\mathcal{A}$ each access different PUFs.

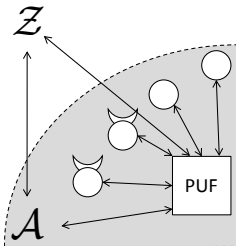

Figure 11: The environment $\mathcal{Z}$, the adversary $\mathcal{A}$ and the protcol participants all access the same PUF.

The first model should generally be discarded. If a PUFs add indeed computational power exceeding PPT, then providing security only against mere PPT algorithms is a weak choice and on top technically challenging, as the achieved security level is insufficient for universal composition in the presence of PUFs. See Subsection B.2 for inspection of the UC theorem and background on this issue.

We now turn to the second model. The adversary $\mathcal{A}$ and $\mathcal{Z}$ may exchange arbitrary information — at a first glance, it seems unnatural to require that they may not share the

[2]More precisely, we are assuming here that it is even hard for PPT algorithms to produce indistinguishable distributions from PUF distributions.

same PUF. On the other hand, one might get the impression that there is no major difference between giving $\mathcal{Z}$ direct access to the PUF and providing the access through $\mathcal{A}$. But indeed, there is. In the second model, the simulator $\mathcal{S}$ is far more powerful. The simulator simulates $\mathcal{A}$ and thus controls $\mathcal{A}$'s access to the PUF. Thus, $\mathcal{S}$ can fiddle with the PUF's output (consult Figure 12) and thereby program the PUF behavior. In the third model, the environment $\mathcal{Z}$ is provided with direct access to the PUF. Thus, the simulator $\mathcal{S}$ may not provide manipulated PUF answers to $\mathcal{A}$, as those get detected by $\mathcal{Z}$, see Figure 13.

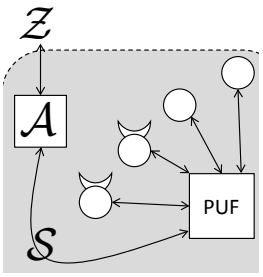

Figure 12: The simulator $\mathcal{S}$ can fiddle with the PUF's answers without being detected by $\mathcal{Z}$.

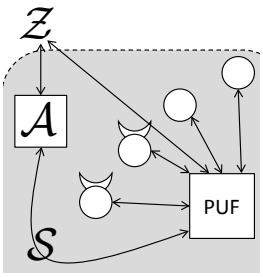

Figure 13: The PUF answers provided to $\mathcal{Z}$ by $\mathcal{A}$ through $\mathcal{S}$ need to be consistent with the answers provided to $\mathcal{Z}$ by PUF directly.

This issue is comparable to programmability of random oracles. As observed by Nielsen [27], programmability of random oracles trivially allows to realize certain cryptographic tasks which are provably not achievable without programmability. For instance, without additional assumptions, commitment schemes are known not to be realizable in the plain UC model [6]. Moreover, Canetti and Fischlin [6] provided a UC-secure commitment scheme in the *programmable* common random string model. Later, Canetti et al. [5] extended the original impossibility result to the *non-programmable* common reference string model thus showing that programmability assumption cannot be removed. One would expect the same results to carry over for the situation of PUFs. However, querying a PUF is not a public operation as opposed to random oracle evaluation and accessing a common reference strings. Hence, the impossibility results do not directly carry through so that we were able to provide secure protocols in the non-programmable model for the three prominent tasks of oblivious transfer (Section 5), commitments (Section 6) and key exchange (Section 7). From now on, the non-programmable UC model with PUFs is called PUC, and the model depicted in Figure 10 is denoted as weak PUC.

All protocols are shown to be secure for static corruptions only. In Section B.5, we show that the requirements to realize oblivious transfer in PUC while allowing adaptive corruptions are very stringent. In Subsection B.2, we show that the composition theorem carries through in PUC as well as in weak PUC.

B.2 The Composition Theorem

We first give a short reminder of UC notations and theorems. A protocol ρ is said to *UC-emulate* a protocol ϕ if for all adversary $\mathcal{A}$ there is a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$, the output of the environment $\mathcal{Z}$ running with ϕ and $\mathcal{A}$, written $\mathrm{EXEC}_{\phi,\mathcal{A},\mathcal{Z}}$ is computationally indistinguishable from the output of the environment $\mathcal{Z}$ running with ρ and $\mathcal{S}$, written $\mathrm{EXEC}_{\rho,\mathcal{S},\mathcal{Z}}$.

Canetti [4] proved that the notion of UC-emulation, i.e.,

$$\forall \mathcal{A}\ \exists \mathcal{S}\ \forall \mathcal{Z}\ \mathrm{EXEC}_{\phi,\mathcal{A},\mathcal{Z}} \approx \mathrm{EXEC}_{\rho,\mathcal{S},\mathcal{Z}}$$

is equivalent to

$$\exists \mathcal{S}\ \forall \mathcal{Z}\ \mathrm{EXEC}_{\phi,\mathcal{D},\mathcal{Z}} \approx \mathrm{EXEC}_{\rho,\mathcal{S},\mathcal{Z}}$$

where $\mathcal{D}$ is the dummy adversary which only executes instructions given by the environment $\mathcal{Z}$ and relays answers to the environment $\mathcal{Z}$. The notation π^{ϕ} denotes that the protocol π invokes ϕ as a subroutine. The notation $\pi^{\rho/\phi}$ means that π invokes ρ instead of ϕ.

Theorem B.1 (Composition Theorem) *Let π^{ϕ}, ρ and ϕ be three protocols. If ρ UC-emulates ϕ, then $\pi^{\rho/\phi}$ UC-emulates π^{ϕ}.*

We then obtain the following corollary which is the claim usually used when considering UC-security.

Corollary B.2 *Let ρ be a protocol which UC-emulates a functionality $\mathcal{F}$ and let $\pi^{\mathcal{F}}$ be a protocol which UC-simulates a functionality $\mathcal{G}$. Then, $\pi^{\rho/\mathcal{F}}$ UC-emulates $\mathcal{G}$.*

We will first shortly summarize the proof of Corollary B.2 and then consider the proof of Theorem B.1. In Section B.4 and Section B.3, we then analyze where to be cautious when interacting with non-PPT algorithms such as PUFs.

Canetti proves Corollary B.2 as follows: Theorem B.1 assures that for every adversary $\mathcal{A}$ there is a simulator $\mathcal{A}_{\pi}$ such that for all environments $\mathcal{Z}$, one has $\mathrm{EXEC}_{\pi^{\mathcal{F}},\mathcal{A}_{\mathcal{F}},\mathcal{Z}} \approx \mathrm{EXEC}_{\pi^{\rho/\mathcal{F}},\mathcal{A},\mathcal{Z}}$. As $\pi^{\mathcal{F}}$ UC-emulates $\mathcal{G}$, for every $\mathcal{A}_{\pi}$ there is a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$, $\mathrm{EXEC}_{\mathcal{G},\mathcal{S},\mathcal{Z}} \approx \mathrm{EXEC}_{\pi^{\mathcal{F}},\mathcal{A}_{\mathcal{F}},\mathcal{Z}}$, and the result follows by transitivity of $\approx$.

The proof of Theorem B.1 is along the following lines: Assume $\mathcal{S}_{\rho}$ is such that for all $\mathcal{Z}_{\rho}$,

$$\mathrm{EXEC}_{\phi,\mathcal{D}_{\rho},\mathcal{Z}_{\rho}} \approx \mathrm{EXEC}_{\rho,\mathcal{S}_{\rho},\mathcal{Z}_{\rho}}\ ,$$

see Figure 14 and Figure 15. Then, one constructs an adversary $\mathcal{A}_{\pi}(\mathcal{S}_{\rho})$ as a black-box construction from $\mathcal{S}_{\rho}$, and one shows that $\mathcal{A}_{\pi}$ satisfies for all environment $\mathcal{Z}_{\pi}$ that $\mathrm{EXEC}_{\pi^{\phi},\mathcal{D}_{\pi},\mathcal{Z}_{\pi}} \approx \mathrm{EXEC}_{\pi^{\rho/\phi},\mathcal{A}_{\pi},\mathcal{Z}_{\pi}}$, see Figure 16 and Figure 17.

Assume, the latter does not hold, and let $\mathcal{Z}_{\pi}$ be an environment that falsifies the statement. Then, one constructs an environment $\mathcal{Z}_{\rho}(\mathcal{Z}_{\pi})$ such that Relation B.2 is falsified. The

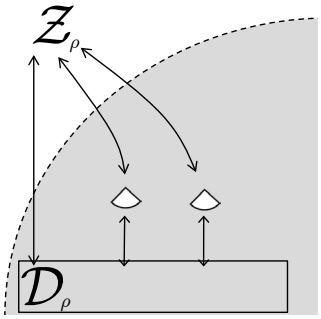

Figure 14: The environment $\mathcal{Z}_\rho$ runs with ρ and the dummy adversary $\mathcal{D}_\rho$.

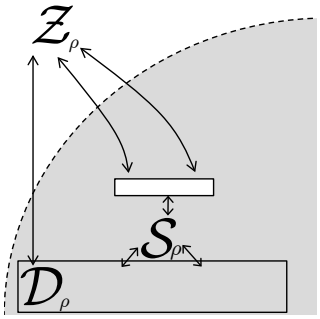

Figure 15: The environment $\mathcal{Z}_\rho$ runs with ϕ and the simulator $\mathcal{S}_\rho$. The latter runs $\mathcal{D}_\rho$ as a subroutine.

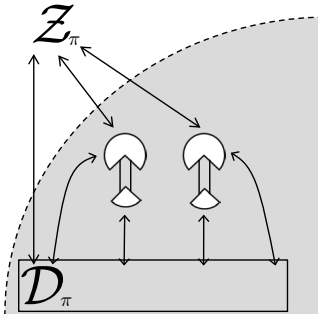

Figure 16: The environment $\mathcal{Z}_\pi$ runs with $\pi^{\rho/\phi}$ and the dummy adversary $\mathcal{D}_\pi$.

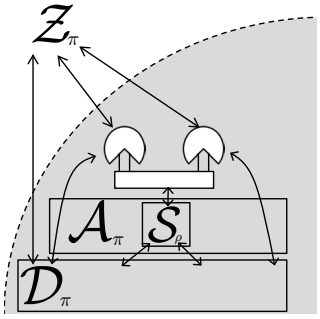

Figure 17: The environment $\mathcal{Z}_\pi$ runs with π^ϕ and $\mathcal{A}_\pi$. The adversary $\mathcal{A}_\pi$ runs $\mathcal{S}$ and the dummy adversary $\mathcal{D}_\pi$ as subroutines.

constructions and their analysis on a high-level work as follows: $\mathcal{A}_\pi$ runs $\mathcal{S}_\rho$ as a subroutine in order to transform network outputs of the protocol ρ into simulated outputs of the protocol ϕ. For network outputs of the protocol π itself, $\mathcal{A}_\pi$ simply relays the outputs of the honest parties to $\mathcal{D}_\pi$. Now, the construction of $\mathcal{Z}_\rho(\mathcal{Z}_\pi)$ is as follows: $\mathcal{Z}_\rho$ simulates the honest parties as the run π. It relays all inputs of $\mathcal{Z}_\pi$ to these simulated parties. Whenever these parties activate an instance of ϕ then $\mathcal{Z}_\pi$ passes these inputs to the real parties (which either run ϕ or ρ dependent on whether one is in the real or in the simulated world). $\mathcal{Z}_\rho$ produces the same final output as $\mathcal{Z}_\pi$. See also Figure 18 and Figure 19. By inspection of the code, one is able to show that the view of $\mathcal{Z}_\pi$ is identical in both situations, and thus, $\mathcal{Z}_\rho(\mathcal{Z}_\pi)$ distinguishes ϕ from ρ whenever $\mathcal{Z}_\pi$ distinguishes π^ϕ from $\pi^{\rho/\phi}$.

We now analyze how the proof is affected when the algorithms are not PPT Turing machines but of complexity PPT+PUF. We consider two different scenarios: PUC and weak

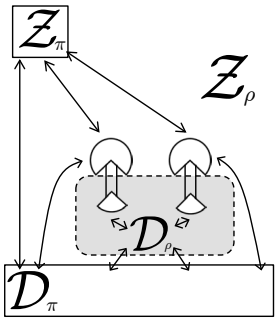

Figure 18: The environment $\mathcal{Z}_\rho$ runs $\mathcal{Z}_\pi$ and $\mathcal{D}_\pi$ as subroutines. It simulates the protocol π and invokes instances of ϕ whenever π intends to do so.

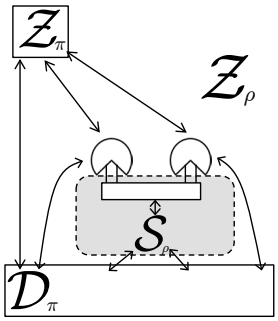

Figure 19: The environment $\mathcal{Z}_\rho$ runs $\mathcal{Z}_\pi$ and $\mathcal{D}_\pi$ as subroutines. It simulates the protocol π and invokes instances of ρ whenever π intends to invoke an instance of ϕ.

PUC.

B.3 The Composition Theorem in weak PUC

First of all, the theorem goes through if all algorithms may use a PUF, as it does not make a difference to which complexity class the algorithms belong as long as they are part of the same class so that simulations are possible, i.e., they can be PPT, unbounded, have additional PSPACE oracles or, as in the present case, access to a PUF. Caution is necessary when hybrid arguments are involved. The composition theorem uses a hybrid argument over the (polynomial) number of instances of an invoked protocol. For PPT+PUF, the polynomial bound on the number of instances is preserved.

Now, we would like to address compatibility issues: If the protocol π uses a PUF, but the subprotocol ρ does not use a PUF, then it is not sufficient for ρ to be proven secure in the classic UC setting. The reason is that it will be used in an environment that has PUF-power. In the proof, this will show up when $\mathcal{Z}_\rho$ is constructed (see Figure 18 and Figure 19). To simulate π, the environment $\mathcal{Z}_\rho$ needs to access a PUF. Thus, $\mathcal{Z}_\rho$ has PUF complexity. To make use of the property that $\mathcal{S}_\rho$ is a good simulator (compare Figure 14 and 15), $\mathcal{Z}_\rho$ should match the corresponding syntactical requirements. If $\mathcal{S}_\rho$ is only good for PPT environments, this does not hold. Thus, if a protocol uses PUFs, all uses subprotocols need to be proven secure against PUF environments.

Let's turn to the converse consideration. If $\pi^{\mathcal{F}}$ emulates functionality $\mathcal{G}$ and does not use a PUF, is UC-emulation sufficient, or is weak PUC-emulation necessary, if $\mathcal{F}$ shall be realized by a protocol ρ that uses PUFs. Again, weak PUC-emulation is necessary. Intuitively, this follows from considering $\mathcal{G}$ and $\pi^{\rho/\mathcal{F}}$. The latter protocol uses a PUF while the first one does not. In the final analysis, thus, both need to be analyzed in the presence of PUFs. Else, they

can be distinguished easily. It is in the proof of Corollary B.2 that this becomes apparent: The proof used that for every PUF-adversary $\mathcal{A}$ there is a PUF-simulator $\mathcal{A}_\pi$ such that for all PUF-environments $\mathcal{Z}$, one has $\text{EXEC}_{\pi^\mathcal{F},\mathcal{A}_\mathcal{F},\mathcal{Z}} \approx \text{EXEC}_{\pi^{\rho/\mathcal{F}},\mathcal{A},\mathcal{Z}}$. Then, one used that $\pi^\mathcal{F}$ emulates $\mathcal{G}$ which means that for every PPT-adversary $\mathcal{A}_\pi$ there is a PPT-simulator $\mathcal{S}$ such that for all PPT-environments $\mathcal{Z}$, $\text{EXEC}_{\mathcal{G},\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi^\mathcal{F},\mathcal{A}_\mathcal{F},\mathcal{Z}}$. Thus, the latter cannot be applied to a PUF-environment $\mathcal{Z}$ and so the proof of corollary B.2 fails, if $\pi^\mathcal{F}$ only provides a UC-emulation of $\mathcal{G}$ instead of a weak PUC-emulation.

We summarize our analysis for weak PUC:

Theorem B.3 (Weak PUC Composition Theorem) *Let π^ϕ, ρ and ϕ be three protocols which potentially may use PUFs. If ρ weak PUC-emulates ϕ, then $\pi^{\rho/\phi}$ weak PUC-emulates π^ϕ.*

Corollary B.4 *Let ρ be a protocol which weak PUC-emulates a functionality $\mathcal{F}$ and let $\pi^\mathcal{F}$ be a protocol which weak PUC-simulates a functionality $\mathcal{G}$. Then, $\pi^{\rho/\mathcal{F}}$ weak PUC-emulates $\mathcal{G}$.*

None of the two conditions can be weakened to UC-emulation.

B.4 The Composition Theorem in PUC

If all algorithms may access a common PUF functionality, then first of all, as observed in Subsection B.3, complexity issues are not problematic. Accessing a common functionality was previously mainly considered for random oracles. The common access provides non-programmability. However, to satisfy UC-syntax, we must alter our description of PUC slightly.

The syntactical tweak consists in granting the environment $\mathcal{Z}$ access to the PUF *through* the adversary but to restrict the adversary to honestly relay PUF queries and answers.

Moreover, the adversary $\mathcal{A}$ refuses to relay queries to PUFs that are currently owned by honest participants, as $\mathcal{F}_{\mathsf{PUF}}$ does not answer those. The $\mathcal{S}$ is required to do exactly the same: The PUF indices which belong to honest users may not be queried by the environment $\mathcal{Z}$.

B.5 Impossibility Results and Adaptive Corruption

In [6], Canetti and Fischlin show that adaptively secure commitments cannot be realized in UC. However, they provide a protocol which is secure in the common reference string (CRS) model. Here, the CRS is considered to be *programmable*, i.e., the simulator $\mathcal{S}$ can use the CRS to embed secret information which it later uses to simulate the protocol, formally backed up by the CRS-hybrid model. In [5], Canetti et al. introduce a generalized UC model with global setup (GUC). In this model, a trusted setup is genuinely generated and available in the ideal world as well as in the real world. As the setup is considered to be realized once and for all,

any external protocol shall be able to use it. Thus, the environment is provided access to the setup as well.

Again, if the trusted setup in GUC is a common reference string, the environment's access prevents the simulator $\mathcal{S}$ from manipulating the CRS; the simulator, too, needs to use the given CRS. Canetti et al. then show that the impossibility result of [6] can be generalized to GUC when the setup is a CRS. Somewhat surprisingly, this does not apply to PUC.

In the original proof, Canetti and Fischlin use that if there is secure commitment scheme in UC then there is a simulator $\mathcal{S}$ which can produce a commitment c which can later be opened to arbitrary values. Now, if such a $\mathcal{S}$ exists, then the commitment scheme cannot have been secure from the beginning, as any malicious participant can use the algorithm of $\mathcal{S}$ to produce a fake commitment. The result is based on the fact that all information available to the $\mathcal{S}$ is also available to a malicious participant. Thus, the latter can use a copy of $\mathcal{S}$ to cheat. The proof is based on the assumption that a malicious user (respectively, the adversary $\mathcal{A}$) has access to the same information as the simulator $\mathcal{S}$. For PUFs, this is not the case.

This implies certain properties of PUC-secure protocols: During the execution of an oblivious transfer protocol (OT), the sender must have a secret which is not available to the receiver, as else, the extraction algorithm of the simulator $\mathcal{S}$ could be used by the receiver to break the protocol. We develop this thought further and consider adaptive corruption: We noted that the OT-protocol becomes insecure when the receiver has access to the PUF *during* the protocol execution. Note that usually this also holds after termination of the protocol: Consider the OT-protocol provided in Section 5. Any party which has access to the transcript of the protocol *and* the sender's PUF can extract both secrets from the transcript. Hence, the protocol is insecure w.r.t. adaptive corruption as corrupting the sender also provides access to the sender's PUF — as the simulator $\mathcal{S}$ provides a simulation without knowing both secrets, thus would reveal to the environment that it interacts with the ideal world. Moreover, this holds for *all* protocols where access to the PUF allows to extract both secrets from the protocol transcript. It is thus necessary to provide a non-committing transcript.

The transcript of the presented OT-protocol is not non-committing w.r.t. to the secrets of the sender. However, it is non-committing w.r.t. to the secret bit of the receiver, i.e., as the simulator $\mathcal{S}$ has access to a PUF, for each message transcript, it can provide two states of the receiver, one for each secret bit b. Thus, the commitment scheme which we built in Section 6 is secure against adaptive attacks when the OT building block is replaced via the OT-protocol presented in 5. We her re-state the protocol depicted in Figure 20 with explicit OT subprotocol.

We now show how to react to adaptive corruptions and how to provide state for any of the two parties at any stage of the protocol. Note that the sender does not have any inputs from the environment. Thus, all simulations of the receiver (recall that the receiver of the commitment scheme plays the role of the $\mathcal{S}$ in the OT scheme) can be done genuinely and its genuine state can be passed to the adversary $\mathcal{A}$. For simulation of the sender's state, note that the simulator has access to the PUF and gets the secret bit of the committer. The $\mathcal{S}$

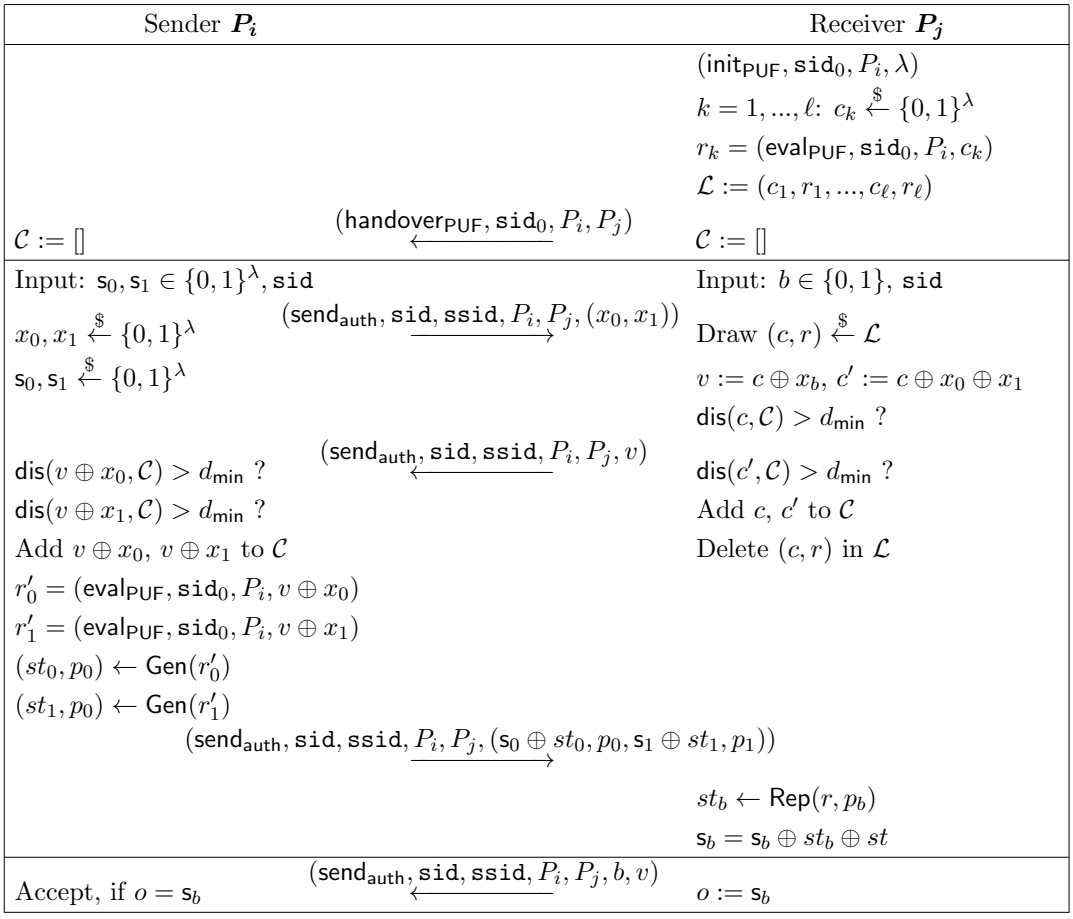

Figure 20: Adaptively secure commitment scheme with PUFs.

simulates the committer's state as follows (We simulate the state chronologically in order to capture all possible corruption moments): The list $\mathcal{L}$ can be genuinely generated. The values c and c' are set to be $c := x_b \oplus v$ and $c' := c \oplus x_0 \oplus x_1$. Set r to be the value returned by PUF when being evaluated on c. Set $o := \mathsf{s}_b$. Note that before the final message is sent, the simulator $\mathcal{S}$ learns the challenge bit b and can compute o accordingly.

C Related Security Properties of PUFs

In this section, we review security properties which are frequently mentioned in the context of PUFs. We investigate how they relate to our notion and provide complementary formalization. For an overview on PUF implementations as well as their properties, we refer the reader to

[25] for a comprehensive overview.

C.1 Uncloneability of PUFs

Uncloneability is usually considered in two flavors: *software-uncloneability* and *hardware-uncloneability* [29, 39, 25]. Hardware uncloneability, also called *physical uncloneability*, requires that it is difficult to create a second device whose behavior is indistinguishable from the first one. For each PUF implementation the authors showed that physical cloning of their implementation is impossible, see for example [29, 24, 35, 17, 37, 25]. Note that *physical* properties are usually infinite properties. However, for cryptographic application, we are interested in its discrete mathematical behavior as a (noisy) function and thus consider cloneability merely with respect to its input/output behavior. Software uncloneability, also called *model building* requires that it should be hard to provide a software model of a PUF. Model building usually considers mere software models, while we additionally allow the software to use a second, different PUF to achieve its cloning goal.

Our entropy-based notion of unpredictability automatically assures both, software uncloneability and hardware uncloneability assuming that the creation process is not controllable. For a game-based formalization of hardware uncloneability, we refer the reader to [32]. As software uncloneability implies hardware uncloneability, the below game-based definition is strictly stronger than the one presented in [32]. Cryptographic applications that allow the adversary to read out arbitrarily many challenge pairs, require software uncloneability. Note that, however, in a weakened adversarial model, physical uncloneability can be sufficient.

A PPT algorithm $\mathcal{A}$ which intends to clone a PUF_1 while having access to arbitrarily many PUF_2,...,PUF_n. Now, $\mathcal{A}$ might measure the PUF_1 on a couple of challenges $c_1, ..., c_k$. The algorithm $\mathcal{A}$ then finishes the cloning process. From now on, $\mathcal{A}$ is denied access to the PUF_1 and shall simulate its input/output behavior. We now draw a random challenge c. With high probability (see Appendix C.3), the challenge c is not close to any of the c_i. Thus, from $\mathcal{A}$'s point of view, the entropy on the response value r is high. Thus, information-theoretically, $\mathcal{A}$ has low probability in providing a suitable response. Note that we restrict the observable parameters of the PUF to be its input/output behavior.

Game UNC$(\mathsf{PUF}, \mathcal{A}, \lambda)$

Learning Phase Proceeding adaptively, the adversary has access to an oracle Sample which draws a PUF index i from the PUF family according to Sample as well as to Eval oracles that evaluate PUF_i. The index of the first sampling query is denoted by i_0.

Simulation Phase Eventually, we withdraw the adversary's possibility to query PUF_{i_0}. The adversary can now be queried with input challenge values c. We denote this adversary by $\mathcal{A}$. For all challenges c, he is required to output answers r that are (computationally) indistinguishable from answers given by PUF_{i_0}. An (efficient) distinguisher $\mathcal{D}$ is given

two oracles that are either both equal to PUF_{i_0}, or one oracle equal to $\mathcal{A}$, and one oracle equal to PUF_{i_0}.

For each (efficient) distinguisher $\mathcal{D}$, we denote by $\mathbf{Unc}_{\mathcal{A},\mathcal{D}}(\lambda)$ the difference

$$\mathrm{Prob}\left[\mathcal{D}^{\mathcal{A},\mathsf{PUF}_{i_0}} = 1\right] - \mathrm{Prob}\left[\mathcal{D}^{\mathsf{PUF}_{i_0},\mathsf{PUF}_{i_0}} = 1\right].$$

Definition C.1 (Cloneability of PUFs) *A PUF is cloneable if there exists an efficient algorithms $\mathcal{A}$ such that for all (efficient) distinguishers $\mathcal{D}$, the distinguishing advantage $\mathbf{Unc}_{\mathcal{A},\mathcal{D}}(\lambda)$ is negligible in λ.*

Definition C.2 (Uncloneability of PUFs) *A PUF is uncloneable if it is not cloneable.*

The latter definition is equivalent to the game-based definition of unpredictability provided in a concurrent paper by Armknecht et al. [1]. Note that Armknecht et al. consider a combination of a fuzzy extractor and a PUF, while the above definition considers a mere PUF.

C.2 Unpredictability of PUFs

As considered earlier, unpredictability of PUFs states that it is difficult to compute the output to a chosen stimulus without evaluating the PUF. This property holds even if further CRPs are given (e.g., one collected several CRPs before). The difference to "regular" unpredictable functions is that the attacker may indeed be able to predict response values for challenges which are close to those it measured the PUF on. That is, some PUFs map closely related measurements to closely related outputs.

We provided a statistical and a computational variant of unpredictability. An alternative would be to use a game-based definition. Unfortunately, this is not suitable for UC-application. Note that our UC-definition implies the security notion defined by the following game: The attacker first collects several PUF measurements for challenge value of its choice up to a certain moment where it has no longer access to the device. Subsequently, the attacker has to output a valid challenge/response pair (c^*, r^*) such that the distance between its prediction and all previously measurements is at least ϵ. If this conditions is fulfilled and if $r^* \leftarrow \mathsf{PUF}(c^*)$, then the attacker wins the game. We define unpredictability of PUFs in the following game between a challenger and an adversary $\mathcal{A}$:

Game PRE$(\mathsf{PUF}, \mathcal{A}, \lambda)$

Setup The challenger generates a PUF by running the generation procedure Sample to obtain a PUF index x.

Queries Proceeding adaptively, the adversary $\mathcal{A}$ may query the PUF oracle $\mathcal{O}_{\mathsf{PUF}}$ on challenges $c_i \in \{0,1\}^\lambda$. It returns $r_i \leftarrow \mathsf{PUF}(c_i)$ to the attacker and adds c_i to a list $\mathcal{Q}$ (list of challenge/response pairs queried by the attacker).

OUTPUT Eventually, the adversary outputs a pair (c^*, r^*) with $c^* \notin \mathcal{Q}$. The game evaluates the PUF on $r \leftarrow \mathsf{PUF}(c^*)$. It outputs 1 if the metric distance $\mathsf{dis}(c^*, c) > \epsilon(\lambda)$ for all $c \in \mathcal{Q}$ and $r = r^*$.

We define $\mathbf{Pre}_{\mathcal{A}}(\lambda)$ be the probability that $\mathbf{PRE}(\mathsf{PUF}, \mathcal{A}, \lambda)$ outputs 1.

Definition C.3 (Unpredictability of PUFs) *A PUF is unpredictable with respect to the game* $\mathbf{PRE}(\mathsf{PUF}, \mathcal{A}, \lambda)$ *if for all efficient algorithms* $\mathcal{A}$ *the probability* $\mathbf{Pre}_{\mathcal{A}}(\lambda)$ *is negligible in* λ*.*

C.3 Uncorrelated Output of PUFs

If challenges are members of close-by regions, their corresponding responses are usually correlated, for example in the case of optical PUFs [29, 36]. However, if two challenges c_1, c_2 are far away from each other, namely more than d_{min}, then their responses should not be close to each other. This property is implied by unpredictability.

If the outputs of two such different challenges were closely correlated, then knowing the PUF response to challenge c_1 would provide more information about the PUF response to c_2 than allowed by unpredictability. We now define the intuition of uncorrelated outputs in the following game:

Game UNCOR$(\mathsf{PUF}, d_{\mathsf{min}}, \mathcal{A}, \lambda)$

SETUP The challenger generates a PUF by running the generation procedure Sample to obtain a PUF index x. The adversary adaptively generates challenges c_i which it may query to a PUF several times in arbitrary order. The tth response to the challenge c_i is denoted by $r_{i,t}$. Each time the game returns $r_{i,t} \leftarrow \mathsf{PUF}(c_i)$ to the attacker, it adds $c_i, r_{i,t}$ to a list $\mathcal{Q}$.

OUTPUT Eventually, the adversary outputs $(c_i^*, r_i^*), (c_j^*, r_j^*)$. The game outputs 1 if the queries output by the adversary are in $\mathcal{Q}$, $\mathsf{dis}(c_i^*, c_j^*) > d_{\mathsf{min}}(\lambda)$ and the metric distance between the two responses is at most $\rho(\lambda)$, e.g, $\mathsf{dis}(r_i^*, r_j^*) \leq \rho(\lambda)$.

We define $\mathbf{Cor}_{\mathcal{A}}(\lambda)$ as the probability that $\mathbf{UNCOR}(\mathsf{PUF}, d_{\mathsf{min}}, \mathcal{A}, \lambda)$ returns 1.

Definition C.4 (Uncorrelated Output of PUFs) *A PUF satisfies* d_{min} *uncorrelated output if for all efficient adversaries* $\mathcal{A}$ *the probability* $\mathbf{Cor}_{\mathcal{A}}(\lambda)$ *is negligible in* λ*.*

C.4 One-Wayness of PUFs

One-wayness of PUFs means that evaluating a PUF is easy while inverting it remains hard. That is, given a response r it should be hard to find a corresponding challenge c such that

for $r' \leftarrow \mathsf{PUF}(c)$, the responses r and r' are close. A PUF family is one-way, if no algorithm $\mathcal{A}$ outputs the challenge c with more than non-negligible probability. Note that for PUFs with a small input ranges one-wayness is not achievable as outputting a random input already yields a non-negligible probability in inverting the PUF, e.g., simple arbiter PUFs. Moreover, measuring the PUF on all input values increases the probability to 1.

Our notion of unpredictability is incomparable to one-wayness, i.e., a PUF may satisfy unpredictability without being one-way, and conversely, a PUF can be one-way while not satisfying unpreditability. This shows that one-wayness does not imply that the PUF has any entropy at all which is, however, one of the main stated goals in PUF design[3]. This shows that one-wayness as a mere assumption is not sufficient. However, one-wayness is a desirable property which, however, seems to be hard to realize, as many of the common PUF types are not one-way, such as arbiter PUFs, ring-oscillator PUFs, SRAM PUFs and coating PUFs. Towards applicability for many PUF types, we thus discard the assumption of one-wayness.

We now first define one-wayness formally and then turn to the bidirectional separation between unpredictability and one-wayness.

Game INV$(\mathsf{PUF}, \mathcal{A}, \lambda)$

SETUP The challenger generates a PUF by running the generation procedure Sample to obtain a PUF index x. It picks a challenge c at random and evaluates $r \leftarrow \mathsf{PUF}(c)$. It hands r to the adversary.

OUTPUT The adversary may adaptively query the PUF and finally outputs a challenge c. The challenger computes $r' \leftarrow \mathsf{PUF}(c)$ and outputs 1 if $\mathsf{dis}(r', r) < d_{\mathsf{noise}}$.

We define $\mathbf{Inv}_{\mathcal{A}}(\lambda)$ as the probability that $\mathbf{INV}(\mathsf{PUF}, \mathcal{A}, \lambda)$ returns 1.

Definition C.5 *A PUF is one-way with respect to the game* $\mathbf{INV}(\mathsf{PUF}, \mathcal{A}, \lambda)$ *if for all efficient algorithm* $\mathcal{A}$ *the probability* $\mathbf{Inv}_{\mathcal{A}}(\lambda)$ *is at most* $\delta_{\mathbf{Inv}}$.

The following separations shall illustrate that unpredictability and one-wayness do not imply each other.

Unpredictability does not imply One-Wayness Assume, a PUF-family satisfies our notion of unpredictability for suitable parameters. Then, we can define a new PUF-family PUF* which on input challenge c queries PUF on challenge c. The PUF then returns a response r, and PUF* returns as its response R the pair $R = (c, r)$. The high-entropy condition of our unpredictability notion is still satisfied. However, PUF* is not one-way.

[3]In the first work, Pappu considered PUFs in the context of entropy [29]. Later on, people studied/revisited the notion of entropy of optical PUFs as well as of further PUF implementations [36, 33, 34, 21, 38]

One-Wayness does not imply Unpredictability Let PUF be a family of one-way-functions $\{f_i\}_{\mathcal{I}}$ with efficient descriptions. We define a new function family $\{f_i^*\}_{\mathcal{I}}$ as follows: On input a value c, f_i^* outputs $f_i(c)$ concatenated with a function description of f_i. This construction is still one-way. However, it is not unpredictable. After querying the function on c, one can easily derive the function value on any other input c'. This concludes the separation.

C.5 Tamper-Resistance

Tamper-resistance means that physical manipulation should be either impossible or easy to detect, e.g., tampering changes the input/output behavior of the PUF. Our modeling of PUFs implicitly assumes tamper-resistance, as the attacker is bound on querying the PUF. There are two ways to approach this topic: One may consider this as a human-related property, i.e., manipulating is easy to detect for a human. The other approach would be to assume that the mathematical behavior becomes atypical, or, that the mathematical behavior becomes very different. In the latter case, when receiving a PUF, the sender and the receiver needs to exchange a number of challenge-response pairs, i.e., the sender sends a couple of challenges, and the receiver returns the PUF's responses. If they are close enough to the values the sender has stored in its challenge-response pair list, the sender will confirm that no tempering has occurred. In our model, this property is implicit. When desired, one can define a more basic PUF definition which allows the adversary mathematical tempering. One would then show that this definition can be composed with the above protocol to emulate our PUF definition. A formalization of this approach is left for future work.