Universally Composable Secure Computation with (Malicious) Physically Uncloneable Functions

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

Abstract

The use of Physically Uncloneable Functions (PUFs) in Cryptography is a recent breakthrough that has caught the interest of both theoreticians and practitioners. A major step towards understanding and securely using PUFs has been done in [CRYPTO 2011] where Brzuska, Fischlin, Schröder and Katzenbeisser augmented the Universal Composition (UC) Framework of Canetti [FOCS 2001] by considering Physically Uncloneable Functions (PUFs). Their model considers *trusted* PUFs only (i.e., adversaries are assumed to be unable to produce fake/malicious PUFs). Moreover they assumed that the simulator can observe queries made by an adversary to a PUF (i.e., an adversary can access a PUF only in a prescribed detectable way). Since the study of PUFs to achieve cryptographic tasks is still in its infancy, assuming such limitations on the capabilities of the adversaries in misbehaving with PUFs might not correspond to real-world scenarios.

In this work we continue this research direction by focusing on relaxing the above two restrictions. We first present models that are resilient to possible real-world attacks that have not been addressed by the model (and constructions) of Brzuska et al. Next, we give positive answers to the question of achieving universally composable secure computation with PUFs under the new security models. In particular we consider first the case of fully *malicious* PUFs and show how to obtain UC security for any functionality using computational assumptions. We then achieve *unconditional* UC security for any functionality with *trusted* PUFs but allowing the adversary an *oblivious* (undetected by any simulator) access to them. The latter result improves the work of Brzuska et al. since we obtain the same result but considering a relaxed assumption (i.e., modeling a more powerful adversary).

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

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

1 Introduction

The impossibility of secure computation in the universal composability framework has been

*Departments of Computer Science and Department of Mathematics, UCLA, 3732D Boelter Hall, Los Angeles CA 90095-1596, U.S. 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, 3771 Boelter Hall, Los Angeles CA 90095-1596, U.S. Email: awadia@cs.ucla.edu

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].

More recently, Katz in [Kat07] showed that UC security is possible by relying on the existence of tamper-proof programmable hardware tokens. 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]).

Recently a different hardware component has gained a lot of attention and has been concretely designed and constructed by using different technologies: Physically Uncloneable Functions (PUFs) [Pap01, PRTG02]. A PUF is a device generated through a special physical process such that producing a clone is considered infeasible. Once a PUF has been constructed, there is a physical procedure to query it, and to measure its answers.

The apparent similarity of PUFs with programmable tamper-proof hardware tokens 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[1], since it knows the program that the token runs. Tamper-proof tokens are realized through ad-hoc procedures that models them as black boxes, their internal content is protected from physically 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, 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.

The use of PUFs in cryptographic protocols is rapidly increasing but various incomparable models have been used so far, therefore producing incomparable results that moreover do not necessarily hold when some real-world PUFs are used. A very recent breakthrough in modeling PUFs for cryptographic protocols has been presented in [BFSK11].

The model of Brzuska et al. [BFSK11]. In the model of Brzuska et al. the Universal Composition (UC) Framework of Canetti is augmented by including PUFs modeled with only basic properties. Even if there exist technologies for the construction of PUFs that enjoy other advanced properties, the choice of Brzuska et al. is to consider only the basic properties obtained by all current realizations of PUFs. Such properties are unpredictability and uncloneability. The first property informally means that the only way to know the output of a PUF on input a given value, consists in querying it. The second property is that PUFs are unique in their input/output behavior, therefore without possession of a given PUF, it is not possible to predict its output to a query, even in case outputs of other queries have been previously obtained. Since the PUF model of Brzuska et al. is integrated in the UC framework, PUFs are defined through an ideal functionality.

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

1.1 Our Contribution

Relaxing the model of Brzuska et al. The model of Brzuska et al. considers *trusted* PUFs only (i.e., adversaries are assumed to be unable to produce fake/malicious PUFs). Moreover it is assumed that the simulator can observe queries made by an adversary to a trusted PUF (i.e., an adversary can access a honest PUF only through a detectable and understandable process).

In this work we continue this research direction by focusing on relaxing the two above restrictions. Indeed given that the study of PUFs is still in its infancy, it is risky to rely on assumptions on the capabilities of the adversaries in generating and accessing PUFs adversarially. Our goal consists in presenting security models that are resilient to the above plausible real-world attacks (viz., adversaries producing malicious PUFs, or adversaries making "hidden" queries to PUFs so that the simulator can not observe these queries).

We present two relaxations of the model of Brzuska et al. and in both cases we give positive answers to the question of achieving universally composable secure computation with PUFs. The importance of our contribution lies precisely in improving the goal of Brzuska et al. that chose to give a "very minimalistic model" (see discussion in the second page of [BFSK11]). Indeed their formulation has two strong requirements that could not be achieved by real-world PUFs, namely: 1) impossibility of generating malicious PUFs; 2) impossibility of finding alternative procedures (unknown to the others) to query a honest PUF. Not only we present two interesting relaxed formulations, but we also show that in both cases UC secure computation is possible. More details follow below.

Malicious PUF generation. We augment the UC framework with untrusted (malicious) PUFs, so that an adversary is assumed to be able to produce fake PUFs (e.g., stateful, programmed with malicious code). The natural question is whether UC security can be achieved in such a much more hostile setting. We give a positive answer by relying on the (realistic) assumption that standard computational assumptions still hold in presence of PUFs.

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 constructions are secure. 2) There exists a PPT adversary that can generate a PUF that breaks standard assumptions; in this case our constructions are 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.

Malicious access to PUFs. 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. The reason is that we *do not* want to make the assumption that an adver-

sary has *only black-box access* to the PUF[2]. 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. In this case, it is not clear to someone observing the system what the corresponding query/response pair is. 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]), 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 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, in presence of such non-black-box simulators, it should still be hard for the simulator to understand a query, when a protocol is executed in the universal composability framework. Indeed the order of quantifiers in the UC definition is $\forall A \exists S$, and therefore if an adversary is able to query a PUF in some special way, there should exist a simulator that can understand the query and the answer. However this objection fails when one would like to follow the spirit of UC security. Indeed 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: $\forall A \exists S \forall \mathcal{Z}$. It is therefore possible that an adversary does not know how to query a PUF in some alternative way, and therefore the simulator does not know it either[3]. 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. Notice that the UC definition allows the environment to use the adversary as a proxy. Therefore following this modeling, when the adversary holds a PUF the environment is allowed to ask the adversary to query the PUF in a different way, sending some data X. The adversary performs this procedure and sends back to the environment some data Y. Now the environment knows that X correspond to a query x, and the obtained data Y corresponds to an output y. However both the adversary and the simulator are unaware of these mappings, and therefore they have no idea about which query has been asked and which answer has been obtained[4].

Clearly because of this lack of information, the adversary could not continue a protocol that needs the input/output of the PUF (i.e., x/y in the above example) without knowing such values. However this is not an issue for the adversary in the UC framework. Indeed it can still continue the protocol acting as a proxy with the environment that therefore would be the actual adversary using

[2]Contrast this scenario with the case of tamper-proof hardware tokens. In the case of these tokens, due to the "tramper-proof" assumption, an adversary can only observe the input/output behavior of the token, and thus it is justified for the simulator to observe an adversary's queries. However, as we argue next, the case of PUFs is different.

[3]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.

[4]We stress that this behavior does not violate unpredictability since the PUF has been actually queried with input x, but since a different procedure has been used, the query is not detected by the simulator.

a PUF but hiding his queries. The PUF is however still accessible by the adversary, and therefore still accessible by the simulator too (that can still query it as already allowed in the definition of [BFSK11]).

Obviously if the environment instructs the adversary to physically access the PUF in some special way, one should also take into account the fact that the environment might ask the adversary to destroy the PUF[5]. Therefore one might have an adversary that is able to access the PUF in some undetectable way (i.e., the simulator does not catch queries asked by the adversary and the corresponding answers given by the PUF), and then immediately destroys the PUF (i.e., later on the simulator will no be able to query the PUF again). In this case it seems that both advantages of the simulator with respect to malicious players considered in the definition of [BFSK11] disappear and therefore UC security can not be achieved. Indeed, since the simulator (because of the oblivious access to a PUF) can not observe the queries asked by the adversary and (because of the capability of destroying a PUF after its use) can not query a PUF after adversary's queries, it is not clear how one can get any reasonable advantage to achieve UC security, which requires straight-line simulation (i.e., straight-line extraction of the input of the adversary and straight-line equivocality of the input played on behalf of the honest party).

Very surprisingly we show that in this model unconditional UC secure computation is possible! The construction given in [BFSK11] clearly fails in such a relaxed model, but we use it as a building block along with other techniques to design our construction therefore obtaining security against stronger adversary and thus improving their result.

2 Definitions

Notation. We denote by n the security parameter and by PPT the property of an algorithm of running in probabilistic polynomial-time.

We denote by $(v_A, v_B) \leftarrow \langle A(a), B(b)\rangle(x)$ the random process obtained by having A and B interacting on common input x and on (private) auxiliary inputs a and b to A and B, respectively (if any), and with independent random coin tosses for A and B, and by (v_A, v_B) the local outputs of parties A and B, respectively, after the completion of their interaction. When the common input x is the security parameter, we omit it. We use $v \stackrel{\$}{\leftarrow} \mathsf{Alg}()$ when the algorithm $\mathsf{Alg}()$ is randomized. We denote by $\mathsf{view}_A(A(a), B(b))(x)$ the view of A of the interaction with player B, i.e., its values is the transcript $(\gamma_1, \gamma_2, ..., \gamma_t; r)$, where the γ_i's are all the messages exchanged and r is A's coin tosses. Finally, 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.

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, $SD(X,Y) = \frac{1}{2}\sum_{z\in\{0,1\}^n} |\Pr[\, X = z\,] - \Pr[\, Y = z\,]|$.

2.1 Physically Uncloneable Functions

In this section we follow the definitions given in [BFSK11]. A Physically Uncloneable Function (PUF) is a noisy physical source of randomness. The randomness property comes from an uncon-

[5]Indeed in practice PUFs are very easy to break, and destroying a PUF does not violate the assumption that all PUFs are honest. Instead, there is no way for the adversary to tweak the PUF changing its input/output behavior, since this would violate the assumption that there exists honest PUFs only.

trollable 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. Algorithm Sample abstracts the PUF fabrication process and works as follows. Given the security parameter in input, 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. Without loss of generality, 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 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 samplable 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 following we use $\mathsf{PUF}_{\mathsf{id}}(q)$ to denote $\mathcal{D}_{\mathsf{id}}$. 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. Furthermore, we assume that PUFs are tamper-evident. In the following we recall the concept of average min-entropy, and the formal definition of PUF-unpredictability.

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

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

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 $\mathsf{PUF}(q_1), \ldots, \mathsf{PUF}(q_\ell)$ each corresponding to an evaluation of the PUF on challenge q_k.

Definition 3 (Unpredictability). *A (rg, d_{noise})-PUF family $p = (\mathsf{Sample}, \mathsf{Eval})$ for security parameter n is $(d_{\mathsf{min}}(n), m(n))$-unpredictable if for any $q \in \{0,1\}^n$ and challenge list $\mathcal{Q} = (q_1, \ldots, q_n)$, one has that, if for all $1 \leq k \leq n$ the Hamming distance satisfies $\mathsf{dis}_{\mathsf{ham}}(q, q_k) \geq d_{\mathsf{min}}(n)$, then the average min-entropy satisfies $\tilde{H}_\infty(\mathsf{PUF}(q)|\mathsf{PUF}(\mathcal{Q})) \geq m(n)$. 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). *Let dis be a distance function for metric space $\mathcal{M}$. 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 $d_{\mathsf{min}} = o(n/\log n)$, a *matching* fuzzy extractor has the following parameters: $\ell(n) = n$, $t(n) = d_{\mathsf{noise}}(n)$, and ϵ is a negligible function in 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 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 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 we show that as long as standard computational assumptions still hold when PPT adversary 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. 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 the parties to rely on only the PUFs that they themselves manufacture, and not on the ones they receive. We think of a malicious-PUF family as a PUF-family that does not guarantee the security property of Definition 3. Of course, we also want the honest parties to be able to obtain and send honestly generated PUFs. Thus our ideal functionality for PUFs, $\mathcal{F}_{\mathsf{PUF}}$ (Figure 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 specifies a $\mathtt{mode} \in \{\mathtt{normal}, \mathtt{mal}\}$ to $\mathcal{F}_{\mathsf{PUF}}$, and the ideal functionality uses the corresponding 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.

The handover procedure remains the same as in the original formulation of Brzuska et al. [BFSK11]. Each PUF has a status flag which determines whether it is in transit or not. We allow the PUF to be queried by the adversary while it is in transit. Thus, when a party P_i hands over a PUF to party P_j, that PUF's flag is changed from $\mathsf{notrans}$ to trans, 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 handover is complete. The ideal functionality is described formally in Figure 1.

3.2 Constructions for UC Security in the Malicious PUFs model

In this section we start presenting a construction that UC-realizes the $\mathcal{F}_{\mathsf{com}}$ functionality. We then show how to obtain UC security for any functionality.

Recall that the major difficulty when using (honest) PUFs (in contrast to tamper-proof tokens) is that PUFs are *not programmable*. This means that the simulator must honestly answer all queries made by the adversary by forwarding them to the actual PUF. Thus, the simulator can only exploit the power of observing the queries made by the adversary to the PUF received from the honest party, and of running the PUF when it is still possessed by the adversary. In the honest PUF model, the simulator is guaranteed that any PUF sent by the adversary is honest. This means that the answers received from such a PUF are unpredictable for the adversary as well, and more importantly, the adversary has no control of the behaviour of the PUF. This fact is of great help in designing protocols with honest PUFs.

In the malicious PUFs model instead, the behaviour of a PUF sent by an adversary is entirely in the adversary's control. This means that the malicious PUF can answer (or even abort) adaptively on the query according to some pre-shared strategy with the malicious creator. Also, even though the malicious PUF can now be programmable, the honest PUF is still unpredictable. In particular, the simulator is still constrained to send an honest PUF which outputs are therefore out of the control of the simulator.

$\mathcal{F}_{\mathsf{PUF}}$ is parameterized by PUF families $\mathcal{P}_1 = (\mathsf{Sample}_{\mathtt{normal}}, \mathsf{Eval}_{\mathtt{normal}})$ with parameters $(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, m)$, and $\mathcal{P}_2 = (\mathsf{Sample}_{\mathtt{mal}}, \mathsf{Eval}_{\mathtt{mal}})$, and 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 $(\mathsf{init}_{\mathsf{PUF}}, \mathsf{sid}, \mathtt{mode}, P_i)$ 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 $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, P_i, \mathsf{notrans})$ 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 $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, P_i, \mathsf{notrans})$ 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 $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \perp, \mathsf{trans}(*))$.
  - 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 $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ in $\mathcal{L}$.
- 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 $\mathcal{L}$. If not, return to the waiting state. Else, write this tuple to the communication input tape of P_i.

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

The high-level idea behind our protocol. The protocol is shown in Figure 2. We start providing the high-level idea of the protocol, and then we show how to actually implement it.

To commit to a bit, the committer C sends a (regular[6]) commitment of that bit over the channel, while it sends the opening of such commitment to a PUF received from the receiver R. The idea is that the simulator will observe the adversary's PUF queries (i.e., the opening of the regular commitment) and extract the committer's bit. To enforce C to query the PUF *before* the commitment phase is over (otherwise simulator will not be able to extract), we require it to also send the commitment to the answer that the PUF gives on input the opening of the commitment. Note that the committer can not send the answer itself as the PUF is malicious. Later, in the decommitment phase, the committer will send the openings of both commitments: the commitment of the bit and the commitment of the answer of the PUF. By the binding of commitment scheme, and by the unpredictability of PUFs, the committer is forced to query the PUF with the valid opening already in the commitment phase, in order to be able to provide an accepting decommitment phase. Hence, the simulator extracts the bit committed with all but negligible probability. Thus, this is an extractable commitment scheme.

Now let us look at a malicious receiver. In this case, the simulator needs to equivocate. To achieve this, instead of using a regular commitment as described above, we use an *equivocal* regular commitment from the committer to the receiver. The problem is that the simulator must equivocate in *straight line*. To achieve this, we augment an existing equivocal commitment scheme in the standard model with the use of PUFs to get the desired result. In particular, we use the equivocal commitment scheme from [CO99], which in turn is based on Naor's scheme [Nao89]. Naor's scheme consists of two messages, where the fist message is a randomly chosen string r that the receiver sends to the committer. The scheme has the property that if the string r is crafted appropriately, then the commitment is equivocal. Di Crescenzo and Ostrovsky [CO99] show how this can be achieved by adding a coin-tossing phase before the commitment. To make this approach work in our case, we need to implement the coin tossing in such a way that a straight-line simulator can force the output to the desired result. In particular, the coin tossing in the scheme 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, so that the Naor parameter is set as $\alpha \oplus \beta$. The simulator would be able to equivocate if it can extract the value α committed by R, before having to send β. For this purpose, we construct a straight-line extractable statistically hiding commitment scheme in the malicious PUF model. This scheme follows the idea of our extractable commitment described above but is tailored to be also statistically-hiding. It will be used by the receiver to commit to α. Once the string r is properly crafted, the simulator completes the commitment phase as follows: 1) it commits to a random bit, 2) it does not query the PUF with the opening of such commitment (indeed, since we are in the malicious PUF model, querying the PUF more that an honest party would do, can be fatal for the simulation), 3) it commits to a random string instead of the answer of the PUF. In the decommitment phase, once it receives the bit b to open to, the simulator plays the decommitment as follows. It first computes the equivocation of the commitment of the bit such that it opens to b, then it queries the PUF with such decommitment and obtains the answer. Finally it computes the equivocation of the commitment of the string, such that it opens to the answer received from the PUF.

The final construction consists of the extractable commitment of α sent by R to C, the message β sent by C to R and the extractable/equivocal commitment of the bit, and the answer of the PUF,

[6]By "regular" commitment, we mean a commitment scheme in the plain model.

sent by C to R.

The protocol sketched so far involves back-and-forth for PUFs. This is because the protocol involves a party P_i committing to the response of a random query (secret to P_j) to the PUF created by P_j, and then later proving that it had indeed committed to the correct response. However, because of unpredictability, to check the correctness of such response, P_j needs to query its own PUF and this involves P_i sending the PUF back to P_j. In the actual implementation we show how to overcome this problem.

The actual implementation. The receiver starts by querying its PUF with a pair of queries (q_0, q_1), and then sends the PUF to C. Then, to commit to a bit b the committer queries the PUF with query q_b (instead of the opening of the commitment of b), and sends to R only the commitment of the answer received from the PUF. C obtains the desired query q_b by running an Oblivious Transfer (OT) protocol with R. 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, C can not get both queries (thus confusing the simulator), neither R can detect which query has been transferred. This idea improves the straight-line extractable commitment scheme discussed above (for now just consider the originally proposed computationally hiding version) since this implementation does not require back-and-forth of PUFs.

Implementing the statistically-binding straight-line extractable and equivocal commitment $\mathsf{Com}_{\mathsf{equiv}}$. Using the above implementation of a straight-line extractable (computationally hiding) commitment scheme that avoids back-and-forth of PUFs, new issues arises. The first issue concerns the equivocation. Indeed, now the simulator has to query the token with a particular string, chosen by the receiver, and it has to choose it already in the commitment phase, when it does not know yet which is the bit to open to. We overcome this difficulty as follows. After C has sent its commitment of the answer, R reveals both queries (q_0, q_1) and also randomness used to run the OT protocol. In this way the simulator gets both strings and, in the opening phase, will query the PUF with the one corresponding to the bit to open. Another subtle issue is the selective abort of a malicious PUF. If the PUF aborts when queried with a particular string, then we have that the sender would abort already in the commitment phase, while the simulator aborts only in the decommitment phase. We avoid such problem by requiring that the sender 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 the output of the coin flipping, it is also straight-line equivocal. Since to commit to a bit we are basically committing to the l-bit string answer of the PUF, the size of Naor's parameter is $N = (3n)l$. We denote such a protocol as $\mathsf{Com}_{\mathsf{equiv}}$. The formal specification of the protocol is given in Figure 7, and the proof is shown in Appendix B.2.

Implementing the statistically-hiding extractable commitment scheme $\mathsf{Com}_{\mathsf{shext}}$. Another issue concerns the fact that, in order to use the arguments for binding of Naor's commitment, we need that the extractable commitment sent by the receiver, i.e., the commitment of the string α for the coin flipping, to be statistically hiding. Thus we can not reuse the same protocol $\mathsf{Com}_{\mathsf{equiv}}$ discussed above. We will obtain the desired scheme by applying following modification to the straight-line extractable (but only computationally hiding) commitment scheme that avoids back-and-forth of PUFs, as we discussed above.

First, the answer received from the PUF is committed using a statistically hiding commitment scheme, and the OT protocol must be such that the receiver's privacy is statistical. Second, after

the commitment, the receiver (that is playing as committer for α) has to provide a statistical zero-knowledge argument of knowledge of the message committed. This turns out to be necessary to argue about binding. Such protocol is statistically hiding and straight-line extractable. We denote such a protocol as $\mathsf{Com}_{\mathsf{shext}}$. The formal specification of $\mathsf{Com}_{\mathsf{shext}}$ is given in Figure 6, while the formal proof is shown in Appendix B.1.

The final protocol $\mathsf{Com}_{\mathsf{uc}} = (\mathsf{C}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}})$ is shown in Figure 2 and consists of the receiver committing to α using the statistically hiding straight-line extractable commitment scheme $\mathsf{Com}_{\mathsf{shext}} = (\mathsf{C}_{\mathsf{shext}}, \mathsf{R}_{\mathsf{shext}})$, the sender choosing β and the result of the coin flipping, $\sigma = \alpha \oplus \beta$, is used as *common input* to run the statistically binding straight-line extractable and straight-line equivocal $\mathsf{Com}_{\mathsf{equiv}} = (\mathsf{C}_{\mathsf{equiv}}, \mathsf{R}_{\mathsf{equiv}})(\sigma)$. The protocol needs two PUFs (one for each party) that are exchanged only at the beginning of the protocol.

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 2: 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 PUF 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 PUF model, then Protocol $\mathsf{Com}_{\mathsf{uc}} = (\mathsf{C}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}})$ depicted in Figure 2, UC-realizes the $\mathcal{F}_{\mathsf{com}}$ functionality.*

The proof is provided in Appendix C.1.

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* [7]. 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.

[7] We remark that however our protocol enjoys parallel composition and reusing of the same PUF.

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 the 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 values, 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.

4 Honest PUFs with Oblivious Queries

We describe a more relaxed (we consider stronger adversaries) PUF access model in this section. In particular, we study the question: what is the power of the simulator in the PUF-model? In their original formulation, Brzuska et al.([BFSK11]) made the model *non-programmable*. Their motivation was that PUFs are intrinsically non-programmable in the sense that even the manufacturer does not have control over the PUF functionality. Moreover, as a PUF is considered to implement a physical process, it can not be assumed that a PPT simulator will be able to reproduce that distribution indistinguishably. Thus, the simulator must answer the adversary's queries faithfully by forwarding them to the PUF.

A different axis along which we can analyze the power of the simulator is whether it can observe the queries made by the adversary to the PUF or not. The protocols in Brzuska et al.([BFSK11]) crucially rely on the simulator's ability to observe the adversary's queries to the PUF.

We consider a relaxation of the model where the simulator may not observe the adversary's queries. Our motivation of this model comes from the possibility of an adversary to maliciously run the Eval process. Even for an honest PUF, an adversary may use a different physical process (that is, different from the prescribed procedure Eval) to obtain an *obfuscated* query-response pair that is hidden from anyone observing the system. Indeed, as PUFs are generated by a known physical process, we assume that the adversary has a *non black-box* access to the PUF. Thus a more realistic model would be allowing the adversary's queries to remain hidden from the simulator. In particular this is the case when considering the UC framework and the fact that the simulator does not know the code of the environment, and the environment can use the adversary as a proxy, therefore using procedures to query the PUF that are not understood by the simulator. See Section 1 for a more detailed discussion.

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. When the adversary makes a PUF query, the simulator sends it to $\mathcal{F}_{\mathsf{hPUF}}$, and sends the response back to the adversary. However, the simulator's next message function does not depend upon the query or the response. We call such simulators *oblivious-query* simulators. The ideal functionality $\mathcal{F}_{\mathsf{hPUF}}$ is described in Figure 9 in Appendix E.

Before we begin, we address one final technical issue concerning oblivious-query simulators. As the simulator 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, else we are in the plain model where UC is impossible). Now in the real physical 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 $\mathsf{augmented} - \mathcal{F}_{\mathsf{hPUF}}$ hybrid model. However, this creates a 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 the PUF, the PUF is already dead and can not be accessed and the simulator is stuck. This attack must be handled explicitly in the protocol.

However, there exists a simple compiler that transforms a protocol in $\mathcal{F}_{\mathsf{hPUF}}$ hybrid model (where parties are not allowed to send kill message to $\mathcal{F}_{\mathsf{hPUF}}$) to a protocol in the $\mathsf{augmented} - \mathcal{F}_{\mathsf{hPUF}}$ 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 hasn't 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 $\mathsf{augmented} - \mathcal{F}_{\mathsf{hPUF}}$ 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 Query Model

In this section, we construct an unconditional UC protocol for OT in the oblivious-query PUF model. Recall that in this 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. A formal description of our protocol is given in Figure 3.

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 the 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[8]. 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[9]. 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 of d_i to 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 of 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.

Combining OTs. If the above cut-and-choose does not abort, consider the indices that were not "revealed". We have k pairs of queries $(v_j \oplus x^0_j, v_j \oplus x^1_j)$, such that for almost all pairs, (1) the receiver knows the response (of $\mathsf{sid}^{\mathsf{R}}$) to only one query in each pair, and the sender does not know which query the receiver knows, and (2) the simulator knows the query whose response is known by the receiver. Now we run the original OT protocol of Brzuska et al. [BFSK11] on these pairs to implement k random OTs. As the final step, we use the well-known reduction from random OTs to OT.

The proof of the following theorem is deferred to Appendix C.2.

[8] 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.

[9] 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.

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 R.
2. **[($\mathsf{S}_{\mathsf{uncOT}} \Leftarrow \mathsf{R}_{\mathsf{uncOT}}$): Receiver PUF initialization]** R initializes a PUFs $\mathsf{sid}^{\mathsf{R}}$. It uniformly chooses $2k$ queries $q_1, \ldots, q_k$ and obtains responses $a_1, \ldots, a_k$. 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 $\delta \in \{0,1\}$, choose random strings $s_1^\delta, \ldots, s_k^\delta$ such that $s^\delta = \bigoplus_{j=1}^k s_j^\delta$.
   - for $\delta \in \{0,1\}$, 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 sid_{i_j} to query q_{i_j}.
   - for $\delta \in \{0,1\}$ 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 3: Unconditional OT protocol $(\mathsf{S}_{\mathsf{uncOT}}, \mathsf{R}_{\mathsf{uncOT}})$ in the Oblivious Query Model.

Theorem 2. *The protocol* $(\mathsf{S}_{\mathsf{uncOT}}, \mathsf{R}_{\mathsf{uncOT}})$ *in Figure 3, UC-realizes the* $\mathcal{F}_{\mathsf{OT}}$ *functionality in the* $\mathcal{F}_{\mathsf{PUF}}$*-hybrid model.*

5 Unconditional Security with Malicious PUFs

We turn our attention back to the malicious PUF model. In Section 3.2 we provided a computational commitment scheme using malicious PUFs. This combined with the results in Canetti et al. [CLOS02] gives us computational UC security with malicious PUFs. The natural question to ask is whether we can leverage the power of PUFs to obtain *unconditional* UC security. We leave this as an intriguing 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.

Our protocol is the straightforward 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[10], 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, along with security proofs, are given in Appendix D.

Acknowledgments

Supported in part by NSF grants 0830803, 09165174,1065276, 1118126 and 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, and Lockheed-Martin Corporation Research Award. 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. The work of the second and third authors has been done while visiting UCLA and is supported in part by the European Commission through the FP7 programme under contract 216676 ECRYPT II.

References

[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.

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

[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.

[BG93] Mihir Bellare and Oded Goldreich. On Defining Proofs of Knowledge. In *Advances in Cryptology – Crypto '92*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420. Springer Verlag, 1993.

[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.

[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.

[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 pre-processing. 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 *Proceedings of the 8th conference on Theory of cryptography*, TCC'11, Berlin, Heidelberg, 2011. Springer-Verlag. To appear.

[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.

[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.

[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.

[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.

[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.

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

[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.

[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 $\epsilon(n) < 1/p(n)$.

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 5. *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., $\{X_w\}_{w\in\mathcal{W}} \overset{\mathsf{C}}{\equiv} \{Y_w\}_{w\in\mathcal{W}}$ (resp., $\{X_w\}_{w\in\mathcal{W}} \overset{\mathsf{S}}{\equiv} \{Y_w\}_{w\in\mathcal{W}}$) 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 6 (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 (with probability 1).*

Statistical (resp., Computational) Hiding. *For every (resp., 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 statistically (resp., 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 $\mathsf{C}(\mathtt{com}, b)$.*

Statistical (resp., Computational) Binding. *For every (resp., 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.*

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 6 are required to hold over the random choice of σ.

Definition 7 (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* $\mathbf{Exp}^{\mathcal{S}}_{\mathsf{R}^*}(n)$:
$\sigma \xleftarrow{\$} \{0,1\}^{\ell(n)}$;	$(\tilde{\sigma}, \mathsf{state}_1) \xleftarrow{\$} \mathcal{S}_1(1^n)$;
$((c,d),c) \leftarrow \langle \mathsf{C}(\mathtt{com}, b), \mathsf{R}^*(\mathtt{recv})\rangle(\sigma)$;	$(\mathsf{state}_2, \tilde{c}) \leftarrow \langle \mathcal{S}_2(\mathsf{state}_1), \mathsf{R}^*(\mathtt{recv})\rangle(\tilde{\sigma})$;
$\mathtt{return}\ \mathsf{R}^*(\sigma, b, c, d)\rangle$;	$\tilde{d} \leftarrow \mathcal{S}_3(\sigma, \mathsf{state}_2, b); \mathtt{return}\ \mathsf{R}^*(\tilde{\sigma}, \tilde{c}, b, \tilde{d})$;

Note that in this definition, the verification of the receiver R *is computed upon the common input also.*

Definition 8 (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* $b^\star$ *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* c *is* accepting *then* $b^\star \neq \perp$*.*
- *(binding) if* $b^\star \neq \perp$ *the probability that* C^* *decommit to a bit* $b \neq b^\star$ *is negligible.*

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 9. *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*

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

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

Argument of knowledge. Informally, a proof system is an argument of knowledge if for any probabilistic polynomial-time interactive algorithm prover P^* that convinces an honest verifier with non-negligible probability, there exists a probabilistic polynomial-time algorithm called the extractor, that outputs a valid witness for the statement proved by P^* with roughly equivalent probability. Formally, we have the following definition.

Definition 10. *(adapted from [BG93] with negligible knowledge error) A proof (resp., argument) system* $\langle P(\cdot), V\rangle(x)$ *for an* **NP***-language* L *is a* proof (resp., argument) system of knowledge *if there exists a probabilistic polynomial-time algorithm* E *such that for every (resp., every polynomial-sized) circuit family* $\{P_n^*\}_{n\in N}$*, there exists a negligible function* ϵ *such that for any* x *of size* n*, if* $\Pr[\,\mathsf{out}(\langle P_n^*, V\rangle(x)) = 1\,] = p(|x|)$*, then* $\Pr\left[\, w \leftarrow \mathsf{E}^{P_n^*(x)}(x) : R_L(x, w) = 1\,\right] = p(|x|) - \epsilon(|x|)$ *and the expected running time of* E *is polynomial in* $|x|$.

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 Π **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 $\mathcal{Z}$,*

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

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. 4.

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

Commitment Functionality. The ideal functionality for Commitment Scheme as presented in [CF01], is depicted in Fig. 5. Such definition captures the hiding and binding property defined in Definition 6.

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

Figure 5: 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 Section 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 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. 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 Sub-Protocols

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 provide in Section D.

B.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: $\mathcal{R}_{\mathtt{com}} = \{(c, (s, d)) \text{ such that } \mathsf{R}_{\mathsf{SH}}(c, s, d) = 1\}$. Protocol $\mathsf{Com}_{\mathsf{shext}} = (\mathsf{C}_{\mathsf{shext}}, \mathsf{R}_{\mathsf{shext}})$ is depicted in Fig.6.

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 $\mathsf{C}_{\mathsf{shext}}$.

$\mathsf{R}_{\mathsf{shext}} \Leftrightarrow \mathsf{C}_{\mathsf{shext}}$: (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, $a'_b \xleftarrow{\$} \{0,1\}^n$.

$\mathsf{C}_{\mathsf{shext}} \Leftrightarrow \mathsf{R}_{\mathsf{shext}}$: (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}.

$\mathsf{C}_{\mathsf{shext}} \Leftrightarrow \mathsf{R}_{\mathsf{shext}}$: ($\mathcal{SZK}$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 6: Statistically Hiding Straight-line Extractable Bit Commitment Scheme $(\mathsf{C}_{\mathsf{shext}}, \mathsf{R}_{\mathsf{shext}})$.

Theorem 3. *If* $\mathsf{Com_{SH}} = (\mathsf{C_{SH}}, \mathsf{R_{SH}})$ *is a statistically-hiding commitment scheme,* $(\mathsf{S_{OT}}, \mathsf{R_{OT}})$ *is a statistical receiver OT protocol and* (P, V) *is a* $\mathcal{SZK}\mathsf{AoK}$*, then* $\mathsf{Com_{shext}}$ *is a statistically hiding straight-line extractable bit commitment scheme in the malicious PUFs model.*

Proof. **Completeness.** Before delivering its own PUF $\mathsf{PUF_R}$, $\mathsf{R_{shext}}$ 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, $\mathsf{C_{shext}}$ has to commit to the output a_b of $\mathsf{PUF_R}$.

By the completeness of the OT protocol, $\mathsf{C_{shext}}$ obtains the query q_b corresponding to its secret bit. Then $\mathsf{C_{shext}}$ queries $\mathsf{PUF_R}$ with q_b and commits to the response a'_b running $\mathsf{C_{SH}}$. Furthermore, $\mathsf{C_{shext}}$ proves using $\mathcal{SZK}\mathsf{AoK}$ the knowledge of the opening. By the completeness of $\mathcal{SZK}\mathsf{AoK}$ and $\mathsf{Com_{SH}}$ the commitment phase is concluded without aborts. In the opening phase, $\mathsf{C_{shext}}$ sends b and opens the commitment to a'_b, and $\mathsf{R_{shext}}$ 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, $\mathsf{R_{shext}}$ gets the correct answer and accept the decommitment for the bit b.

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

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

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

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

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

H_3**:** In this hybrid, $\mathsf{C_{shext}}$ 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 $(\mathsf{S_{OT}}, \mathsf{R_{OT}})$, hybrids H_2 and H_3 are statistically indistinguishable.

H_4**:** In this hybrid, $\mathsf{C_{shext}}$ 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 $\mathsf{PUF^*_R}$ aborts, then $\mathsf{C_{shext}}$ 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, $\mathsf{C_{shext}}$ proceeds as in H_4 except that it commits to the string a'_1. By the statistically hiding property of protocol $\mathsf{Com_{SH}}$, 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 8. 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.

<u>Extractor E</u>

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 1 ($\Pr[\, \mathbf{Fail1} \,]$ is negligible)**.** *If* $(\mathsf{S_{OT}}, \mathsf{R_{OT}})$ *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 $\mathsf{S_{OT}}$, on input auxiliary information $z = (s_0, s_1)$, while it runs $\mathsf{C}^*_{\mathsf{shext}}$ internally. $\mathsf{R}^*_{\mathsf{OT}}$ initializes and sends $\mathsf{PUF_R}$ to $\mathsf{C}^*_{\mathsf{shext}}$, then it runs the OT protocol forwarding the messages received from the external sender $\mathsf{S_{OT}}$ 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 $\mathsf{PUF_R}$. 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 $(\mathsf{S_{OT}}, \mathsf{R_{OT}})$ is a stand-alone secure OT protocol, δ must be negligible. □

Lemma 2 ($\Pr[\, \mathbf{Fail2} \,]$ is negligible)**.** *Assume that* τ_c *is an accepting transcript. If* $\mathsf{Com_{SH}} = (\mathsf{C_{SH}}, \mathsf{R_{SH}})$ *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 $\mathsf{R_{shext}}$ accepts, i.e., the opening (d) of the statistically hiding commitment is valid *and* corresponds to an answer (a'_b) of $\mathsf{PUF_R}$ upon one of the queries played by the $\mathsf{R_{shext}}$ in the OT protocol. Formally, $\mathsf{R_{SH}}(c, a'_b, d) = 1$ 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 $\mathsf{PUF_R}$ 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 $\mathsf{PUF_R}$. By the unpredictability of $\mathsf{PUF_R}$ we have that this case has negligible probability to happen.
2. $\mathsf{C}^*_{\mathsf{shext}}$ queries $\mathsf{PUF_R}$ 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 $\mathsf{Com_{SH}}$ 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 $\mathsf{Com_{SH}}$ with probability δ. $\mathsf{C}^*_{\mathsf{SH}}$ runs $\mathsf{C}^*_{\mathsf{shext}}$ internally, simulating the honest receiver $\mathsf{R_{shext}}$ to it, and forwarding only the messages belonging to $\mathsf{Com_{SH}}$ to an external receiver $\mathsf{R_{SH}}$, and vice versa. Let c denote the transcript of $\mathsf{Com_{SH}}$. When the commitment phase of $\mathsf{Com_{shext}}$ is successfully completed, and therefore $\mathsf{C}^*_{\mathsf{shext}}$ has provided an accepting proof for the theorem $(c, \cdot) \in \mathcal{R}_{\mathsf{com}}$, $\mathsf{C}^*_{\mathsf{SH}}$ 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 $c \in \mathcal{R}_{\mathsf{com}}$ 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 2, 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 3. *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 2 we use the same notation $\mathsf{Com}_{\mathsf{shext}}$ to refer to a string commitment scheme.

B.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. 7.

Theorem 4. *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:

- **Fail1.** If there exists a pair $q'_0, q'_1 \in \mathcal{Q}_{\mathsf{precom}}$ such that $\mathsf{dis}_{\mathsf{ham}}(q_0, q'_0) \leq d_{\mathsf{min}}$ and $\mathsf{dis}_{\mathsf{ham}}(q_1, q'_1) \leq d_{\mathsf{min}}$, output $b^\star = \perp$.
- **Fail2.** If for all $q' \in \mathcal{Q}_{\mathsf{precom}}$ 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 $\mathsf{dis}_{\mathsf{ham}}(q_0, q') \leq d_{\mathsf{min}}$ then output $b^\star = 0$;

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 Biding Commitment)

1. $a'_b \leftarrow \mathsf{PUF}_{\mathsf{R}}(q'_b)$. If $\mathsf{PUF}_{\mathsf{R}}$ aborts, $a'_b \xleftarrow{\$} \{0,1\}^n$.
2. for $1 \leq i \leq l$, 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 7: Statistically Binding Straight-line Extractable and Equivocal Commitment $(\mathsf{C}_{\mathsf{equiv}}, \mathsf{R}_{\mathsf{equiv}})$.

2. If there exists $q' \in \mathcal{Q}_{\mathsf{precom}}$ such that $\mathsf{dis}_{\mathsf{ham}}(q_1, q') \leq d_{\mathsf{min}}$ 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 3, 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,

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

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 1. 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 biding 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 $s^i_0 \leftarrow \{0,1\}^n$, $\alpha^i_0 \leftarrow G(s^i_0)$;
2. pick $s^i_1 \leftarrow \{0,1\}^n$, $\alpha^i_1 \leftarrow G(s^i_1)$;
3. $r_i = \alpha^i_0 \oplus \alpha^i_1$.

Output $r_1, \ldots, r_l$, $\mathsf{state}_1 = \{s^i_0, s^i_1\}_{i \in l}$;

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

- 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 $i = 1, \ldots, l$, $\tilde{c}_i \leftarrow G(s^i_0)$. 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 4. *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 $\mathbf{Exp}_{\mathsf{R}_{\mathsf{equiv}}^*}^{\mathcal{S}}$.

□

□

C UC Security Proofs

C.1 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 2 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 an simulator Sim is almost straightforward.

Receiver is corrupted. 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 $\mathsf{Com}_{\mathsf{shext}}$.

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 5. *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}. □

Committer is corrupted. 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 6. *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.

□

C.2 Proof of Theorem 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 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 $st^0_{i_j}$ and $st^1_{i_j}$. 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 [BFSK11] Appendix A, 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 $\hat{q}^0_{i_j}$ and $\hat{q}^1_{i_j}$.*

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 $q_{i_j}^{\rho}$, and not $q_{i_j}^{1-\rho}$.
- As $c_{i_j} = 0$, we have $b'_{i_j} = \rho$.

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

D On Unconditional Security with Malicious PUFs

The formal description of the protocol $(\mathsf{C}_{\mathsf{uncon}}, \mathsf{R}_{\mathsf{uncon}})$ is given in Figure 8. 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 extractor $(\mathsf{FuzGen}, \mathsf{FuzRep})$ of appropriate matching parameters such that $\ell = 3n$.

Completeness of protocol $(\mathsf{C}_{\mathsf{uncon}}, \mathsf{R}_{\mathsf{uncon}})$ follows from response consistency. We focus on hiding and binding properties.

Lemma 7 (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. □

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

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[11], 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 8 (Binding)**.** *For any malicious committer $\mathsf{C}^*_{\mathsf{uncon}}$, the probability that it wins in the binding game of Definition 6 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

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

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}.

□

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

$\mathcal{F}_{\mathsf{PUF}}(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, m)$ 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 9: The ideal functionality $\mathcal{F}_{\mathsf{hPUF}}$ from Brzuska et.al. [BFSK11] .