Physically Uncloneable Functions in the Universal Composition Framework

Christina Brzuska, Marc Fischlin, Heike Schröder, and 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. In this paper, 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.

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 [21]. 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

In this paper, 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) [27,26]. Basically, a PUF is a noisy device 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.

P. Rogaway (Ed.): CRYPTO 2011, LNCS 6841, pp. 51–70, 2011.

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 (e.g., whether a PUF is one-way or not, or if the output is pseudorandom). See [29,23] for more information. 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, see [27,17,12,2,29,11,30]. 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 [21,19,24,16,15] 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. During transition, 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 in the full version 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 the full version of this paper 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 of the PUF. 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 [25] 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 corresponds to an honest user who prevents further unauthorized access. In a stronger version one could also allow further "uncontrolled" interaction between the environment, i.e., other protocols, and the PUF even when in possession of the honest user. 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. [21,14]). 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 [21], 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 [28]. His protocol, however, has a high round complexity due to an interactive hashing step. Still, [28] points out that, using symmetry of oblivious transfer [32] 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 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 recovers the image with the help of the PUF. Both parties use the images as the key, after applying a fuzzy extractor for error correction 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. Note that in this case, attacks such as described in [31] will also be covered by the security proof. An interesting feature of PUFs is that, unlike other hardware tokens (e.g., [21]), 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 For others types this can be achieved through appropriate encoding, as for angles in optical PUFs.

Definition 1 (Physically Uncloneable Functions). *Let rg indicate the dimension of the range of the PUF responses of 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 [1,23,29] for overviews) such as unpredictability, uncloneability, bounded noise, uncorrelated outputs, one-wayness, and tamper-evidence. We give a detailed analysis of these properties in the full version of this paper 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. 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-evidencen.

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 the 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, as the challenges c are chosen by the adversary, we ask the average min-entropy to be high for all challenges and defined by the maximal probability of a possible response r.

Definition 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*

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

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 occasionally 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 3 (Unpredictability). *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 full version), 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 full version); 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.

2.3 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.

An (m, ℓ, t, ϵ)-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 an ℓ-bit secret st together with helper data p. The helper data can be stored publicly, since it does not reveal information about the secret: as long as the measurement contains m bits of min-entropy the secret has statistical distance ϵ to the uniform distribution, even if given p. The helper data is later used to reproduce the same secret st from related measurements within a certain distance t according to some metric space.

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. 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.

Definition 4. *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 following properties of a well-spread domain, extraction independence and response consistency hold. A formal proof is given in the full version of this paper.

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 maps 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 $st \leftarrow \mathsf{Rep}(r', p)$.

3 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.

3.1 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 1 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 $(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, m)$ 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 $(\mathtt{sid}, \mathsf{id}, \bot, \mathsf{trans}(P_j))$. 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 $(\mathtt{sid}, \mathsf{id}, \bot, \mathsf{trans}(*))$:
  - ⋆ 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 $(\mathtt{sid}, \mathsf{id}, \bot, \mathsf{trans}(P_j))$ in $\mathcal{L}$:
  - ⋆ If such a tuple does not exist then turn into the waiting state.
  - ⋆ Else, modify the tuple $(\mathtt{sid}, \mathsf{id}, \bot, \mathsf{trans}(P_j))$ 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.

Fig. 1. 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. Our functionality also implies that the sender knows when the PUF has been delivered to the receiver. Relaxing this requirement is delicate as the adversary could then still be in possession of the PUF. Formally, delivery confirmation can be ensured by having the receiver send an acknowledgment message (via an authenticated channel).

3.2 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.

4 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 [22,9,13]. 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.

4.1 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 2. 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 `ssid`.

4.2 Oblivious Transfer Scheme

In Figure 3, we provide an oblivious transfer protocol. For simplicity of exposition, we use the following notation. For a possibly empty set $\mathcal{C}$ we let

$\mathcal{F}_{\mathsf{OT}}$ 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 $(\mathsf{choose\text{-}secret}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, b)$ on the input tape of $\mathcal{F}_{\mathsf{OT}}$, the functionality $\mathcal{F}_{\mathsf{OT}}$ stores this tuple and writes $(\mathsf{choose\text{-}secret}_{\mathsf{OT}}, \mathtt{ssid}, \mathtt{sid}, P_i, P_j)$ 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 $(\mathsf{choose\text{-}secret}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, b)$ 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.

Fig. 2. The ideal functionality for oblivious transfer adapted from [4]

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

Fig. 3. Oblivious transfer scheme with PUFs

$\mathsf{dis}(c, \mathcal{C}) > d_{\mathsf{min}}$ 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.

We now sketch the security arguments for the OT-protocol in Figure 3, 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 4), then with overwhelming probability this is the case, as — due to the well-spread domain property (see Subsection 2.3) — 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 1. *Assuming that* $(\mathsf{Gen}, \mathsf{Rep})$ *is a* (m, ℓ, t, ϵ)*-fuzzy generator and that* $\mathsf{PUF} = (\mathsf{Sample}, \mathsf{Eval})$ *is a PUF-family with matching parameters (see Definition 4), 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 the full version of the paper.

4.3 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 [32], 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 [32] 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 $\mathcal{F}_{\mathsf{OT}}$.

5 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 $(\mathsf{com}, \mathsf{decom})$ such that com reveals nothing about the value msg but using the pair $(\mathsf{com}, \mathsf{decom})$ one can open msg. Moreover it should be infeasible to find a value decom' such that $(\mathsf{com}, \mathsf{decom}')$ reveals $\mathsf{msg}' \neq \mathsf{msg}$.

5.1 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 $(\mathsf{corrupt\text{-}committer}, \mathtt{sid}, \mathtt{ssid})$ to $\mathcal{F}_{\mathsf{com}}$, the functionality reveals the recorded value msg to the adversary $\mathcal{S}$. Furthermore, if the $\mathsf{receipt}$ 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 4.

$\mathcal{F}_{\mathsf{com}}$ 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 $(\mathsf{corrupt\text{-}committer}, \mathtt{sid}, \mathtt{ssid})$ 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 $\mathsf{receipt}$ output was not yet written on P_j's tape, then $\mathcal{S}$ can change the recorded value to msg'.

Fig. 4. The ideal functionality for commitment schemes adapted from [4]

5.2 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 [22,9] 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 5 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.

The main idea is to inverse the roles of the sender and the receiver. The commitment protocol uses the OT-protocol as a building block or, more precisely, since we work in the UC framework, the corresponding ideal OT-functionality. Consider a commitment scheme with OT-sender P_i and OT-receiver P_j. Then, P_j is the committer and has a secret bit b which it submits to the OT-functionality. The receiver P_i draws two sufficiently long random strings s_0 and s_1 which it submits to the OT-functionality. The OT-functionality then provides P_j with the secret s_b. This terminates the commitment phase.

In the opening phase, the committer P_j sends the pair (b, s_b). The receiver P_i then checks whether s_b matches the b-th secret. The protocol is binding, as the OT-functionality does not allow to modify the secret bit b and the secret s_{1-b} is statistically hidden from P_i. Thus, P_i can determine s_{1-b} only with negligible probability. The protocol is hiding, as the OT-functionality does not reveal information about the bit b to P_i.

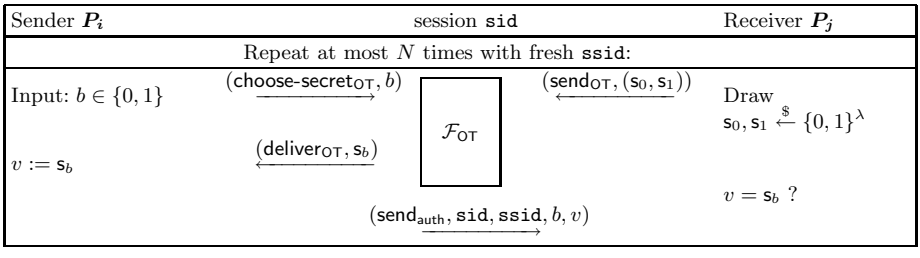

Fig. 5. Commitment scheme with $\mathcal{F}_{\mathsf{OT}}$

Theorem 2. *The commitment protocol in Figure 5 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 2 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.

5.3 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 (the 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.

6 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.

6.1 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 6, adapted from [7].

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

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

Fig. 6. The key exchange ideal functionality adapted from [7]

6.2 Minimal Requirements

We present a key exchange protocol in Section 6.3 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.

6.3 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. We again note that

Server $\boldsymbol{P_i}$		Client $\boldsymbol{P_j}$
$(\mathsf{init}_{\mathsf{PUF}}, \mathtt{sid}, P_i, \lambda)$		
Repeat N times:		
$r \leftarrow (\mathsf{eval}_{\mathsf{PUF}}, \mathtt{sid}, P_i, c)$		
$(st, p) \leftarrow \mathsf{Gen}(r)$		
add (c, r, st, p) to $\mathcal{L}$	$(\mathsf{handover}_{\mathsf{PUF}}, \mathtt{sid}, P_i, P_j) \longrightarrow$	
	Repeat at most N times	
pick $(c, r, st, p) \leftarrow \mathcal{L}$		
remove the entry from $\mathcal{L}$	$(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_j, (c, p)) \longrightarrow$	$r' \leftarrow (\mathsf{eval}_{\mathsf{PUF}}, \mathtt{sid}, P_j, c)$
$\kappa = st$		$st \leftarrow \mathsf{Rep}(r', p)$
		$\kappa = st$

Fig. 7. Key exchange scheme with PUFs

the sender is informed about the point in time when the receiver is in possession of the PUF.

Theorem 3. *Protocol* PUFKE *securely realizes functionality* $\mathcal{F}_{\mathsf{ke}}$ *in the* $\mathcal{F}_{\mathsf{PUF}}$*-hybrid model.*

In the following we merely provide a proof sketch for Theorem 3. The idea is that, for an honest sender, the simulator can easily emulate the setup phase by simply querying the PUF honestly. The simulator simply reveals these samples step by step. If the receiver is honest then, due to the well-spread domain, the adversary will most likely not have queried the PUF about any of the sampled values during the transition phase, such that all the derived keys are statistically indistinguishable from random. It follows that the simulation is statistically close to an actual protocol execution. Finally note that, if one of the parties in the key exchange protocol is corrupt, then the simulator can easily set the key to one of the obtained PUF measurements (after running the fuzzy extractor).

Acknowledgments. We thank the anonymous reviewers for valuable comments. Marc Fischlin is supported by the Emmy Noether Program Fi 940/2-1 of the German Research Foundation (DFG). This work was supported by CASED (`http://www.cased.de`).

References

1. Armknecht, F., Maes, R., Sadeghi, A.-R., Standaert, F.-X., Wachsmann, C.: A formal foundation for the security features of physical functions. To appear at IEEE S&P (2011)
2. Armknecht, F., Maes, R., Sadeghi, A.-R., Sunar, B., Tuyls, P.: Memory Leakage-Resilient Encryption Based on Physically Unclonable Functions. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 685–702. Springer, Heidelberg (2009)
3. Brassard, G., Crépeau, C., Robert, J.-M.: Information theoretic reductions among disclosure problems. In: FOCS, pp. 168–173. IEEE, Los Alamitos (1986)

4. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS, pp. 136–145 (2001)
5. Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally Composable Security with Global Setup. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 61–85. Springer, Heidelberg (2007)
6. Canetti, R., Fischlin, M.: Universally composable commitments. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 19–40. Springer, Heidelberg (2001)
7. Canetti, R., Krawczyk, H.: Universally composable notions of key exchange and secure channels. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 337–351. Springer, Heidelberg (2002)
8. Canetti, R., Rabin, T.: Universal Composition with Joint State. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 265–281. Springer, Heidelberg (2003)
9. Crépeau, C.: Equivalence between Two Flavours of Oblivious Transfers. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 350–354. Springer, Heidelberg (1988)
10. Dodis, Y., Ostrovsky, R., Reyzin, L., Smith, A.: Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. SIAM J. Comput. 38, 97–139 (2008)
11. Frikken, K.B., Blanton, M., Atallah, M.J.: Robust Authentication Using Physically Unclonable Functions. In: Samarati, P., Yung, M., Martinelli, F., Ardagna, C.A. (eds.) ISC 2009. LNCS, vol. 5735, pp. 262–277. Springer, Heidelberg (2009)
12. Gassend, B., van Dijk, M., Clarke, D.E., Torlak, E., Devadas, S., Tuyls, P.: Controlled physical random functions and applications. ACM Trans. Inf. Syst. Secur. 10(4) (2008)
13. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC, pp. 218–229. ACM, New York (1987)
14. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: One-Time Programs. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 39–56. Springer, Heidelberg (2008)
15. Goyal, V., Ishai, Y., Mahmoody, M., Sahai, A.: Interactive Locking, Zero-Knowledge PCPs, and Unconditional Cryptography. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 173–190. Springer, Heidelberg (2010)
16. Goyal, V., Ishai, Y., Sahai, A., Venkatesan, R., Wadia, A.: Founding Cryptography on Tamper-Proof Hardware Tokens. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 308–326. Springer, Heidelberg (2010)
17. Guajardo, J., Kumar, S.S., Schrijen, G.-J., Tuyls, P.: FPGA Intrinsic PUFs and Their Use for IP Protection. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 63–80. Springer, Heidelberg (2007)
18. Håstad, J., Impagliazzo, R., Levin, L.A., Luby, M.: A pseudorandom generator from any one-way function. SIAM J. Comput. 28(4), 1364–1396 (1999)
19. Hazay, C., Lindell, Y.: Constructions of truly practical secure protocols using standard smartcards. In: ACM CCS, pp. 491–500. ACM, New York (2008)
20. Hofheinz, D., Unruh, D., Müller-Quade, J.: Universally composable zero-knowledge arguments and commitments from signature cards. Tatra Mt. Math. Pub., 93–103 (2007)
21. Katz, J.: Universally Composable Multi-party Computation Using Tamper-Proof Hardware. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 115–128. Springer, Heidelberg (2007)
22. Kilian, J.: Founding cryptography on oblivious transfer. In: STOC, pp. 20–31. ACM, New York (1988)

23. Maes, R., Verbauwhede, I.: hysically Unclonable Functions: a Study on the State of the Art and Future Research Directions, section 1. Towards Hardware-Intrinsic Security. Springer, Heidelberg (2010)
24. Moran, T., Segev, G.: David and Goliath Commitments: UC Computation for Asymmetric Parties Using Tamper-Proof Hardware. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 527–544. Springer, Heidelberg (2008)
25. Nielsen, J.B.: Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 111–126. Springer, Heidelberg (2002)
26. Pappu, R., Recht, B., Taylor, J., Gershenfeld, N.: Physical one-way functions. Science 297, 2026–2030 (2002)
27. Pappu, R.S.: Physical One-Way Functions. Phd thesis, Massachusetts Institut of Technology (2001)
28. Rührmair, U.: Oblivious Transfer Based on Physical Unclonable Functions. In: Acquisti, A., Smith, S.W., Sadeghi, A.-R. (eds.) TRUST 2010. LNCS, vol. 6101, pp. 430–440. Springer, Heidelberg (2010)
29. Rührmair, U., Sölter, J., Sehnke, F.: On the foundations of physical unclonable functions. Cryptology ePrint Archive, Report 2009/277 (2009)
30. Sadeghi, A.-R., Visconti, I., Wachsmann, C.: Enhancing RFID Security and Privacy by Physically Unclonable Functions. Towards Hardware-Intrinsic Security. Springer, Heidelberg (2010)
31. Rührmair, C.J.U., Algasinger, M.: An attack on puf-based session key exchange and a hardware-based countermeasure: Erasable pufs. In: Proc. Financial Cryptoghraphy (2011)
32. Wolf, S., Wullschleger, J.: Oblivious Transfer Is Symmetric. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 222–232. Springer, Heidelberg (2006)