Universally Composable Secure Computation with (Malicious) Physically Uncloneable Functions

Rafail Ostrovsky * Alessandra Scafuro † Ivan Visconti ‡
Akshay Wadia §

Abstract

Physically Uncloneable Functions (PUFs) [Pap01] are noisy physical sources of randomness. As such, they are naturally appealing for cryptographic applications, and have caught the interest of both theoreticians and practitioners. A major step towards understanding and securely using PUFs was recently taken in [Crypto 2011] where Brzuska, Fischlin, Schröder and Katzenbeisser model PUFs in the Universal Composition (UC) framework of Canetti [FOCS 2001]. Their model considers *trusted* PUFs only, and thus real-world adversaries can not create malicious PUFs, and can access the physical object only via the prescribed procedure. However, this does not accurately reflect real-life scenarios, where an adversary could be able to create and use malicious PUFs, or access the PUF through other procedures.

The goal of this work is to extend the model proposed in [Crypto 2011] in order to capture such real-world attacks. The main contribution of this work is the study of the Malicious PUFs model. Namely, we extend the PUF functionality of Brzuska et al. so that it allows the adversary to create arbitrarily malicious PUFs. Then, we provide positive results in this, more realistic, model. We show that, under computational assumptions, it is possible to UC-securely realize any functionality. Furthermore, we achieve *unconditional* (not UC) security with malicious PUFs, by showing a statistically hiding statistically binding commitment scheme that uses one PUF only, and such PUF can be malicious.

As an additional contribution, we investigate another attack model, where adversaries access to a trusted PUF in a different way (i.e., not following the prescribed procedure). Technically this attack translates into the fact that the simulator cannot observe the queries made to an honest PUF. In this model, queries are *oblivious* to the simulator, and we call it the "Oblivious Query" model. We are able to achieve *unconditionally* UC-secure computation, even in this more severe model. This protocol is secure against stronger adversaries compared to the ones of Brzuska et al.

Finally, we show the impossibility of UC secure computation in the combination of the above two new models, where the real-world adversary can create malicious PUFs and maliciously access to honest PUFs.

Our work sheds light on the significant power and applicability of PUFs in the design of cryptographic protocols modeling adversaries that misbehave with PUFs.

Keywords: Physically uncloneable functions, UC security, hardware set-up assumptions.

*Departments of Computer Science and Department of Mathematics, UCLA, Email: rafail@cs.ucla.edu
†Dipartimento di Informatica, University of Salerno, Italy. Email. scafuro@dia.unisa.it
‡Dipartimento di Informatica, University of Salerno, Italy. Email. visconti@dia.unisa.it
§Department of Computer Science, UCLA, Email: awadia@cs.ucla.edu

1 Introduction

The impossibility of secure computation in the universal composability framework was proved first by Canetti and Fischlin [CF01], and then strengthened by Canetti et al. in [CKL03]. As a consequence, several setup assumptions, and relaxations of the UC framework have been proposed to achieve UC security [CLOS02, BCNP04, PS04, KLP05].

In recent years, researchers have started exploring the use of secure hardware in protocol design. The idea is to achieve protocols with strong security guarantees (like UC) by allowing parties to use hardware boxes that have certain security properties. An example of the kind of security required from such a hardware box is that of *tamper-proofness*; i.e., the receiver of the box can only observe the input/output behaviour of the functionality that the box implements. This property was formalized by Katz in [Kat07], and it was shown that UC security is possible by relying on the existence of tamper-proof programmable hardware tokens, and computational assumptions. Smart cards are well understood examples of such tokens, since they have been used in practice in the last decades. Several improvements and variations of Katz's model have been then proposed in follow up papers (e.g., [CGS08, MS08, GKR08, GIS+10, DKMQ11, CKS+11, DKMQ12]).

Spurred by technological advances in manufacturing, recently a new hardware component has gained a lot of attention: Physically Uncloneable Functions (PUFs) [Pap01, PRTG02]. A PUF is a hardware device generated through a special physical process that implements a "random" function[1] that depends upon the physical parameters of the process. These parameters can not be "controlled", and producing a clone of the device is considered infeasible. Once a PUF has been constructed, there is a physical procedure to query it, and to measure its answers. The answer of a PUF depends on the physical behavior of the PUF itself, and is assumed to be unpredictable, or to have high min-entropy. Namely, even after obtaining many challenge-response pairs, it is infeasible to predict the response to a new challenge.

Since their introduction by Pappu in 2001, PUFs have gained a lot of attention for cryptographic applications like anti-counterfeiting mechanisms, secure storage, RFID applications, identification and authentication protocols [TB06, GKST07, GKST07, SVW10, EKvdL11, KSWS11]. More recently PUFs have been used for designing more advanced cryptographic primitives. In [Rüh10] Rührmair shows the first construction of Oblivious Transfer, the security proof of which is later provided in [RKB10]. In [AMS+09], Armknecht et al. deploy PUFs for the construction of memory leakage-resilient encryption schemes. In [MHV12] Maes et al. provide construction and implementation of PUFKY, a design for PUF-based cryptographic key generators. There exist several implementations of PUFs, often exhibiting different properties. The work of Armknecht et al. [AMS+11] formalizes the security features of physical functions in accordance to existing literature on PUFs and proposes a general security framework for physical functions. A survey on PUF implementations is given in [MV10]. Very recently in [KKR+12] Katzenbeisser et al. presented the first large scale evaluation of the security properties of some popular PUFs implementations (i.e., intrinsic electronic PUFs).

Modeling PUFs in the UC framework. Only very recently, Brzuska et al. [BFSK11] suggested a model for using PUFs in the UC setting that aims at abstracting real-world implementations. The unpredictability and uncloneability properties are modeled through an ideal functionality. Such functionality allows only the creation of trusted PUFs. In [BFSK11] PUFs are thought as non-PPT

[1] Technically, a PUF does not implement a function in the mathematical sense, as the same input might produce different responses.

setup assumptions. As such, a PPT simulator cannot simulate a PUF, that is, PUFs are non-programmable. Although non-programmable, PUFs are not modeled as global setup [CDPW07]. [BFSK11] shows how to achieve unconditional UC secure Oblivious Transfer, Bit Commitment and Key Agreement with trusted PUFs.

PUFs vs tamper-proof hardware tokens. The apparent similarity of PUFs with programmable tamper-proof hardware tokens [Kat07] vanishes immediately when one compares in detail the two physical devices. Indeed, PUFs are non-programmable and thus provide unpredictability only. Instead tokens are programmable and can run sophisticated code. Moreover, PUFs are stateless, while tokens can be stateful. When a PUF is not physically available, it is not possible to know the output of new queries it received. Instead the answer of a stateless token to a query is always known to its creator[2], since it knows the program embedded in the token. Tamper-proof tokens are realized through ad-hoc procedures that model them as black boxes, their internal content is protected from physical attacks and thus the functionalities that they implement can be accessed only through the prescribed input/output interface provided by the token designer. Instead, PUFs do not necessarily require such a hardware protection (which moreover could be in contrast with the need of running a physical procedure to query the PUF), and their design is associated to recommended procedures to generate and query a PUF, guaranteeing uncloneability and unpredictability. Finally, in contrast to tokens that correspond to PPT machines, PUFs are not simulatable since it is not clear if one can produce an (even computationally) indistinguishable distribution.

1.1 Our Contribution

We observe that the UC formulation of PUFs proposed by Brzuska et al. makes the following assumptions. First, the model considers *trusted* PUFs only, that is, adversaries are assumed to be unable to produce fake/malicious PUFs. Second, a PUF can be queried only using the prescribed evaluation process[3]. As we argue below, we feel that assuming that an adversary cannot misbehave in any of the above ways, might be unrealistic. Given that the study of PUFs is still in its infancy, it is risky to rely on assumptions on the impossibility of the adversaries in generating and accessing PUFs adversarially. Therefore in this paper, we will focus on studying security models that capture such plausible real-world attacks.

The main contribution of this work consists in studying the security of protocols in presence of adversaries that can create malicious PUFs. Additionally, we will consider adversaries making "hidden" queries to PUFs (so that the simulator can not observe these queries). We will present two modifications of the model of Brzuska et al. that formalize security with respect to such stronger adversaries and in both cases we give positive answers to the question of achieving universally composable secure computation with PUFs. More in details, our contributions are listed below.

Modeling malicious PUFs. We augment the UC framework so to enable the adversary to create untrusted (malicious) PUFs. But what exactly are malicious PUFs? In real life, an adversary could tamper with a PUF in such a way that the PUF loses any of its security properties. Or the adversary may introduce new behaviours; for example, the PUF may start logging its queries. To keep the treatment of malicious behaviour as general as possible, we allow the adversary to send as PUF any hardware token that meets the syntactical requirements of a PUF. Thus, an adversary is assumed to be able to even produce fake PUFs that might be stateful and programmed with malicious code.

[2] This is true for stateful tokens too, provided that one knows the sequence of inputs received by the token.

[3] This property technically manifests in the security proofs as the ability of the simulator to observe queries made by an adversary to a trusted PUF.

We assume that a malicious PUF however cannot interact with its creator once is sent away to another party. If this was not the case, then we are back in the standard model, where UC security is impossible to achieve has argued below.

UC secure computation is impossible when malicious PUFs can interact with their creator. The impossibility is straight forward. Consider any functionality that protects the privacy of the input of a player P_1. Comparing to the plain model (where UC is impossible), the only advantage of the simulator to extract the input of the real-world adversary P_1^*, is to read the challenge/answer pairs generated by P_1^* when using the honest PUF created by the simulator that plays on behalf of P_2. If such a simulator exists, then an adversary P_2^* can generate a malicious PUF that just plays as proxy and forwards back and forth what P_2^* wishes to play. P_2^* can locally use one more honest PUF in order to compute the answers that the (remote) malicious PUF is supposed to give. Clearly P_2^* will have a full view of all challenge/answer pairs generated by honest P_1 and running the simulator's code, P_2^* will extract the input of P_1, therefore contradicting input privacy.

UC secure computation with malicious PUFs. The natural question is whether UC security can be achieved in such a much more hostile setting. We give a positive answer to this question by constructing a *computational* UC commitment scheme in the malicious PUFs model. Our commitment scheme needs two PUFs that are transferred only once (PUFs do not go back-and-forth), at the beginning of the protocol and it requires computational assumptions. We avoid that PUFs go back-and-forth by employing a technique that requires OT. The results of Canetti, et al. [CLOS02] shows how to achieve general UC computation from computational UC commitments. Whether *unconditional* UC secure computation is possible in the malicious PUF model, is still an open problem.

Hardness assumptions with PUFs. Notice that as correctly observed in [BFSK11], since PUFs are not PPT machines, it is not clear if standard complexity-theoretic assumptions still hold in presence of PUFs. We agree with this observation. However the critical point is that even though there can exist a PUF that helps to break in polynomial time a standard complexity-theoretic assumptions, it is still unlikely that a PPT adversary can find such a PUF. Indeed a PPT machine can only generate a polynomial number of PUFs, therefore obtaining the one that allows to break complexity assumptions is an event that happens with negligible probability and thus it does not effect the concrete security of the protocols.

In light of the above discussion, only one of the following two cases is possible. 1) Standard complexity-theoretic assumptions still hold in presence of PPT adversaries that generate PUFs; in this case our construction is secure. 2) There exists a PPT adversary that can generate a PUF that breaks standard assumptions; in this case our construction is not secure, but the whole foundations of complexity-theoretic cryptography would fall down (which is quite unlikely to happen) with respect to real-world adversaries. We elaborate on this issue in Section 3.1.

Unconditional cryptography with malicious PUFs. We show a commitment scheme that is unconditionally secure (statistically hiding and statistically binding), even with malicious PUFs. The scheme requires one PUF only, sent by the committer at the beginning of the protocol. Achieving unconditional *UC secure* commitments is left as open problem.

Malicious access to PUFs. When using the simulation-based security paradigm, the actual security obtained depends on the power of the simulator in the ideal world. A good model should not give unwarranted capabilities to the simulator. For example, allowing the simulator to program the outputs of a PUF assumes implicitly that the output of a PUF can be simulated by a PPT

machine. However, this is not necessarily the case in the real life, as PUFs are complex physical devices not necessarily simulatable in PPT. For this reason in [BFSK11] the simulator is not allowed to program PUFs.

Another power that the simulator has in the original formulation of Brzuska et al. is that it can *observe* queries made by the adversary to PUFs. In real life, this translates to requiring that the adversary queries the PUF *only using the prescribed, honest query process*. However, in reality an adversary can subject the PUF to some maliciously chosen physical stimulation, deviating from the prescribed process (we discuss it in details in Section 4). Therefore, giving the simulator the ability to observe adversary's queries could be an unwarranted capability, which is at odds with the real-life execution. Moreover note that in UC the environment could be the player that knows such different procedures and thus one can not assume that they are known to the simulator too, since in UC the same simulator must work for every environment.

This leads us to formalize a model in which the simulator *is not allowed to observe* the adversary's queries, and the natural question is whether any security is still possible in such a model. Note that the original protocols of Brzuska et al. [BFSK11] clearly fail in this model. Such a formulation is interesting from a theoretical point of view, to better understand the question of the achievability of UC security with minimal restrictions on the adversary's (mis)behavior. For instance, Nielsen [Nie02] mentions this as an interesting open problem in the random oracle model.

We qualitatively strengthen the UC feasibility result of Brzuska et al. by achieving unconditional UC secure computation with a weaker simulator. In particular, we construct a simulator that, in addition to being disallowed to program the output of a PUF as in the original formulation of [BFSK11], is *not even allowed to observe the PUF queries made by the adversary*. The concrete requirements of such a construction (e.g., several PUFs that go back and forth) make it merely a feasibility result that contributes to the understanding of the foundational concept of achieving UC security with PUFs.

Impossibility of UC secure computation in the combined model. The positive results noted above lead to the natural question of feasibility of UC in a model where the real-world adversary can create malicious PUFs *and* can query honest PUFs with non-prescribed evaluation procedures. In the malicious PUF model the power of the simulator is to see the queries made by the real-world adversary to the honest PUFs and to have permanent access to the PUFs present in the system (i.e., the simulator can always query the PUFs). In the oblivious queries model, challenge/response pairs corresponding to queries done by the real-world adversary are not visible to the simulator. Therefore, when considering the combined model where the adversary can create malicious PUFs and the simulator can not see queries made by the adversary to PUFs, the only power left to the simulator is the permanent access to the PUFs present in the system. In Section 5 we show that it is impossible to achieve UC secure computation in this combined model.

Further details on our work. In our protocols after an execution the PUF can not be reused in another protocol execution. We explain how reuse of PUFs makes our commitment protocol insecure at the end of Section 3.2. However, this inability to reuse PUFs is not an artifact of our protocols, but is inherent in the UC formulation. In particular, the proof of the UC Composition Theorem [Can01] requires that different executions use independently created PUFs. Finding a formulation of security that allows reuse of PUFs (for e.g., by moving to the *Global* UC framework [CDPW07]) is an interesting open question. In the meantime, we feel using new PUFs for each execution is a cost we must bear for achieving UC. Another limitation of our protocol in the oblivious queries model is that it requires a large number of PUFs, but the idea there is to establish a feasibility result in

that severely adversarial model.

Independent work. Very recently, and independently of us, Rührmair and van Dijk in [vDR12] also investigate security under malicious PUFs (which they call "bad PUFs"), and show impossibility of some security definitions in certain settings that are very different from ours, as we will explain this below. They also provide constructions using honest, but less demanding PUFs, that however are secure only in the indistinguishability stand-alone sense (as opposed to being UC).

In more details [vDR12] develops three main directions of inquiry. Firstly, they propose a new definition for (honest) PUFs that is weaker than the original formulation of [BFSK11]. In particular, the min-entropy condition is required to hold only for randomly chosen challenges (instead of for *every* challenge vector). They show constructions for Oblivious Transfer (OT), Key agreement, and Bit Commitment (BC) using honest PUFs with this less demanding unpredictability property. However, such constructions are secure only in the indistinguishability stand-alone sense, (i.e., no UC security). Secondly, they propose two "attack models" for bad PUF behaviour, and give impossibility results. The posterior access model (PAM) considers a scenario in which all PUFs in the system are trusted, but an adversary can access the PUF after a protocol execution is completed, even if at the end of the protocol the PUF is physically in the hands of the honest player[4]. Their first impossibility result shows that *unconditional* OT in the PAM model is impossible. Their second impossibility result shows that *unconditional* OT is impossible also in the "bad PUF model" under the assumption that even honest players generate malicious PUFs. Indeed, while a bad PUF is very similar to our malicious PUF, in their impossibility proof, they seems to require that *all* PUFs in the system are bad, *even those created by the honest parties*. This is in contrast with real-life scenarios when we assume that honest PUFs are available to honest parties (e.g., a honest party can generate a PUF by himself, or can obtain it from a trusted source). Our formalization of malicious PUFs does allow an honest party to create honest PUFs (while the adversary can create arbitrary malicious PUFs), so the impossibility result of [vDR12] has little bearing on our work. They also show a construction of unconditional (not UC) Bit Commitment with bad PUFs. Their construction is more demanding compared to our BC construction. Indeed their construction requires two PUFs, and one PUF goes back and forth. The security proof of the BC protocol assumes that honest parties create honest PUFs.

Finally, [vDR12] (see also [RvD12]) showed that if a malicious receiver in the OT protocol of [BFSK11] (see the OT scheme in Fig. 3, [BFSK11]) is allowed to make $2^{n/2}$ queries to the (honest) PUF, where n is the length of the query, then he can obtain both the strings of the sender (thus achieving a quadratic speed-up over the brute force attack). This is also applicable to our OT protocol in the oblivious queries model. This attack becomes relevant when the challenge space of the actual PUF families is fairly small and it becomes feasible for the adversary to make $2^{n/2}$ queries. The fact that both Brzuska et al. protocol and our protocol in the oblivious query model require PUFs with large input space gives an insight that there is still a gap between those foundational results and their concrete use in the real world. We feel this provides further impetus to practitioners towards realizing more sophisticated (i.e., with larger input space) PUFs, and to theoreticians to provide constructions where security is maintained even when PUFs are assumed to have small input space.

[4] In our UC formulation, the honest players makes the PUF unavailable (i.e., it can be destroyed) to the adversary and therefore such an attack is not possible. This is similar to the original formulation of [BFSK11].

2 Definitions

Notation. We let n be the security parameter and PPT be the class of probabilistic polynomial time Turing machines. For PPT algorithms A and B, we denote by $(v_A, v_B) \leftarrow \langle A(a), B(b)\rangle(x)$ the random process obtained by having A and B interact on common input x and on (private) auxiliary inputs a and b, respectively, and with independent random coin tosses for A and B. We use $v \xleftarrow{\$} A()$ when the algorithm $A()$ is randomized. Let P_1 and P_2 be two parties running a protocol that uses protocol (A,B) as sub-protocol. When we say that party "P_1 runs $\langle A(\cdot), B(\cdot)\rangle(\cdot)$ with P_2" we always mean that P_1 executes the procedure of party A and P_2 executes the procedure of party B. We will denote by $\mathsf{dis}_{\mathsf{ham}}(a, b)$ the Hamming distance of a and b. For an n bit string x, $\mathsf{Parity}(x)$ is the sum of the bits of x mod 2.

Physically uncloneable functions. In this section we follow definitions given in [BFSK11]. A PUF is a noisy physical source of randomness. The randomness property comes from an uncontrollable manufacturing process. A PUF is evaluated with a physical stimulus, called the *challenge*, and its physical output, called the *response*, is measured. Because the processes involved are physical, the function implemented by a PUF can not (necessarily) be modeled as a mathematical function, neither can be considered computable in PPT. Moreover, the output of a PUF is noisy, namely, querying a PUF twice with the same challenge, could yield to different outputs. The mathematical formalization of a PUF due to [BFSK11] is the following.

A PUF-family $\mathcal{P}$ is a pair of (not necessarily efficient) algorithms Sample and Eval, and is parameterized by the bound on the noise of PUF's response d_{noise} and the range of the PUF's output rg. Algorithm Sample abstracts the PUF fabrication process and works as follows. On input the security parameter, it outputs a PUF-index id from the PUF-family satisfying the security property (that we define soon) according to the security parameter. Algorithm Eval abstracts the PUF-evaluation process. On input a challenge q, it evaluates the PUF on q and outputs the response a of length rg. The output is guaranteed to have bounded noise d_{noise}, meaning that, when running $\mathsf{Eval}(1^n, \mathsf{id}, q)$ twice, the Hamming distance of any two responses a_1, a_2 is smaller than $d_{\mathsf{noise}}(n)$. Wlog, we assume that the challenge space of a PUF is a full set of strings of a certain length.

Definition 1 (Physically Uncloneable Functions)**.** *Let rg denote the size of the range of the PUF responses of a PUF-family and d_{noise} denote a bound of the PUF's noise. $\mathcal{P} = (\mathsf{Sample}, \mathsf{Eval})$ is a family of (rg, d_{noise})-PUF if it satisfies the following properties.*

- **Index Sampling.** *Let $\mathcal{I}_n$ be an index set. On input the security parameter n, the sampling algorithm Sample outputs an index $\mathsf{id} \in \mathcal{I}_n$ following a not necessarily efficient procedure. Each $\mathsf{id} \in \mathcal{I}_n$ corresponds to a set of distributions $\mathcal{D}_{\mathsf{id}}$. For each challenge $q \in \{0,1\}^n$, $\mathcal{D}_{\mathsf{id}}$ contains a distribution $\mathcal{D}_{\mathsf{id}}(q)$ on $\{0,1\}^{rg(n)}$. $\mathcal{D}_{\mathsf{id}}$ is not necessarily an efficiently sampleable distribution.*
- **Evaluation.** *On input the tuple $(1^n, \mathsf{id}, q)$, where $q \in \{0,1\}^n$, the evaluation algorithm Eval outputs a response $a \in \{0,1\}^{rg(n)}$ according to distribution $\mathcal{D}_{\mathsf{id}}(q)$. It is not required that Eval is a PPT algorithm.*
- **Bounded Noise.** *For all indexes $\mathsf{id} \in \mathcal{I}_n$, for all challenges $q \in \{0,1\}^n$, when running $\mathsf{Eval}(1^n, \mathsf{id}, q)$ twice, the Hamming distance of any two responses a_1, a_2 is smaller than $d_{\mathsf{noise}}(n)$.*

In the paper we use $\mathsf{PUF}_{\mathsf{id}}(q)$ to denote $\mathcal{D}_{\mathsf{id}}(q)$. When not misleading, we omit id from $\mathsf{PUF}_{\mathsf{id}}$, using only the notation PUF.

Security of PUFs. We assume that PUFs enjoy the properties of *uncloneability* and *unpredictability*. Unpredictability is modeled via an entropy condition on the PUF distribution. Namely, given

that a PUF has been measured on a polynomial number of challenges, the response of the PUF evaluated on a new challenge has still a significant amount of entropy. In the following we recall the concept of average min-entropy.

Definition 2 (Average min-entropy). *The average min-entropy of the measurement* $\mathsf{PUF}(q)$ *conditioned on the measurements of challenges* $\mathcal{Q} = (q_1, \ldots, q_{\mathtt{poly}(n)})$ *is defined by*

$$\begin{aligned}\tilde{H}_\infty(\mathsf{PUF}(q)|\mathsf{PUF}(\mathcal{Q})) &= -log\big(\mathbb{E}_{a_k \leftarrow \mathsf{PUF}(q_k)}[\max_a \Pr\left[\,\mathsf{PUF}(q) = a | a_1 = \mathsf{PUF}(q_1), \ldots, a_{\mathtt{poly}(n)} = \mathsf{PUF}(q_{\mathtt{poly}(n)})\,\right]]\big) \\ &= -log\big(\mathbb{E}_{a_k \leftarrow \mathsf{PUF}(q_k)}[2^{H_\infty(\mathsf{PUF}(q)=a|a_1=\mathsf{PUF}(q_1),\ldots,a_{\mathtt{poly}(n)}=\mathsf{PUF}(q_{\mathtt{poly}(n)})}]\big)\end{aligned}$$

where the probability is taken over the choice of id *from* $\mathcal{I}_n$ *and the choice of possible PUF responses on challenge* q*. The term* $\mathsf{PUF}(\mathcal{Q})$ *denotes a sequence of random variables* $\mathsf{PUF}(q_1), \ldots, \mathsf{PUF}(q_{\mathtt{poly}(n)})$ *each corresponding to an evaluation of the PUF on challenge* q_k*, for* $1 \leq k \leq \mathtt{poly}(n)$*.*

Definition 3 (Unpredictability). *A* (rg, d_{noise})*-PUF family* $\mathcal{P} = (\mathsf{Sample}, \mathsf{Eval})$ *for security parameter* n *is* $(d_{\mathsf{min}}(n), m(n))$*-unpredictable if for any* $q \in \{0,1\}^n$ *and challenge list* $\mathcal{Q} = (q_1, \ldots, q_{\mathtt{poly}(n)})$*, one has that, if for all* $1 \leq k \leq \mathtt{poly}(n)$ *the Hamming distance satisfies* $\mathsf{dis}_{\mathsf{ham}}(q, q_k) \geq d_{\mathsf{min}}(n)$*, then the average min-entropy satisfies* $\tilde{H}_\infty(\mathsf{PUF}(q)|\mathsf{PUF}(\mathcal{Q})) \geq m(n)$*, where* $\mathsf{PUF}(\mathcal{Q})$ *denotes a sequence of random variables* $\mathsf{PUF}(q_1), \ldots, \mathsf{PUF}(q_{\mathtt{poly}(n)})$ *each corresponding to an evaluation of the PUF on challenge* q_k*. Such a PUF-family is called a* $(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, m)$*-PUF family.*

Fuzzy Extractors. The output of a PUF is noisy, that is, feeding it with the same challenge twice may yield distinct, but still close, responses. Fuzzy extractors of Dodis et al. [DORS08] are applied to the outputs of the PUF to convert such noisy, high-entropy measurements into *reproducible* randomness.

Let U_ℓ denote the uniform distribution on ℓ-bit strings. Let $\mathcal{M}$ be a metric space with the distance function dis: $\mathcal{M} \times \mathcal{M} \to \mathbb{R}^+$.

Definition 4 (Fuzzy Extractors). *A* (m, ℓ, t, ϵ)*-fuzzy extractor is a pair of efficient randomized algorithms* $(\mathsf{FuzGen}, \mathsf{FuzRep})$*. The algorithm* FuzGen *on input* $w \in \mathcal{M}$*, outputs a pair* (p, st)*, where* $st \in \{0,1\}^\ell$ *is a secret string and* $p \in \{0,1\}^*$ *is a helper data string. The algorithm* FuzRep*, on input an element* $w' \in \mathcal{M}$ *and a helper data string* $p \in \{0,1\}^*$ *outputs a string* st*. A fuzzy extractor satisfies the following properties.*

Correctness. *For all* $w, w' \in \mathcal{M}$*, if* $\mathsf{dis}(w, w') \leq t$ *and* $(st, p) \xleftarrow{\$} \mathsf{FuzGen}$*, then* $\mathsf{FuzRep}(w', p) = st$*.*

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

Given a $(rg(n), d_{\mathsf{noise}}(n), d_{\mathsf{min}}(n), m(n))$-PUF family with $d_{\mathsf{min}}(n) = o(n/\log n)$, a *matching* fuzzy extractor has as parameters $\ell(n) = n$ and $t(n) = d_{\mathsf{noise}}(n)$. The metric space $\mathcal{M}$ is the range $\{0,1\}^{rg}$ with Hamming distance $\mathsf{dis}_{\mathsf{ham}}$. We call such PUF family and fuzzy extractor as having matching parameters, and the following properties are guaranteed.

Well-Spread Domain. For all polynomial $p(n)$ and all set of challenges $q_1, \ldots, q_{p(n)}$, the probability that a randomly chosen challenge is within distance smaller than d_{min} with any q_k, for $1 \leq k \leq n$ is negligible.

Extraction Independence. For all challenges $q_1, \ldots, q_{p(n)}$, and for a challenge q such that $\mathsf{dis}(q, q_k) > d_{\mathsf{min}}$ for $1 \leq k \leq p(n)$, it holds that the PUF evaluation on q and subsequent application of FuzGen yields an almost uniform value st even if p is observed.

Response consistency. Let a, a' be the responses of PUF when queried twice with the same challenge q, then for $(st, p) \xleftarrow{\$} \mathsf{FuzGen}(a)$ it holds that $st \leftarrow \mathsf{FuzRep}(a', p)$.

3 UC Security with Malicious PUFs

In Section 1 we have motivated the need of a different formulation of UC security with PUFs that allows the adversary to generate malicious PUFs. In this section we first show how to model malicious PUFs in the UC framework, and then show that as long as standard computational assumptions still hold when PPT adversaries can generate (even malicious) PUFs, there exist protocols for UC realizing any functionality with (malicious) PUFs.

3.1 Modeling Malicious PUFs

We allow our adversaries to send malicious PUFs to honest parties[5]. As discussed before, the motivation for malicious PUFs is that the adversary may have some control over the manufacturing process and may be able to produce errors in the process that break the PUF's security properties. Thus, we would like parties to rely on only the PUFs that they themselves manufacture (or obtain from a source that they trust), and not on the ones they receive from other (possibly adversarial) parties.

Malicious PUF Families. In the real world, an adversary may create a malicious PUF in a number of ways. For example, it can tamper with the manufacturing process for an honestly-generated PUF to compromise its security properties (unpredictability, for instance). It may also introduce additional behaviour into the PUF token, like logging of queries. Taking inspiration from the literature on modeling tamper-proof hardware tokens, one might be tempted to model malicious PUFs analogously in the following way: to create a malicious PUF, the adversary simply specifies to the ideal functionality, the (malicious) code it wants to be executed instead of an honest PUF. But, note that as the adversary is a PPT machine, the malicious code it specifies must also be PPT. However, PUFs are *not* modeled as PPT machines, so this places severe restrictions on the adversaries[6]. In particular, modeling malicious PUFs in this way would disallow the adversary from modifying honest PUFs (or more precisely, the honest PUF manufacturing process). Instead, we allow the adversary to specify a "malicious PUF family", that the ideal functionality uses. To keep our treatment as general as possible, we do not place any restriction on a malicious PUF, except that it should have the same syntax as that of an honest PUF family, as specified in Definition 1. Of course, in the protocol, we also want the honest parties to be able to obtain and send honestly generated PUFs. Thus our ideal functionality for PUFs, $\mathcal{F}_{\mathsf{PUF}}$ (Fig. 1) is parameterized by two PUF families: the normal (or honest) family $(\mathsf{Sample}_{\mathtt{normal}}, \mathsf{Eval}_{\mathtt{normal}})$ and the possibly malicious family $(\mathsf{Sample}_{\mathtt{mal}}, \mathsf{Eval}_{\mathtt{mal}})$. When a party P_i wants to initialize a PUF, it sends a $\mathsf{init}_{\mathsf{PUF}}$ message to $\mathcal{F}_{\mathsf{PUF}}$ in which it specifies the $\mathtt{mode} \in \{\, \mathtt{normal}, \mathtt{mal} \,\}$, and the ideal functionality uses the corresponding

[5] Throughout this section, we assume the reader is familiar with the original UC PUF formulation of Brzuska et al. ([BFSK11], Section 4.2).

[6] Observe that allowing the adversary to specify the malicious code enables the simulator to "rewind" the malicious PUF. However, in the model we use, such rewinding is not possible.

family for initializing the PUF. For each initialized PUF, the ideal functionality $\mathcal{F}_{\mathsf{PUF}}$ also stores a tag representing the family (i.e., `mal` or `normal`) from which it was initialized. Thus, when the PUF needs to be evaluated, $\mathcal{F}_{\mathsf{PUF}}$ runs the evaluation algorithm corresponding to the tag.

As in the original formulation of Brzuska et al., the ideal functionality $\mathcal{F}_{\mathsf{PUF}}$ keeps a list $\mathcal{L}$ of tuples $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \hat{P}, \tau)$. Here, sid is the session identifier of the protocol and id is the PUF identifier output by the $\mathsf{Sample}_{\mathtt{mode}}$ algorithm. As discussed above $\mathtt{mode} \in \{\, \mathtt{normal}, \mathtt{mal} \,\}$ indicates the mode of the PUF, and $\hat{P}$ identifies the party that currently holds the PUF. The final argument τ specifies transition of PUFs: $\tau = \mathsf{notrans}$ indicates the PUF is not in transition, while $\tau = \mathsf{trans}(P_j)$ indicates that the PUF is in transition to party P_j. Only the adversary may query the PUF during the transition period. Thus, when a party P_i hands over a PUF to party P_j, the corresponding τ value for that PUF is changed from $\mathsf{notrans}$ to $\mathsf{trans}(P_j)$, and the adversary is allowed to send evaluation queries to this PUF. When the adversary is done with querying the PUF, it sends a $\mathsf{ready}_{\mathsf{PUF}}$ message to the ideal functionality, which hands over the PUF to P_j and changes the PUFs transit flag back to $\mathsf{notrans}$. The party P_j may now query the PUF. The ideal functionality now waits for a $\mathsf{received}_{\mathsf{PUF}}$ message from the adversary, at which point it sends a $\mathsf{received}_{\mathsf{PUF}}$ message to P_i informing it that the hand over is complete. The ideal functionality is described formally in Fig. 1.

Allowing Adversary to Create PUFs. We deviate from the original formulation of $\mathcal{F}_{\mathsf{PUF}}$ of Brzuska et al. [BFSK11] in one crucial way: we allow the ideal-world adversary $\mathcal{S}$ to create new PUFs. That is, $\mathcal{S}$ can send a $\mathsf{init}_{\mathsf{PUF}}$ message to $\mathcal{F}_{\mathsf{PUF}}$. In the original formulation of Brzuska et al., $\mathcal{S}$ could not create its own PUFs, and this has serious implications for the composition theorem. We thank Margarita Vald [Val12] for pointing out this issue. We elaborate on this in Appendix H. Also, it should be noted that the PUF set-up is non-programmable, but not *global* [CDPW07]. The environment must go via the adversary to query PUFs, and may only query PUFs in transit or held by the adversary at that time.

We remark that the OT protocol of [BFSK11] for honest PUFs, fails in the presence of malicious PUFs. Consider the OT protocol in Fig. 3 in [BFSK11]. The security crucially relies on the fact that the receiver P_j *can not* query the PUF after receiving sender's first message, i.e., the pair (x_0, x_1). If it could do so, then it would query the PUF on both $x_0 \oplus v$ and $x_1 \oplus v$ and learn both s_0 and s_1. In the malicious PUF model however, as there is no guarantee that the receiver can not learn query/answer pairs when a malicious PUF that he created is not in its hands, the protocol no longer remains secure.

PUFs and computational assumptions. The protocol we present in the next section will use computational hardness assumptions. These assumptions hold against probabilistic polynomial-time adversaries. However, PUFs use physical components and are not modeled as PPT machines, and thus, the computational assumptions must additionally be secure against PPT adversaries that have access to PUFs. We remark that this is a reasonable assumption to make, as if this is not the case, then PUFs can be used to invert one-way functions, to find collisions in CRHFs and so on, therefore not only our protocol, but any computational-complexity based protocol would be insecure. Note that PUFs are physical devices that actually exist in the real world, and thus all real-world adversaries could use them.

To formalize this, we define the notion of "admissible" PUF families. Informally, a PUF family (regardless of whether it is honest or malicious) is called *admissible* with respect to a hardness assumption if that assumption holds even when the adversary has access to PUFs from this family. We will prove that our protocol is secure when the $\mathcal{F}_{\mathsf{PUF}}$ ideal functionality is instantiated with

$\mathcal{F}_{\mathsf{PUF}}$ uses PUF families $\mathcal{P}_1 = (\mathsf{Sample}_{\mathtt{normal}}, \mathsf{Eval}_{\mathtt{normal}})$ with parameters $(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, m)$, and $\mathcal{P}_2 = (\mathsf{Sample}_{\mathtt{mal}}, \mathsf{Eval}_{\mathtt{mal}})$. It runs on input the security parameter 1^n, with parties $\mathbb{P} = \{ P_1, \ldots, P_n \}$ and adversary $\mathcal{S}$.

- When a party $\hat{P} \in \mathbb{P} \cup \{ \mathcal{S} \}$ writes $(\mathsf{init}_{\mathsf{PUF}}, \mathsf{sid}, \mathtt{mode}, \hat{P})$ on the input tape of $\mathcal{F}_{\mathsf{PUF}}$, where $\mathtt{mode} \in \{ \mathtt{normal}, \mathtt{mal} \}$, then $\mathcal{F}_{\mathsf{PUF}}$ checks whether $\mathcal{L}$ already contains a tuple $(\mathsf{sid}, *, *, *, *)$:
  - If this is the case, then turn into the waiting state.
  - Else, draw $\mathsf{id} \leftarrow \mathsf{Sample}_{\mathtt{mode}}(1^n)$ from the PUF family. Put $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \hat{P}, \mathsf{notrans})$ in $\mathcal{L}$ and write $(\mathsf{initialized}_{\mathsf{PUF}}, \mathsf{sid})$ on the communication tape of $\hat{P}$.
- When party $P_i \in \mathbb{P}$ writes $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{sid}, P_i, q)$ on $\mathcal{F}_{\mathsf{PUF}}$'s input tape, check if there exists a tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, P_i, \mathsf{notrans})$ in $\mathcal{L}$.
  - If not, then turn into waiting state.
  - Else, run $a \leftarrow \mathsf{Eval}_{\mathtt{mode}}(1^n, \mathsf{id}, q)$. Write $(\mathsf{response}_{\mathsf{PUF}}, \mathsf{sid}, q, a)$ on P_i's communication input tape.
- When a party P_i sends $(\mathsf{handover}_{\mathsf{PUF}}, \mathsf{sid}, P_i, P_j)$ to $\mathcal{F}_{\mathsf{PUF}}$, check if there exists a tuple $(\mathsf{sid}, *, *, P_i, \mathsf{notrans})$ in $\mathcal{L}$.
  - If not, then turn into waiting state.
  - Else, modify the tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, P_i, \mathsf{notrans})$ to the updated tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \perp, \mathsf{trans}(P_j))$. Write $(\mathsf{invoke}_{\mathsf{PUF}}, \mathsf{sid}, P_i, P_j)$ on $\mathcal{S}$'s communication input tape.
- When the adversary sends $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{sid}, \mathcal{S}, q)$ to $\mathcal{F}_{\mathsf{PUF}}$, check if $\mathcal{L}$ contains a tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \perp, \mathsf{trans}(*))$ or $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \mathcal{S}, \mathsf{notrans})$.
  - If not, then turn into waiting state.
  - Else, run $a \leftarrow \mathsf{Eval}_{\mathtt{mode}}(1^n, \mathsf{id}, q)$ and return $(\mathsf{response}_{\mathsf{PUF}}, \mathsf{sid}, q, a)$ to $\mathcal{S}$.
- When $\mathcal{S}$ sends $(\mathsf{ready}_{\mathsf{PUF}}, \mathsf{sid}, \mathcal{S})$ to $\mathcal{F}_{\mathsf{PUF}}$, check if $\mathcal{L}$ contains the tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \perp, \mathsf{trans}(P_j))$.
  - If not found, turn into the waiting state.
  - Else, change the tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \perp, \mathsf{trans}(P_j))$ to $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, P_j, \mathsf{notrans})$ and write $(\mathsf{handover}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ on P_j's communication input tape and store the tuple $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$.
- When the adversary sends $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ to $\mathcal{F}_{\mathsf{PUF}}$, check if the tuple $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ has been stored. If not, return to the waiting state. Else, write this tuple to the input tape of P_i.

Figure 1: The ideal functionality $\mathcal{F}_{\mathsf{PUF}}$ for malicious PUFs.

admissible PUF families.

For our purpose, all the cryptographic tools that we use to construct our protocols can be based on the DDH assumption. Thus, we define PUF families that are admissible only with respect to DDH, but note that the definition can be generalized to other cryptographic primitives. This is a straightforward generalization of the standard DDH definition, and is given in Appendix B. From this point on in this paper, we only talk of admissible families.

3.2 Constructions for UC Security in the Malicious PUFs model

In this section we present a construction that UC-realizes the commitment functionality $\mathcal{F}_{\mathsf{com}}$ in the malicious PUFs model, and thus UC security for any PPT functionality using the compiler

of [CLOS02].

Let us recall some of the peculiarities of the PUFs model. A major difficulty when using PUFs, in contrast to tamper-proof tokens, is that PUFs are *not programmable*. That is, the simulator can not simulate the answer of a PUF, and it must honestly forward the queries to the $\mathcal{F}_{\mathsf{PUF}}$ functionality. The only power of the simulator is to intercept the queries made by the adversary to honest PUFs. Thus, in designing the protocol, we shall force parties to query the PUFs with the critical private information related to the protocol, so as to allow the simulator to extract such information in straight-line. In the malicious PUFs model the behaviour of a PUF sent by an adversary is entirely in the adversary's control. A malicious PUF can answer (or even abort) adaptively on the query according to some pre-shared strategy with the malicious creator. Finally, a side effect of the unpredictability of PUFs, is that the creator of a honest PUF is not able to check the authenticity of the answer generated by its own PUF, without having the PUF in its hands (or having queried the PUF previously on the very same value). This might generate the undesired effect of PUFs going back and forth during the protocol.

High-level idea of the protocol. Showing UC security for commitments requires obtaining straight-line extraction and straight-line equivocality. Our starting point is the equivocal commitment scheme of [CO99] which in turn is based on Naor's scheme [Nao89] and that consists of two messages. The first message is a randomly chosen string r that the receiver sends to the committer. The second message is the commitment of the bit b, computed using r. More precisely, it is $G(s) \oplus (r \wedge \mathbf{b})$ ($\mathbf{b}$ is the $|r|$-bit vector containing all bits b), where $G()$ is a PRG, and s a randomly chosen seed. The scheme has the property that if the string r is crafted appropriately, then the commitment is equivocal. [CO99] shows how this can be achieved by adding a coin-tossing phase before the commitment. The coin tossing of [CO99] proceeds as follows: the receiver commits to a random string α (using a statistically hiding commitment scheme), the committer sends a string β, and then the receiver opens the commitment. Naor's parameter r is then set as $\alpha \oplus \beta$.

Against a malicious receiver, observe that if the simulator can choose β after knowing α, then it can control the output of the coin-tossing phase (which is, in turn, sufficient for equivocation). In order to do this, it needs to extract the value α committed by the receiver, before having to send β. Furthermore, besides being equivocal, the second message sent by the Naor's committer must be extractable. For this purpose, we construct a straight-line extractable commitment scheme in the malicious PUFs model. Plugging an extractable commitment in both directions, i.e., for the commitment of α and for the commitment of the bit, the [CO99] construction yields a straight-line equivocal (and extractable) commitment scheme.

Thus, now we need to construct extractable commitments with malicious PUFs. The key idea is the following. The receiver creates a PUF and queries it with challenges (q_0, q_1). The PUF is then sent to the committer. To commit to a bit b, the committer queries such PUF with the string q_b, and commits[7] to the response obtained from the PUF. The committer obtains the desired query q_b by running an OT protocol with the receiver. The simulator can extract the bit by checking the queries sent to the PUF and looking which one is close enough, in Hamming distance, to either q_0 or q_1. Due to the security of OT, the committer can not get both queries (thus confusing the simulator), neither can the receiver detect which query has been transferred. Due to the binding property of the commitment scheme used to commit q_b, a malicious committer can not postpone

[7] The commitment scheme used to commit q_b, is statistically hiding in protocol $\mathsf{Com}_{\mathsf{shext}}$ (run by the receiver to commit to α), and is "equivocal" and statistically binding in protocol $\mathsf{Com}_{\mathsf{equiv}}$ (run by the committer to commit to the bit).

querying the PUF to the decommitment phase (thus preventing the simulator to extract already in the commitment phase). Due to the unpredictability of PUFs, the committer cannot avoid to query the PUF to obtain the correct response.

This idea is used in both directions. From the receiver to the committer, when the receiver is committing to α, and from the committer to the receiver, when the committer has to commit to the bit. However, the commitment of α played by the receiver, must be statistically hiding. While the commitment of the bit, played by the committer, must be equivocal. As we see later, achieving each property requires different techniques. Therefore, we implement two extractable commitment schemes: $\mathsf{Com}_{\mathsf{shext}}$ used by the receiver and $\mathsf{Com}_{\mathsf{equiv}}$ used by the committer. Each protocol involves the use of one PUF, sent by the receiver to the committer. PUFs are transferred only once, at the beginning of the protocol.

The final construction $\mathsf{Com}_{\mathsf{uc}} = (\mathsf{C}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}})$ is formally described in Fig. 4 (a graphical representation is provide in Fig. 5). It consists of the coin-flipping phase, and the (equivocal) commitment phase. In the coin flipping, the receiver commits to α using the statistically hiding straight-line extractable commitment scheme $\mathsf{Com}_{\mathsf{shext}} = (\mathsf{C}_{\mathsf{shext}}, \mathsf{R}_{\mathsf{shext}})$. The output of the coin-flipping is the Naor's parameter $\bar{r}{=}\alpha \oplus \beta$ used as input for the extractable/equivocal commitment scheme $\mathsf{Com}_{\mathsf{equiv}} = (\mathsf{C}_{\mathsf{equiv}}, \mathsf{R}_{\mathsf{equiv}})(\bar{\mathsf{r}})$.

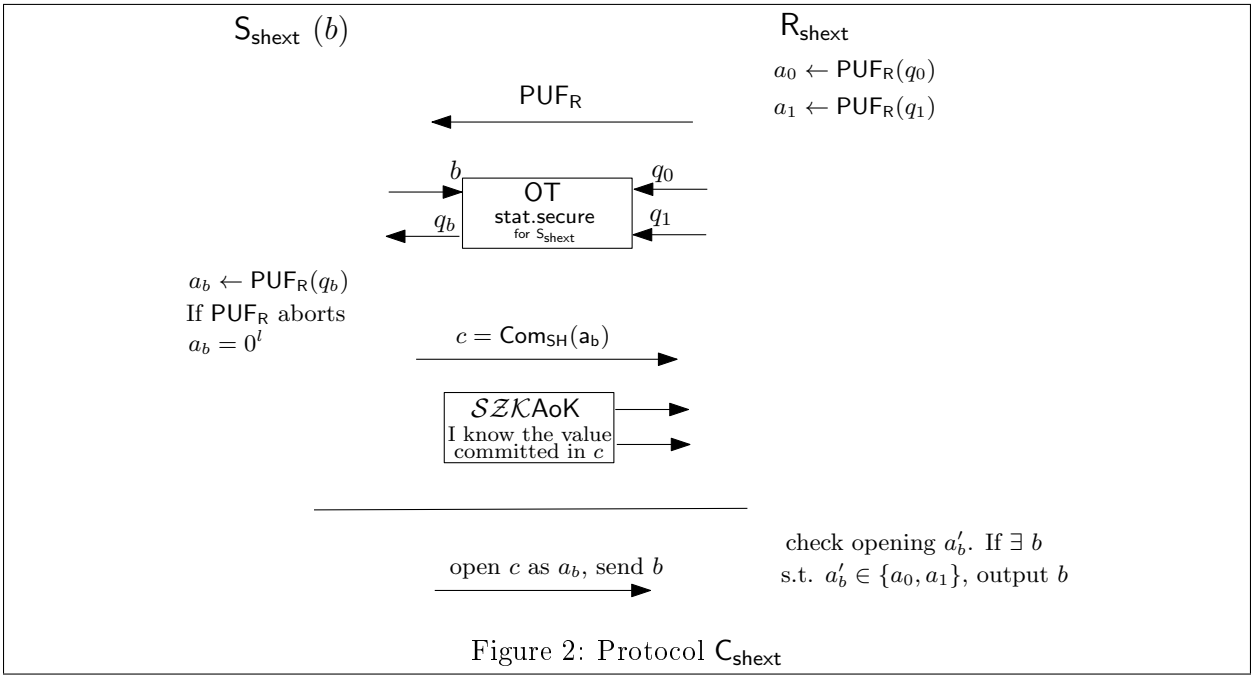

Figure 2: Protocol $\mathsf{C}_{\mathsf{shext}}$

Details on the implementation. Recall the basic steps of the extractable commitment scheme. The receiver queries its own PUF with (q_0, q_1) and transfers the PUF to the committer. The committer, on input a bit b, runs the OT protocol with the receiver, to obtain q_b. It queries the PUF with q_b and it commits to the answer obtained from it. To commit to a n-bit string, the same procedure is repeated in parallel n times. These basic steps are augmented in different ways for the implementation of $\mathsf{Com}_{\mathsf{shext}}$ and $\mathsf{Com}_{\mathsf{equiv}}$.

Implementing the statistically-hiding extractable commitment scheme $\mathsf{Com}_{\mathsf{shext}} = (\mathsf{C}_{\mathsf{shext}}, \mathsf{R}_{\mathsf{shext}})$**.** Protocol $\mathsf{Com}_{\mathsf{shext}}$ is used by R_{uc} to commit to α. The committer $\mathsf{C}_{\mathsf{shext}}$, commits to the PUF-

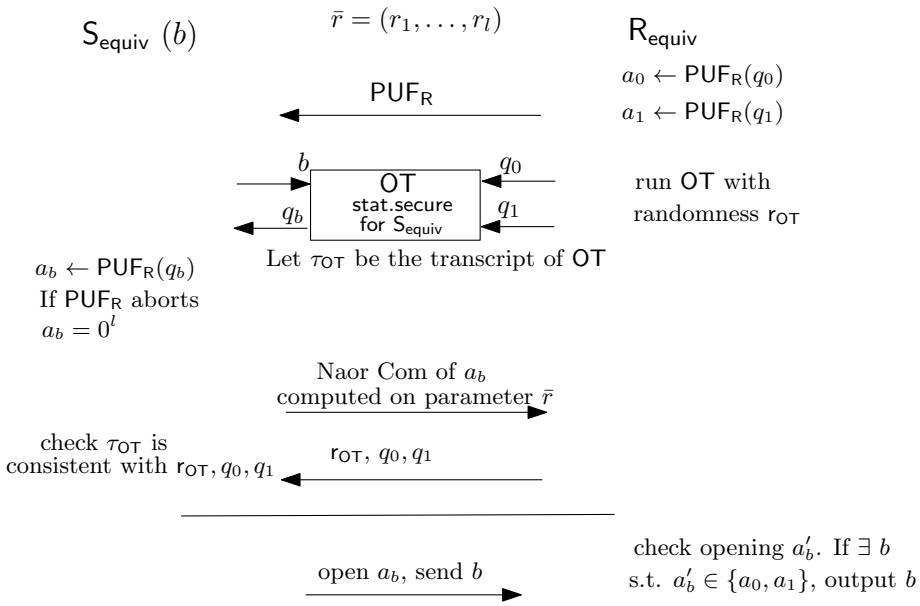

Figure 3: Protocol $\mathsf{Com}_{\mathsf{equiv}}$

response using a statistically hiding commitment scheme, and the OT protocol must be such that the receiver's privacy is statistical. Along with the statistically hiding commitment, $\mathsf{C}_{\mathsf{shext}}$ provides a statistical zero-knowledge argument of knowledge of the message committed. This turns out to be necessary to argue about binding (that is only computational). A graphic high-level description of $\mathsf{Com}_{\mathsf{shext}}$ is given in Fig. 2, while the formal specification is given in Fig. 10. Formal proofs are given in Appendix C.1.

Implementing the statistically-binding straight-line extractable and equivocal commitment $\mathsf{Com}_{\mathsf{equiv}}$**.** Protocol $\mathsf{Com}_{\mathsf{equiv}}$ is run by C_{uc} to commit to the secret bit b. The input of $\mathsf{Com}_{\mathsf{equiv}}$ is the Naor's parameter decided in the coin-flipping phase, and it is used to compute the second message of Naor's commitment. Note that, instead of committing to the single bit b, the committer will actually commit to the answer a_b, thus Naor's parameter is a vector $\bar{r}$ of strings $r_1, \ldots, r_l$. We said that the simulator can properly craft this parameter, so that it will equivocate the commitment, i.e., in the decommitment phase it is able to open to any string both a_0 and a_1.

However, a_0, a_1 are PUF-answers. Due to the security of the OT protocol, the simulator can obtain only one of the PUF-queries among (q_0, q_1), and the choice of the query must be done already in the commitment phase (when the secret bit b is not known to the simulator). Thus, even though the simulator has the power to equivocate the commitment to any string, it might not know the PUF-answer to open to. We solve this problem by asking the receiver to reveal both queries (q_0, q_1), once the committer has committed to the PUF-answer.

Now, the simulator can: play the OT protocol with a random bit, commit to a random string (without querying the PUF), and finally obtain both queries. In the decommitment phase, the simulator gets the actual bit b. Hence, it can query the PUF with input q_b, obtain the PUF-answer, and equivocate the commitment so to open to such PUF-answer.

Another subtle issue is that of a *selective abort* of a malicious PUF. If the PUF aborts when queried with a particular string, then we have that the committer would abort already in the

commitment phase, while the simulator aborts only in the decommitment phase. We avoid such problem by requiring that the committer continues the commitment phase also in case the PUF aborts, by committing to a random value. The above protocol is statistically binding (we are using Naor's commitment), straight-line extractable, and, assuming that Naor's parameter is decided by as explained above, it is also straight-line equivocal. To commit to a bit we are committing to the l-bit PUF-answer, thus the size of Naor's parameter is $N = (3n)l$. We denote such a protocol as $\mathsf{Com}_{\mathsf{equiv}}$. A graphic representation is given in Fig. 3, the formal specification is given in Fig. 11 and proofs are given in Appendix C.2. The proof of the next theorem is provided in Appendix D.

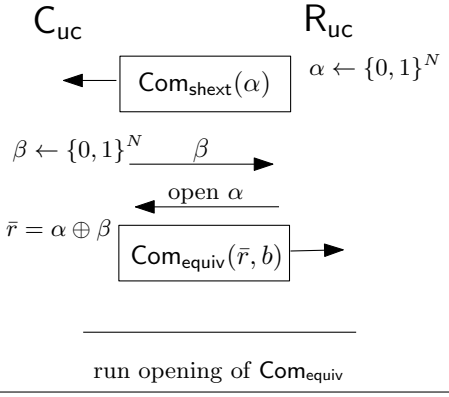

Figure 4: Pictorial representation of Protocol $\mathsf{Com}_{\mathsf{uc}}$.

Committer's Input: Bit $b \in \{0,1\}$.
Commitment Phase

$\mathsf{R}_{\mathsf{uc}} \Leftrightarrow \mathsf{C}_{\mathsf{uc}}$: (Coin Flipping)

1. R_{uc} picks $\alpha \xleftarrow{\$} \{0,1\}^N$; commit to α by running $((c_\alpha, d_\alpha), c_\alpha) \leftarrow \langle \mathsf{C}_{\mathsf{shext}}(\mathtt{com}, \alpha), \mathsf{R}_{\mathsf{shext}}(\mathtt{recv}) \rangle$ with C_{uc}.
2. C_{uc} sends $\beta \xleftarrow{\$} \{0,1\}^N$ to R_{uc}.
3. R_{uc} sends decommitment (α, d_α) to C_{uc}.
4. C_{uc}: if $\mathsf{R}_{\mathsf{shext}}(d_\alpha, \alpha) = 0$, abort.

$\mathsf{C}_{\mathsf{uc}} \Leftrightarrow \mathsf{R}_{\mathsf{uc}}$: (Equivocal Commitment)
C_{uc} commit to b by running $((c_{\mathsf{bit}}, d_{\mathsf{bit}}), c_{\mathsf{bit}}) \leftarrow \langle \mathsf{C}_{\mathsf{equiv}}(\mathtt{com}, b), \mathsf{R}_{\mathsf{equiv}}(\mathtt{recv}) \rangle (\alpha \oplus \beta)$ with R_{uc}.

Decommitment Phase

C_{uc} sends decommitment (b, d_{bit}) to R_{uc}.

R_{uc} accepts iff $\mathsf{R}_{\mathsf{equiv}}(\alpha \oplus \beta, c_{\mathsf{bit}}, b, d_{\mathsf{bit}})$ is accepting. Else, reject.

Figure 5: Computational UC Commitment Scheme $(\mathsf{C}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}})$.

Theorem 1. *If* $\mathsf{Com}_{\mathsf{shext}} = (\mathsf{C}_{\mathsf{shext}}, \mathsf{R}_{\mathsf{shext}})$ *is a statistically hiding straight-line extractable commitment scheme in the malicious PUFs model, and* $\mathsf{Com}_{\mathsf{equiv}} = (\mathsf{C}_{\mathsf{equiv}}, \mathsf{R}_{\mathsf{equiv}})$ *is a statistically binding straight-line extractable and equivocal commitment scheme in the malicious PUFs model, then* $\mathsf{Com}_{\mathsf{uc}} = (\mathsf{C}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}})$ *in Fig. 4, UC-realizes the* $\mathcal{F}_{\mathsf{com}}$ *functionality.*

The above protocol can be used to implement the multiple commitment functionality $\mathcal{F}_{\mathsf{mcom}}$ by using independent PUFs for each commitment. Note that in our construction we can not reuse the *same* PUF when multiple commitments are executed *concurrently*[8]. The reason is that, in both sub-protocols $\mathsf{Com}_{\mathsf{shext}}, \mathsf{Com}_{\mathsf{equiv}}$, in the opening phase the committer forwards the answer obtained by querying the receiver's PUF. The answer of a malicious PUF can then convey information about the value committed in concurrent sessions that have not been opened yet.

When implementing $\mathcal{F}_{\mathsf{mcom}}$ one should also deal with malleability issues. In particular, one should handle the case in which the man-in-the-middle adversary forwards honest PUFs to another party. However such attack can be easily ruled out by exploiting the unpredictability of honest PUFs as follows. Let P_i be the creator of PUF_i, running an execution of the protocol with P_j. Before delivering its own PUF, P_i queries it with the identity of P_j concatenated with a random *nonce*. Then, at some point during the protocol execution with P_j it will ask P_j to evaluate PUF_i on such *nonce* (and the identity). Due to the unpredictability of PUFs, and the fact that *nonce* is a randomly chosen value, P_j is able to answer to such a query only if it *possesses* the PUF. The final step to obtain UC security for any functionality consists in using the compiler of [CLOS02], which only needs a UC secure implementation of the $\mathcal{F}_{\mathsf{mcom}}$ functionality.

3.3 Unconditional Security with Malicious PUFs

We showed already how to implement the commitment functionality with malicious PUFs and computational assumptions, that combined with the compiler of [CLOS02] gives us UC secure computation with malicious PUFs under computational assumptions. The natural question to ask is whether we can leverage the power of PUFs to obtain *unconditional* UC secure computation. We leave this as an open question for future work. As evidence that this endeavour might be fruitful, in this section we provide a statistically hiding and statistically binding commitment scheme in the malicious PUF model.

Our protocol is an adaptation of Naor's commitment protocol [Nao89] in the malicious PUFs model. The sender first queries the PUF with a random string and then sends the PUF to the receiver. Then, as in Naor's protocol, the receiver sends a random string, and finally the sender sends either the response of the PUF to the random query[9], or the response of the PUF to the random query XORed with the string sent by the receiver, depending on whether the bit to be committed is 0 or 1. By extraction independence, it follows that receiver's view in the two cases is identical. To argue binding, we note that the binding argument in Naor's commitment relies only on the expansion property of the PRG. Thus, if we choose the PUF family and a matching fuzzy extractor family of appropriate length, the same argument holds in our case.

The formal description of the protocol $(\mathsf{C}_{\mathsf{uncon}}, \mathsf{R}_{\mathsf{uncon}})$ is given in Fig. 6. In the description of the protocol, the (implicit) security parameter will be denoted by n, and we assume that $\mathcal{F}_{\mathsf{PUF}}$ is parameterized with a PUF family $\mathcal{P}$. Further, the parties also have access to a (m, ℓ, t, ϵ)-fuzzy

[8] Note that however our protocol enjoys parallel composition and reuse of the same PUF, i.e., one can commit to a string reusing the same PUF.

[9] More precisely, the output of the extractor applied to the answer to a random query.

extractor $(\mathsf{FuzGen}, \mathsf{FuzRep})$ of appropriate matching parameters such that $\ell = 3n$. The proof of security is given in Appendix F.

Committer's Input: Bit $b \in \{0,1\}$.
Commitment Phase

$\mathsf{C}_{\mathsf{uncon}} \Rightarrow \mathsf{R}_{\mathsf{uncon}}$: Committer sends $(\mathsf{init}_{\mathsf{PUF}}, \mathtt{normal}, \mathsf{sid}, \mathsf{C}_{\mathsf{uncon}})$ to $\mathcal{F}_{\mathsf{PUF}}$ and obtains response $(\mathsf{initialized}_{\mathsf{PUF}}, \mathsf{sid})$. Committer uniformly selects a query $q \in \{0,1\}^n$ and sends $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{sid}, \mathsf{C}_{\mathsf{uncon}}, q)$ and receives response $(\mathsf{response}_{\mathsf{PUF}}, \mathsf{sid}, q, a)$. Committer obtains $(st, p) \leftarrow \mathsf{FuzGen}(a)$, and sends p to $\mathsf{R}_{\mathsf{uncon}}$. Committer sends $(\mathsf{handover}_{\mathsf{PUF}}, \mathsf{sid}, \mathsf{C}_{\mathsf{uncon}}, \mathsf{R}_{\mathsf{uncon}})$ to $\mathcal{F}_{\mathsf{PUF}}$.

$\mathsf{C}_{\mathsf{uncon}} \Leftarrow \mathsf{R}_{\mathsf{uncon}}$: Receiver receives p' from the committer and $(\mathsf{handover}_{\mathsf{PUF}}, \mathsf{sid}, \mathsf{C}_{\mathsf{uncon}})$ from $\mathcal{F}_{\mathsf{PUF}}$. It uniformly chooses $r \in \{0,1\}^\ell$ and sends it to the committer.

$\mathsf{C}_{\mathsf{uncon}} \Rightarrow \mathsf{R}_{\mathsf{uncon}}$: If $b = 0$, committer sends $y = st$ to the receiver. Else it sends $y = r \oplus st$.

Decommitment Phase

$\mathsf{C}_{\mathsf{uncon}} \Rightarrow \mathsf{R}_{\mathsf{uncon}}$: Committer sends (b, q) to receiver.

$\mathsf{R}_{\mathsf{uncon}}$: Receiver receives (b', q') from the committer and sends $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{sid}, \mathsf{R}_{\mathsf{uncon}}, q')$ to $\mathcal{F}_{\mathsf{PUF}}$ and obtains $(\mathsf{response}_{\mathsf{PUF}}, \mathsf{sid}, q', a')$. It then computes $st' \leftarrow \mathsf{FuzRep}(a', p')$. If $b = 0$, it checks if $st' = y$. Else, it checks if $st' = y \oplus r$. If the check passes, it accepts, else it rejects.

Figure 6: Unconditional Commitment $(\mathsf{C}_{\mathsf{uncon}}, \mathsf{R}_{\mathsf{uncon}})$.

4 Honest PUFs with Oblivious Queries

In this section we continue our foundational study of secure computation in presence of an adversary that misbehave with PUFs. We now present a model that is incomparable to the malicious PUFs model from Section 3, and that moreover is properly stronger[10] than the honest PUF model of Brzuska et al.

The protocols given by Brzuska et al. place another strong restriction on the design of PUFs: the security proofs rely on the simulator's ability to *observe adversary's queries to the PUF*. In real life, this translates to assuming that a party can query the PUF only using the prescribed evaluation procedure. However, current implementations of PUFs do not give any indication why this assumption should be valid.

We augment the UC framework with trusted (honest) PUFs following the spirit of [BFSK11], but we relax the requirement that queries made by an adversary can be observed by a simulator. It is possible that a smart real-world adversary could be able to stimulate the PUF with a physical process that is different from the one prescribed by the PUF designer.

The above point is reinforced by drawing an analogy with the Knowledge of Exponent assumption (KEA) [Dam91, BP04]. Let g be a generator of an "appropriate" group (for details, see [BP04]),

[10] Here by 'stronger' we mean that the real-world adversary has more power

and consider an adversary A that gets as input (g, g^a), and outputs a pair (C, Y). The adversary wins if $C^a = Y$. Roughly, the knowledge of exponent assumption states that the *only* way for the adversary to win is to choose c, and output (g^c, g^{ac}). Or in other words, to win, the adversary *must know* the exponent c. This is formalized by saying that for every adversary, there exists an extractor that can output c. We feel that in the case of PUFs, assuming that the simulator can observe the PUF queries of the adversary is similar in spirit to the KEA (which is a controversial, non-standard and non-falsifiable assumption). Note that the main assumption in KEA is that there is only one way for the adversary to output a winning tuple, and this is debatable. Also, in the case of PUFs, as we have discussed in the previous paragraphs, we can not rule out the existence of malicious procedures for accessing PUFs.

Now, one can object that the above restriction to the simulator is not relevant since we can always have a non-black-box simulator based on the code of the adversary, that by definition knows the querying mechanisms of the adversary. However, this does not help the simulator in the universal composability framework. In the UC framework there is an environment $\mathcal{Z}$ that can not be rewound by the simulator. Technically speaking, in the UC definition, we have the following quantifiers: $\forall A \exists S \forall \mathcal{Z}$, where $\mathcal{Z}$ is the environment. It is therefore possible that an adversary does not know other physical processes to query a PUF, and therefore the simulator does not know any of them either[11]. However the environment is in possession of this knowledge, and can instruct the adversary to query the PUF in some special way getting back the result. To make things more concrete, let us think of a PUF as an exponential size table with challenge-response pairs (x_i, y_i) as its entries (for simplicity, ignore the issue of noisy output for the moment). A PUF comes with a physical procedure that allows a party to stimulate the PUF and read the response y_i corresponding to x_i. However, in real life, there can be super-polynomially many different physical procedures that allow a party to read the same entry (x_i, y_i) from the table. Note that having many different procedures like above does not violate unpredictability. The unpredictability condition says that after reading polynomial number of entries from the table, an unread entry still has sufficient entropy. These different procedures are only different access mechanisms into the table. It does not matter by what *means* the adversary learns the entries of the table, an unread entry still has sufficient entropy.

Now the environment can use any of these super-polynomially many procedures to query the PUF. The simulator, being a PPT machine, can only understand a small fraction of them. Thus, the queries that the environment asks the PUF via the adversary can remain hidden to the simulator. This motivates us to consider a model where the simulator is oblivious to the adversary's queries. Formally, we consider the original PUF ideal functionality of Brzuska et al. [BFSK11], which we call $\mathcal{F}_{\mathsf{hPUF}}$ for honest PUFs. We construct an unconditional UC protocol for Oblivious Transfer functionality in the $\mathcal{F}_{\mathsf{hPUF}}$-hybrid model and show an ideal-model simulator that has the following property: the simulation strategy is *oblivious* to the adversary's PUF queries. A bit more formally, the simulator consists of two algorithms, S_1 and S_2. Simulator S_1 acts as a "wrapper" to S_2 (that is, S_1 executes S_2 internally), and carries out the interaction with the adversary. Whenever S_1 receives a PUF query from the adversary, it queries $\mathcal{F}_{\mathsf{PUF}}$, and returns the response back to the adversary. All other messages from the adversary are sent to S_2. Similarly, whenever S_2 makes a PUF query, S_1 queries $\mathcal{F}_{\mathsf{PUF}}$, and returns the answer to S_2. All other messages by S_2 are sent to the adversary. In this way, S_2 remains oblivious of the adversary's queries, even though S_1 can see the queries. Note that S_1 always performs a fixed function, and its view has no impact on the environment. We

[11] Moreover we can not assume that a simulator has hardwired in all possible physical processes that can be used to query a PUF since there is no evidence that the number of such procedures is polynomially bounded.

call such a simulator $S = (S_1, S_2)$ an *oblivious-query* simulator. The ideal functionality $\mathcal{F}_{\mathsf{hPUF}}$ is described in Fig. 12 in Appendix G.

Before we begin, we address one final technical issue concerning oblivious-query simulators. As the simulator S_2 can not observe the adversary's PUF queries, it must determine its own queries to the PUF only from the transcript (it is clear that the simulator must query the PUF, otherwise we are in the plain model where UC is impossible). Now observe that in the real-world, a party that receives a PUF may certainly *destroy* it after using it. To model this behaviour, we ought to augment our PUF ideal functionality with a kill message, that destroys a PUF so that no party may query that particular PUF anymore. Let us call this the aug-$\mathcal{F}_{\mathsf{hPUF}}$ hybrid model. Apparently, this can be a major problem for the oblivious-query simulator. Consider an adversary the destroys a PUF as soon as it queries it, i.e., before sending any message. Now, by the time the simulator determines what to ask to the PUF, the PUF is already dead and can not be accessed and the simulator is stuck. As such, the oblivious-query PUF model seems to be very demanding and gives the intuition (that we will contradict later) that UC secure computation can not be achieved.

Interestingly, we show a compiler that transforms any protocol in the $\mathcal{F}_{\mathsf{hPUF}}$-hybrid model (where parties can not send a kill message to $\mathcal{F}_{\mathsf{hPUF}}$) to a protocol in the aug-$\mathcal{F}_{\mathsf{hPUF}}$ hybrid model where parties are allowed to destroy PUFs. The point is that it is straight forward for party P_j to check if a party P_i still possesses a PUF that P_j had sent it earlier: before handing over the PUF, party P_j queries the PUF with a random query, say q, and obtains the response, and then hands over the PUF to P_i. When party P_j wishes to check if P_i still possesses the PUF (and has not destroyed it by sending a kill message), party P_j simply sends q to P_i and compares the responses. If P_i is no longer in possession of the PUF (because it had sent a kill message earlier), by the unpredictability of PUFs, it will not be able to give the correct response, and P_j will abort. The compiler works as follows: given any protocol in $\mathcal{F}_{\mathsf{hPUF}}$ hybrid model, the protocol in aug-$\mathcal{F}_{\mathsf{hPUF}}$ follows the same steps as the original protocol, except that after every round of the original protocol, each party verifies that all the recipients of its PUFs are still in possession of those PUFs. Having this compiler in mind, and for the sake of simplicity of notation, we present our protocol in the $\mathcal{F}_{\mathsf{hPUF}}$ hybrid model.

4.1 Unconditional OT in the Oblivious Queries Model

In this section, we provide the formal specification of the protocol described in Section 4.1. Unconditional OT in the oblivious queries model is shown in Fig. 7. We begin by explaining the main ideas of the protocol.

Recall that in the oblivious query model, the simulator can not use the queries that an adversary makes to the PUF. We begin with the original OT protocol of Brzuska et al. [BFSK11], and identify the sections of the protocol where the simulator needs to observe the adversary's queries to extract its input. We then modify these parts by embedding extra information in the transcript which allows extraction without observing the queries. Because of this added information, we also need to add more consistency checks for the final protocol to work. In the following, we give an informal overview of the original protocol of Brzuska et al. [BFSK11], and then describe these steps in more detail.

Overview of the Brzuska et al. [BFSK11] OT protocol. The protocol starts with the receiver initializing a PUF, say $\mathsf{sid}^{\mathsf{R}}$, and querying it on a random query q to obtain response a. The receiver now hands over $\mathsf{sid}^{\mathsf{R}}$ to the sender and can not make any more queries. Now the idea is that the

sender will pick two queries such that the receiver knows the response to only one of them. This is done by the sender picking two random queries x_0, x_1 sending them to the receiver, who responds with $v = x_b \oplus q$, where b is receiver's input. Now, of the two queries $v \oplus x_0$ and $v \oplus x_1$, the receiver knows the response to only one, while the sender has no information about which response the receiver knows. The sender uses the responses to these queries to mask its strings and the receiver can "decrypt" only one of them.

Extracting from Sender without observing queries. This is already possible in the original protocol. Note that the sender masks its strings by responses to the queries $v \oplus x_0$ and $v \oplus x_1$. Both of these queries can be determined from the transcript. The simulator obtains responses to both of these, and thus "decrypts" both strings. We use the same strategy in our protocol.

Extracting from Receiver without observing queries. Consider an adversary that corrupts the receiver. Informally, the simulator in the original protocol of [BFSK11] keeps a list of the queries made by the adversary. When it receives the value v from the adversary, it checks which of $v \oplus x_0$ and $v \oplus x_1$ is in that list[12]. If it is the former, then the adversary's bit is 0, else it is 1. Thus, the simulator relies crucially on observability of queries.

To tackle this, we simply ask the receiver to send, along with v, a query d whose response is the receiver's bit[13]. There are several issues to handle here:

Whose PUF can be used? Note that at the time the receiver sends v to the sender, receiver's PUF is already with the sender. Thus, the query d can not be sent to receiver's PUF anymore, otherwise the sender can evaluate the PUF on d and obtain the receiver's bit. Instead, we make the sender send a PUF, say $\mathsf{sid}^{\mathsf{S}}$, in the beginning of the protocol to the receiver. The receiver queries $\mathsf{sid}^{\mathsf{S}}$ with random queries till it finds a query d whose response is its secret bit. Then it sends d along with v, and because it still holds $\mathsf{sid}^{\mathsf{S}}$, the sender can not query it on d, and receiver's bit is hidden. However, the simulator can query $\mathsf{sid}^{\mathsf{S}}$ with d and extract receiver's bit.

Forcing Receiver to use correct queries: Cut-and-Choose. Of course, a malicious receiver might not use a query d that corresponds to its bit. In this case, the simulator will extract an incorrect bit. However, we can use cut-and-choose to enforce correct behaviour on the receiver. Let k be a statistical security parameter. The sender sends $2k$ PUFs, say $\mathsf{sid}^{\mathsf{S}}_1, \ldots, \mathsf{sid}^{\mathsf{S}}_{2k}$, and $2k$ pairs $(x^0_1, x^1_1), \ldots, (x^0_{2k}, x^1_{2k})$, to the receiver after receiving its PUF $\mathsf{sid}^{\mathsf{R}}$. The receiver chooses *random* bits $b_1, \ldots, b_{2k}$ and prepares v_i and d_i according to b_i (that is, $v_i = q_i \oplus x_{b_i}$ for some query q_i, and the response to d_i of PUF $\mathsf{sid}^{\mathsf{S}}_i$ is the bit b_i.) Now the sender asks the receiver to "reveal" k of these indices chosen at random. To reveal an index i, the receiver sends the query q_i, along with its response (from PUF $\mathsf{sid}^{\mathsf{R}}$) a_i, and also hands over the PUF $\mathsf{sid}^{\mathsf{S}}_i$ back to the sender. The sender first determines the bit b_i from v_i (by checking if $v_i \oplus q_i$ is x^0_i or x^1_i). Then it checks if the response of $\mathsf{sid}^{\mathsf{S}}_i$ matches b_i or not. If the checks pass for a random subset of size k, then the number of indices j where the response of d_j (i.e., response from $\mathsf{sid}^{\mathsf{S}}_j$) *does not* correspond to b_j is very small.

[12] This is a simplification. The simulator actually checks which of $v \oplus x_0$ and $v \oplus x_1$ is within a hamming distance d_{min} of some query in the list.

[13] The response of a PUF is a long string and not a bit, but we can use a suitable Boolean function to map the response to a bit. In our protocol, we use the Parity function for this purpose.

Combining OTs. As above, for a query pair $(v_j \oplus x_j^0, v_j \oplus x_j^1)$, the verifier knows the response to only $v_j \oplus x_j^{b_j}$. Let us call an index j 'bad' if it was not revealed in the cut-and-choose phase, and for which b_j does not correspond with d_j. After the cut-and-choose stage, there are k pairs of queries $(v_j \oplus x_j^0, v_j \oplus x_j^1)$. Now, as in the protocol of Brzuska et al., we would like to use the responses of these queries to mask the sender's strings. The advantage we have over their protocol now is that the simulator has the queries d_j (and access to PUFs $\mathsf{sid}_j^{\mathsf{S}}$), which will potentially reveal the receiver's bit. So the main question is how can we "combine these k OTs", and do so in such a way that the small number of bad indexes left after the cut-and-choose does not matter.

The idea is to use a k-out-of-k additive secret sharing scheme to distribute both of the sender's input strings over these k OTs. Let s_j^0 and s_j^1 be the j^{th} shares of the sender's two strings. The first problem is that receiver runs these k OTs with random selection bits b_js, and thus it is not clear how the sender should distribute the shares over the k OTs. We tackle this by having the receiver send the "correction bit" $b \oplus b_j$, which appears uniformly distributed to the sender. If this bit is 0, the sender uses (s_j^0, s_j^1) as its input in the j^{th} OT, else, if this bit is 1, it flips the order and uses (s_j^1, s_j^0) in the j^{th} OT. It is easy to see that an honest receiver gets all k shares of the string it wants.

The simulator follows the strategy outlined earlier: it uses d_js to compute b_js, XORs it with the correction bit to obtain the bit $\hat{b}'_j$. Then it takes the majority of the bits $\hat{b}'_j$, and uses that as receiver's extracted input. Note that if index j is not bad, then $\hat{b}'_j = b$. We show that if the cut-and-choose succeeds, then the probability that the number of bad indexes is greater than $k/10$ is negligible. Thus, with all but negligible probability, the majority of the bits $\hat{b}'_j$ will be the reciever's actual input, b.

The formal description of our protocol is given in Figure 7. The proof of the following theorem appears in Appendix E.

Theorem 2. *Protocol* $(\mathsf{S}_{\mathsf{uncOT}}, \mathsf{R}_{\mathsf{uncOT}})$ *depicted in Fig. 7, UC-realizes the* $\mathcal{F}_{\mathsf{OT}}$ *functionality in the* $\mathcal{F}_{\mathsf{hPUF}}$*-hybrid model.*

5 Impossibility of UC Security in the Malicious PUFs + Oblivious Queries Model

We have shown the feasibility of UC secure computation under computational assumptions in the malicious PUFs model and of unconditional UC secure computation in the oblivious queries model. A natural question is whether one can achieve UC security in a model that combines both models, i.e., when the adversary can create malicious PUFs *and* obtain challenge/response pairs from the PUF without querying the PUF in the prescribed way. We said that in the malicious PUFs model the power of the simulator is to see queries made by the adversary to the honest PUFs and their answers, and to have permanent access (i.e., the simulator can always query the PUFs) to all PUFs present in the system. In the oblivious queries model, those challenge/response pairs are not available to the simulator. Therefore, when considering the combination of the two models, the only power left to the simulator is the permanent access to the PUFs present in the system. In fact, as shown in Lemma 1, the only power left is the permanent access to PUFs generated by the honest party only. We show that, under the assumption that the adversary can construct malicious PUFs that he can fully predict, UC security in the malicious PUFs + oblivious queries model is impossible. Since

Sender's Input: Strings $s^0, s^1 \in \{0,1\}^n$.
Receiver's Input: Bit $b \in \{0,1\}$.

1. **[($\mathsf{S}_{\mathsf{uncOT}} \Rightarrow \mathsf{R}_{\mathsf{uncOT}}$): Sender PUF initialization]** S initializes $2k$ PUFs $\mathsf{sid}_1^{\mathsf{S}}, \ldots, \mathsf{sid}_{2k}^{\mathsf{S}}$ and sends them to R.
2. **[($\mathsf{S}_{\mathsf{uncOT}} \Leftarrow \mathsf{R}_{\mathsf{uncOT}}$): Receiver PUF initialization]** R initializes a PUF $\mathsf{sid}^{\mathsf{R}}$. It uniformly chooses $2k$ queries $q_1, \ldots, q_{2k}$ and obtains responses $a_1, \ldots, a_{2k}$. It sends the PUF $\mathsf{sid}^{\mathsf{R}}$ to S.
3. **[Cut-and-Choose]**
   (a) ($\mathsf{S}_{\mathsf{uncOT}} \Rightarrow \mathsf{R}_{\mathsf{uncOT}}$)For $1 \leq i \leq 2k$, sender uniformly selects a pair of queries (x_i^0, x_i^1) and sends it to R.
   (b) ($\mathsf{S}_{\mathsf{uncOT}} \Leftarrow \mathsf{R}_{\mathsf{uncOT}}$)For each $1 \leq i \leq 2k$, receiver does the following:
   - select random bit $b_i \in \{0,1\}$.
   - select random query d_i and let da_i be the response of the PUF $\mathsf{sid}_i^{\mathsf{S}}$. Compute $(dst_i, dp_i) \leftarrow \mathsf{FuzGen}(da_i)$. If $\mathsf{Parity}(dst_i) \neq b_i$, repeat this step. Else, continue.
   - compute $v_i := x_i^{b_i} \oplus q_i$.

   For each $1 \leq i \leq 2k$ receiver sends to sender (v_i, d_i, dp_i).
   (c) ($\mathsf{S}_{\mathsf{uncOT}} \Rightarrow \mathsf{R}_{\mathsf{uncOT}}$) Sender selects a random subset $S \subset [2k]$ of size k and sends it to receiver.
   (d) ($\mathsf{S}_{\mathsf{uncOT}} \Leftarrow \mathsf{R}_{\mathsf{uncOT}}$) For all $j \in S$, receiver sends (q_j, a_j) to sender, and also hands over the PUF $\mathsf{sid}_j^{\mathsf{S}}$ to the sender.
   (e) Sender makes the following checks for each $j \in S$:
   - compute the response of PUF $\mathsf{sid}^{\mathsf{R}}$ on query q_j to obtain a_j^*; if $\mathsf{dis}(a_j, a_j^*) > d_{\mathsf{noise}}$, abort.
   - if $v_j \oplus q_j = x_j^0$, set $b_j^* = 0$; if $v_j \oplus q_j = x_j^1$, set $b_j^* = 1$; else abort.
   - query the PUF $\mathsf{sid}_j^{\mathsf{S}}$ with d_j to obtain response da_j^*; if $\mathsf{Parity}(\mathsf{FuzRep}(da_j^*, dp_j)) \neq b_j^*$, abort.
4. **[($\mathsf{S}_{\mathsf{uncOT}} \Leftarrow \mathsf{R}_{\mathsf{uncOT}}$): Receiver sends correction-bits]** Let $i_1, \ldots, i_k$ be the indices *not* in S. For $1 \leq j \leq k$, receiver sends to sender the bit $b'_{i_j} = b_{i_j} \oplus b$.
5. **[($\mathsf{S}_{\mathsf{uncOT}} \Rightarrow \mathsf{R}_{\mathsf{uncOT}}$): Sender's final message]** Sender prepares its final message as follows:
   - for $\delta = 0, 1$, choose random strings $s_1^\delta, \ldots, s_k^\delta$ such that $s^\delta = \bigoplus_{j=1}^k s_j^\delta$.
   - for $\delta = 0, 1$, for $1 \leq j \leq k$, compute $\hat{q}_{i_j}^\delta = v_{i_j} \oplus x_{i_j}^\delta$ and let $(st_{i_j}^\delta, p_{i_j}^\delta)$ be the output of the fuzzy extractor applied to the response of PUF $\mathsf{sid}^{\mathsf{R}}$ to query $\hat{q}_{i_j}^\delta$.
   - for $\delta = 0, 1$ and $1 \leq j \leq k$, set $m_{i_j}^\delta = s_j^\delta \oplus st_{i_j}^{b'_{i_j} \oplus \delta}$.

   Sender sends $(m_{i_1}^0, m_{i_1}^1), \ldots, (m_{i_k}^0, m_{i_k}^1)$ and $(p_{i_1}^0, p_{i_1}^1), \ldots, (p_{i_k}^0, p_{i_k}^1)$ to the receiver.
6. **[Receiver's final step]** For $1 \leq j \leq k$, receiver computes $st_{i_j} \leftarrow \mathsf{FuzRep}(p_{i_j}^{b_{i_j}}, a_{i_j})$. It outputs $s^b = \bigoplus_{j=1}^k (m_{i_j}^b \oplus st_{i_j})$.

Figure 7: Unconditional OT protocol $(\mathsf{S}_{\mathsf{uncOT}}, \mathsf{R}_{\mathsf{uncOT}})$ in the Oblivious Queries Model.

relying on the fact that an adversary can not create such PUFs is a strong assumption, we see our impossibility as a concrete negative result.

Definition 5. *A fully predictable malicious PUF is a physical device that is not unpredictable to the creator. For any challenge, the creator is able to compute in polynomial time an answer that is identical to the answer computed by the device, without having physical access to it.*

Theorem 3. *There exists no UC-secure Bit Commitment Scheme in the malicious PUFs + oblivious queries model, if the adversary can generate PUFs that he can fully predict.*

Proof. The proof is by contradiction. Assume that there exists a bit commitment protocol (S, R) for functionality $\mathcal{F}_{\mathsf{com}}$ that is *UC-secure* in the malicious PUFs and oblivious queries model. This implies the existence of a straight-line simulator $\mathsf{Sim} = (\mathsf{Sim}_S, \mathsf{Sim}_R)$ satisfying two properties. First, it provides a transcript that is indistinguishable from the transcript of the real protocol execution; second, it extracts the input of the malicious sender S^* in straight-line. Specifically, Sim_S corrupts party S in the ideal world, it runs the real-world (possibly dummy) adversary S^* as sub-routine and emulates the execution of the real-world protocol simulating the honest party R to S^*. Sim_S extracts the input that S^* uses in the protocol execution in straight-line, without rewinding and without using its code. The extracted input is given to the ideal functionality $\mathcal{F}_{\mathsf{com}}$.

Given Sim_S, one can construct an adversary R^* that extracts the input of an honest S. R^* starts a protocol with S. Concurrently R^* runs the simulator Sim_S as sub-routine and forwards all the messages from Sim_S to S and vice versa. R^* also simulates functionalities $\mathcal{F}_{\mathsf{PUF}}$ and $\mathcal{F}_{\mathsf{com}}$ to Sim_S. This means that R^* intercepts the queries made by Sim_S to the ideal functionalities and it simulates the answers. At some point during the protocol the simulator extracts the input of the adversary R^* that in this case is the honest S and sends it to the functionality $\mathcal{F}_{\mathsf{com}}$. Therefore, R^* obtains S's secret input and security is violated.

R^* can carry out such attack against S for the following reasons.

- Due to the UC-security it follows that Sim_S is straight-line. It does not rewind S^* and it does not needs its code.
- Due to the oblivious queries model, Sim_S carries out the simulation without seeing the queries that S^* makes to the PUFs sent by R.

But, by definition Sim_S has *permanent* access to all PUFs in the system. Formally this is equivalent to say that Sim_S can make Eval queries to $\mathcal{F}_{\mathsf{PUF}}$ even if the PUF has not been "handover" to the ideal world adversary. Instead, R^* has access to S's PUFs (and her own PUFs) only when they are physically in her hands. Formally this means that the adversary can make Eval queries to $\mathcal{F}_{\mathsf{PUF}}$ only when the PUF is handover to her. Thus, in order for the above attack to be successful, R^* has to simulate the permanent access to all PUFs, (i.e., simulate Eval queries to $\mathcal{F}_{\mathsf{PUF}}$) even when R^* is not allowed to make such queries. This can be done due to the following. For simplicity in the following we talk about queries to PUFs instead of invoking Eval to the $\mathcal{F}_{\mathsf{PUF}}$ functionality.

- Access to PUFs created by S must not be simulated. Due to Lemma 1, in the malicious PUFs and malicious queries model, the simulator queries a PUF sent by the adversary, only when the PUF is in the hands of the honest party. In our case, Sim_S will access to PUFs sent by S only when it is in R^*'s hands.

- Access to PUFs created by R^* can be simulated due to the fact that they are fully predictable. Thus R^* is able to answer to the queries made by Sim_S to R's PUFs, anytime during the simulation.

Hence, given any UC-simulator in the malicious PUF and malicious queries model, it is possible to construct an adversary that extracts the input of the honest party by using such simulator as sub-routine. Therefore, there exists no UC-secure protocol in such a model.

□

Lemma 1. *In the malicious PUFs and malicious queries model, the UC-simulator has access to malicious PUFs only when they are in the hands of the honest party.*

Proof. Assume not, then there exists a UC simulator that, in order to successfully carry out the simulation, it needs to access to a malicious PUF when it is in the hands of the malicious party. Note that the malicious party could simply destroy the PUF as soon as it goes back in her hands. More generally, since the PUF is maliciously generated, the adversary can always deactivate the PUF when it is in her hands, and re-activate it when it is sent to the honest party. Thus, any simulator that needs to query the malicious PUF when it is in the adversary's hand, will fail in obtaining the answer, and completing the simulation. Hence, it must be the case that any simulator uses the access to a malicious PUF only when it is with the honest party. □

Acknowledgments

The authors thank Margarita Vald for pointing out the problem with the original formulation of the [BFSK11] PUF ideal functionality where the adversary is not allowed to create PUFs. The authors also thank Marten van Dijk and Ulrich Rührmair for extensive and insightful discussions on their and our paper.

Research supported in part by NSF grants CCF-0916574; IIS-1065276; CCF-1016540; CNS-1118126; CNS-1136174; US-Israel BSF grant 2008411, OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award, Lockheed-Martin Corporation Research Award and the European Commission through the FP7 programme under contract 216676 ECRYPT II. This material is also based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0392. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

[AMS+09] Frederik Armknecht, Roel Maes, Ahmad-Reza Sadeghi, Berk Sunar, and Pim Tuyls. Memory leakage-resilient encryption based on physically unclonable functions. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 685–702. Springer, 2009.

[AMS+11] Frederik Armknecht, Roel Maes, Ahmad-Reza Sadeghi, Francois-Xavier Standaert, and Christian Wachsmann. A formalization of the security features of physical functions. In *IEEE Symposium on Security and Privacy*, pages 397–412. IEEE Computer Society, 2011.

[BCNP04] Boaz Barak, Ron Canetti, Jesper B. Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *Foundations of Computer Science (FOCS'04)*, pages 394–403, 2004.

[BCR86] Gilles Brassard, Claude Crépeau, and Jean-Marc Robert. Information theoretic reductions among disclosure problems. In *FOCS*, pages 168–173. IEEE Computer Society, 1986.

[BFSK11] Christina Brzuska, Marc Fischlin, Heike Schröder, and Stefan Katzenbeisser. Physically uncloneable functions in the universal composition framework. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 51–70. Springer, 2011.

[BP04] Mihir Bellare and Adriana Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In *CRYPTO*, pages 273–289, 2004.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science (FOCS'01)*, pages 136–145, 2001.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Berlin, Germany.

[CGS08] Nishanth Chandran, Vipul Goyal, and Amit Sahai. New constructions for UC secure computation using tamper-proof hardware. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 545–562, Istanbul, Turkey, 2008. Springer, Berlin, Germany.

[CKL03] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 68–86, Warsaw, Poland, May 4–8, 2003. Springer, Berlin, Germany.

[CKS+11] Seung Geol Choi, Jonathan Katz, Dominique Schröder, Arkady Yerukhimovich, and Hong-Sheng Zhou. (efficient) universally composable two-party computation using a minimal number of stateless tokens. *IACR Cryptology ePrint Archive*, 2011:689, 2011.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing*, Lecture Notes in Computer Science, pages 494–503, Montréal, Québec, Canada, May 19–21, 2002. ACM Press.

[CO99] Giovanni Di Crescenzo and Rafail Ostrovsky. On concurrent zero-knowledge with preprocessing. In *CRYPTO*, pages 485–502, 1999.

[Dam91] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In *CRYPTO*, pages 445–456, 1991.

[DKMQ11] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. Unconditional and composable security using a single stateful tamper-proof hardware token. In Yuval Ishai, editor, *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 164–181. Springer, 2011.

[DKMQ12] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. David & goliath oblivious affine function evaluation - asymptotically optimal building blocks for universally composable two-party computation from a single untrusted stateful tamper-proof hardware token. Cryptology ePrint Archive, Report 2012/135, 2012. `http://eprint.iacr.org/`.

[DORS08] 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.

[EKvdL11] Ilze Eichhorn, Patrick Koeberl, and Vincent van der Leest. Logically reconfigurable pufs: memory-based secure key storage. In *Proceedings of the sixth ACM workshop on Scalable trusted computing*, STC '11, pages 59–64, New York, NY, USA, 2011. ACM.

[GIS+10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 308–326, Zurich, Switzerland, February 9–11, 2010. Springer, Berlin, Germany.

[GKR08] Shafi Goldwasser, Yael T. Kalai, and Guy. N. Rothblum. One-time programs. In *Advances in Cryptology – CRYPTO'08*, volume 5157 of *Lecture Notes in Computer Science*, pages 39–56. Springer, Berlin, Germany, 2008.

[GKST07] Jorge Guajardo, Sandeep S. Kumar, Geert Jan Schrijen, and Pim Tuyls. Fpga intrinsic pufs and their use for ip protection. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 63–80. Springer, 2007.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. *SICOMP*, 18(6):186–208, 1989.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, Cambridge, UK, 2001.

[HL10] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols Techniques and Constructions*. Springer, 2010.

[Kat07] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 115–128, Barcelona, Spain, May 20–24, 2007.

[KKR+12] Stefan Katzenbeisser, Ünal Koccabas, Vladimir Rozic, Ahmad-Reza Sadeghi, Ingrid Verbauwhede, and Christian Wachsmann. Pufs: Myth, fact or busted? a security evaluation of physically unclonable functions (pufs) cast in silicon. In Prouff and Schaumont [PS12], pages 283–301.

[KLP05] Yael Tauman Kalai, Yehuda Lindell, and Manoj Prabhakaran. Concurrent general composition of secure protocols in the timing model. In *37th Annual ACM Symposium on Theory of Computing*, pages 644–653, 2005.

[KSWS11] Ünal Koccabas, Ahmad-Reza Sadeghi, Christian Wachsmann, and Steffen Schulz. Poster: practical embedded remote attestation using physically unclonable functions. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 797–800. ACM, 2011.

[MHV12] Roel Maes, Anthony Van Herrewege, and Ingrid Verbauwhede. Pufky: A fully functional puf-based cryptographic key generator. In Prouff and Schaumont [PS12], pages 302–319.

[MS08] Tal Moran and Gil Segev. David and Goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 527–544, Istanbul, Turkey, April 13–17, 2008. Springer, Berlin, Germany.

[MV10] Roel Maes and Ingrid Verbauwhede. Physically unclonable functions: A study on the state of the art and future research directions. In Ahmad-Reza Sadeghi and David Naccache, editors, *Towards Hardware-Intrinsic Security*, Information Security and Cryptography, pages 3–37. Springer Berlin Heidelberg, 2010.

[Nao89] Moni Naor. Bit commitment using pseudo-randomness. In *CRYPTO*, pages 128–136, 1989.

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *CRYPTO*, pages 111–126, 2002.

[OVY93] Rafail Ostrovsky, Ramarathnam Venkatesan, and Moti Yung. Interactive hashing simplifies zero-knowledge protocol design. In Tor Helleseth, editor, *EUROCRYPT*, volume 765 of *Lecture Notes in Computer Science*, pages 267–273. Springer, 1993.

[Pap01] Ravikanth Srinivasa Pappu. *Physical One-Way Functions*. PhD thesis, MIT, 2001.

[PRTG02] Ravikanth S. Pappu, Ben Recht, Jason Taylor, and Niel Gershenfeld. Physical one-way functions. *Science*, 297:2026–2030, 2002.

[PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: achieving universal composability without trusted setup. In *36th Annual ACM Symposium on Theory of Computing*, pages 242–251, 2004.

[PS12] Emmanuel Prouff and Patrick Schaumont, editors. *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*. Springer, 2012.

[PW08] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In Cynthia Dwork, editor, *STOC*, pages 187–196. ACM, 2008.

[RKB10] Ulrich Rührmair, Stefan Katzenbeisser, and H. Busch. Strong pufs: Models, constructions and security proofs. In A. Sadeghi and P. Tuyls, editors, *Towards Hardware Intrinsic Security: Foundations and Practice*, pages 79–96. Springer, 2010.

[Rüh10] Ulrich Rührmair. Oblivious transfer based on physical unclonable functions. In Alessandro Acquisti, Sean W. Smith, and Ahmad-Reza Sadeghi, editors, *TRUST*, volume 6101 of *Lecture Notes in Computer Science*, pages 430–440. Springer, 2010.

[RvD12] Ulrich Rührmair and Marten van Dijk. Practical security analysis of puf-based two-player protocols. In Prouff and Schaumont [PS12], pages 251–267.

[SVW10] Ahmad-Reza Sadeghi, Ivan Visconti, and Christian Wachsmann. Enhancing rfid security and privacy by physically unclonable functions. In Ahmad-Reza Sadeghi and David Naccache, editors, *Towards Hardware-Intrinsic Security*, Information Security and Cryptography, pages 281–305. Springer Berlin Heidelberg, 2010.

[TB06] Pim Tuyls and Lejla Batina. Rfid-tags for anti-counterfeiting. In David Pointcheval, editor, *CT-RSA*, volume 3860 of *Lecture Notes in Computer Science*, pages 115–131. Springer, 2006.

[Val12] Margarita Vald. Private Communication, 2012.

[vDR12] Marten van Dijk and Ulrich Rührmair. Physical unclonable functions in cryptographic protocols: Security proofs and impossibility results. *IACR Cryptology ePrint Archive*, 2012:228, 2012.

[WW06] Stefan Wolf and Jürg Wullschleger. Oblivious transfer is symmetric. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 222–232. Springer, 2006.

A Missing Definitions and Tools

Notation. A function ϵ is negligible in n (or just negligible) if for every polynomial $p(\cdot)$ there exists a value $n_0 \in \mathbb{N}$ such that for all $n > n_0$ it holds that $\epsilon(n) < 1/p(n)$. The *view* of A in interaction $(A(a), B(b))(x)$ is denoted by $\mathsf{view}_A(A(a), B(b))(x)$ and consists of the common input x, the private input a, the private random tape of A, and all the messages received by A during the course of the interaction. For two random variables X and Y with supports in $\{0,1\}^n$, the *statistical difference* between X and Y, denoted by $SD(X,Y)$, is defined as, $SD(X,Y) = \frac{1}{2}\sum_{z\in\{0,1\}^n} |\Pr[\,X = z\,] - \Pr[\,Y = z\,]|$.

In the following definitions we assume that parties are stateful and that malicious parties obtain auxiliary inputs, although for better readability we omit them.

Indistinguishability. Let $\mathcal{W}$ be a set of strings. An *ensemble* of random variables $X = \{X_w\}_{w\in\mathcal{W}}$ is a sequence of random variables indexed by elements of $\mathcal{W}$.

Definition 6. *Two ensembles of random variables* $X = \{X_w\}_{w\in\mathcal{W}}$ *and* $Y = \{Y_w\}_{w\in\mathcal{W}}$ *are* computationally indistinguishable *(resp.,* statistically indistinguishable*), i.e.,* $\{X_w\}_{w\in\mathcal{W}} \stackrel{\mathsf{C}}{\equiv} \{Y_w\}_{w\in\mathcal{W}}$ *(resp.,* $\{X_w\}_{w\in\mathcal{W}} \stackrel{\mathsf{S}}{\equiv} \{Y_w\}_{w\in\mathcal{W}}$*) if for any polynomial-sized circuit (resp., unbounded)* D *there exists a negligible function* ϵ *such that*

$$\left|\Pr\left[\,\alpha \leftarrow X_w : D(w,\alpha) = 1\,\right] - \Pr\left[\,\alpha \leftarrow Y_w : D(w,\alpha) = 1\,\right]\right| < \epsilon(w).$$

A.1 Commitment Schemes

Definition 7 (Bit Commitment Scheme)**.** *A commitment scheme is a tuple of PPT algorithms* $\mathsf{Com} = (\mathsf{C}, \mathsf{R})$ *implementing the following two-phase functionality. Given to* C *an input* $b \in \{0,1\}$*, in the first phase (*`commitment phase`*)* C *interacts with* R *to commit to the bit* b*, we denote this interaction as* $((c,d),c) \leftarrow \langle \mathsf{C}(\mathtt{com}, b), \mathsf{R}(\mathtt{recv})\rangle$ *where* c *is the transcript of the commitment phase and* d *is the decommitment. In the second phase (*`opening phase`*)* C *sends* (b,d) *and* R *finally accepts or rejects according to* (c,b,d)*.*

$\mathsf{Com} = (\mathsf{C}, \mathsf{R})$ *is a commitment scheme if it satisfies the following properties.*

Completeness. *If* C *and* R *follow their prescribed strategy then* R *will always accept the commitment and the decommitment (with probability 1).*

Computational Hiding. *For every PPT* R^* *the ensembles* $\{\mathsf{view}_{\mathsf{R}^*}\ (\mathsf{C}(\mathtt{com}, 0),\ \mathsf{R}^*)\ (1^n)\}_{n\in\mathbb{N}}$ *and* $\{\mathsf{view}_{\mathsf{R}^*}(\mathsf{C}(\mathtt{com},\ 0),\ \mathsf{R}^*)\ (1^n)\}_{n\in\mathbb{N}}$ *are computationally indistinguishable, where* $\mathsf{view}_{\mathsf{R}^*}\ (\mathsf{C}(\mathtt{com}, b),\ \mathsf{R}^*)$ *denotes the view of* R^* *in the commit stage interacting with* $\mathsf{C}(\mathtt{com}, b)$*.*

Computational Binding. *For every PPT* C^**, there exists a negligible function* ϵ *such that the malicious sender* C^* *succeeds in the following game with probability at most* $\epsilon(n)$*: On security parameter* 1^n*,* C^* *interacts with* R *in the commit stage obtaining the transcript* c *. Then* C^* *outputs pairs* $(0, d_0)$ *and* $(1, d_1)$*, and succeeds if in the opening phase,* $\mathsf{R}(0, d_0, c) = \mathsf{R}(1, d_1, c) =$ *accept.*

When hiding (resp., binding) holds even against an unbounded adversary, then the commitment scheme enjoys statistical hiding (resp., binding).

It will be helpful to consider commitment schemes in which the committer and receiver take an additional common input, denoted by σ. This additional common input is drawn fresh for each execution from a specified distribution. In our case, this additional common input is always drawn from the uniform distribution of appropriate length. We will denote such commitment schemes with $\mathsf{Com} = (\mathsf{C}, \mathsf{R})(\sigma)$. The properties of Definition 7 are required to hold over the random choice of σ.

Definition 8 (Straight-line Equivocal Commitment Scheme.)**.** *A commitment scheme* $\mathsf{Com} = (\mathsf{C}, \mathsf{R})(\sigma)$ *with common input* σ*, is a* straight-line equivocal commitment scheme *if there exists a straight-line strict polynomial-time simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ *such that for any* $b \in \{0,1\}$*, and for all PPT* R^**, the output of the following two experiments is computationally indistinguishable:*

Experiment $\mathbf{Exp}^{\mathsf{C}}_{\mathsf{R}^*}(n)$:	*Experiment* $\mathbf{Exp}^{\mathcal{S}}_{\mathsf{R}^*}(n)$:
$\sigma \stackrel{\$}{\leftarrow} \{0,1\}^{\ell(n)}$;	$(\tilde{\sigma}, \mathsf{state}_1) \stackrel{\$}{\leftarrow} \mathcal{S}_1(1^n)$;
$((c,d),c) \leftarrow \langle \mathsf{C}(\mathtt{com}, b), \mathsf{R}^*(\mathtt{recv})\rangle(\sigma)$;	$(\mathsf{state}_2, \tilde{c}) \leftarrow \langle \mathcal{S}_2(\mathsf{state}_1), \mathsf{R}^*(\mathtt{recv})\rangle(\tilde{\sigma})$;
`return` $\mathsf{R}^*(\sigma, b, c, d)\rangle$;	$\tilde{d} \leftarrow \mathcal{S}_3(\sigma, \mathsf{state}_2, b)$; `return` $\mathsf{R}^*(\tilde{\sigma}, \tilde{c}, b, \tilde{d})$;

Note that in this definition, the verification of the receiver R is computed by using also the common input. Further, this definition can easily be extended to the setting where all parties have access to PUFs.

Definition 9 (Straight-line Extractable Commitment Scheme in the Malicious PUF model)**.** *A commitment scheme* $\mathsf{Com} = (\mathsf{C}, \mathsf{R})$ *is a* straight-line extractable commitment scheme in the malicious PUF model *if there exists a straight-line strict polynomial-time extractor* E *that, having on-line access to the queries made by any PPT malicious committer* C^* *to the PUFs sent by the honest receiver* R*, and running only the commitment phase, it outputs a bit* $b^\star$ *or the special symbol* $\perp$ *such that:*

- *(simulation) the views* $\mathsf{view}_{\mathsf{C}^*}(\mathsf{C}^*(\mathtt{com}, \star), \mathsf{R}(\mathtt{recv}))$ *and* $\mathsf{view}_{\mathsf{C}^*}(\mathsf{C}^*(\mathtt{com}, \star), \mathsf{E})$ *are identical;*
- *(extraction) let* c *be the transcript obtained from the commitment phase run between* C^* *and* E*. If* E *outputs* $\perp$ *then the probability that* C^* *will provide an accepting decommitment is negligible.*
- *(binding) if* $b^\star \neq \perp$ *the probability that* C^* *decommit to a bit* $b \neq b^\star$ *is negligible.*

Remark 1. *The standard definition of extractable commitments in the plain model requires that, if the* commitment *is accepting then, probability that the extractor fails and outputs* $\perp$*, is negligible. Our definition is in the* $\mathcal{F}_{\mathsf{PUF}}$*-hybrid model, and straight-line extractability is achieved using access to* $\mathcal{F}_{\mathsf{PUF}}$*. Here, we require that, if the* decommitment *is accepting, then probability that the extractor fails and output* $\perp$ *is negligible. Therefore, to establish that an extractor fails, we have to consider the probability of success in the decommitment phase. To see why this definition is necessary, consider the following protocol. To commit to a bit b, the committer sends COM(b) to the receiver, then it queries a PUF (received from the receiver) with the opening of COM(b), and finally it commits to the response received from such PUF. Then in the decommitment phase, the committer has to open both commitments, and send the PUF back to the receiver, who accepts iff the openings are accepting and the response committed by the committer corresponds to the response of the PUF on input the opening of COM(b). In this case, a committer can always provide an accepting commitment, without querying the PUF (making the extractor output* $\perp$*). Indeed, it can just commit to junk. However, in the decommitment phase, the committer will not be able to open to the correct response, and the receiver will not accept. In such case, the extractor did not fail, since the committer did not actually commit to any value that could have been opened.*

A.2 Statistical Zero-Knowledge Argument of Knowledge

A polynomial-time relation R is a relation for which it is possible to verify in time polynomial in $|x|$ whether $R(x, w) = 1$. Let us consider an **NP**-language L and denote by R_L the corresponding polynomial-time relation such that $x \in L$ if and only if there exists w such that $R_L(x, w) = 1$. We will call such a w a *valid witness for* $x \in L$. We will denote by $\mathrm{Prob}_r[\ X\ ]$ the probability of an event X over coins r.

Interactive proof/argument systems with efficient prover strategies. An *interactive proof (resp., argument) system* for a language L is a pair of probabilistic polynomial-time interactive algorithms P and V, satisfying the requirements of *completeness* and *soundness*. Informally, completeness requires that for any $x \in L$, at the end of the interaction between P and V, where P has as input a valid witness for $x \in L$, V rejects with negligible probability. Soundness requires that for any $x \notin L$, for any (resp., any polynomial-sized) circuit P^*, at the end of the interaction between P^* and V, V accepts with negligible probability. We denote by $\mathsf{out}(\langle P(w), V\rangle(x))$ the output of

the verifier V when interacting on common input x with prover P that also receives as additional input a witness w for $x \in L$. Moreover we denote by $\mathsf{out}(\langle P^*, V\rangle(x))$ the output of the verifier V when interacting on common input x with an adversarial prover P^*.

Formally, we have the following definition.

Definition 10. *A pair of interactive algorithms $\langle P(\cdot), V(\cdot)\rangle(\cdot)$ is an* interactive proof (resp., argument) system *for the language L, if V runs in probabilistic polynomial-time and*

1. *Completeness: For every $x \in L$, $|x| = n$, and for every* **NP** *witness w for $x \in L$*

$$\Pr\left[\, \mathsf{out}(\langle P(w), V\rangle(x) = 1 \,\right] = 1.$$

2. *Soundness (resp. computational soundness): For every (resp., every polynomial-sized) circuit family $\{P_n^*\}_{n\in N}$ there exists a negligible function $\epsilon(\cdot)$ such that*

$$\Pr\left[\, \mathsf{out}(\langle P_n^*, V\rangle(x)) = 1 \,\right] < \epsilon(|x|).$$

for every $x \notin L$ of size n.

Arguments of knowledge. Informally, an argument system is an argument of knowledge if for any probabilistic polynomial-time interactive algorithm P^* there exists a probabilistic algorithm called the extractor, such that 1) the expected running time of the extractor is polynomial-time regardless of the success probability of P^*; 2) if P^* has non-negligible probability of convincing a honest verifier for proving that $x \in L$, where L is an **NP** language, then the extractor with overwhelming probability outputs a valid witness for $x \in L$.

Zero knowledge. The classical notion of zero knowledge has been introduced in [GMR89]. In a zero-knowledge argument system a prover can prove the validity of a statement to a verifier without releasing any additional information. This concept is formalized by requiring the existence of an expected polynomial-time algorithm, called the *simulator*, whose output is indistinguishable from the view of the verifier.

Definition 11. *An interactive argument system $\langle P(\cdot,\cdot), V(\cdot)\rangle$ for a language L is computational (resp., statistical, perfect)* zero-knowledge *if for all polynomial-time verifiers V^*, there exists an expected polynomial-time algorithm S such that the following ensembles are computationally (resp., statistically, perfectly) indistinguishable:*

$$\mathsf{view}_{V^*}((P(w), V^*(z))(x))_{x\in L, w\in W(x), z\in\{0,1\}^*} \text{ and } \{\mathsf{S}(x,z)\}_{x\in L, z\in\{0,1\}^*}.$$

A.3 The UC Framework and the Ideal Functionalities

For simplicity, we define the two-party protocol syntax, and then informally review the two-party UC-framework, which can be extended to the multi-party case. For more details, see [Can01].

Protocol syntax. Following [GMR89] and [Gol01], a protocol is represented as a system of probabilistic interactive Turing machines (ITMs), where each ITM represents the program to be run within a different party. Specifically, the input and output tapes model inputs and outputs that are received from and given to other programs running on the same machine, and the communication

tapes model messages sent to and received from the network. Adversarial entities are also modeled as ITMs.

The construction of a protocol in the UC-framework proceeds as follows: first, an *ideal functionality* is defined, which is a "trusted party" that is guaranteed to accurately capture the desired functionality. Then, the process of executing a protocol in the presence of an adversary and in a given computational environment is formalized. This is called the *real-life* model. Finally, an *ideal process* is considered, where the parties only interact with the ideal functionality, and not amongst themselves. Informally, a protocol realizes an ideal functionality if running of the protocol amounts to "emulating" the ideal process for that functionality.

Let $\Pi = (P_1, P_2)$ be a protocol, and $\mathcal{F}$ be the ideal-functionality. We describe the ideal and real world executions.

The real-life process. The real-life process consists of the two parties P_1 and P_2, the environment $\mathcal{Z}$, and the adversary $\mathcal{A}$. Adversary $\mathcal{A}$ can communicate with environment $\mathcal{Z}$ and can corrupt any party. When $\mathcal{A}$ corrupts party P_i, it learns P_i's entire internal state, and takes complete control of P_i's input/output behavior. The environment $\mathcal{Z}$ sets the parties' initial inputs. Let $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}$ be the distribution ensemble that describes the environment's output when protocol Π is run with adversary $\mathcal{A}$.

We also consider a *$\mathcal{G}$-hybrid model*, where the real-world parties are additionally given access to an ideal functionality $\mathcal{G}$. During the execution of the protocol, the parties can send inputs to, and receive outputs from, the functionality $\mathcal{G}$. We will use $\mathsf{REAL}^{\mathcal{G}}_{\Pi,\mathcal{A},\mathcal{Z}}$ to denote the distribution of the environment's output in this hybrid execution.

The ideal process. The ideal process consists of two "dummy parties" $\hat{P}_1$ and $\hat{P}_2$, the ideal functionality $\mathcal{F}$, the environment $\mathcal{Z}$, and the ideal world adversary Sim, called the simulator. In the ideal world, the uncorrupted dummy parties obtain their inputs from environment $\mathcal{Z}$ and simply hand them over to $\mathcal{F}$. As in the real world, adversary Sim can corrupt any party. Once it corrupts party $\hat{P}_i$, it learns $\hat{P}_i$'s input, and takes complete control of its input/output behavior. Let $\mathsf{IDEAL}^{\mathcal{F}}_{\mathsf{Sim},\mathcal{Z}}$ be the distribution ensemble that describes the environment's output in the ideal process.

Definition 12. *(UC-Realizing an Ideal Functionality) Let $\mathcal{F}$ be an ideal functionality, and Π be a protocol. We say that Π UC-**realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model** if for any hybrid-model PPT adversary $\mathcal{A}$, there exists an ideal process expected PPT adversary Sim such that for every PPT environment $\mathcal{Z}$, it holds that:*

$$\{\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}(n,z)\}_{n\in\mathbb{N},z\in\{0,1\}^*} \sim \{\mathsf{REAL}^{\mathcal{G}}_{\Pi,\mathcal{A},\mathcal{Z}}(n,z)\}_{n\in\mathbb{N},z\in\{0,1\}^*} \tag{1}$$

Note that the above equation, says that in the ideal world, the simulator Sim has no access to the ideal functionality $\mathcal{G}$. However, when $\mathcal{G}$ is a set-up assumption, this is not necessarily true and the simulator may have access to $\mathcal{G}$ even in the ideal world. Indeed, there exist different formulations of the UC framework, capturing different requirements on the set-assumptions (e.g., [CDPW07, BFSK11]). In [CDPW07] for example, the set-up assumption is global, which means that the environment has direct access to the set-up functionality $\mathcal{G}$. Hence, the simulator Sim needs to have oracle access to $\mathcal{G}$ as well. In [BFSK11] they assume that Sim cannot simulate (*program*) a PUF, and thus it needs access to the ideal functionality $\mathcal{F}_{\mathsf{PUF}}$. [BFSK11] however restricts the access of the environment to $\mathcal{F}_{\mathsf{PUF}}$. $\mathcal{Z}$ has not permanent access to $\mathcal{F}_{\mathsf{PUF}}$.

Oblivious Transfer Functionality. Oblivious Transfer (OT) is a two-party game in which a sender holds a pair of strings (s_0, s_1), and a receiver needs to obtain one string according to its input bit b. The transfer of the desired string is oblivious in the sense that the sender does not know the string obtained by the receiver, while the receiver obtaining one string gains no information about the other one. The OT Functionality $\mathcal{F}_{\mathsf{OT}}$ is shown in Fig. 8.

Functionality $\mathcal{F}_{\mathsf{OT}}$

$\mathcal{F}_{\mathsf{OT}}$ running with an oblivious sender S a receiver R and an adversary Sim proceeds as follows:

- Upon receiving a message $(\mathsf{send}, \mathsf{sid}, s_0, s_1, \mathsf{S}, \mathsf{R})$ from S where each $s_0, s_1 \in \{0,1\}^n$, record the tuple (sid,s_0, s_1) and send $(\mathsf{send}, \mathsf{sid})$ to R and Sim. Ignore any subsequent send messages.
- Upon receiving a message $(\mathsf{receive}, \mathsf{sid}, b)$ from R, where $b \in \{0,1\}$ send (sid, s_b) to R and Sim and halt. (If no (send,$\cdot$) message was previously sent do nothing).

Figure 8: The Oblivious Transfer Functionality $\mathcal{F}_{\mathsf{OT}}$.

Commitment Functionality. The ideal functionality for Commitment Scheme as presented in [CF01], is depicted in Fig. 9. Such definition captures the hiding and binding property defined in Definition 7.

Functionality $\mathcal{F}_{\mathsf{com}}$

$\mathcal{F}_{\mathsf{com}}$ running with parties $P_1, \ldots, P_m$ and an adversary Sim proceeds as follows:

- **Commitment Phase:** Upon receiving a message $(\mathsf{commit}, \mathsf{sid}, P_i, P_j, b)$ from P_i where $b \in \{0,1\}$, record the tuple $(\mathsf{sid}, P_i, P_j, b)$ and send the message ($\mathsf{receipt}$, sid, P_i, P_j) to P_j and Sim. Ignore any subsequent commit messages.
- **Decommit Phase:** Upon receiving (open,sid, P_i, P_j) from P_i, if the tuple $(\mathsf{sid}, P_i, P_j, b)$ is recorded then send (open,$\mathsf{sid}, P_i, P_j, b$) to P_j and to Sim and halt. Otherwise ignore the message.

Figure 9: The Commitment Functionality $\mathcal{F}_{\mathsf{com}}$.

A.4 Security in Presence of Malicious Adversary in the Stand-alone Model

In this paragraph we recall the definition of security in presence of malicious adversary in the stand-alone model. The security in the stand-alone model is defined as a comparison of the output of two experiments, the real-life experiment and the ideal process, as for the UC-model in Appendix A.3, except that, in the stand-alone model there is no environment $\mathcal{Z}$. In this weaker model, $\mathsf{REAL}_{\Pi,\mathcal{A}}$ is defined as the output pair of the honest party and the adversary $\mathcal{A}$ from the real-life execution of Π (instead of the real-time view of the environment $\mathcal{Z}$), while $\mathsf{IDEAL}^{\mathcal{F}}_{\mathsf{Sim}}$, is defined as the output pair of the honest party and the ideal adversary Sim from the above ideal execution. In the following definition for simplicity of notation, we use the same notation used for definition of UC-security.

Definition 13. *(Security in presence of Malicious adversary in the Stand-alone Model). Let $\mathcal{F}$ be an ideal functionality, and Π be a protocol. We say that Π* ***securely computes*** *$\mathcal{F}$ with abort in presence*

of malicious adversary, if for any non-uniform adversary PPT $\mathcal{A}$, there exists a non-uniform PPT ideal process adversary Sim *such that*

$$\mathsf{IDEAL}^{\mathcal{F}}_{\mathsf{Sim}} \sim \mathsf{REAL}_{\Pi,\mathcal{A}}$$

Stand-alone Secure Statistical Receiver Oblivious Transfer. A statistical receiver OT is an Oblivious Transfer protocol in which the security of the receiver is preserved statistically, and is one of the ingredients of our constructions: $\mathsf{Com}_{\mathsf{shext}}$ and $\mathsf{Com}_{\mathsf{equiv}}$. One can obtain a statistical receiver string OT protocol from any statistical sender bit OT protocol as follows. First, from bit statistical sender OT obtain a bit statistical receiver OT, by applying the OT-reverse transformation shown by Wolf and Wullschleger in [WW06]. Then, obtain string statistical receiver OT from bit statistical receiver OT, by using the technique shown by Brassard et al. in [BCR86]. Finally note that a construction for stand-alone statistical sender bit OT is provided by Lindell and Hazay in [HL10] under the DDH assumption.

Alternatively, one can achieve OT statically secure for the receiver, by using the interactive hashing technique introduced in [OVY93]. In [OVY93], interactive hashing is used to achieve statistically hiding commitment scheme from any one-way permutation (OWP). The idea for commitments is that, the sender picks an x, and sends to the receiver $y = f(x)$, where f is a OWP. Then sender and receiver engage in an interactive hashing phase, at the end of which they both agree on values (y_0, y_1) such that $y \in \{y_0, y_1\}$, and from the receiver's view, y is equally likely to be any of the two strings. This is why hiding holds statistically. Finally, to commit, the sender sends a bit d that is equal to $b \oplus c$ where c is such that $y = y_c$.

This idea can be used to implement statistical receiver OT as follows. Let b the input of the OT receiver, and s_0, s_1 the inputs of the OT sender. First, we replace the OWP with a *trapdoor* OWP. The sender of OT, chooses a trapdoor OWP f and sends it to the OT receiver. The OT receiver picks a x, computes $y = f(x)$ and sends it to the OT sender. They engage in an interactive hashing, which output is the pair (y_0, y_1) as before. Let $y_c = y$, as before, the OT receiver knows $x_c = f^{-1}(y_c)$, and sends $d = b \oplus c$ to the sender. In the oblivious transfer phase, the sender computes $x_0 = f^{-1}(y_0)$ and $x_1 = f^{-1}(y_1)$, and sends $s_d \oplus x_d$ and $s_{\bar{d}} \oplus x_{\bar{d}}$.

The above OT protocol is statistically secure for the receiver, for the same argument of the statistically hiding commitment of [OVY93]. The security of the sender instead is preserved under the assumption that f is a OWP.

In the rest of the paper we will write statistical receiver OT to refer to a stand-alone secure OT protocol in which the security of the receiver is preserved statistically.

B Admissible PUF Families

The following formulation is adapted from [PW08].

Let G be an algorithm that takes as input a security parameter n and outputs a tuple $\mathbb{G} = (p, \langle G \rangle, g)$, where p is a prime, $\langle G \rangle$ is the description of a cyclic multiplicative group G of order p, and g is a generator of G.

Definition 14. *Let* G *be as defined above, and let* $(\mathsf{Sample}, \mathsf{Eval})$ *be a PUF family. The tuple* $(\mathsf{G}, (\mathsf{Sample}, \mathsf{Eval}))$ *is called* DDH-admissible *if for every oracle PPT distinguisher D, for every polynomial $p(\cdot)$, and for sufficiently large n,*

$$\left|\Pr\left[\, D^{\mathsf{Sample}(\cdot),\mathsf{Eval}(\cdot)}(1^n,(\mathbb{G},g^a,g^b,g^{ab}))=1\,\right]-\Pr\left[\, D^{\mathsf{Sample}(\cdot),\mathsf{Eval}(\cdot)}(1^n,(\mathbb{G},g^a,g^b,g^c))=1\,\right]\right| \;\leq\; \frac{1}{p(n)},$$

where $\mathbb{G}\leftarrow\mathsf{G}(1^n)$ *and* $a,b,c\leftarrow\mathbb{Z}_p$ *are uniform and independent.*

For succinctness, we will often keep G implicit in our discussion, and refer to a PUF family for which $(\mathsf{G},(\mathsf{Sample},\mathsf{Eval}))$ is DDH-admissible simply as "admissible".

C Sub-Protocols of $\mathsf{Com}_{\mathsf{uc}}$

In this section we show the main ingredients of Protocol $\mathsf{Com}_{\mathsf{uc}}$, i.e., Protocol $\mathsf{Com}_{\mathsf{shext}}$ and Protocol $\mathsf{Com}_{\mathsf{equiv}}$. For simplicity, in this section we use the following informal notation. We refer to a PUF created by party A as PUF_A, and we denote by $v\leftarrow\mathsf{PUF}_A(q)$ the evaluation of the PUF PUF_A on challenge q. An example of the formal notation involving the invocation of the ideal functionality $\mathcal{F}_{\mathsf{PUF}}$ is provided in Appendix F.

C.1 Statistically Hiding Straight-line Extractable Commitment Scheme

Let $\mathsf{Com}_{\mathsf{SH}}=(\mathsf{C}_{\mathsf{SH}},\mathsf{R}_{\mathsf{SH}})$ be a Statistically Hiding string commitment scheme, $(\mathsf{S}_{\mathsf{OT}},\mathsf{R}_{\mathsf{OT}})$ be a statistical receiver OT protocol (namely, an OT protocol where the receiver's privacy is statistically preserved). Let (P,V) be a Statistical Zero Knowledge Argument of Knowledge ($\mathcal{SZK}$AoK) for the following relation: $\mathcal{R}_{\mathtt{com}}=\{(c,(s,d))$ such that $\mathsf{R}_{\mathsf{SH}}(c,s,d)=1\}$. A pictorial description of protocol $\mathsf{Com}_{\mathsf{shext}}=(\mathsf{C}_{\mathsf{shext}},\mathsf{R}_{\mathsf{shext}})$ was given in Fig. 2. In Fig. 10 we provide the formal specification.

Theorem 4. *If* $\mathsf{Com}_{\mathsf{SH}}=(\mathsf{C}_{\mathsf{SH}},\mathsf{R}_{\mathsf{SH}})$ *is a statistically-hiding commitment scheme,* $(\mathsf{S}_{\mathsf{OT}},\mathsf{R}_{\mathsf{OT}})$ *is a statistical receiver OT protocol and* (P,V) *is a* $\mathcal{SZK}$AoK*, then* $\mathsf{Com}_{\mathsf{shext}}$ *is a statistically hiding straight-line extractable bit commitment scheme in the malicious PUFs model.*

Proof. **Completeness.** Before delivering its own PUF $\mathsf{PUF}_{\mathsf{R}}$, $\mathsf{R}_{\mathsf{shext}}$ queries it with a pair of random challenges (q_0,q_1) and gets answers (a_0,a_1). To commit to a bit b, $\mathsf{C}_{\mathsf{shext}}$ has to commit to the output a_b of $\mathsf{PUF}_{\mathsf{R}}$.

By the completeness of the OT protocol, $\mathsf{C}_{\mathsf{shext}}$ obtains the query q_b corresponding to its secret bit. Then $\mathsf{C}_{\mathsf{shext}}$ queries $\mathsf{PUF}_{\mathsf{R}}$ with q_b and commits to the response a'_b running C_{SH}. Furthermore, $\mathsf{C}_{\mathsf{shext}}$ proves using $\mathcal{SZK}$AoK the knowledge of the opening. By the completeness of $\mathcal{SZK}$AoK and $\mathsf{Com}_{\mathsf{SH}}$ the commitment phase is concluded without aborts. In the opening phase, $\mathsf{C}_{\mathsf{shext}}$ sends b and opens the commitment to a'_b, and $\mathsf{R}_{\mathsf{shext}}$ checks whether the string a'_b matches the answer a_b obtained by its own PUF applying the fuzzy extractor. By the response consistency property, $\mathsf{R}_{\mathsf{shext}}$ gets the correct answer and accept the decommitment for the bit b.

Statistically Hiding. We show that, for all $\mathsf{R}^*_{\mathsf{shext}}$ it holds that:

$$\mathsf{view}_{\mathsf{R}^*_{\mathsf{shext}}}(\mathsf{C}_{\mathsf{shext}}(\mathtt{com},0),\mathsf{R}_{\mathsf{shext}})\stackrel{\mathsf{S}}{\equiv}\mathsf{view}_{\mathsf{R}^*_{\mathsf{shext}}}(\mathsf{C}_{\mathsf{shext}}(\mathtt{com},1),\mathsf{R}^*_{\mathsf{shext}}).$$

This follows from the statistical security of the three sub-protocols run in the commitment phase by $\mathsf{C}_{\mathsf{shext}}$. More specifically, recall that the view of $\mathsf{R}^*_{\mathsf{shext}}$ in the commitment phase consists of the transcript of the execution of the OT protocol $(\mathsf{S}_{\mathsf{OT}},\mathsf{R}_{\mathsf{OT}})$, the transcript of the Statistically Hiding commitment scheme $\mathsf{Com}_{\mathsf{SH}}$ and the transcript of the execution of the $\mathcal{SZK}$AoK protocol. The proof goes by hybrids.

Committer's Input: Bit $b \in \{0,1\}$.

Commitment Phase

$\mathsf{R}_{\mathsf{shext}}$: Initialize $\mathsf{PUF}_{\mathsf{R}}$.

1. obtain $a_0 \leftarrow \mathsf{PUF}_{\mathsf{R}}(q_0)$, $a_1 \leftarrow \mathsf{PUF}_{\mathsf{R}}(q_1)$, for $(q_0, q_1) \xleftarrow{\$} \{0,1\}^n$.
2. $(st_0, p_0) \leftarrow \mathsf{FuzGen}(a_0)$, $(st_1, p_1) \leftarrow \mathsf{FuzGen}(a_1)$.
3. handover $\mathsf{PUF}_{\mathsf{R}}$ to $\mathsf{C}_{\mathsf{shext}}$.

$\mathsf{R}_{\mathsf{shext}} \Leftrightarrow \mathsf{C}_{\mathsf{shext}}$: (Statistical OT phase)
$\langle \mathsf{S}_{\mathsf{OT}}(q_0, q_1), \mathsf{R}_{\mathsf{OT}}(b) \rangle$ is run by $\mathsf{R}_{\mathsf{shext}}$ as S_{OT} with input (q_0, q_1), and $\mathsf{C}_{\mathsf{shext}}$ as R_{OT} with input b. Let q'_b be the local output of $\mathsf{C}_{\mathsf{shext}}$.

$\mathsf{C}_{\mathsf{shext}}$: $a'_b \leftarrow \mathsf{PUF}_{\mathsf{R}}(q'_b)$. If $\mathsf{PUF}_{\mathsf{R}}$ aborts, $a'_b \xleftarrow{\$} \{0,1\}^l$.

$\mathsf{C}_{\mathsf{shext}} \Leftrightarrow \mathsf{R}_{\mathsf{shext}}$: (Statistically Hiding Commitment)
$((c,d),c) \leftarrow \langle \mathsf{C}_{\mathsf{SH}}(\mathtt{com}, a'_b), \mathsf{R}_{\mathsf{SH}}(\mathtt{recv}) \rangle$ is run by $\mathsf{C}_{\mathsf{shext}}$ as C_{SH} to commit to a'_b, and by $\mathsf{R}_{\mathsf{shext}}$ as R_{SH}.

$\mathsf{C}_{\mathsf{shext}} \Leftrightarrow \mathsf{R}_{\mathsf{shext}}$: ($\mathcal{SZK}\mathsf{AoK}$)
$\langle P(d, a'_b), V \rangle(c)$ is run by $\mathsf{C}_{\mathsf{shext}}$ playing as prover P for the theorem $(c,(c,d)) \in \mathcal{R}_{\mathtt{com}}$ and by $\mathsf{R}_{\mathsf{shext}}$ playing as verifier V on input c. If the proof is not accepting, $\mathsf{R}_{\mathsf{shext}}$ aborts.

Decommitment Phase

$\mathsf{C}_{\mathsf{shext}}$: if $\mathsf{PUF}_{\mathsf{R}}$ did not abort, send opening (d, a'_b, b) to $\mathsf{R}_{\mathsf{shext}}$.

$\mathsf{R}_{\mathsf{shext}}$: if $\mathsf{R}_{\mathsf{SH}}(c, a'_b, d) = 1$ and $\mathsf{FuzRep}(a'_b, p_b) = st_b$ then accept. Else reject.

Figure 10: Statistically Hiding Straight-Line Extractable Bit Commitment Scheme $(\mathsf{C}_{\mathsf{shext}}, \mathsf{R}_{\mathsf{shext}})$.

H_0: In this hybrid the sender $\mathsf{C}_{\mathsf{shext}}$ commits to bit 0. Namely, it plays the OT protocol with the bit 0 to obtain q'_0, then it queries the malicious $\mathsf{PUF}^*_{\mathsf{R}}$ to obtain a string a'_0, then it commits to a'_0 executing C_{SH} and finally it runs the honest prover P to prove knowledge of the decommitment.

H_1: In this hybrid, $\mathsf{C}_{\mathsf{shext}}$ proceeds as in H_0, except that it executes the zero knowledge protocol by running the zero knowledge simulator S. By the statistical zero knowledge property of (P, V), hybrids H_0 and H_1 are statistically indistinguishable.

H_2: In this hybrid, $\mathsf{C}_{\mathsf{shext}}$ proceeds as in H_1, excepts that it runs C_{SH} to commit to a random string s instead of a'_0. By the statistically hiding property of protocol $\mathsf{Com}_{\mathsf{SH}}$, hybrids H_1 and H_2 are statistically indistinguishable.

H_3: In this hybrid, $\mathsf{C}_{\mathsf{shext}}$ proceeds as in H_2, except that in OT protocol it plays with bit 1, obtaining query q'_1. By the receiver security of protocol $(\mathsf{S}_{\mathsf{OT}}, \mathsf{R}_{\mathsf{OT}})$, hybrids H_2 and H_3 are statistically indistinguishable.

H_4: In this hybrid, $\mathsf{C}_{\mathsf{shext}}$ proceeds as in H_3, except that here it queries the PUF with string q'_1 to obtain a'_1 (however it still commits to the random string s). If the $\mathsf{PUF}^*_{\mathsf{R}}$ aborts, then $\mathsf{C}_{\mathsf{shext}}$

sets $a'_1 \leftarrow \{0,1\}^l$. Note that any malicious behavior does not effect the transcript generated in H_4. Thus, hybrids H_3 and H_3 are identical.

H_5**:** In this hybrid, $\mathsf{C}_{\mathsf{shext}}$ proceeds as in H_4 except that it commits to the string a'_1. By the statistically hiding property of protocol $\mathsf{Com}_{\mathsf{SH}}$, hybrids H_4 and H_5 are statistically indistinguishable.

H_6**:** In this hybrid, $\mathsf{C}_{\mathsf{shext}}$ proceeds as in H_5, except that it executes the zero knowledge protocol running as the honest prover P. By the statistical zero knowledge property of (P, V), hybrids H_5 and H_6 are statistically indistinguishable.

By observing that hybrid H_0 corresponds to the case in which $\mathsf{C}_{\mathsf{shext}}$ commits to 0 and hybrid H_6 corresponds to the case in which $\mathsf{C}_{\mathsf{shext}}$ commits to 1, the hiding property is proved.

Straight-line Extractability. To prove extractability we show a straight-line strict polynomial-time extractor E that satisfies the properties required by Definition 9. Recall that, in the commitment scheme $\mathsf{Com}_{\mathsf{shext}}$, the sender basically commits to the answer a_b received from $\mathsf{PUF}_{\mathsf{R}}$. By the unpredictability of PUF, the sender needs to get the right query q_b from $\mathsf{R}_{\mathsf{shext}}$ in order to obtain the value to commit to. Such q_b is obliviously retrieved by $\mathsf{C}_{\mathsf{shext}}$ running OT with the bit b. The strategy of the extractor, that we show below, is very simple. It consists of running the commitment phase as the honest receiver, and then looking at the queries made by $\mathsf{C}^*_{\mathsf{shext}}$ to $\mathsf{PUF}_{\mathsf{R}}$ to detect which among q_0, q_1 has been asked and thus extract the bit. The extraction of the bit fails when one of the following two cases happens. Case **Fail1**: the set of queries contains both (q_0, q_1) (or at least a pair that is within their hamming distance); in this case E cannot tell which is the bit played by $\mathsf{C}^*_{\mathsf{shext}}$ and therefore outputs $\perp$. By the sender's security of OT this case happens only with negligible probability. Case **Fail2**: the set of queries does not contain any query close (within hamming distance) to neither q_0 nor q_1. This is also a bad case since E cannot extract any information. However, if there exists such a $\mathsf{C}^*_{\mathsf{shext}}$ that produces an accepting commitment without querying the PUF in the commitment phase (but perhaps it makes queries in the decommitment phase only) then, given that responses of honest PUFs are unpredictable, one can break either the binding property of the underlying commitment scheme $\mathsf{Com}_{\mathsf{SH}}$ or the argument of knowledge property of (P, V). The formal description of E is given below. Formal arguments follow.

Extractor E

Commitment Phase. Run the commitment phase following the honest receiver procedure. We denote by (q_0, q_1) the queries made by the extractor E to the honest PUF before delivering it to $\mathsf{C}^*_{\mathsf{shext}}$. E uses such a pair when running as S_{OT} in OT protocol. If all sub-protocols (OT, $\mathsf{Com}_{\mathsf{SH}}, \mathcal{SZK}\mathsf{AoK}$) are successfully completed go the extraction phase. Else, abort.

Extraction phase. Let $\mathcal{Q}$ be the set of queries asked by $\mathsf{C}^*_{\mathsf{shext}}$ to $\mathsf{PUF}_{\mathsf{R}}$ during the commitment phase.

Fail1. If there exists a pair $q'_0, q'_1 \in \mathcal{Q}$ such that $\mathsf{dis}_{\mathsf{ham}}(q_0, q'_0) \leq d_{\mathsf{min}}$ and $\mathsf{dis}_{\mathsf{ham}}(q_1, q'_1) \leq d_{\mathsf{min}}$, output $b^\star = \perp$.

Fail2. If for all $q' \in \mathcal{Q}$ it holds that $\mathsf{dis}_{\mathsf{ham}}(q_0, q') > d_{\mathsf{min}}$ and $\mathsf{dis}_{\mathsf{ham}}(q_1, q') > d_{\mathsf{min}}$, output $b^\star = \perp$.

Good. 1. If there exists $q' \in \mathcal{Q}$ such that $\mathsf{dis}_{\mathsf{ham}}(q_0, q') \leq d_{\mathsf{min}}$ then output $b^\star = 0$.
2. If there exists $q' \in \mathcal{Q}$ such that $\mathsf{dis}_{\mathsf{ham}}(q_1, q') \leq d_{\mathsf{min}}$ then output $b^\star = 1$.

The above extractor E satisfies the following three properties.
Simulation. E follows the procedure of the honest receiver $\mathsf{R}_{\mathsf{shext}}$. Thus the view of $\mathsf{C}^*_{\mathsf{shext}}$ playing with E is identical to the view of $\mathsf{C}^*_{\mathsf{shext}}$ playing with $\mathsf{R}_{\mathsf{shext}}$.
Extraction. Let τ_c the transcript of the commitment phase. For the extraction property we have to show that if τ_c is accepting, then the probability that E outputs $\perp$ is negligible. Note that E outputs $\perp$ if and only if one of the event between **Fail1** and **Fail2** happens. Thus,

$$\Pr[\, b^\star = \perp \,] = \Pr[\, \mathbf{Fail1} \,] + \Pr[\, \mathbf{Fail2} \,]$$

In the following we show that, if τ_c is accepting, then $\Pr[\, b^\star = \perp \,]$ is negligible by showing separately that $\Pr[\, \mathbf{Fail1} \,]$ and $\Pr[\, \mathbf{Fail2} \,]$ are negligible.

Lemma 2 ($\Pr[\, \mathbf{Fail1} \,]$ is negligible)**.** *If* $(\mathsf{S}_{\mathsf{OT}}, \mathsf{R}_{\mathsf{OT}})$ *is an Oblivious Transfer protocol, then* $\Pr[\, \mathbf{Fail1} \,]$ *is negligible.*

Proof. Assume that there exists a PPT $\mathsf{C}^*_{\mathsf{shext}}$ such that event **Fail1** happens with non-negligible probability δ. Then it is possible to construct $\mathsf{R}^*_{\mathsf{OT}}$ that uses $\mathsf{C}^*_{\mathsf{shext}}$ to break the sender's security of the OT protocol. $\mathsf{R}^*_{\mathsf{OT}}$ interacts with an external OT sender S_{OT}, on input auxiliary information $z = (s_0, s_1)$, while it runs $\mathsf{C}^*_{\mathsf{shext}}$ internally. $\mathsf{R}^*_{\mathsf{OT}}$ initializes and sends $\mathsf{PUF}_{\mathsf{R}}$ to $\mathsf{C}^*_{\mathsf{shext}}$, then it runs the OT protocol forwarding the messages received from the external sender S_{OT} to $\mathsf{C}^*_{\mathsf{shext}}$ and vice versa. When the OT protocol is completed, $\mathsf{R}^*_{\mathsf{OT}}$ continues the internal execution with $\mathsf{C}^*_{\mathsf{shext}}$ emulating the honest receiver. When the commitment phase is successfully completed, $\mathsf{R}^*_{\mathsf{OT}}$ analyses the set $\mathcal{Q}$ of queries made by $\mathsf{C}^*_{\mathsf{shext}}$ to $\mathsf{PUF}_{\mathsf{R}}$. If there exists a pair (q'_0, q'_1) within hamming distance with strings (s_0, s_1) then $\mathsf{R}^*_{\mathsf{OT}}$ outputs (s_0, s_1), therefore breaking the sender's security of OT with probability δ (indeed, there exists no simulator that can simulate such attack since in the ideal world Sim gets only one input among (s_0, s_1)). Since by assumption $(\mathsf{S}_{\mathsf{OT}}, \mathsf{R}_{\mathsf{OT}})$ is a stand-alone secure OT protocol, δ must be negligible. □

Lemma 3 ($\Pr[\, \mathbf{Fail2} \,]$ is negligible)**.** *Assume that* τ_c *is an accepting transcript. If* $\mathsf{Com}_{\mathsf{SH}} = (\mathsf{C}_{\mathsf{SH}}, \mathsf{R}_{\mathsf{SH}})$ *is a commitment scheme and if* (P, V) *is a* $\mathcal{SZK}\mathsf{AoK}$ *then* $\Pr[\, \mathbf{Fail2} \,]$ *is negligible.*

Proof. If transcript τ_c is accepting then it holds that $\mathsf{C}^*_{\mathsf{shext}}$ in the decommitment phase will send a tuple (b, d, a'_b) for which, given τ_c, the receiver $\mathsf{R}_{\mathsf{shext}}$ accepts, i.e., the opening (d) of the statistically hiding commitment is valid *and* corresponds to an answer (a'_b) of $\mathsf{PUF}_{\mathsf{R}}$ upon one of the queries played by the $\mathsf{R}_{\mathsf{shext}}$ in the OT protocol. Formally, $\mathsf{R}_{\mathsf{SH}}(c, a'_b, d) = 1$ and $\mathsf{FuzRep}(a'_b, p_b) = st_b$.

Toward a contradiction, assume that $\Pr[\, \mathbf{Fail2} \,] = \delta$ and is not-negligible. Recall that the event **Fail2** happens when $\mathsf{C}^*_{\mathsf{shext}}$ successfully completed the commitment phase, without querying $\mathsf{PUF}_{\mathsf{R}}$ with any of (q_0, q_1). Given that τ_c is accepting, let (b, d, a'_b) be an accepting decommitment, we have the following cases:

1. $\mathsf{C}^*_{\mathsf{shext}}$ honestly committed to the correct a'_b without having queried $\mathsf{PUF}_{\mathsf{R}}$. By the unpredictability of $\mathsf{PUF}_{\mathsf{R}}$ we have that this case has negligible probability to happen.
2. $\mathsf{C}^*_{\mathsf{shext}}$ queries $\mathsf{PUF}_{\mathsf{R}}$ in the *decommitment* phase to obtain the value a'_b to be opened. Thus $\mathsf{C}^*_{\mathsf{shext}}$ opens commitment c (sent in the commitment phase) as string a'_b. We argue that by the computational binding of $\mathsf{Com}_{\mathsf{SH}}$ and by the argument of knowledge property of (P, V) this case also happens with negligible probability.
   First, we show and adversary $\mathsf{C}^*_{\mathsf{SH}}$ that uses $\mathsf{C}^*_{\mathsf{shext}}$ as a black-box to break the binding of the commitment scheme $\mathsf{Com}_{\mathsf{SH}}$ with probability δ. $\mathsf{C}^*_{\mathsf{SH}}$ runs $\mathsf{C}^*_{\mathsf{shext}}$ internally, simulating the honest receiver $\mathsf{R}_{\mathsf{shext}}$ to it, and forwarding only the messages belonging to $\mathsf{Com}_{\mathsf{SH}}$ to

an external receiver R_{SH}, and vice versa. Let c denote the transcript of $\mathsf{Com}_{\mathsf{SH}}$. When the commitment phase of $\mathsf{Com}_{\mathsf{shext}}$ is successfully completed, and therefore $\mathsf{C}^*_{\mathsf{shext}}$ has provided an accepting proof for the theorem $(c,\cdot) \in \mathcal{R}_{\mathtt{com}}$, $\mathsf{C}^*_{\mathsf{SH}}$ runs the extractor E_P associated to the protocol (P,V). By the argument of knowledge property, E_P, having oracle access to $\mathsf{C}^*_{\mathsf{shext}}$, extracts the witness $(\tilde{a}_b, \tilde{d})$ used by $\mathsf{C}^*_{\mathsf{shext}}$ to prove theorem $c \in \mathcal{R}_{\mathtt{com}}$ w.h.p. If the witness extracted is not a valid decommitment of c, then $\mathsf{C}^*_{\mathsf{shext}}$ can be used to break the soundness of (P,V).
Else, $\mathsf{C}^*_{\mathsf{SH}}$ proceeds to the decommitment phase, and as by hypothesis of Lemma 3, since the commitment τ_c is accepting, $\mathsf{C}^*_{\mathsf{shext}}$ provides a valid opening (a_b, d).
If $(\tilde{a}_b, \tilde{d}) \neq (a_b, d)$ are two valid openings for c then $\mathsf{C}^*_{\mathsf{SH}}$ outputs such tuple breaking the binding property of $\mathsf{Com}_{\mathsf{SH}}$ with probability δ.
If $(\tilde{a}_b, \tilde{d}) = (a_b, d)$ with non-negligible probability, then consider the following analysis. By assumption, event **Fail2** happens when $\mathsf{C}^*_{\mathsf{shext}}$ does not query $\mathsf{PUF}_{\mathsf{R}}$ with none among (q_0, q_1). By the unpredictability property, it holds that without querying the PUF, $\mathsf{C}^*_{\mathsf{shext}}$ cannot guess the values a_b, thus w.h.p. the commitment c played by $\mathsf{C}^*_{\mathsf{shext}}$ in the commitment phase, does not hide the value a_b. However, since the output of the extraction is a valid opening for a_b, then it must have been the case that in one of the rewinding attempts of the black-box extractor E_P, $\mathsf{C}^*_{\mathsf{shext}}$ has obtained a_b by asking $\mathsf{PUF}_{\mathsf{R}}$. Indeed, upon each rewind E_P very luckily changes the messages played by the verifier of the ZK protocol, and $\mathsf{C}^*_{\mathsf{shext}}$ could choose the queries for $\mathsf{PUF}_{\mathsf{R}}$ adaptively on such messages. However, recalling that E_P is run by $\mathsf{C}^*_{\mathsf{SH}}$ to extract from $\mathsf{C}^*_{\mathsf{shext}}$, $\mathsf{C}^*_{\mathsf{SH}}$ can avoid such failure by following this strategy: when a rewinding thread leads $\mathsf{C}^*_{\mathsf{shext}}$ to ask the PUF with query q_b, then abort such thread and start a new one. By noticing that in the commitment phase, $\mathsf{C}^*_{\mathsf{shext}}$ did not query the PUF with q_b, we have that, by the argument of knowledge property of (P,V) this event happens again in the rewinding threads w.h.p. Thus, by discarding the rewinding thread in which $\mathsf{C}^*_{\mathsf{shext}}$ asks for query q_b, $\mathsf{C}^*_{\mathsf{SH}}$ is still be able to extract the witness in polynomial time (again, if this was not the case then one can use $\mathsf{C}^*_{\mathsf{shext}}$ to break the argument of knowledge property). With this strategy, the event $(\tilde{a}_b, \tilde{d}) = (a_b, d)$ is ruled out.

□

Binding. Let $b^\star = b_0$ the bit extracted by E, given the transcript τ_c. Assume that in the decommitment phase $\mathsf{C}^*_{\mathsf{shext}}$ provides a valid opening of τ_c as b_1 and $b_0 \neq b_1$. If such an event happens, the the following three events happened: 1) in the commitment phase $\mathsf{C}^*_{\mathsf{shext}}$ queried $\mathsf{PUF}_{\mathsf{R}}$ with query q_{b_0} only; 2) in decommitment phase $\mathsf{C}^*_{\mathsf{shext}}$ queried $\mathsf{PUF}_{\mathsf{R}}$ with q_{b_1}, let a_{b_1} be the answer; 3)$\mathsf{C}^*_{\mathsf{shext}}$ opens the commitment c (that is the commitment of the answer of $\mathsf{PUF}_{\mathsf{R}}$ received in the commitment phase), as a_{b_1}, but c was computed without knowledge of $\mathsf{PUF}_{\mathsf{R}}(q_{b_1})$.

By the security of the OT protocol and by the computational binding of the commitment scheme $\mathsf{Com}_{\mathsf{SH}}$, the above cases happen with negligible probability. Formal arguments follow previous discussions and are therefore omitted. □

Lemma 4. *Protocol* $\mathsf{Com}_{\mathsf{shext}}$ *is close under parallel repetition using the same PUF.*

Sketch. The proof comes straightforwardly by the fact that all sub-protocols used in protocol $\mathsf{Com}_{\mathsf{shext}}$ are close under parallel repetition. However, issues can arise when the same, possibly malicious and stateful PUF, is reused. Note that, the output of the (malicious) PUF is statistically hidden in the commitment phase and that it is revealed only in the decommitment phase. Thus, any side information that is leaked by a dishonest PUF, cannot be used by the malicious creator,

before the decommitment phase. At the decommitment stage however, the input of the committer is already revealed, and no more information is therefore gained by the malicious party. We stress out that re-usability is possible only when many instances of $\mathsf{Com}_{\mathsf{shext}}$ are run in *parallel*, i.e., only when all decommitment happen simultaneously. If decommitment phases are interleaved with commitment phase of other sessions, then reusing the same PUF, allow the malicious creator to gain information about sessions that are not open yet. To see why, let i and j be two concurrent executions. Assume that the commitment of i and j is done in parallel but session j is decommitted before session i. Then, a malicious PUF can send information on the bit committed in the session i through the string sent back for the decommitment of j. □

Statistically Hiding Straight-line Extractable String Commitment Scheme. We obtain statistically hiding straight-line extractable *string* commitment scheme, for n-bit string, by running n execution of $\mathsf{Com}_{\mathsf{shext}}$ in parallel and reusing the same PUF. In the main protocol shown in Figure 4 we use the same notation $\mathsf{Com}_{\mathsf{shext}}$ to refer to a string commitment scheme.

C.2 Statistically Binding Straight-line Extractable and Equivocal Commitment Scheme

Let $l = rg(n)$ be the range of the PUF, $(\mathsf{S}_{\mathsf{OT}}, \mathsf{R}_{\mathsf{OT}})$ be a statistical receiver OT protocol and let $G : \{0,1\}^n \to \{0,1\}^{3n}$ be a PRG. The commitment scheme that we present, takes as common input a string $\bar{r} = r_1, \ldots, r_l$, that is *uniformly chosen* in the set $(\{0,1\}^{3n})^l$. This string can be seen as l distinct parameters for Naor's commitment, and indeed it is used to commit bit-by-bit to an l-bit string (i.e., the answer received from the PUF). Our statistically binding straight-line extractable and equivocal commitment scheme $\mathsf{Com}_{\mathsf{equiv}} = (\mathsf{C}_{\mathsf{equiv}}, \mathsf{R}_{\mathsf{equiv}})$ is depicted in Fig. 11. A graphical representation was provided in Fig. 3.

Theorem 5. *If G is a PRG and $(\mathsf{S}_{\mathsf{OT}}, \mathsf{R}_{\mathsf{OT}})$ is statistical receiver OT protocol, then $\mathsf{Com}_{\mathsf{equiv}} = (\mathsf{C}_{\mathsf{equiv}}, \mathsf{R}_{\mathsf{equiv}})$ is a statistically binding straight-line extractable and equivocal commitment scheme in the malicious PUFs model.*

Proof. **Completeness.** It follows from the completeness of the OT protocol, the correctness of Naor's commitment and the response consistency property of PUFs with fuzzy extractors. To commit to the bit b, the sender $\mathsf{C}_{\mathsf{equiv}}$ is required to commit to the answer of $\mathsf{PUF}_{\mathsf{R}}$ on input q_b. Therefore, $\mathsf{C}_{\mathsf{equiv}}$ runs the OT protocol with input b and obtains the query q_b and thus the value to commit to using Naor's commitments. The correctness of OT guarantees that the consistency check performed by $\mathsf{C}_{\mathsf{equiv}}$ goes through. In the decommitment phase, the response consistency property along with correctness of Naor, allow the receiver $\mathsf{R}_{\mathsf{equiv}}$ to obtain the string a_b and in therefore the bit decommitted to by $\mathsf{C}_{\mathsf{equiv}}$.

Straight-line Extractability.
Extractor E

Commitment Phase. Run the commitment phase following the honest receiver procedure: E queries $\mathsf{PUF}_{\mathsf{R}}$ with (q_0, q_1) before delivering it to $\mathsf{C}^*_{\mathsf{equiv}}$, and uses such a pair when running as S_{OT} in OT protocol. If OT protocol is not successfully completed then abort. Else, let $\mathcal{Q}_{\mathsf{precom}}$ be the set of queries asked by $\mathsf{C}^*_{\mathsf{equiv}}$ to $\mathsf{PUF}_{\mathsf{R}}$ *before* sending the commitments $c_1, \ldots, c_l$ to E. Upon receiving such commitments, do as follows:

Committer's Input: Bit $b \in \{0,1\}$. **Common Input:** $\bar{r} = (r_1, \ldots, r_l)$

Commitment Phase

$\mathsf{R}_{\mathsf{equiv}}$: Initialize $\mathsf{PUF}_{\mathsf{R}}$;

1. obtain $a_0 \leftarrow \mathsf{PUF}_{\mathsf{R}}(q_0)$, $a_1 \leftarrow \mathsf{PUF}_{\mathsf{R}}(q_1)$, for $(q_0, q_1) \xleftarrow{\$} \{0,1\}^n$.
2. $(st_0, p_0) \leftarrow \mathsf{FuzGen}(a_0)$, $(st_1, p_1) \leftarrow \mathsf{FuzGen}(a_1)$.
3. handover $\mathsf{PUF}_{\mathsf{R}}$ to $\mathsf{C}_{\mathsf{equiv}}$;
4. choose random tape $\mathsf{ran}_{\mathsf{OT}} \xleftarrow{\$} \{0,1\}^*$.

$\mathsf{R}_{\mathsf{equiv}} \Leftrightarrow \mathsf{C}_{\mathsf{equiv}}$: (OT phase)
$\langle \mathsf{S}_{\mathsf{OT}}(q_0, q_1), \mathsf{R}_{\mathsf{OT}}(b) \rangle$ is run by $\mathsf{R}_{\mathsf{equiv}}$ as S_{OT} with input (q_0, q_1) and randomness $\mathsf{ran}_{\mathsf{OT}}$, while $\mathsf{C}_{\mathsf{equiv}}$ runs as R_{OT} with input b. Let q'_b be the local output of $\mathsf{C}_{\mathsf{equiv}}$, and τ_{OT} be the transcript of the execution of the OT protocol.

$\mathsf{C}_{\mathsf{equiv}}$:(Statistically Binding Commitment)

1. $a'_b \leftarrow \mathsf{PUF}_{\mathsf{R}}(q'_b)$. If $\mathsf{PUF}_{\mathsf{R}}$ aborts, $a'_b \xleftarrow{\$} \{0,1\}^l$.
2. for $1 \le i \le l$, pick $s_i \xleftarrow{\$} \{0,1\}^n$, $c_i = G(s_i) \oplus (r_i \wedge a'_b[i])$. [a]
3. send $c_1, \ldots, c_l$ to $\mathsf{R}_{\mathsf{equiv}}$.

$\mathsf{R}_{\mathsf{equiv}}$: upon receiving $c_1, \ldots, c_l$, send $\mathsf{ran}_{\mathsf{OT}}, q_0, q_1$ to $\mathsf{C}_{\mathsf{equiv}}$.

$\mathsf{C}_{\mathsf{equiv}}$: check if transcript τ_{OT} is consistent with $(\mathsf{ran}_{\mathsf{OT}}, q_0, q_1, b)$. If the check fails abort.

Decommitment Phase

$\mathsf{C}_{\mathsf{equiv}}$: if $\mathsf{PUF}_{\mathsf{R}}$ did not abort, send $((s_1, \ldots, s_l), a'_b), b$ to $\mathsf{R}_{\mathsf{shext}}$.

$\mathsf{R}_{\mathsf{equiv}}$: if for all i, it holds that $(c_i = G(s_i) \oplus (r_i \wedge a'_b[i])$ and $\mathsf{FuzRep}(a'_b, p_b) = st_b)$ then accept. Else reject.

[a] where $(r_i \wedge a'_b[i])_j = r_i[j] \wedge a'_b[i]$.

Figure 11: Statistically Binding Straight-line Extractable and Equivocal Commitment $(\mathsf{C}_{\mathsf{equiv}}, \mathsf{R}_{\mathsf{equiv}})$.

Fail1. If there exists a pair $q'_0, q'_1 \in \mathcal{Q}_{\mathsf{precom}}$ such that $\mathsf{dis}_{\mathsf{ham}}(q_0, q'_0) \le d_{\mathsf{min}}$ and $\mathsf{dis}_{\mathsf{ham}}(q_1, q'_1) \le d_{\mathsf{min}}$, output $b^\star = \perp$.

Fail2. If for all $q' \in \mathcal{Q}_{\mathsf{precom}}$ it holds that $\mathsf{dis}_{\mathsf{ham}}(q_0, q') > d_{\mathsf{min}}$ and $\mathsf{dis}_{\mathsf{ham}}(q_1, q') > d_{\mathsf{min}}$, output $b^\star = \perp$.

Good. 1. If there exists $q' \in \mathcal{Q}_{\mathsf{precom}}$ such that $\mathsf{dis}_{\mathsf{ham}}(q_0, q') \le d_{\mathsf{min}}$ then output $b^\star = 0$;
2. If there exists $q' \in \mathcal{Q}_{\mathsf{precom}}$ such that $\mathsf{dis}_{\mathsf{ham}}(q_1, q') \le d_{\mathsf{min}}$ then output $b^\star = 1$;

Finally sends $\mathsf{ran}_{\mathsf{OT}}, q_0, q_1$ to $\mathsf{C}^*_{\mathsf{equiv}}$.

Simulation. E follows the procedure of the honest receiver $\mathsf{R}_{\mathsf{equiv}}$. Thus the view of $\mathsf{C}^*_{\mathsf{equiv}}$ playing with E is identical to the view of $\mathsf{C}^*_{\mathsf{equiv}}$ playing with $\mathsf{R}_{\mathsf{equiv}}$.

Extraction. The proof of extraction follows from the same arguments shown in the proof of

Theorem 4, and it is simpler since in protocol $\mathsf{Com}_{\mathsf{equiv}}$ we use statistically binding commitments (given that the common parameter $\bar{r}$ is uniformly chosen).

Let τ_c the transcript of the commitment phase. For the extraction property we have to show that if τ_c is accepting, then the probability that E outputs $\perp$ is negligible. Note that E outputs $\perp$ if and only if one event between **Fail1** and **Fail2** happens. Thus,

$$\Pr[\, b^\star = \perp \,] = \Pr[\, \mathbf{Fail1} \,] + \Pr[\, \mathbf{Fail2} \,]$$

By the sender's security property of the OT protocol, event **Fail1** happens with negligible probability. The formal proof follows the same arguments given in Lemma 2. Given that the common parameter $\bar{r}$ is uniformly chosen, we have that the Naor's commitments (i.e., $c_1, \ldots, c_l$) sent by $\mathsf{C}^*_{\mathsf{equiv}}$ in the commitment phase, are statistically binding. Thus, by the unpredictability property of PUFs and the by the statistically binding property of Naor's commitment scheme, event **Fail2** also happens with negligible probability only.

Binding. Given that the common input $\bar{r}$ is uniformly chosen, binding of $\mathsf{Com}_{\mathsf{equiv}}$ follows from the statistically binding property of Naor's commitment scheme.

Straight-line Equivocality. In the following we show a straight-line simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ and we prove that the view generated by the interaction between $\mathcal{S}$ and $\mathsf{R}^*_{\mathsf{equiv}}$ is computationally indistinguishable from the view generated by the interaction between $\mathsf{C}_{\mathsf{equiv}}$ and $\mathsf{R}^*_{\mathsf{equiv}}$.

$\underline{\mathcal{S}_1}$. $(\bar{r} = r_1, \ldots, r_l, \mathsf{state}_1) \leftarrow \mathcal{S}_1(1^{ln})$:

For $i = 1, \ldots, l$.

1. pick $s_0^i \leftarrow \{0,1\}^n$, $\alpha_0^i \leftarrow G(s_0^i)$;
2. pick $s_1^i \leftarrow \{0,1\}^n$, $\alpha_1^i \leftarrow G(s_1^i)$;
3. $r_i = \alpha_0^i \oplus \alpha_1^i$.

Output $r_1, \ldots, r_l$, $\mathsf{state}_1 = \{s_0^i, s_1^i\}_{i \in l}$;

$\underline{\mathcal{S}_2}$. $(\mathsf{state}_2) \leftarrow \mathcal{S}_2(\mathsf{state}_1)$:

- obtain $\mathsf{PUF}^*_{\mathsf{R}}$ from $\mathsf{R}^*_{\mathsf{equiv}}$.
- run OT protocol with input a random bit $\tilde{b}$; if the OT protocol is not successfully completed, abort.
- computes commitments as follows: for $i = 1, \ldots, l$, $\tilde{c}_i \leftarrow G(s_0^i)$. Send $\tilde{c}_1, \ldots, \tilde{c}_l$ to $\mathsf{R}^*_{\mathsf{equiv}}$.
- Obtain $(\mathsf{ran}_{\mathsf{OT}}, q'_0, q'_1)$ from $\mathsf{R}^*_{\mathsf{equiv}}$ and check if the transcript τ_{OT} is consistent with it. If the check fails, abort. Else, output $\mathsf{state}_2 = \{\mathsf{state}_1, (q'_0, q'_1)\}$.

$\underline{\mathcal{S}_3}$. $\mathcal{S}_3(\mathsf{state}_2, b)$:

- query $\mathsf{PUF}^*_{\mathsf{R}}$ with input q'_b. If $\mathsf{PUF}^*_{\mathsf{R}}$ aborts, abort. Otherwise, let a'_b denote the answer of $\mathsf{PUF}^*_{\mathsf{R}}$.
- for $i = 1, \ldots, l$: send $(s^i_{a_b[i]}, a_b[i])$ to $\mathsf{R}^*_{\mathsf{equiv}}$.

Lemma 5. *If $(\mathsf{S}_{\mathsf{OT}}, \mathsf{R}_{\mathsf{OT}})$ is a statistical receiver OT protocol and G is a pseudo-random generator, then for all PPT $\mathsf{R}^*_{\mathsf{equiv}}$ it holds that, $\{\mathsf{out}(\mathbf{Exp}^{\mathsf{C}_{\mathsf{equiv}}}_{\mathsf{R}^*_{\mathsf{equiv}}}(n))\} \stackrel{\mathsf{C}}{\equiv} \mathsf{out}\{(\mathbf{Exp}^{\mathcal{S}}_{\mathsf{R}^*_{\mathsf{equiv}}}(n)\}$.*

Proof. The proof goes by hybrids arguments.

H_0. This is the real world experiment $\mathbf{Exp}^{\mathsf{C}_{\mathsf{equiv}}}_{\mathsf{R}^*_{\mathsf{equiv}}}$.

H_1. In this hybrid the common parameter $\bar{r}$ is chosen running algorithm $\mathcal{S}_1$. The only difference between experiment H_0 and H_1 is in the fact that in H_1 each string $r_i \in \bar{r}$ is pseudo-random. By the pseudo-randomness of PRG H_0 and H_1 are computationally indistinguishable.

H_2. In this hybrid, the commitments $c_1, \ldots, c_l$ are computed as in $\mathcal{S}_2$, that is, for all i, c_i corresponds to an evaluation of the PRG i.e., $c_i = G(s_0^i)$, regardless of the bit that is committed. Then in the decommitment phase the sender uses knowledge of s_1^i, in case the i-th commitment of a'_b is the bit 1. (Each pair (s_0^i, s_1^i) is inherited from the output of $\mathcal{S}_1$). The difference between experiment H_1 and experiment H_2 is in the fact that in H_2 all commitments are pseudo-random, while in H_1, pseudo-random values are used only to commit to bit 0. By the pseudo-randomness of PRG, experiments H_1 and H_2 are computationally indistinguishable. Note that in this experiment, the sender is not actually committing to the output obtained by querying $\mathsf{PUF}_{\mathsf{R}}^*$.

H_3. In this experiment the sender queries $\mathsf{PUF}_{\mathsf{R}}^*$ on input q_b only in the decommitment phase. The only difference between this experiment and the previous one is that in H_3, the sender is able to detect if $\mathsf{PUF}_{\mathsf{R}}^*$ aborts, only in the decommitment phase. However, in experiment H_2, if the PUF aborts, the sender continues the execution of the commitment phase, committing to a random string, ad aborts only in the decommitment phase. Therefore, hybrids H_2 and H_3 are identical.

H_4. In this experiment, the sender executes the OT protocol with a random bit $\tilde{b}$, obtaining $q_{\tilde{b}}$, but it does not use such a query to evaluate $\mathsf{PUF}_{\mathsf{R}}^*$. Instead it uses the string q'_b received from $\mathsf{R}_{\mathsf{equiv}}^*$ in the last round of the commitment phase.

We stress out that, due to the correctness of the OT protocol and to the statistical receiver's security, the case in which $\mathsf{R}_{\mathsf{equiv}}^*$ plays the OT protocol with a pair $(q_b, q_{\bar{b}})$ and then is able to compute randomness $\mathsf{ran}_{\mathsf{OT}}$ and a different pair $((q'_b, q_{\bar{b}})$ that are still consistent with the transcript obtained in the OT execution, is statistically impossible . By the statistical receiver security of the OT protocol, H_3 and H_4 are statistically indistinguishable.

H_5. This is the ideal world experiment $\mathbf{Exp}_{\mathsf{R}_{\mathsf{equiv}}^*}^{\mathcal{S}}$.

□

□

D Proof of Theorem 1

In this section we show that protocol $\mathsf{Com}_{\mathsf{uc}} = (\mathsf{C}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}})$ depicted in Figure 4 is UC-secure, by showing a PPT ideal world adversary Sim such that for all PPT environment $\mathcal{Z}$, the view of the environment in the ideal process is indistinguishable from the view of the environment in the real process, in the $\mathcal{F}_{\mathsf{PUF}}$ hybrid model. Due to the straight-line extractability of $\mathsf{Com}_{\mathsf{shext}}$ and to the straight-line extractability and equivocality of $\mathsf{Com}_{\mathsf{equiv}}$, showing such a simulator Sim is almost straightforward.

Receiver is corrupt. Let $\mathsf{R}_{\mathsf{uc}}^*$ a malicious receiver. We show a PPT simulator Sim whose output is computational indistinguishable from the output obtained by $\mathsf{R}_{\mathsf{uc}}^*$ when interacting with the honest

committer C_{uc}. The goal of Sim is to use the straight-line equivocator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ associated to protocol $\mathsf{Com}_{\mathsf{equiv}}$. To accomplish that, Sim has to force the output of the coin flipping, to the parameter generated by $\mathcal{S}_1$. Once this is done, then Sim can use $\mathcal{S}_2$ to complete the commitment phase, and $\mathcal{S}_3$ to equivocate the commitment. In order to force the output of the coin flipping, Sim extracts the commitment of α sent by $\mathsf{R}^*_{\mathsf{uc}}$ so that it can compute β appropriately. The extraction is done by running the extractor $\mathsf{E}_{\mathsf{C}_{\mathsf{shext}}}$ associated to the protocol $\mathsf{Com}_{\mathsf{shext}}$.

Simulator 1.

Commitment Phase

- Run $(\bar{r}, \mathsf{state}_1) \leftarrow \mathcal{S}_1(1^{ln})$.
- Execute protocol $\mathsf{Com}_{\mathsf{shext}}$ by running the associated extractor $\mathsf{E}_{\mathsf{C}_{\mathsf{shext}}}$. If the output of the extractor is $\perp$, then abort. Else, let $\alpha^\star$ be the string extracted by $\mathsf{E}_{\mathsf{C}_{\mathsf{shext}}}$. Set $\beta = \bar{r} \oplus \alpha^\star$, and send β to $\mathsf{R}^*_{\mathsf{uc}}$. If $\mathsf{R}^*_{\mathsf{uc}}$ aborts, then abort.
- When receiving the opening to α from $\mathsf{R}^*_{\mathsf{uc}}$, if the opening is not accepting, or if $\alpha \neq \alpha^\star$ then abort.
- Execute the commitment phase of protocol $\mathsf{Com}_{\mathsf{equiv}}$, on common input $\alpha \oplus \beta = \bar{r}$, by running $\mathcal{S}_2(\mathsf{state}_1)$, and obtain state_2 as local output.

Decommitment Phase

- On input the bit b. Execute the decommitment phase of protocol $\mathsf{Com}_{\mathsf{equiv}}$ by running $\mathcal{S}_3(\mathsf{state}_2, b)$.
- Output whatever R_{uc} outputs.

Lemma 6. *For all PPT real-world malicious receiver $\mathsf{R}^*_{\mathsf{uc}}$, for all PPT adversary $\mathcal{Z}$, it holds that:*

$$\mathsf{IDEAL}^{\mathcal{F}_{\mathsf{com}}}_{\mathsf{Sim},\mathcal{Z}} \sim \mathsf{REAL}^{\mathcal{F}_{\mathsf{PUF}}}_{\mathsf{Com}_{\mathsf{uc}},\mathsf{R}^*_{\mathsf{uc}},\mathcal{Z}}$$

Proof. It follows from the straight-line extractability of $\mathsf{Com}_{\mathsf{shext}}$ and from the straight-line equivocality of $\mathsf{Com}_{\mathsf{equiv}}$.

By the straight-line extractability of $\mathsf{Com}_{\mathsf{shext}}$ it holds that, with overwhelming probability, Sim obtains the value $\alpha^\star$ that will be later opened by $\mathsf{R}^*_{\mathsf{uc}}$, before it has to send the message β. Hence, Sim is able to force the output of the coin flipping to the value determined by $\mathcal{S}_1$. Then Sim just runs the simulator $\mathcal{S}_2$ in the commitment phase, and $\mathcal{S}_3$ in the decommitment phase. By the straight-line equivocality property of $\mathsf{Com}_{\mathsf{equiv}}$ the view generated by the interaction between $\mathsf{R}^*_{\mathsf{uc}}$ and Sim is computationally indistinguishable from the view generated by the interaction between $\mathsf{R}^*_{\mathsf{uc}}$ and an honest sender C_{uc}. □

Receiver and Committer are honest. In this case, $\mathcal{Z}$ feeds the parties with their inputs, and activates the dummy adversary $\mathcal{A}$. $\mathcal{A}$ does not corrupt any party, but just observes the conversation between the committer and the receiver, forwarding every message to $\mathcal{Z}$.

In this case the simulator is almost equal to the simulator shown in Simulator 1 (when the receiver is corrupt). The only difference in this case is that, the receiver is also simulated by Sim. Therefore, Sim chooses both the strings used in the coin flipping by himself (α, β). Thus, there is no need for extraction.

More specifically, upon receiving the message $(\mathsf{receipt}, \mathsf{sid}, P_i, \mathsf{C}_{\mathsf{uc}})$ from $\mathcal{F}_{\mathsf{com}}$ in the ideal world, Sim draws a random tape to simulate the receiver, and runs the commitment phase as in Simulator 1,

except for the second step. Instead of using the extractor associated to $\mathsf{Com}_{\mathsf{shext}}$ run by the receiver, Sim just picks values α and β so that $\bar{r} = \beta \oplus \alpha$ (where $\bar{r}$ is the value given in output by $\mathcal{S}_1$), and continues the commitment phase using such values. The decommitment phase is run identically to the decommitment phase of Simulator 1.

From the same argument of the previous case, the transcript provided by Sim is indistinguishable from the transcript provided by the dummy adversary $\mathcal{A}$ running with honest sender and receiver.

Committer is corrupt. In this case, the task of Sim is to extract the bit of the malicious committer $\mathsf{C}^*_{\mathsf{uc}}$ already in the commitment phase. This task is easily accomplished by running the straight-line extractor $\mathsf{E}_{\mathsf{equiv}}$ associated to protocol $\mathsf{Com}_{\mathsf{equiv}}$. However, note that the binding property and thus the extractability property hold only when the common parameter $\bar{r}$ is uniformly chosen, while in protocol $\mathsf{Com}_{\mathsf{uc}}$ the common parameter is dictated by the coin flipping.

However, by the statistically hiding property of $\mathsf{Com}_{\mathsf{shext}}$, any unbounded adversary can not guess α better than guessing at random. Therefore for any $\mathsf{C}^*_{\mathsf{uc}}$ the distribution of $\alpha \oplus \beta$ is uniformly chosen over $\{0,1\}^{3nl}$, and thus the statistically binding property of $\mathsf{Com}_{\mathsf{equiv}}$ still holds.

Commitment Phase

- Pick a random α^{ln} and executes $\mathsf{Com}_{\mathsf{shext}}$ as the honest receiver.
- Obtain β from $\mathsf{C}^*_{\mathsf{uc}}$ and let $r = \alpha \oplus \beta$.
- Execute protocol $\mathsf{Com}_{\mathsf{equiv}}$ by running the associated extractor $\mathsf{E}_{\mathsf{equiv}}$. If the extractor aborts, abort. Else, let $b^\star$ the output of $\mathsf{E}_{\mathsf{equiv}}$. Send $(\mathsf{commit}, \mathsf{sid}, \mathsf{C}_{\mathsf{equiv}}, \mathsf{R}_{\mathsf{equiv}}, b^\star)$ to $\mathcal{F}_{\mathsf{com}}$

Lemma 7. *For all PPT real-world malicious committer $\mathsf{C}^*_{\mathsf{uc}}$, for all PPT adversary $\mathcal{Z}$, it holds that:*

$$\mathsf{IDEAL}^{\mathcal{F}_{\mathsf{com}}}_{\mathsf{Sim},\mathcal{Z}} \sim \mathsf{REAL}^{\mathcal{F}_{\mathsf{PUF}}}_{\mathsf{Com}_{\mathsf{uc}},\mathsf{C}^*_{\mathsf{uc}},\mathcal{Z}}$$

Proof. As mentioned before, the common input $\bar{r}$ computed through the coin-flipping, is uniformly distributed. Therefore the binding and the extractability property of $\mathsf{Com}_{\mathsf{equiv}}$ hold. The simulator runs protocol $\mathsf{Com}_{\mathsf{shext}}$ following the honest receiver, and runs the protocol $\mathsf{Com}_{\mathsf{equiv}}$ activating the straight-line extractor associated. By the simulation property of the extractor, the transcript generated by Sim is indistinguishable from the transcript generated by the honest receiver R_{uc}. From the extraction property satisfied by $\mathsf{E}_{\mathsf{equiv}}$, we have that Sim extracts the input bit of the adversary $\mathsf{C}^*_{\mathsf{uc}}$ and plays it in the ideal functionality, w.h.p.

□

E Proof of Theorem 2

In the discussion below, we construct oblivious query simulators for various adversaries. Recall that an oblivious query simulator is a pair of algorithms (S_1, S_2), where S_1 acts as a wrapper, and runs S_2 internally. Since the behaviour of S_1 is fixed, in the discussion below, by "simulator" we will mean the algorithm S_2.

Correctness. Consider the case when the receiver's input bit b is 0. The other case follows from a similar argument. If both parties are honest, then it follows from the bounded noise and response consistency properties that the protocol does not abort in the cut-and-choose phase. Let i be an

index that survives cut-and-choose, that is, $i \notin S$. Let q_i, a_i, b_i be the query, response and random bit chosen for index i by the receiver, as defined in the protocol. Consider Step 5 of the protocol: note that the $\hat{q}_i$s computed by the sender are such that $\hat{q}_i^{b_i} = q_i$. Thus, $m_i^0 = s_i^0 \oplus st^{b_i}$, where st^{b_i} is the output of the fuzzy extractor applied to the response of q_i. From the reconstruction information $p_i^{b_i}$, the receiver can compute $st_i^{b_i}$, and thus obtain the correct s_i^0. In this way, receiver obtains all the correct shares, and can reconstruct s^0.

Malicious Sender. The simulator runs the protocol honestly with receiver's input bit as 0. However, it makes additional queries to learn the responses of both $q_{i_j}^0$ and $q_{i_j}^1$ from Step 5 of the protocol. Thus, it can compute both $st_{i_j}^0$ and $st_{i_j}^1$ and extract both the strings s^0 and s^1.

Malicious Receiver. The simulator $\mathsf{Sim}_{\mathsf{uncOT}}$ starts an internal interaction with adversary $\mathsf{R}^*_{\mathsf{uncOT}}$ and proceeds as follows:

1. $\mathsf{Sim}_{\mathsf{uncOT}}$ plays the part of the sender and executes Steps 1-3 (i.e., till the end of cut-and-choose) of the protocol honestly with $\mathsf{R}^*_{\mathsf{uncOT}}$. Note that up to this point, sender's messages do not depend on its input, so the simulator can reproduce this execution perfectly.

2. Let $i_1, \ldots, i_k$ be the indices *not* in S. The simulator receives bits $b'_{i_1}, \ldots, b'_{i_k}$ from the adversary, and for $1 \leq j \leq k$, does the following:

   - query $\mathsf{PUF}\ \mathsf{sid}^{\mathsf{S}}_{i_j}$ with d_{i_j} and obtain response da_{i_j}.
   - compute $dst_{i_j} \leftarrow \mathsf{FuzRep}(dp_{i_j}, da_{i_j})$.
   - compute $c_{i_j} = b'_{i_j} \oplus \mathsf{Parity}(dst_{i_j})$.

3. $\mathsf{Sim}_{\mathsf{uncOT}}$ sets bit $\hat{c}$ to the majority of $c_{i_1}, \ldots, c_{i_k}$ (ties are broken arbitrarily).

4. Simulator $\mathsf{Sim}_{\mathsf{uncOT}}$ queries the ideal functionality with bit $\hat{c}$ and obtains a string s. It chooses a random string $\hat{s} \in \{0,1\}^n$ and sets $s^{\hat{c}} = s$ and $s^{1-\hat{c}} = \hat{s}$. Then it runs Step 5 of the protocol with the pair (s^0, s^1).

We first prove that for each $1 \leq j \leq k$, the receiver knows only one of $st_{i_j}^0$ and $st_{i_j}^1$. This already implies that the adversary learns only one of the sender's strings, while the other one remains information-theoretically hidden. However, we must show that the adversary learns the same string in both the real and ideal worlds - it should not be the case that in the real execution, the adversary gets s^c, while in the simulation it gets s^{1-c}. We show that if the cut-and-choose succeeds, then with high probability the simulator extracts the correct bit.

Let $\mathcal{Q}$ be the set of queries the adversary makes to $\mathsf{PUF}\ \mathsf{sid}^{\mathsf{R}}$. For a query q, we say "$\mathcal{Q}$ *covers* q" if there exists $q' \in \mathcal{Q}$ such that $\mathsf{dis}(q, q') < d_{\mathsf{min}}$. The proof of the following claim appears in Appendix A of [BFSK11], and we sketch it here for completeness.

Claim 1 (from [BFSK11])**.** *For all $j \in [k]$, with overwhelming probability, $\mathcal{Q}$ covers at most one of $\hat{q}_{i_j}^0$ and $\hat{q}_{i_j}^1$.*

Proof. We first show that for a particular $q' \in \mathcal{Q}$, it can not be the case that both $\mathsf{dis}(q', \hat{q}^0_{i_j}) < d_{\mathsf{min}}$ and $\mathsf{dis}(q', \hat{q}^1_{i_j}) < d_{\mathsf{min}}$. Indeed, this would imply $\mathsf{dis}(x^0_{i_j}, x^1_{i_j}) < 2d_{\mathsf{min}}$, which happens with negligible probability due to the well-spread domain property. The second case to consider is when two different queries in $\mathcal{Q}$ say q' and q'' are close to $\hat{q}^0_{i_j}$ and $\hat{q}^1_{i_j}$. That is, $\mathsf{dis}(q', \hat{q}^0_{i_j}) < d_{\mathsf{min}}$ and $\mathsf{dis}(q'', \hat{q}^1_{i_j}) < d_{\mathsf{min}}$. However, this implies that $\mathsf{dis}(x^0_{i_j} \oplus x^1_{i_j}, q' \oplus q'') < 2d_{\mathsf{min}}$. As the size of $\mathcal{Q}$ is polynomial in the security parameter, and $x^0_{i_j}$ and $x^1_{i_j}$ are chosen randomly, the probability of this happening is negligible. Thus, $\mathcal{Q}$ covers both $\hat{q}^0_{i_j}$ and $\hat{q}^1_{i_j}$ with negligible probability. □

Fix receiver's message in Step 3(b). For $1 \leq i \leq 2k$, set $\hat{b}_i = 0$ if $\mathcal{Q}$ covers $\hat{q}^0_i$, else set $\hat{b}_i = 1$ if $\mathcal{Q}$ covers $\hat{q}^1_i$, else let $\hat{b}_i = \perp$. Let da_i be response of query d_i to $\mathsf{PUF}\ \mathsf{sid}^{\mathsf{S}}_i$. We call an index $i \in [2k]$ "bad" if $\hat{b}_i \neq \mathsf{Parity}(\mathsf{FuzRep}(da_i, dp_i))$. Let η be the number of bad indices , i.e., $\eta = |\{\, i \in [2k] \mid i \text{ is bad. } \}|$. We bound the probability that the cut-and-choose succeeds *and* $\eta \geq \gamma$, for some parameter γ. This probability is upper bounded by the probability that cut-and-choose succeeds *given* $\eta \geq \gamma$. This probability, in turn, can be computed by counting the number of subsets of size k that do not contain any of the γ bad indices. Thus, this probability is:

$$\begin{aligned}
\frac{\binom{2k-\gamma}{k}}{\binom{2k}{k}} &= \frac{(2k-\gamma)!}{(k-\gamma)!}\frac{k!}{(2k)!} \\
&= \frac{k(k-1)\cdots(k-(\gamma-1))}{(2k)(2k-1)\cdots(2k-(\gamma-1))} \\
&= \left(1-\frac{k}{2k}\right)\left(1-\frac{k}{2k-1}\right)\cdots\left(1-\frac{k}{2k-(\gamma-1)}\right) \\
&\leq e^{-k\left(\frac{1}{2k}+\frac{1}{2k-1}+\cdots+\frac{1}{2k-(\gamma-1)}\right)} \\
&< e^{-\gamma/2}.
\end{aligned}$$

Setting $\gamma = k/10$, we get that the probability that the cut-and-choose succeeds and $\eta \geq k/10$ is at most $e^{-k/20}$.

Now condition on the event that the cut-and-choose succeeds and the number of bad indices is less than $k/10$. Let the bit $\hat{c}$ extracted by the simulator in Step 3 be 0 (the case when $\hat{c} = 1$ is handled analogously). We will argue that in the real execution, s^1 is information theoretically hidden from the receiver. As the number of zeros in the sequence $c_{i_1}, \ldots, c_{i_k}$ is more than $k/2$, and then number of bad indices in $[2k]$ is at most $k/10$, there must exist an index i_j such that (1) $c_{i_j} = 0$ and, (2) i_j is not bad. In fact, there will be a large number of such indices. Fix such an index i_j. Observe the following:

- Let $\rho = \mathsf{Parity}(\mathsf{FuzRep}(da_{i_j}, dp_{i_j}))$. As i_j is not bad, we have that $\mathcal{Q}$ covers $q^{\rho}_{i_j}$, and not $q^{1-\rho}_{i_j}$.
- As $c_{i_j} = 0$, we have $b'_{i_j} = \rho$.

In the real execution, the sender prepares the message $m^1_{i_j} = s^1_j \oplus st^{1-b'_j}_{i_j} = s^1_j \oplus st^{1-\rho}_{i_j}$. As $q^{1-\rho}_{i_j}$ is not covered by $\mathcal{Q}$, we have that in the real execution, $st^{1-\rho}_{i_j}$ is information-theoretically hidden from the adversary, which implies that the share s^1_j is hidden, which in turn implies that s^1 is information-theoretically hidden.

The Honest-Honest Case. Now we handle the case when the adversary does not corrupt either of the parties. The simulator in this case is very simple: it mimics an honest execution between $\mathsf{S}_{\mathsf{uncOT}}$ with both inputs set to 0^n, and $\mathsf{R}_{\mathsf{uncOT}}$ with input bit 0. Let $\tau(s^0, s^1, b)$ be the adversary's view during an honest execution of $(\mathsf{S}_{\mathsf{uncOT}}, \mathsf{R}_{\mathsf{uncOT}})$ with sender inputs (s^0, s^1), and receiver's input b (note that τ consists of the transcript of interaction between the honest sender and receiver, along with the PUF challenge-response pairs the adversary obtains during handover of various PUFs). To show correctness of simulation, we will argue that for every (s^0, s^1, b), the distributions $\tau(s^0, s^1, b)$ and $\tau(0^n, 0^n, 0)$ are statistically close.

The argument goes via a hybrid argument. The first hybrid is $\tau(s^0, s^1, b)$. In the second hybrid, we change receiver's input to 0, i.e., the second hybrid is $\tau(s^0, s^1, 0)$. The only message in the transcript where b is used in Step 4 of the protocol, where the receiver sends the correction bits b'_{i_j}. Let $\mathcal{Q}_1$ be the set of queries the adversary makes to the PUFs $\mathsf{sid}^{\mathsf{S}}_1, \ldots, \mathsf{sid}^{\mathsf{S}}_{2k}$ during $\mathsf{handover}_{\mathsf{PUF}}$ in Step 1 of the protocol. As the receiver uniformly chooses queries d_i for $1 \leq i \leq 2k$ after the handover is complete, the probability that $\mathcal{Q}_1$ covers any d_i is negligible. Thus, the bits b'_{i_j} are uniform and distributed independently of b. Therefore, $\tau(s^0, s^1, b)$ and $\tau(s^0, s^1, 0)$ are statistically close.

In the third hybrid, we replace the sender's inputs with 0^n, i.e., the third hybrid is $\tau(0^n, 0^n, 0)$. Let $\mathcal{Q}_2$ be the set of queries that the adversary makes to the PUF $\mathsf{sid}^{\mathsf{R}}$ during the $\mathsf{handover}_{\mathsf{PUF}}$ in Step 2 of the protocol. Again, as $q_1, \ldots, q_{2k}$ are chosen uniformly by the receiver, the probability that $\mathcal{Q}_2$ covers any one of $q_1, \ldots, q_{2k}$ is negligible. Further, as (x^0_i, x^1_i) are also chosen uniformly, this implies that $\mathcal{Q}_2$ covers any of $(\hat{q}^0_{i_j}, \hat{q}^1_{i_j})$ with negligible probability. Thus, the distribution of the final message is independent of the sender's inputs. Therefore, $\tau(s^0, s^1, 0)$ and $\tau(0^n, 0^n, 0)$ are statistically close.

The Malicious-Malicious Case. If the adversary corrupts both the parties, then the simulator simply acts as a wire between the adversary and the environment. In this case, the simulation is perfect.

F On Unconditional Security with Malicious PUFs

In this section we prove the security of the unconditional commitment scheme in Figure 6. Completeness of protocol $(\mathsf{C}_{\mathsf{uncon}}, \mathsf{R}_{\mathsf{uncon}})$ follows from response consistency. We focus on hiding and binding properties.

Lemma 8 (Hiding). *For any malicious receiver $\mathsf{R}^*_{\mathsf{uncon}}$, the statistical difference between the ensembles*

$$\{\, \mathsf{view}_{\mathsf{R}^*}(\mathsf{C}(\mathtt{com}, 0), \mathsf{R}^*(\mathtt{recv}))(1^n) \,\}_{n\in\mathbb{N}} \text{ and } \{\, \mathsf{view}_{\mathsf{R}^*}(\mathsf{C}(\mathtt{com}, 1), \mathsf{R}^*(\mathtt{recv}))(1^n)) \,\}_{n\in\mathbb{N}}$$

is negligible in n.

Proof. Let $\mathcal{Q}$ be the set of queries that the receiver R^* makes to the committer's PUF sid and let q be the query made by the committer before sending the PUF sid. First consider the case that there exists $q' \in \mathcal{Q}$ such that $\mathsf{dis}(q', q) < d_{\mathsf{min}}$. As the receiver is polynomially bounded, the number of queries in $\mathcal{Q}$ is a polynomial, say $p(n)$. The total number of queries within a distance d_{min} of queries in $\mathcal{Q}$ can be bounded by $p(n)n^{d_{\mathsf{min}}(n)}$, which is a negligible fraction of 2^n. Thus, this event happens with negligible probability.

Now consider the case that $q \notin \mathcal{Q}$. By the extraction independence property, st is statistically close to the uniform distribution, U_ℓ. As, for any string r, the distributions U_ℓ and $r \oplus U_\ell$ are identical, thus it follows from transitivity that the distributions st and $st \oplus r$ are statistically close. □

We now turn to the binding property. The proof follows a similar path as in the proof of statistical binding in Naor's commitment [Nao89]. Informally[14], Naor's argument counts the number of 'bad' strings in the range of the PRG. These are the strings r in the range of a PRG $G(\cdot)$ for which there exist two seeds s_0, s_1 such that $r = G(s_0) \oplus G(s_1)$. For these strings r, equivocation is possible. But because of the expansion property of PRG, the number of bad strings is small. Similarly, in our proof of binding, we use the fact that we have set the parameters of the PUF family and fuzzy extractor such that $\ell = 3n$. We use the same arguments to define 'bad' strings and show that because of expansion, their number is bounded. Care has to be taken to handle the fact that the output of the PUF is noisy.

Lemma 9 (Binding)**.** *For any malicious committer $\mathsf{C}^*_{\mathsf{uncon}}$, the probability that it wins in the binding game of Definition 7 is negligible in n.*

Proof. Recall that we have chosen the parameters of the PUF family and fuzzy extractor such that $\ell = 3n$. We can think of the adversary choosing the malicious PUF as picking a set of distributions $\mathcal{D}_{q_1}, \ldots, \mathcal{D}_{q_N}$, where $N = 2^n$. For a fixed p, call a string $st \in \{0,1\}^\ell$ "heavy" if there exists query q such that $\Pr[\,\mathsf{FuzRep}(p, \mathcal{D}_q) = st\,] \geq 2^{-\log^2(n)}$.

Now we will show that the probability of the adversary breaking the binding is negligible. For a fixed first message of the malicious committer, call a string $r \in \{0,1\}^\ell$ 'bad' if the probability that the adversary breaks binding on receiving r in the second step of the protocol is at least $2^{-2\log^2(n)}$. For this to happen, it must be the case that there exist heavy strings st_0 and st_1 such that $r = st_0 \oplus st_1$. Thus, to bound the number of bad strings in $\{0,1\}^\ell$, we simply need to bound the number of pairs of heavy strings. By the definition of heavy string, each query can produce at most $2^{\log^2(n)}$ heavy strings for one PUF. As the total number of queries is 2^n, the total number of pairs of heavy strings is bounded by $2^{2(n+\log^2(n))}$, which is a negligible fraction of 2^{3n}. □

G The Brzuska et.al. [BFSK11] Ideal Functionality $\mathcal{F}_{\mathsf{hPUF}}$

H Allowing Adversary to Create PUFs

In this section, we explain why it is crucial for the UC-composition theorem to allow the adversary to create PUFs of its own. We begin by sketching the proof of the UC composition theorem from Section 5.2, [Can01]. Those familiar with the proof of the composition theorem can skip to the end of this section.

We say that a protocol ρ "UC-emulates" a protocol ϕ if there exists an adversary S such that no environment can tell whether it is interacting with ρ and the dummy adversary, or ϕ and S. That is for every environment $\mathcal{Z}$,

[14] The following assumes familiarity with Naor's commitment scheme [Nao89].

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

- When a party P_i writes $(\mathsf{init}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ on the input tape of $\mathcal{F}_{\mathsf{PUF}}$, then $\mathcal{F}_{\mathsf{PUF}}$ checks whether $\mathcal{L}$ already contains a tuple $(\mathsf{sid}, *, *, *, *)$:
  - If this is the case, then turn into the waiting state.
  - Else, draw $\mathsf{id} \leftarrow \mathsf{Sample}(1^n)$ from the PUF-family. Put $(\mathsf{sid}, \mathsf{id}, P_i, *, \mathsf{notrans})$ in $\mathcal{L}$ and write $(\mathsf{initialized}_{\mathsf{PUF}}, \mathsf{sid})$ on the communication tape of P_i.
- When party P_i writes $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{sid}, P_i, q)$ on $\mathcal{F}_{\mathsf{PUF}}$'s input tape, check if there exists a tuple $(\mathsf{sid}, \mathsf{id}, P_i, \mathsf{notrans})$ in $\mathcal{L}$.
  - If not, then turn into waiting state.
  - Else, run $a \leftarrow \mathsf{Eval}(1^n, \mathsf{id}, q)$. Write $(\mathsf{response}_{\mathsf{PUF}}, \mathsf{sid}, q, a)$ on P_i's communication input tape.
- When a party P_i sends $(\mathsf{handover}_{\mathsf{PUF}}, \mathsf{sid}, P_i, P_j)$ to $\mathcal{F}_{\mathsf{PUF}}$, check if there exists a tuple $(\mathsf{sid}, *, P_i, \mathsf{notrans})$ in $\mathcal{L}$.
  - If not, then turn into waiting state.
  - Else, modify the tuple $(\mathsf{sid}, \mathsf{id}, P_i, \mathsf{notrans})$ to the updated tuple $(\mathsf{sid}, \mathsf{id}, \perp, \mathsf{trans}(P_j))$. Write $(\mathsf{invoke}_{\mathsf{PUF}}, \mathsf{sid}, P_i, P_j)$ on $\mathcal{S}$'s communication input tape.
- When the adversary sends $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{sid}, \mathcal{S}, q)$ to $\mathcal{F}_{\mathsf{PUF}}$, check if $\mathcal{L}$ contains a tuple $(\mathsf{sid}, \mathsf{id}, \perp, \mathsf{trans}(*))$.
  - If not, then turn into waiting state.
  - Else, run $a \leftarrow \mathsf{Eval}(1^n, \mathsf{id}, q)$ and return $(\mathsf{response}_{\mathsf{PUF}}, \mathsf{sid}, q, a)$ to $\mathcal{S}$.
- When $\mathcal{S}$ sends $(\mathsf{ready}_{\mathsf{PUF}}, \mathsf{sid}, \mathcal{S})$ to $\mathcal{F}_{\mathsf{PUF}}$, check if $\mathcal{L}$ contains the tuple $(\mathsf{sid}, \mathsf{id}, \perp, \mathsf{trans}(P_j))$.
  - If not found, turn into the waiting state.
  - Else, change the tuple $(\mathsf{sid}, \mathsf{id}, \perp, \mathsf{trans}(P_j))$ to $(\mathsf{sid}, \mathsf{id}, P_j, \mathsf{notrans})$ and write $(\mathsf{handover}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ on P_j's communication input tape and store the tuple $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$.
- When the adversary sends $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ to $\mathcal{F}_{\mathsf{PUF}}$, check if the tuple $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ exists in L. If not, return to the waiting state. Else, write this tuple to the communication input tape of P_i.

Figure 12: The ideal functionality $\mathcal{F}_{\mathsf{hPUF}}$ from Brzuska et.al. [BFSK11] .

$$\mathsf{REAL}_{\rho,D,\mathcal{Z}} \sim \mathsf{REAL}_{\phi,S,\mathcal{Z}},$$

where D is a dummy adversary. Let π, ρ, ϕ be protocols, where π may call ϕ as a sub-routine. By $\pi^{\rho/\phi}$ we denote the protocol which is the same as π, except that each call to ϕ is replaced by a call to ρ.

The UC composition theorem [Can01] states that if ρ UC emulates ϕ, then $\pi^{\rho/\phi}$ UC emulates π. The proof of the composition theorem goes as follows: we need to show that there exists an adversary S_π such that for every environment $\mathcal{Z}$,

$$\mathsf{REAL}_{\pi^{\rho/\phi},D,\mathcal{Z}} \sim \mathsf{REAL}_{\pi,S_\pi,\mathcal{Z}}.$$

Let S be the adversary that follows from the fact the ρ UC emulates ϕ. We now outline the construction of adversary S_π that uses S. Adversary S_π divides the messages into two parts: those that belong to an execution of ρ, and those that belong to execution of the rest of π. The adversary S_π handles the messages pertaining to ρ by passing them to an instance of the adversary S. For the messages of π that don't belong to ρ, adversary S_π simply passes them to the parties they are intended for.

To see the correctness of the above construction, assume that there exists an environment $\mathcal{Z}$ that distinguishes between $\pi^{\rho/\phi}$ with a dummy adversary, and π with S_π as the adversary. We will construct an environment $\mathcal{Z}_\rho$ that distinguishes between ρ with the dummy adversary and ϕ with the adversary S. This is a contradiction, as ρ UC-emulates ϕ.

The environment $\mathcal{Z}_\rho$ internally runs an execution of $\mathcal{Z}$, S_π and all the parties of π. It faithfully follows the actions of these parties except for messages concerning the sub-protocol ϕ, for which it uses the actual external parties (which are either running an instance of ϕ, or an instance of ρ). In particular, whenever an internal party of π generates an input for an instance of ρ, the environment $\mathcal{Z}_\rho$ passes that input to the external party. Similarly, any output sent by an external party to $\mathcal{Z}_\rho$ is treated as an output of ϕ. Further, any message sent by S_π to a simulator S is forwarded by $\mathcal{Z}_\rho$ to the external adversary, and the response of the external adversary is conveyed to S_π as though it was sent by S. Finally, the environment $\mathcal{Z}_\rho$ outputs whatever $\mathcal{Z}$ outputs.

Now, note that if the external parties are running ρ with the dummy adversary, then the view of the simulated environment $\mathcal{Z}$ is identical to $\mathsf{REAL}_{\pi^{\rho/\phi},D,\mathcal{Z}}$. On the other hand, if the external parties are running ϕ with S, then the view of the simulated environment $\mathcal{Z}$ is identical to $\mathsf{REAL}_{\pi,S_\pi,\mathcal{Z}}$. This implies that $\mathsf{REAL}_{\rho,D,\mathcal{Z}_\rho}$ and $\mathsf{REAL}_{\phi,S,\mathcal{Z}_\rho}$ are distinguishable, which contradicts the hypothesis that ρ UC-emulates ϕ.

Composition Theorem and PUFs. Now let us consider composition in the presence of PUFs. Recall that in the $\mathcal{F}_{\mathsf{PUF}}$-hybrid model, the environment does not have direct access to the ideal $\mathcal{F}_{\mathsf{PUF}}$ functionality (see Section 4.3 and Appendix B of [BFSK11] for details on the PUF access model). However, looking at the proof of the composition theorem, we immediately notice that the environment $\mathcal{Z}_\rho$ must have the ability to create PUFs. This is because to carry out the internal simulation of $\mathcal{Z}$ and π, environment $\mathcal{Z}_\rho$ must be able to handle PUF requests by the parties of π. Since PUFs are not programmable, $\mathcal{Z}_\rho$ can not simulate PUF responses on its own. It is to tackle this very issue that we allow the adversary to create new PUF's in our $\mathcal{F}_{\mathsf{PUF}}$ ideal functionality in Figure 1. This is sufficient for the composition theorem: when a simulated party in π requests a PUF, the environment $\mathcal{Z}_\rho$ asks the adversary to create a new PUF, and uses that PUF to handle the simulated party's requests.