Unconditional UC-Secure Computation with (Stronger-Malicious) PUFs

Saikrishna Badrinarayanan* Dakshita Khurana† Rafail Ostrovsky‡
Ivan Visconti§

Abstract

Brzuska et. al. (Crypto 2011) proved that unconditional UC-secure computation is possible if parties have access to honestly generated physically unclonable functions (PUFs). Dachman-Soled et. al. (Crypto 2014) then showed how to obtain unconditional UC secure computation based on malicious PUFs, assuming such PUFs are stateless. They also showed that unconditional oblivious transfer is impossible against an adversary that creates malicious stateful PUFs.

- In this work, we go beyond this seemingly tight result, by allowing any adversary to create stateful PUFs with a-priori bounded state. This relaxes the restriction on the power of the adversary (limited to stateless PUFs in previous feasibility results), therefore achieving improved security guarantees. This is also motivated by practical scenarios, where the size of a physical object may be used to compute an upper bound on the size of its memory.
- As a second contribution, we introduce a new model where any adversary is allowed to generate a malicious PUF that may encapsulate other (honestly generated) PUFs within it, such that the outer PUF has oracle access to all the inner PUFs. This is again a natural scenario, and in fact, similar adversaries have been studied in the tamper-proof hardware-token model (e.g., Chandran et. al. (Eurocrypt 2008)), but no such notion has ever been considered with respect to PUFs. All previous constructions of UC secure protocols suffer from explicit attacks in this stronger model.

In a direct improvement over previous results, we construct *UC protocols with unconditional security* in both these models.

*UCLA. Email: saikrishna@cs.ucla.edu.
†UCLA. Email: dakshita@cs.ucla.edu.
‡UCLA. Email: rafail@cs.ucla.edu.
§University of Salerno. Email: visconti@unisa.it.

Contents

1 Introduction

In recent years, there has been a rich line of work studying how to enhance the computational capabilities of probabilistic polynomial-time players by making assumptions on hardware[41]. Two types of hardware assumptions in particular have had tremendous impact on recent research: tamper-proof hardware tokens and physically unclonable functions (PUFs).

The tamper-proof hardware token model introduced by Katz [31] relies on the simple and well accepted assumption that it is possible to physically protect a computing machine so that it can only be accessed as a black box, via oracle calls (as an example, think of smart cards). Immediately after its introduction, this model has been studied and its power is now understood in large part. Tamper-proof hardware tokens allow to obtain strong security notions and very efficient constructions, in some cases without requiring computational assumptions. In particular, the even more challenging case of stateless tokens started by [9] has been investigated further in [25, 20, 30, 33, 29, 21, 14, 1, 13].

Physically Unclonable Functions. Physically Unclonable Functions (PUFs) were introduced by Pappu et al. [37, 36] but their actual potential has been understood only in recent years[1]. Increasing excitement over such *physical random oracles* generated various different (and sometimes incompatible) interpretations about the actual features and formalizations of PUFs.

Very roughly, a PUF is an object that can be queried by translating an input into a specific physical stimulation, and then by translating the physical effects of the stimulation to an output through a measurement. The primary appealing properties of PUFs include: (1) constructing two PUFs with similar input-output behavior is believed to be impossible (i.e. unclonability), and (2) the output of a PUF on a given input is seemingly unpredictable, i.e., one cannot "learn" the behavior of an honestly-generated PUF on any specific input without actually querying the PUF on that input.

There is a lot of ongoing exciting research on concrete constructions of PUFs, based on various technologies. As such, a PUF can only be described in an abstract way with the attempt to establish some target properties for PUF designers.

However, while formally modeling a PUF, one might (incorrectly) assume that a PUF guarantees some properties that unfortunately exceed the state of affairs in real-world scenarios. For example, assuming that the output of a genuine PUF is purely random is clearly excessive, while relying on min-entropy is certainly a safer and more conservative assumption. Various papers have proposed different models and even attempts to unify them. The interested reader can refer to [2] for detailed discussions about PUF models and their connections to properties of actual PUFs. We stress that in this work we will consider the use of PUFs in the UC model of [6]. Informally, this means that we want to study protocols that can securely compose with other protocols that may be executing concurrently.

1.1 UC security based on Physically Unclonable Functions

Starting with the work of Brzuska et al. [5], a series of papers have explored UC-secure computation based on physically unclonable functions. The goal of this line of cryptographic research has been to build protocols secure in progressively stronger models.

[1] PUFs are used in several applications like secure storage, RFID systems, anti-counterfeiting mechanisms, identification and authentication protocols [43, 22, 40, 39, 16, 32].

The trusted PUFs of Brzuska et al. [5]. Brzuska et al. [5] began the first general attempts to add PUFs to the simulation paradigm of secure computation. They allowed any player (malicious or honest) to create only well-formed PUFs. As already mentioned, the output of a well-formed PUF on any arbitrary input is typically assumed to have sufficient min-entropy. Furthermore, on being queried with the same input, a well-formed PUF can be assumed to always produce identical (or sufficiently close) outputs. Applying error-tolerant fuzzy extractors [12] to the output ensures that each invocation of the PUF generates a (non-programmable) random string that can be reproduced by querying the PUF again with the same input. Brzuska et al. demonstrated how to obtain *unconditional* UC secure computation for any functionality in this model.

The malicious PUFs of Ostrovsky et al. [35]. Ostrovsky et al. [35] then showed that the constructions of [5] become insecure in case the adversary can produce a *malicious* PUF that deviates from the behavior of an honest PUF. For instance, a malicious PUF could produce outputs according to a pseudo-random function rather than relying on physical phenomena, or it could just refuse to answer to a query. They also showed that it is possible to UC-securely compute any functionality using (potentially malicious) PUFs if one is willing to additionally make computational assumptions. They left open the problem of achieving *unconditional* UC-secure computation for any functionality using malicious PUFs.

Damgård and Scafuro [11] showed that *unconditional* UC secure commitments can be obtained even in the presence of malicious PUFs[2].

The fully malicious but stateless PUFs of Dachman-Soled et al. [10]. More recently, it was shown by Dachman-Soled et al. [10] that unconditional UC security for general functionalities is impossible if the adversary is allowed to create malicious PUFs that can maintain state. They also gave a complementary feasibility result in an intermediate model where PUFs are allowed to be malicious, but are required to be stateless.

We note that the impossibility result of [10]crucially relies on (malicious) PUFs being able to maintain a priori unbounded state.

Thus, the impossibility seems interesting theoretically, but its impact to practical scenarios is unclear. In the real world, this result implies that unconditional UC secure computation of all functionalities is impossible in a model where an honest player is unable to distinguish maliciously created PUFs *with gigantic memory*, from honest (and therefore completely stateless) PUFs. One could argue that this allows the power of the adversary to go beyond the reach of current technology. On the other hand, the protocol of [10] breaks down completely if the adversary can generate a maliciously created PUF with even one bit of memory, and pass it off as a stateless (honest) PUF. This gap forms the starting point for our work.

1.2 Our Contributions

The current state-of-the-art leaves open the following question:

Can we achieve UC-secure computation with malicious PUFs that are allowed to have a priori bounded state?

[2]This can be extended to other functionalities but not to all functionalities.

In the main contribution of this work we answer this question in the affirmative. We show that not only it is possible to obtain UC-secure computation for any functionality as proven in [35] with computational assumptions, but we prove that this can be done with unconditional security, without relying on any computational assumptions. This brings us to our first main result, which we now state informally.

Informal Theorem 1. *For any two party functionality $\mathcal{F}$, there exists a protocol π that unconditionally and UC-securely realizes $\mathcal{F}$ in the malicious bounded-stateful PUF model.*

As our second contribution, we introduce a new adversarial model for PUF-based protocols. Here, in addition to allowing the adversary to generate malicious stateless PUFs, we also allow him to encapsulate other (honestly generated) PUFs inside his own (malicious, stateless) PUF, even without the knowledge of the functionality of the inner PUFs. This allows the outer malicious PUF to make black-box (or oracle) calls to the inner PUFs that it encapsulates. In particular, the outer malicious PUF could answer honest queries by first making oracle calls to its inner PUFs, and generating its own output as a function of the output of the inner PUFs on these queries. An honest party interacting with such a malicious PUF need not be able to tell whether the PUF is malicious and possibly encapsulates other PUFs in it, or it is honest.

In this new adversarial model[3], we require all PUFs to be stateless. We will refer to this as the malicious encapsulated PUF model. It is interesting to note that all previously known protocols (even for limited functionalities such as commitments) suffer explicit attacks in this stronger malicious encapsulated (stateless) PUF model.

As our other main result, we develop techniques to obtain unconditional UC-secure computation in the malicious encapsulated PUF model.

Informal Theorem 2. *For any two party functionality $\mathcal{F}$, there exists a protocol π that unconditionally and UC-securely realizes $\mathcal{F}$ in the malicious encapsulated (stateless) PUF model.*

Table 1 compares our results with prior work. Our feasibility result in the malicious bound-stateful PUF model and our feasibility result in the malicious encapsulated-stateless PUF model directly improve the works of [5, 10]. Indeed each of our two results strengthen the power of the adversaries of [5, 10] in one meaningful and natural direction still achieving the same unconditional results of [5, 10]. A natural question is whether our techniques defeating malicious bounded-stateful PUFs can be composed with our techniques defeating malicious encapsulated-stateless PUFs to obtain unconditional UC-security for any functionality against adversaries that can construct malicious bounded-stateful encapsulated PUFs. While we do not see a priori any conceptual obstacle in obtaining such even stronger feasibility result, the resulting construction would be extremely complex and heavily tedious to analyze. Therefore we defer such a stronger claim to future work hoping that follow up research will achieve a more direct and elegant construction.

1.3 Our Techniques

The starting point for our constructions is the UC-secure OT protocol of [10], which itself builds upon the works of [35, 5]. We begin by giving a simplified description of the construction in [10].

[3] A concurrent and independent work [38] considers an adversary that can encapsulate PUFs but does not propose UC-secure definitions/constructions.

Reference	Unconditional UC for any Functionality	UC with Stateless Mal. PUFs	UC with Bounded Stateful Mal. PUFs	UC with Encapsulated Stateless Mal. PUFs
[5]	✓	×	×	×
[35]	×	✓	✓	×
[10]	✓	✓	×	×
This Work	✓	✓	✓	×
This Work	✓	✓	×	✓

Table 1: The symbol ✓ (resp. ×) indicates that the construction satisfies (resp. does not satisfy) the corresponding security guarantee.

Suppose a sender $\mathcal{S}$ with inputs (m_0, m_1) and a receiver R with input bit b want to run a UC secure OT protocol in the malicious stateless PUF model. Then, $\mathcal{S}$ generates a PUF and sends it to the receiver. The receiver queries the PUF on a random challenge string c, records the output r and then returns the PUF to $\mathcal{S}$. Then, the sender sends two random strings (x_0, x_1) to the receiver. In turn, the receiver picks x_b, and sends $v = c \oplus x_b$ to the sender. The sender uses $\mathsf{PUF}(v \oplus x_0)$ to mask his input m_0 and $\mathsf{PUF}(v \oplus x_1)$, to mask his input m_1; and sends both masked values to the receiver. Here $\mathsf{PUF}(\cdot)$ denotes the output of the PUF on the given input. Since $\mathcal{R}$ had to return the PUF before (x_0, x_1) were revealed, with overwhelming probability, $\mathcal{R}$ only knows $r = \mathsf{PUF}(v \oplus x_b)$, and can output one and only one of the masked sender inputs.

1.3.1 Enhancing [10] in the Stateless PUF Model.

Though this was a simplified overview of the protocol in [10], it helps us to explain a subtle assumption required in their simulation strategy against a malicious sender. In particular, the simulator against a malicious sender must return the PUF to the sender before the sender picks random messages (x_0, x_1). However, it is evident that in order to extract both messages (m_0, m_1), the simulator must know $(x_0 \oplus x_1)$, and in particular know the response of the PUF on challenges $(c, c \oplus x_0 \oplus x_1)$ for some known string c.

But the simulator only learns (x_0, x_1) after sending the PUF back to $\mathcal{S}$. Thus, in order to successfully extract the input of $\mathcal{S}$, the simulator should have the ability to make these queries *even after* the PUF has been returned to the malicious sender. This means that the PUF is supposed to remain accessible and untouched even when it is again in the hands of its malicious creator. We believe this is a very strong assumption that clearly deviates from real scenarios where the state of a PUF can easily be changed (e.g., by damaging it).

Our protocol in Figure 1 gets rid of this strong assumption on the simulator, and we give a new sender simulation strategy that does not need to query the PUF when it is back in the hands of the malicious sender $\mathcal{S}$. This is also a first step in obtaining security against bounded-stateful PUFs. In the protocol of [10], if the PUF created by a malicious $\mathcal{S}$ is stateful, $\mathcal{S}$ on receiving the PUF can *first* change the state of the PUF (say, to output $\perp$ everywhere), and then output values (x_0, x_1). In this case, no simulation strategy will be able to extract the inputs of the sender.

We change the protocol in [10], by having $\mathcal{S}$ commit to the random values (x_0, x_1) at the beginning of the protocol, using a UC-secure commitment scheme. These values are decommitted

Inputs: Sender $\mathcal{S}$ has private inputs $(\mathsf{m}_0, \mathsf{m}_1) \in \{0,1\}^{2n}$ and Receiver $\mathcal{R}$ has private input $\mathsf{b} \in \{0,1\}$.

1. **Sender Message:** $\mathcal{S}$ does the following:
   - Generate a PUF $\mathsf{PUF}_s : \{0,1\}^n \to \{0,1\}^n$.
   - Choose a pair of random strings $(\mathsf{x}_0, \mathsf{x}_1) \xleftarrow{\$} \{0,1\}^{2n}$.
   - Send PUF_s and $(\mathsf{t}_0, \mathsf{t}_1) = \mathsf{UC\text{-}Com.Commit}(\mathsf{x}_0, \mathsf{x}_1)$ to $\mathcal{R}$.
2. **Receiver Message:** $\mathcal{R}$ does the following:
   - Choose a pair of random strings $(\mathsf{c}_0, \mathsf{c}_1) \xleftarrow{\$} \{0,1\}^{2n}$.
   - Compute $\mathsf{r}_0 = \mathsf{PUF}_s(\mathsf{c}_0), \mathsf{r}_1 = \mathsf{PUF}_s(\mathsf{c}_1)$.
   - Set $\mathsf{c} = \mathsf{c}_\mathsf{p}$ and $\mathsf{r} = \mathsf{r}_\mathsf{p}$ for $\mathsf{p} \xleftarrow{\$} \{0,1\}$.
   - Store the pair (c, r) and send PUF_s to $\mathcal{S}$.
3. **Sender Message:**
   - $\mathcal{S}$ sends $(\mathsf{x}_0, \mathsf{x}_1) = \mathsf{UC\text{-}Com.Decommit}(\mathsf{t}_0, \mathsf{t}_1)$ to $\mathcal{R}$.
4. **Receiver Message:** $\mathcal{R}$ does the following:
   - Abort if the decommitment does not verify correctly.
   - Compute and send $\mathsf{val} = \mathsf{c} \oplus \mathsf{x}_\mathsf{b}$ to $\mathcal{S}$.
5. **Sender Message:**
   - $\mathcal{S}$ computes $\mathsf{S}_0 = \mathsf{m}_0 \oplus \mathsf{PUF}_s(\mathsf{val} \oplus \mathsf{x}_0)$, $\mathsf{S}_1 = \mathsf{m}_1 \oplus \mathsf{PUF}_s(\mathsf{val} \oplus \mathsf{x}_1)$ and sends $(\mathsf{S}_0, \mathsf{S}_1)$ to $\mathcal{R}$.

Outputs: $\mathcal{S}$ has no output. $\mathcal{R}$ outputs m_b which is computed as $(\mathsf{S}_\mathsf{b} \oplus \mathsf{r})$.

Figure 1: Protocol Π^1 for 2-choose-1 OT in the malicious stateless PUF model.

only after $\mathcal{R}$ returns the PUF back to $\mathcal{S}$, so the scheme still remains UC-secure against a malicious receiver. Moreover, now the simulator against a malicious sender can use the straight-line extractor guaranteed by the UC-secure commitment scheme, to extract values (x_0, x_1), and query the PUF on challenges of the form $(c, c \oplus x_0 \oplus x_1)$ for some string c. It then sets $v = c \oplus x_0$ and sends it to $\mathcal{S}$. Now, the sender masks are $\mathsf{PUF}(v \oplus x_0)$ and $\mathsf{PUF}(v \oplus x_1)$, which is nothing but $\mathsf{PUF}(c)$ and $\mathsf{PUF}(c \oplus x_0 \oplus x_1)$, which was already known to the sender simulator before returning the PUF

to $\mathcal{S}$. This simulation strategy works (with the simulator requiring only black-box access to the malicious PUF's code) even if the PUF is later broken or its state is reset in any way. This protocol is described formally and proven secure in Section 3.

1.3.2 UC Security with Bounded Stateful PUFs.

A malicious PUF is allowed to maintain *state*, and can generate outputs (including $\perp$) as a function of not only the current query but also the previous queries that it received as input. This allows for some attacks on the protocol we just described, but they can be prevented by carefully interspersing coin-tossing with the protocol. Please see Section 4 for more details.

A stateful PUF created by the sender can also record information about the queries made by the receiver, and replay this information to a malicious sender when he inputs a secret challenge. Indeed, for PUFs with unbounded state, it is this ability to record queries that makes oblivious transfer impossible. However, we only consider PUFs that have a-priori bounded state. In this case, it is possible to design a protocol, parameterized by an upper bound on the size of the state of the PUF, that in effect exhausts the possible state space of such a malicious PUF. Our protocol then carefully uses this additional entropy to mask the inputs of the honest party.

More specifically, we repeat the OT protocol described before (with an additional coin-tossing phase) K times in parallel, using the same (possibly malicious, stateful) PUF, for sufficiently large $K > \ell$ (where ℓ denotes the upper bound on the state of the PUF). At this point, what we require essentially boils down to a *one-sided* malicious oblivious transfer extractor. This is a gadget that would yield a single OT from K leaky OTs, such that the single OT remains secure even when a malicious sender can ask for ℓ bits of universal leakage across all these OTs. This setting is incomparable to previously studied OT extractors [27, 23] because: a) we require a protocol that is secure against malicious (not just semi-honest) adversaries, and b) the system has only one-sided leakage, i.e., a corrupt sender can request ℓ bits of leakage, but a corrupt receiver does not obtain any leakage at all.

For simplicity, we consider the setting of one-sided receiver leakage (instead of sender leakage). It is possible to consider this because OT is reversible. To protect against a malicious receiver that may obtain ℓ bits of universal leakage, the sender picks different random inputs for each OT execution, and then uses a strong randomness extractor to extract min-entropy and mask his inputs. We show that this in fact suffices to statistically hide the input messages of the sender. Please see Section 5 for a more detailed overview and construction.

1.3.3 UC Security with Encapsulated PUFs.

We demonstrate the feasibility of UC secure computation, in a model where a party may (maliciously) encapsulate one or more PUFs that it obtained from honest parties, inside a malicious stateless PUF of its choice. We stress that our protocol itself does not require honest parties to encapsulate PUFs within each other.

To describe our techniques, we begin by revisiting the protocol in Figure 1, that we described at the beginning of this overview. Suppose parties could maliciously encapsulate some honest PUFs inside a malicious PUF. Then a malicious receiver in this protocol, when it is supposed to return the sender's PUF PUF_s, could instead return a *different* malicious PUF $\widehat{\mathsf{PUF}}_s$. In this case, the receiver would easily learn both inputs of the sender. But as correctly pointed out in prior work [11, 10], the sender can deflect such attacks by probing and recording the output of PUF_s on some random

input(s) (known as Test Queries) before sending it to the receiver. Later the sender can check whether $\widehat{\mathsf{PUF}}_s$ correctly answers to all Test Queries.

However, a malicious receiver may create $\widehat{\mathsf{PUF}}_s$ that encapsulates PUF_s, such that $\widehat{\mathsf{PUF}}_s$ is programmed to send most outer queries to PUF_s and echo its output externally; in order to pass the sender's test. However, $\widehat{\mathsf{PUF}}_s$ may have its own malicious procedure to evaluate some of the other external queries. In particular, the "unpredictability" of $\widehat{\mathsf{PUF}}_s$ may break down completely on these queries.

It turns out that the security of the sender in the basic OT protocol of Figure 1 hinges on the unpredictability of the output of PUF_s (in this situation, $\widehat{\mathsf{PUF}}_s$) on a "special challenge query" only, which we will denote by $\mathfrak{s}$. It is completely feasible for a receiver to create a malicious encapsulating PUF $\widehat{\mathsf{PUF}}_s$ that passes the sender tests, and yet its output on this special query $\mathfrak{s}$ is completely known to the receiver, therefore breaking sender security.

We overcome this issue by ensuring that $\mathfrak{s}$ is chosen using a coin toss, and is completely unknown to the receiver until after he has sent $\widehat{\mathsf{PUF}}_s$ (possibly a malicious encapsulating PUF) back to the sender. Intuitively, this means that $\widehat{\mathsf{PUF}}_s$ will either not pass the sender tests, or will be highly likely to deflect this the query $\mathfrak{s}$ to the inner PUF and echo its output (thereby ensuring that the output of the PUF on input $\mathfrak{s}$ is unpredictable for the receiver). An additional subtlety that arises is that the receiver might use an incorrect $\mathfrak{s}$ in the protocol (instead of using the output of the coin toss): the receiver is forced to use the correct $\mathfrak{s}$ via a special cut-and-choose mechanism. For a more detailed overview and construction, please see to Section 6.

1.3.4 UC-Secure Commitments Against Encapsulation Attacks.

Finally, UC-secure commitments against encapsulation attacks play a crucial role in our UC-secure OT protocol in the encapsulation model. But, we note that the basic commitment protocol of [11] is insecure in this stronger model, and therefore we modify the protocol of [11] to achieve UC-security in this scenario. In a nutshell, this is done by having the receiver send an additional PUF at the end of the protocol, and forcing any malicious committer to query this additional PUF on the committer's input bit. We then show that even an encapsulating (malicious) committer will have to carry out this step honestly in order to complete the commit phase. Then, a simulator can extract the adversary's committed value by observing the queries of the malicious committer to this additional PUF. We illustrate in detail, how prior constructions of UC-secure commitments fail in the PUF encapsulation model in Section 7. Our UC-secure commitment protocol in the encapsulated malicious (stateless) PUF model is also described in Section 7.

1.4 Organization

The rest of this paper is organized as follows. In Section 2, we discuss PUFs and other preliminaries relevant to our protocols. In Section 3, we describe an improved version of the protocol in [10], in the stateless PUF model. In Section 4 and Section 5, we boost this protocol to obtain security in the bounded stateful PUF model. In Section 6 and Section 7, we discuss protocols that are secure in the PUF encapsulation model. In Appendix A, we discuss the formal modelling of our PUFs. In additional supplementary material (Appendices B - I) we complete models and proofs that could not be included in the paper owing to space restrictions.

2 Preliminaries

2.1 Physically Unclonable Functions

A PUF is a noisy physical source of randomness. The randomness property comes from an uncontrollable manufacturing process. A PUF is evaluated with a physical stimulus, called the *challenge*, and its physical output, called the *response*, is measured. Since 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 distinct responses within a small Hamming distance to each other. Moreover, the response need not be random-looking; rather, it is a string drawn from a distribution with high min-entropy. Prior work has shown that, using fuzzy extractors, one can eliminate the noisiness of the PUF and make its output uniformly random. For simplicity, we assume this in the body of the paper and give a detailed description in Appendix B.

A PUF-family is a pair of (not necessarily efficient) algorithms Sample and Eval. Algorithm Sample abstracts the PUF fabrication process and works as follows. On input the security parameter, it outputs a PUF-index id from the PUF-family satisfying the security properties (that we define soon) according to the security parameter. Algorithm Eval abstracts the PUF-evaluation process. On input a challenge q, it evaluates the PUF on q and outputs the response a of length rg, denoting the range. Without loss of generality, we assume that the challenge space of a PUF is a full set of strings of a certain length.

Security of PUFs. Following [5], we consider only the two main security properties of PUFs: *unclonability* and *unpredictability*. Informally, unpredictability means that the output of the PUF is statistically indistinguishable from a uniform random string. Formally, 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 still has a significant amount of entropy. For simplicity, a PUF is unpredictable if its output on any given input appears uniformly random.

Informally, unclonability states that in a protocol consisting of several parties, only the party in whose possession the PUF is, can evaluate the PUF. When a party sends a PUF to a different party, it can no longer evaluate the PUF till the time it gets the PUF back. Thus a party not in possession of a PUF cannot predict the output of the PUF on an input for which it did not query the PUF, unless it maliciously created the PUF. We discuss this in more detail in Appendix A.

A PUF can be modeled as an ideal functionality $\mathcal{F}_{\mathsf{PUF}}$, which mimics the behavior of the PUF in the real world. We formally define ideal functionalities corresponding to honestly generated and various kinds of maliciously generated PUFs in Appendix A. We summarize these here: the model for honestly generated PUFs and for malicious stateless/stateful PUFs has been explored in prior work [35, 10], and we introduce the model for encapsulated PUFs.

- **An honestly generated PUF** can be created according to a sampling algorithm Samp, and evaluated honestly using an evaluation algorithm Eval. The output of an honestly generated PUF is *unpredictable* even to the party that created it, i.e., even the creator cannot predict the output of an honestly generated PUF on any given input without querying the PUF on that input.
- **A malicious stateless PUF**, on the other hand, can be created by the adversary substituting an Eval_{mal} procedure of his choice for the honest Eval procedure. Whenever a (honest)

party in possession of this PUF evaluates the PUF, it runs the stateless procedure $\mathsf{Eval}_{mal}(c)$ instead of $\mathsf{Eval}(c)$ (and cannot distinguish $\mathsf{Eval}_{mal}(c)$ from $\mathsf{Eval}(c)$ unless they are distinguishable with black-box access to the PUF). The output of such a PUF cannot depend on previous queries, moreover no adversary that creates the PUF but does not possess it, can learn previous queries made to the PUF when it was not in its possession. We adapt the definitions from [35], where Eval_{mal} is a polynomial-time algorithm with oracle access to Eval. This is done to model the fact that the Eval_{mal} algorithm can access an (honest) source of randomness Eval, and can arbitrarily modify its output using any polynomial-time strategy.

- **A malicious stateful PUF** can be created by the adversary substituting a stateful Eval_{mal} procedure of his choice for the honest Eval procedure. Whenever a party in possession of this PUF evaluates the PUF, it runs the stateful procedure $\mathsf{Eval}_{mal}(c)$ instead of $\mathsf{Eval}(c)$. Thus, the output of a stateful malicious PUF can possibly depend on previous queries, moreover an adversary that created a PUF can learn previous queries made to the PUF by querying it, say, on a secret input. Eval_{mal} is a polynomial-time stateful Turing Machine with oracle access to Eval. Again, this is done to model the fact that the Eval_{mal} algorithm can access an (honest) source of randomness, Eval, and arbitrarily modify its output using any polynomial-time strategy. Malicious stateful PUFs can further be of two types:
  - *Bounded Stateful.* Such a PUF can maintain a-priori bounded memory/state (which it may rewrite, as long as the total memory is bounded).
  - *Unbounded Stateful.* Such a PUF can maintain unbounded memory/state.
- **A malicious encapsulating PUF** can possibly encapsulate other (honestly generated) PUFs inside it[4], without knowing the functionality of these inner PUFs. Such a PUF $\mathsf{PUF}_{\mathtt{mal}}$ can make black-box calls to the inner PUFs, and generate its outputs as a function of the output of the inner (honest) PUFs.

  This is modeled by having the adversary substitute an Eval_{mal} procedure of his choice for the honest Eval procedure in the $\mathsf{PUF}_{\mathtt{mal}}$ that it creates, where as usual Eval_{mal} is a polynomial-time Turing Machine with oracle access to Eval. Similar to the two previous bullets, this is done to model the fact that the Eval_{mal} algorithm can access an (honest) source of randomness, Eval, and arbitrarily modify its output using any polynomial-time strategy.

  In addition, Eval_{mal} can also make oracle calls to polynomially many other (honestly generated) procedures $\mathsf{Eval}_1, \mathsf{Eval}_2, \ldots \mathsf{Eval}_M$ that are contained in PUFs $\mathsf{PUF}_1, \mathsf{PUF}_2, \ldots \mathsf{PUF}_M$, for any a-priori unbounded $M = \mathsf{poly}(n)$. These correspond to honestly generated PUFs that the adversary may be encapsulating within its own malicious PUF. Thus on some input c, the Eval_{mal} procedure may make oracle calls to $\mathsf{Eval}_1, \mathsf{Eval}_2, \ldots \mathsf{Eval}_M$ on polynomially many inputs, and compute its output as a function of the outputs of the $\mathsf{Eval}, \mathsf{Eval}_1, \mathsf{Eval}_2, \ldots \mathsf{Eval}_M$ procedures. Of course, we ensure that the adversary's Eval_{mal} procedure can make calls to some honestly generated procedure Eval_i *only if* the adversary owns the PUF PUF_i implementing the Eval_i procedure when creating the encapsulating malicious PUF. Furthermore, when the adversary passes such a PUF to an honest party, the adversary "loses ownership" of PUF_i and is no longer allowed to access the Eval_i procedure, this is similar to the unclonability requirement. This is modeled by assigning an owner to each PUF, and on passing

[4]Since the adversary knows the code of maliciously generated PUFs, this model automatically captures real-world scenarios where an adversary may be encapsulating other malicious PUFs inside its own.

an outer (encapsulating) PUF to an honest party, the adversary must automatically pass all the inner (encapsulated) honest PUFs. Whenever an honest party is in possession of such an adversarial PUF $\mathsf{PUF}_{\mathtt{mal}}$ and evaluates it, it receives the output of Eval_{mal}. When the adversary is allowed to construct encapsulating PUFS, we restrict all PUFs to be stateless. Therefore the model with encapsulating PUFs is incomparable with the model with bounded-stateful malicious PUFs. Further details on the modeling of malicious stateless PUFs that may encapsulate other stateless PUFs, are provided in Appendix A.

To simplify notation, we write $\mathsf{PUF} \leftarrow \mathsf{Sample}(1^K)$, $r = \mathsf{PUF}(c)$ and assume that PUF is a deterministic function with random output.

2.2 UC Secure Computation

The UC framework, introduced by [6] is a strong framework which gives security guarantees even when protocols may be arbitrarily composed. In Appendix C, we give a formal definition of the UC framework.

Commitments. A UC-secure commitment scheme $\mathsf{UC\text{-}Com}$ consists of the usual commitment and decommitment algorithms, along with (straight-line) procedures allowing the simulator to extract the committed value of the adversary and to equivocate a value that the simulator committed to. We denote these by ($\mathsf{UC\text{-}Com.Commit}$, $\mathsf{UC\text{-}Com.Decommit}$, $\mathsf{UC\text{-}Com.Extract}$, $\mathsf{UC\text{-}Com.Equivocate}$). A formal definition of UC secure Commitments can be found in Appendix D. Damgård and Scafuro [11] realized unconditional UC secure commitments using stateless PUFs, in the malicious stateful PUF model. However, the scheme of [11] is insecure when adversarial parties create malicious encapsulated PUFs. In Appendix K, we construct a UC secure commitment scheme that is secure in the malicious stateless encapsulated PUF model.

OT. Ideal 2-choose-1 oblivious transfer (OT) is a two-party functionality that takes two inputs m_0, m_1 from a sender and a bit b from a receiver. It outputs m_b to the receiver and $\perp$ to the sender. We use $\mathcal{F}_{\mathsf{ot}}$ to denote this functionality. The ideal oblivious transfer(OT) functionality $\mathcal{F}_{\mathsf{ot}}$ and the ideal commitment functionality $\mathcal{F}_{\mathsf{com}}$ are formally defined in Appendix C. Given UC oblivious transfer, it is possible to obtain UC secure two-party computation of any functionality. These results were obtained in prior works and are stated for completeness in Appendix I.

3 Unconditional UC Security with (Malicious) Stateless PUFs

As a warm up, we start by considering malicious stateless PUFs as in [10] and we strengthen their protocol in order to achieve security even when the simulator does not have access to a malicious PUF that is in possession of the adversary that created it.

Construction. Let n denote the security parameter. The protocol Π^1 in Figure 2 UC-securely and unconditionally realizes 2-choose-1 OT in the malicious stateless PUF model, between a sender $\mathcal{S}$ and receiver $\mathcal{R}$, with the following restrictions:

1. The random variables (x_0, x_1) are chosen by $\mathcal{S}$ independently of PUF_s[5].

[5]This is fixed later by using coin-tossing to generate (x_0, x_1), see Section 4.

2. A (malicious) $\mathcal{R}$ returns to $\mathcal{S}$ the same PUF, PUF_s that it received[6].

We enforce these restrictions in this section only for simplicity and modularity purposes. We remove them in Section 4 and Section 6 respectively.

Our protocol makes black-box use of a UC-commitment scheme, denoted by the algorithms $\mathsf{UC\text{-}Com.Commit}$ and $\mathsf{UC\text{-}Com.Decommit}$. We use $\mathsf{UC\text{-}Com.Commit}(a, b)$ to denote a commitment to the concatenation of strings a and b. UC-secure commitments can be unconditionally realized in the malicious stateless PUF model [11]. Formally, we prove the following theorem:

Theorem 1. *The protocol Π^1 in Figure 2 unconditionally UC-securely realizes $\mathcal{F}_{\mathsf{ot}}$ in the malicious stateless PUF model.*

This protocol is essentially the protocol of Dachman-Soled et al. [10], modified to enable correct extraction of the sender's input. The protocol as specified in [10], even though private, does not allow for straight-line extraction of the sender's input messages, unless one is willing to make the strong assumption that the simulator can make queries to a (malicious) PUF that an adversary created, even when this malicious PUF is in the adversary's possession (i.e., the adversary is forced not to update nor to damage/destroy the PUF).

Our main modification is to have the sender commit to his values (x_0, x_1) using a UC-secure commitment scheme. In this case, it is possible for the simulator to extract (x_0, x_1) in a straight-line manner from the commitment, and therefore extract the sender's input while it remains hidden from a real receiver. The rest of the proof follows in the same manner as [10]; recall that we already gave an overview in Section 1.3. We defer the formal proofs of correctness and security of this protocol to Appendix E.

4 UC-Security with (Bounded-Stateful Malicious) PUFs

Overview. A malicious *stateful* PUF can generate outputs as a function of its previous input queries. For the (previous) protocol in Figure 2, note that in Step 2, $\mathsf{Sim}_{\mathcal{S}}$ makes two queries (c_1, c_2) to the PUF such that $(c_1 \oplus c_2) = (x_1 \oplus x_2)$, where (x_1, x_2) are the sender's random messages. On the other hand, an honest receiver makes two queries (c_1, c_2) to the PUF such that $(c_1 \oplus c_2) = \mathsf{rv}$, for an independent random variable rv.

Therefore, when combined with the sender's view, the joint distribution of the evaluation queries made to the PUF by $\mathsf{Sim}_{\mathcal{S}}$, differs from the joint distribution of the evaluation queries made to the PUF by an honest receiver. Thus, a malicious sender can distinguish the two worlds by having a malicious stateful PUF compute a reply to c_2 depending on the value of the previous challenge c_1. We will call these attacks of Type I. In this section, we will describe a protocol secure against all possible attacks where a stateful PUF computes responses to future queries as a function of prior queries.

A stateful PUF created by the sender can also record information about the queries made by the receiver, and replay this information to a malicious sender when he inputs a secret challenge. For PUFs with bounded state, we view these as 'leakage' attacks, by considering all information

[6] In Section 6, we consider an even stronger model where $\mathcal{R}$ may encapsulate PUF_s within a possibly malicious $\widehat{\mathsf{PUF}}_s$. $\widehat{\mathsf{PUF}}_s$ externally forwards some queries to PUF_s and forwards the outputs to the evaluator, while possibly replacing some or all of these outputs with other arbitrary values. We note that this covers the case where the receiver generates $\widehat{\mathsf{PUF}}_s$ malicious and independently of PUF_s.

Inputs: Sender $\mathcal{S}$ has private inputs $(\mathsf{m}_0, \mathsf{m}_1) \in \{0,1\}^{2n}$ and Receiver $\mathcal{R}$ has private input $\mathsf{b} \in \{0,1\}$.

1. **Sender Message:** $\mathcal{S}$ does the following:
   - Generate a PUF $\mathsf{PUF}_s : \{0,1\}^n \to \{0,1\}^n$.
   - Choose a pair of random strings $(\mathsf{x}_0, \mathsf{x}_1) \xleftarrow{\$} \{0,1\}^{2n}$.
   - Send PUF_s and $(\mathsf{t}_0, \mathsf{t}_1) = \mathsf{UC\text{-}Com.Commit}(\mathsf{x}_0, \mathsf{x}_1)$ to $\mathcal{R}$.
2. **Receiver Message:** $\mathcal{R}$ does the following:
   - Choose a pair of random strings $(\mathsf{c}_0, \mathsf{c}_1) \xleftarrow{\$} \{0,1\}^{2n}$.
   - Compute $\mathsf{r}_0 = \mathsf{PUF}_s(\mathsf{c}_0), \mathsf{r}_1 = \mathsf{PUF}_s(\mathsf{c}_1)$.
   - Set $\mathsf{c} = \mathsf{c}_\mathsf{p}$ and $\mathsf{r} = \mathsf{r}_\mathsf{p}$ for $\mathsf{p} \xleftarrow{\$} \{0,1\}$.
   - Store the pair (c, r) and send PUF_s to $\mathcal{S}$.
3. **Sender Message:**
   - $\mathcal{S}$ sends $(\mathsf{x}_0, \mathsf{x}_1) = \mathsf{UC\text{-}Com.Decommit}(\mathsf{t}_0, \mathsf{t}_1)$ to $\mathcal{R}$.
4. **Receiver Message:** $\mathcal{R}$ does the following:
   - Abort if the decommitment does not verify correctly.
   - Compute and send $\mathsf{val} = \mathsf{c} \oplus \mathsf{x}_\mathsf{b}$ to $\mathcal{S}$.
5. **Sender Message:**
   - $\mathcal{S}$ computes $\mathsf{S}_0 = \mathsf{m}_0 \oplus \mathsf{PUF}_s(\mathsf{val} \oplus \mathsf{x}_0)$, $\mathsf{S}_1 = \mathsf{m}_1 \oplus \mathsf{PUF}_s(\mathsf{val} \oplus \mathsf{x}_1)$ and sends $(\mathsf{S}_0, \mathsf{S}_1)$ to $\mathcal{R}$.

Outputs: $\mathcal{S}$ has no output. $\mathcal{R}$ outputs m_b which is computed as $(\mathsf{S}_\mathsf{b} \oplus \mathsf{r})$.

Figure 2: Protocol Π^1 for 2-choose-1 OT in the malicious stateless PUF model.

recorded and replayed by a PUF as leakage. We will call these attacks of Type II. We describe a protocol secure against general bounded stateful PUFs (i.e., secure against attacks of both Type I and Type II) in Section 5.

Our Strategy. Let ℓ denote a polynomial upper bound on the size of the memory of any malicious PUF created by the sender $\mathcal{S}$. Our strategy to obtain secure oblivious transfer from any

PUF with ℓ-bounded state is as follows: We use (the same) PUF_s created by the sender, to execute $K = \Theta(\ell)$ oblivious transfers in parallel. In our new protocol in Figure 3, we carefully intersperse an additional round of coin tossing with our basic protocol from Figure 2, to obtain security against attacks of Type I.

Specifically, we modify the protocol of Figure 2 as follows: instead of having $\mathcal{S}$ generate the random strings (x_0, x_1), we set the protocol up so that *both* $\mathcal{S}$ and the receiver $\mathcal{R}$ generate XOR shares of (x_0, x_1). Furthermore, $\mathcal{R}$ generates his shares only after obtaining the PUF and a commitment to sender shares from $\mathcal{S}$. In such a case, the PUF created by $\mathcal{S}$ must necessarily be independent of the receiver shares and consequently, also independent of (x_0, x_1).

Recall from Section 3, that the simulator against a malicious sender succeeds if it can obtain the output of the PUF to queries of the form $(c, c \oplus x_0 \oplus x_1)$ for a random c, whereas an honest receiver can only make queries of the form (c_1, c_2) for randomly chosen (c_1, c_2). Since (x_0, x_1) appear to be distributed uniformly at random to the PUF, the distributions of $(c, c \oplus x_0 \oplus x_1)$ and (c_1, c_2) are also statistically indistinguishable to the PUF[7]. Therefore, the sender simulator succeeds whenever the honest receiver does not abort and this suffices to prove security against a malicious sender.

Finally, we note that the simulation strategy against a malicious receiver remains similar to one of Section 3, even if the receiver has the ability to create PUFs with unbounded state.

Construction. The protocol Π_K in Figure 3 allows us to use an ℓ-bounded stateful PUF to obtain K secure (but one-sided leaky) oblivious transfers, such that a malicious sender can obtain at most ℓ bits of additional universal leakage on the joint distribution of the receiver's choice input bits $(b_1, b_2, \ldots b_K)$. Our protocol makes black-box use of a UC-commitment scheme, denoted by the algorithms $\mathsf{UC\text{-}Com.Commit}$ and $\mathsf{UC\text{-}Com.Decommit}$[8]. UC-secure commitments can be unconditionally realized in the malicious *stateful* PUF model [11].

Theorem 2. *The protocol Π_K unconditionally UC-securely realizes K instances of OT($\mathcal{F}_{\mathsf{ot}}^{[\otimes K]}$) in an ℓ-bounded-stateful PUF model, except that a malicious sender can obtain at most ℓ bits of*

[7]We assume the simulator can control which simulator queries the adversary's PUF records (but an honest party cannot). Indeed, without our assumption, if a stateful PUF recorded every simulator query, a malicious sender on getting back PUF_s may observe the correlation between queries (c, c') recorded by the PUF when the simulator queried it, versus two random queries when an actual honest party queried it. Ours is a natural assumption and obtaining secure OT remains extremely non-trivial even with this assumption. We note that this requirement can be removed using standard secret sharing along with cut-and-choose, but at the cost of a more complicated protocol with a worse OT production rate. For completeness, we describe this protocol in Appendix L.

[8]The UC framework (and its variants) seemingly fail to capture the possibility of transfer of physical devices like PUFs across different protocols, to the best of our knowledge. Within our OT protocol, we invoke the ideal functionality for UC-secure commitments. Thus, we would like to ensure that our UC-secure commitment scheme composes with the rest of the protocol even if PUFs created in the commitment scheme are used elsewhere in the OT protocol and vice versa. In our protocol, the only situation where such an issue might arise, is if one of the parties in the main OT protocol, later maliciously passes a PUF that it received from the honest party during a commitment phase. This is avoided by requiring all parties to return the PUFs to their original creator at the end of the decommitment phase. Note that this does not violate security even if the PUFs are malicious and stateful. The creating party, like in previous works [11, 10] can probe a random point before sending the PUF, and then check this point again on receiving the PUF, to ensure that they received the correct PUF. Generic results attempting to model UC security in presence of physical devices that can be transferred across different protocol executions have been presented in [4, 26].

Repeat the following protocol K times in parallel for fresh private inputs $(\mathsf{m}_0^i, \mathsf{m}_1^i)$ of the sender and b_i of the receiver for $i \in [K]$.
Inputs: Sender $\mathcal{S}$ has private inputs $(\mathsf{m}_0, \mathsf{m}_1) = (\mathsf{m}_0^i, \mathsf{m}_1^i) \in \{0,1\}^{2n}$ and Receiver $\mathcal{R}$ has private input $\mathsf{b} = \mathsf{b}^i \in \{0,1\}$.

1. **Sender Message:** $\mathcal{S}$ does the following.
   - Generate a PUF $\mathsf{PUF}_s : \{0,1\}^n \to \{0,1\}^n$.(Use the same PUF for all the K parallel sessions).
   - Choose a pair of random strings $(\mathsf{x}_0, \mathsf{x}_1) \xleftarrow{\$} \{0,1\}^{2n}$.
   - Send PUF_s and $(\mathsf{t}_0, \mathsf{t}_1) = \mathsf{UC\text{-}Com.Commit}(\mathsf{x}_0, \mathsf{x}_1)$ to $\mathcal{R}$.
2. **Receiver Message:** $\mathcal{R}$ does the following.
   - Choose a pair of random strings $(\mathsf{c}_0, \mathsf{c}_1) \xleftarrow{\$} \{0,1\}^{2n}$.
   - Compute $\mathsf{r}_0 = \mathsf{PUF}_s(\mathsf{c}_0), \mathsf{r}_1 = \mathsf{PUF}_s(\mathsf{c}_1)$.
   - Set $\mathsf{c} = \mathsf{c}_\mathsf{p}$ and $\mathsf{r} = \mathsf{r}_\mathsf{p}$ for $\mathsf{p} \xleftarrow{\$} \{0,1\}$ and store the pair (c, r).
   - Pick and send $(\hat{\mathsf{x}}_0, \hat{\mathsf{x}}_1) \xleftarrow{\$} \{0,1\}^{2n}$ along with PUF_s, to $\mathcal{S}$.
3. **Sender Message:**
   $\mathcal{S}$ sends $(\mathsf{x}_0, \mathsf{x}_1) = \mathsf{UC\text{-}Com.Decommit}(\mathsf{t}_0, \mathsf{t}_1)$ to $\mathcal{R}$.
4. **Receiver Message:** If $\mathsf{UC\text{-}Com.Decommit}(\mathsf{t}_0, \mathsf{t}_1)$ does not verify, abort. Else, compute and send $\mathsf{val} = \mathsf{c} \oplus \mathsf{x}_\mathsf{b} \oplus \hat{\mathsf{x}}_\mathsf{b}$ to $\mathcal{S}$.
5. **Sender Message:** $\mathcal{S}$ does the following.
   - Compute
     $\mathsf{S}_0 = \mathsf{m}_0 \oplus \mathsf{PUF}_s(\mathsf{val} \oplus \mathsf{x}_0 \oplus \hat{\mathsf{x}}_0)$ and $\mathsf{S}_1 = \mathsf{m}_1 \oplus \mathsf{PUF}_s(\mathsf{val} \oplus \mathsf{x}_1 \oplus \hat{\mathsf{x}}_1)$.
   - Send $(\mathsf{S}_0, \mathsf{S}_1)$ to $\mathcal{R}$.

Outputs: $\mathcal{S}$ has no output. $\mathcal{R}$ outputs m_b which is computed as $(\mathsf{S}_\mathsf{b} \oplus \mathsf{r})$.

Figure 3: Protocol Π_K for K 2-choose-1 OTs (with at most ℓ-bounded leakage) in the malicious stateful PUF model. The changes from the protocol in Figure 2 are underlined.

additional universal leakage on joint distribution of the receiver's choice bits over all $\mathcal{F}_{\mathsf{ot}}^{[\otimes K]}$.

Correctness is immediate from inspection.We defer the full proof of security to Appendix F.

5 One-Sided Correlation Extractors with Malicious Security

From Section 4, in the ℓ-bounded stateful PUF model, we obtain K leaky oblivious transfers, such that the sender can obtain ℓ bits of universal leakage on the joint distribution of the receiver's choice bits over all K oblivious transfers.

Because OT is reversible [45], it suffices to consider a reversed version of the above setting, i.e., where the receiver can obtain ℓ bits of additional universal leakage on the joint distribution of all the sender's messages over all K oblivious transfers. More formally, the leakage model we consider is as follows:

One-Sided Leakage Model for Correlation Extractors. Here, we begin by describing our leakage model for OT correlations formally, and then we define one-sided correlation extractors for OT. Our leakage model is as follows:

1. **K-OT Correlation Generation Phase:** For $i \in [K]$, the sender $\mathcal{S}$ obtains $(x_0^i, x_1^i) \in \{0,1\}^2$ and the receiver $\mathcal{R}$ gets $(b_i, x_{b_i}^i)$.
2. **Corruption and Leakage Phase:** A malicious adversary corrupts the receiver and sends a leakage function $L : \{0,1\}^K \to \{0,1\}^{t_R}$. It receives $L(\{(x_0^i, x_1^i)\}_{i\in[K]})$.

Let (X, Y) be a random OT correlation (i.e., $X = (x_0, x_1), Y = (r, x_r)$, where (x_0, x_1, r) are sampled uniformly at random.) We denote a t_R-leaky version of $(X, Y)^K$ described above as $((X, Y)^K)^{[t_R]}$.

Definition 1 ((n, p, t_R, ϵ) **One-Sided Malicious OT-Extractor).** *An (n, p, t_R, ϵ) one-sided malicious OT-extractor is an interactive protocol between 2 parties S and R with access to $((X, Y)^n)^{[t_R]}$ described above. The protocol implements p independent copies of secure oblivious transfer instances with error ϵ.*

In other words, we want the output oblivious transfer instances to satisfy the standard ϵ-correctness and ϵ-privacy requirements for OT. In more detail, the correctness requirement is that the receiver output is correct in all p instances of OT with probability at least $(1 - \epsilon)$. The privacy requirement is that in every instance of the output OT protocol, a corrupt sender cannot output the receiver's choice bit, and a corrupt receiver cannot output the 'other message' of the sender with probability more than $\frac{1}{2} + \epsilon$.

Theorem 3 (Extracting a Single OT)**.** *There exists a $(2\ell+2n+1, 1, \ell, 2^{-n})$ one-sided OT extractor according to Definition 1.*

Theorem 4 (High Production Rate)**.** *There exists a $(2\ell+2n, \frac{n}{\log^2 n}, \ell, \frac{1}{n \log n})$ one-sided OT extractor according to Definition 1.*

We prove these theorems by giving a construction and proof of security of such extractors in the following sections. We will make use of strong seeded extractors in our construction, and we define such extractors below.

Definition 2 (Strong seeded extractors)**.** *A function $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is called a strong seeded extractor for entropy k if for any (n, k)-source X and an independent random variable Y that is uniform over $\{0,1\}^d$, it holds that $(\mathsf{Ext}(X, Y), Y) \approx (U_m, Y)$.*

Here, U_m is a random variable that is uniformly distributed over m bit strings and is independent of Y, namely (U_m, Y) is a product distribution. In particular, it is known [24, 42, 15] how to construct strong seeded extractors for any entropy $k = \Omega(1)$ with seed length $d = O(\log n)$ and $m = 0.99k$ output bits.

Construction. In Figure 4, we give the basic construction of an OT extractor that securely obtains a single oblivious transfer from $K = (2\ell + 2n)$ OTs, when a receiver can obtain at most ℓ bits of universal leakage from the joint distribution of sender inputs over all the OTs.

Let $\mathcal{E} : \{0,1\}^K \times \{0,1\}^n \to \{0,1\}$ be a strong randomness $(K, 2^{-n})$-extractor for seed length $d = O(n)$.
Inputs: Sender $\mathcal{S}$ has inputs $(x_0, x_1) \in \{0,1\}^{2n}$ and receiver $\mathcal{R}$ has input $\mathsf{b} \in \{0,1\}$.
Given: $K = 2\ell + 2n$ OTs, such that a malicious receiver can obtain additional ℓ bits of leakage on the joint distribution of all sender inputs.

1. **Invoking OT Correlations:**
   - For $i \in [K]$, $\mathcal{S}$ picks inputs $m_0^i, m_1^i \xleftarrow{\$} \{0,1\}$.
   - For $i \in [K]$, $\mathcal{S}$ invokes the i^{th} OT on input m_0^i, m_1^i.
   - For $i \in [K]$, $\mathcal{R}$ invokes the i^{th} OT on input (the same) choice bit b.
2. **Sender Message:**
   - $\mathcal{S}$ picks random seed $s \xleftarrow{\$} \{0,1\}^d$ for the strong seeded extractor $\mathcal{E}$, and computes $M_0 = \mathcal{E}.\mathsf{Ext}(m_0^1||m_0^2||m_0^3 \ldots m_0^K, s)$ and $M_1 = \mathcal{E}.\mathsf{Ext}(m_1^1||m_1^2||m_1^3 \ldots m_1^K, s)$, where $||$ denotes the concatenation operator.
   - $\mathcal{S}$ sends $y_0 = M_0 \oplus x_0, y_1 = M_1 \oplus x_1$ to $\mathcal{R}$, along with seed s.
3. **Output:** $\mathcal{R}$ computes $x_{\mathsf{b}} = y_{\mathsf{b}} \oplus \mathcal{E}.\mathsf{Ext}(m_{\mathsf{b}}^1||m_{\mathsf{b}}^2||m_{\mathsf{b}}^3 \ldots ||m_{\mathsf{b}}^K, s)$.

Figure 4: $(2\ell + 2n, 1, \ell, 2^{-n})$ One-Sided Malicious Correlation Extractor.

Correctness is immediate from inspection. Intuitively, the protocol is secure against ℓ bits of universal (joint) leakage because setting $K = 2\ell + 2n$ still leaves n bits of high entropy even when the receiver can obtain $2\ell + n$ bits of leakage. Moreover, with ℓ bits of additional universal leakage over all pairs of sender inputs $(m_0^1, m_1^1, m_0^2, m_1^2, \ldots m_0^K, m_1^K)$, the strong seeded extractor extracts an output that is statistically close to uniform, and this suffices to mask the sender input.

We defer the formal proof of security of our protocol to Appendix G.

High Production Rate: It is possible to obtain an improved production rate at the cost of higher simulation error. This follows using techniques developed in prior work [44, 23], and the details can be found in Appendix H.

6 UC Secure Computation in the Malicious Encapsulation Model

Let us consider the stateless protocol described in Section 3. In this protocol, the receiver must query PUF_s that he obtained from the sender on a random challenge c, before returning PUF_s to the sender. A malicious receiver cannot have queried PUF_s on both c and $(c \oplus x_0 \oplus x_1)$, because $(x_0 \oplus x_1)$ is chosen by the sender, independently and uniformly at random, and is revealed *only after* the receiver has returned PUF_s. If a malicious receiver was restricted to honestly returning the PUF generated by the sender, by unpredictability of PUF_s, the output of PUF_s on $(c \oplus x_0 \oplus x_1)$ would be a completely unpredictable uniform random variable from the point of view of the receiver, and this sufficed to prove sender security.

However, if a malicious receiver had no such restriction, it could possibly generate a malicious PUF $\widehat{\mathsf{PUF}}$ of his own and give it to the sender, in place of the sender's PUF that it was actually supposed to return. The output of $\widehat{\mathsf{PUF}}$ would no longer remain unpredictable to the receiver and this would lead to a total break of security. As already pointed out in [10], this can be fixed by having the sender make "test queries" to the PUF he generates, before sending the PUF to the receiver. Indeed, when $\widehat{\mathsf{PUF}}$ is generated by the receiver independently of PUF_s, the response of $\widehat{\mathsf{PUF}}$ on the sender's random test query will not match the response of PUF_s and the sender will catch such a cheating receiver with overwhelming probability.

However there could be a different attack: a malicious receiver can construct $\widehat{\mathsf{PUF}}$ encapsulating PUF_s, such that $\widehat{\mathsf{PUF}}$ redirects all test queries to PUF_s (and outputs the value output by PUF_s on the evaluation query), whereas it maliciously answers all protocol queries. In order to rule this out, we ensure that the protocol queries (i.e., the input c that the receiver must query PUF_s with) are generated uniformly at random, by using coin-tossing, combined with cut-and-choose tests to ensure that they are properly used. This is done carefully to ensure that the test queries and protocol queries are identically distributed in the view of $\widehat{\mathsf{PUF}}$ (and are revealed only after the receiver has sent $\widehat{\mathsf{PUF}}$ to the sender).

This ensures that if a maliciously generated $\widehat{\mathsf{PUF}}$ correctly answers all test queries, then with overwhelming probability it must necessarily have answered at least one evaluation query correctly according to the output of PUF_s. At this point, an OT combiner is used to obtain one secure instance of OT.

Let the security parameter be n. The protocol in Figure 5 UC-securely realizes 2-choose-1 OT in a stronger model, where a malicious party is allowed to create malicious PUFs that encapsulate other honest PUFs (see Section 2.1). We emphasize that our protocol does not require that honest parties must have the capability to encapsulate PUFs, yet it is secure even when adversarial parties can create encapsulated PUFs. The protocol uses a UC-commitment scheme, secure in the malicious stateless encapsulated PUF model. We use Com to denote the ideal functionality for such a scheme. We construct such a scheme in Section 7.

Inputs: Sender $\mathcal{S}$ has private inputs $(\mathsf{m}_0, \mathsf{m}_1) \in \{0,1\}^{2n}$ and Receiver $\mathcal{R}$ has private input $\mathsf{b} \in \{0,1\}$.

1. **Coin Flip I:** For $i \in [n]$, $\mathcal{R}$ picks $\mathsf{c}_1^i \xleftarrow{\$} \{0,1\}^n$, sends $\mathsf{d}^i = \mathsf{UC\text{-}Com.Commit}(\mathsf{c}_1^i)$ to $\mathcal{S}$. $\mathcal{S}$ chooses $\mathsf{c}_2^i \xleftarrow{\$} \{0,1\}^n$, sends c_2^i to $\mathcal{R}$. $\mathcal{R}$ computes $\mathsf{c}^i = \mathsf{c}_1^i \oplus \mathsf{c}_2^i$.
2. **Sender Message:** $\mathcal{S}$ generates $\mathsf{PUF}_s : \{0,1\}^n \to \{0,1\}^n$, and does:
   - **Test Queries:** For each $i \in [n]$, choose $\mathsf{TQ}_i \xleftarrow{\$} \{0,1\}^n$ and compute $\mathsf{TR}_i = \mathsf{PUF}_s(\mathsf{TQ}_i)$. Store the pair $(\mathsf{TQ}_i, \mathsf{TR}_i)$.
   - For each $i \in [n]$, choose a pair of random strings $(\mathsf{x}_0^i, \mathsf{x}_1^i) \xleftarrow{\$} \{0,1\}^{2n}$. Compute $(\mathsf{t}_0^i, \mathsf{t}_1^i) = \mathsf{UC\text{-}Com.Commit}(\mathsf{x}_0^i, \mathsf{x}_1^i)$. Send $(\mathsf{t}_0^i, \mathsf{t}_1^i)$ and PUF_s to $\mathcal{R}$.
3. **Receiver Message:** For each $i \in [n]$, choose a random string $(\mathsf{c}_0^i) \xleftarrow{\$} \{0,1\}^n$ and obtain $\mathsf{r}^i = \mathsf{PUF}_s(\mathsf{c}^i), \mathsf{r}_0^i = \mathsf{PUF}_s(\mathsf{c}_0^i)$. Abort if PUF_s aborts, else send PUF_s to $\mathcal{S}$. For $i \in [n]$, pick and send $(\hat{\mathsf{x}}_0^i, \hat{\mathsf{x}}_1^i) \xleftarrow{\$} \{0,1\}^{2n}$.
4. **Sender Message:** $\mathcal{S}$ does the following.
   - **Verification of TQ:** For each $i \in [n]$, if $\mathsf{TR}_i \neq \mathsf{PUF}_s(\mathsf{TQ}_i)$, abort.
   - For each $i \in [n]$, send $(\mathsf{x}_0^i, \mathsf{x}_1^i) = \mathsf{UC\text{-}Com.Decommit}(\mathsf{t}_0^i, \mathsf{t}_1^i)$ to $\mathcal{R}$.
5. **Receiver Message:** Abort if $\mathsf{UC\text{-}Com.Decommit}(\mathsf{t}_0^i, \mathsf{t}_1^i)$ does not verify for any $i \in [n]$. Else pick $\mathsf{b}_i \xleftarrow{\$} \{0,1\}$, compute and send $\mathsf{val}^i = \mathsf{c}^i \oplus \mathsf{x}_{\mathsf{b}_i}^i \oplus \hat{\mathsf{x}}_{\mathsf{b}_i}^i$ to $\mathcal{S}$.
6. **Cut-and-choose:**
   - **Coin Flip II:** $\mathcal{S}$ picks $r_{\mathcal{S}} \xleftarrow{\$} \{0,1\}^{2K}$, sends $t_{\mathcal{S}} = \mathsf{UC\text{-}Com.Commit}(r_{\mathcal{S}})$. $\mathcal{R}$ picks and sends $r_{\mathcal{R}} \xleftarrow{\$} \{0,1\}^{2K}$. $\mathcal{S}$ sends $r_{\mathcal{S}} = \mathsf{UC\text{-}Com.Decommit}(t_S)$, and $(\mathcal{S}, \mathcal{R})$ use $(r_{\mathcal{S}} \oplus r_{\mathcal{R}})$ to pick a subset I of indices $i \in [n]$, of size $\frac{K}{2}$.
   - For $i \in [I]$, $\mathcal{R}$ sends $\mathsf{c}_1^i = \mathsf{UC\text{-}Com.Decommit}(\mathsf{d}^i)$.
   - **Verification:** $\mathcal{S}$ computes $\mathsf{c}^i = \mathsf{c}_1^i \oplus \mathsf{c}_2^i$ and checks if either $\mathsf{val}^i = \mathsf{c}^i \oplus \mathsf{x}_0^i \oplus \hat{\mathsf{x}}_0^i$ OR $\mathsf{val}^i = \mathsf{c}^i \oplus \mathsf{x}_1^i \oplus \hat{\mathsf{x}}_1^i$. If not, $\mathcal{S}$ aborts.
7. **Receiver Message:** For each $i \in [n] \setminus I$, $\mathcal{R}$ sends $\mathsf{bc}_i = \mathsf{b}_i \oplus \mathsf{b}$ to $\mathcal{S}$.
8. **Sender Message:** $\mathcal{S}$ computes $\mathsf{S}_0 = \mathsf{m}_0 \bigoplus_{i \in n \setminus I} \mathsf{PUF}_s(\mathsf{val}^i \oplus \mathsf{x}_{\mathsf{bc}_i}^i \oplus \hat{\mathsf{x}}_{\mathsf{bc}_i}^i)$, $\mathsf{S}_1 = \mathsf{m}_1 \bigoplus_{i \in n \setminus I} \mathsf{PUF}_s(\mathsf{val}^i \oplus \mathsf{x}_{1-\mathsf{bc}_i}^i \oplus \hat{\mathsf{x}}_{1-\mathsf{bc}_i}^i)$. $\mathcal{S}$ sends $(\mathsf{S}_0, \mathsf{S}_1)$ to $\mathcal{R}$.

Outputs: $\mathcal{S}$ has no output. $\mathcal{R}$ outputs $\mathsf{m}_{\mathsf{b}} := (\mathsf{S}_{\mathsf{b}} \oplus \mathsf{r}^1 \oplus \ldots \oplus \mathsf{r}^n)$.

Figure 5: OT in the malicious stateless PUF model with encapsulation. We underline all differences from the protocol in the stateless malicious PUF model.

Though the commitment scheme we construct is UC-secure, it is not immediately clear that it composes with the rest of the OT protocol for the same reasons as were described in Section 4.

Namely, the UC framework seemingly does not capture the possibility of transfer of PUFs across sub-protocols, thus we would like to ensure that our UC-commitment scheme composes with the rest of the protocol even if PUFs created for the commitment scheme are used elsewhere.

Like in Section 4, this can be resolved by requiring both parties to return PUFs back to the respective creators at the end of the decommitment phase, and the creators performing simple verification checks to ensure that the correct PUF was returned. If any party fails to return the PUF, the other party aborts the protocol. Therefore, parties cannot pass off PUFs used by some party in a previous sub-protocol as a new PUF in a different sub-protocol.

Correctness.

Claim 1. *For all* $(m_0, m_1) \in \{0,1\}^2$ *and* $\mathsf{b} \in \{0,1\}$*, the output of* $\mathcal{R}$ *equals* m_b*.*

Proof. If $b = 0$, $\mathsf{bc}^i = \mathsf{b}^i$ for all i, and the receiver computes:
$m'_0 = S_0 \bigoplus_{i \in n \setminus I} r^i = S_0 \bigoplus_{i \in n \setminus I} \mathsf{PUF}_s(c^i) = S_0 \bigoplus_{i \in n \setminus I} \mathsf{PUF}_s(\mathsf{val}^i \oplus x^i_{\mathsf{b}_i})$
$= m_0 \bigoplus_{i \in n \setminus I} \mathsf{PUF}_s(\mathsf{val}^i \oplus x^i_{\mathsf{bc}_i}) \bigoplus_{i \in n \setminus I} \mathsf{PUF}_s(\mathsf{val}^i \oplus x^i_{\mathsf{b}_i}) = m_0.$
If $b = 1$, $1 - \mathsf{bc}^i = \mathsf{b}^i$ for all i, and the receiver computes:
$m'_1 = S_1 \bigoplus_{i \in n \setminus I} r^i = S_1 \bigoplus_{i \in n \setminus I} \mathsf{PUF}_s(c^i) = S_1 \bigoplus_{i \in n \setminus I} \mathsf{PUF}_s(\mathsf{val}^i \oplus x^i_{b_i})$
$= m_1 \bigoplus_{i \in n \setminus I} \mathsf{PUF}_s(\mathsf{val}^i \oplus x^i_{1-\mathsf{bc}_i}) \bigoplus_{i \in n \setminus I} \mathsf{PUF}_s(\mathsf{val}^i \oplus x^i_{\mathsf{b}_i}) = m_1.$ □

We defer the formal proof of security of our protocol to Appendix J.

7 UC Commitments in the Malicious Encapsulation Model

In this section we construct unconditional UC commitments using stateless PUFs. The model we consider is incomparable with respect to the one of [11] since in our model an adversary can encapsulate honest PUFs (see Section 2.1) when creating malicious *stateless encapsulated* PUFs. Note that the protocol does not require any honest party to have the ability to encapsulate PUFs, but is secure against parties that do have this ability.

We note that it suffices to construct an extractable commitment scheme that is secure against encapsulation. Indeed, given such a scheme, Damgård and Scafuro [11] show that it is possible to compile the extractable commitment scheme using an additional ideal commitment scheme, to obtain a UC commitment scheme that is secure in the malicious stateless PUF model. Since the compiler of [11] does not require any additional PUFs at all, if the extractable commitment and the ideal commitment are secure against encapsulation attacks, then so is the resulting UC commitment.

Extractable Commitments. We describe how to construct an extractable bit commitment scheme $\mathsf{ExtCom} = (\mathsf{ExtCom.Commit}, \mathsf{ExtCom.Decommit}, \mathsf{ExtCom.Extract})$ that is secure in the malicious stateless PUFs model with encapsulation. We start with the extractable commitment scheme of [11] that is secure against malicious PUFs in the non-encapsulated setting. They crucially rely on the fact that the initial PUF (let's call it PUF_r) sent by the receiver can not be replaced by the committer (as that would be caught using a previously computed test query). To perform extraction, the simulator against a malicious committer observes the queries made by the committer to PUF_r and extracts the committer's bit. However, in the encapsulated setting, the malicious committer could encapsulate the receiver's PUF inside another PUF (let's call it $\widehat{\mathsf{PUF}_r}$) that, for

all but one query, answers with the output of PUF_r. For the value that the committer is actually required to query on, $\widehat{\mathsf{PUF}_r}$ responds with a maliciously chosen value. Observe that in the protocol description, this query is chosen only by the committer and hence this is an actual attack. Therefore, except with negligible probability, all the receiver's test queries will be answered by $\widehat{\mathsf{PUF}_r}$ with the output of the receiver's original PUF PUF_r. On the other hand, since the target query is no longer forwarded by $\widehat{\mathsf{PUF}_r}$ to the receiver's original PUF, the simulator does not get access to the target query and hence can not extract the committer's bit.

To overcome this issue, we develop a new technique that forces the malicious committer to reveal the target query to the simulator (but not to the honest receiver). After the committer returns $\widehat{\mathsf{PUF}_r}$, the receiver creates a new PUF (let's call it PUF_R). Now, using the commitment, the receiver queries PUF_R on two values, one of which is guaranteed to be the output of $\widehat{\mathsf{PUF}_r}$ on the target query. The receiver stores these two outputs and sends PUF_R to the committer. The malicious committer now has to query PUF_R with $\widehat{\mathsf{PUF}_r}$'s output on his target query and commit to the value that is given in output by PUF_R (using an ideal commitment scheme). In the decommitment phase, using the previously stored values and the committer's input bit, the receiver can verify that the committer indeed queried PUF_R on the correct value. Observe that since the receiver has precomputed the desired output, the malicious committer will not be able to produce an honest decommitment if he tampers with PUF_R and produces a different output. Therefore, the malicious committer *must* indeed query PUF_R and this can be observed by the simulator and used to extract the committer's bit. Our scheme is described in Figure 6. We show that this scheme is correct, statistically hiding, and extractable; and give further details in Appendix K.

8 Acknowledgements

Research supported in part by "GNCS - INdAM", EU COST Action IC1306, NSF grants 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 based upon work supported in part by DARPA Safeware program. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

The work of the 4th author has been done in part while visiting UCLA.

We thank the anonymous reviewers for valuable comments, and in particular for suggesting some important updates to our functionality for encapsulated PUFs.

References

[1] Agrawal, S., Ananth, P., Goyal, V., Prabhakaran, M., Rosen, A.: Lower bounds in the hardware token model. In: Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8349, pp. 663–687. Springer (2014)

[2] Armknecht, F., Moriyama, D., Sadeghi, A., Yung, M.: Towards a unified security model for physically unclonable functions. In: Topics in Cryptology - CT-RSA 2016 - The Cryptogra-

Inputs: Committer $\mathcal{C}$ has private input $\mathsf{b} \in \{0,1\}$ and receiver $\mathcal{R}$ has no input.

Commitment Phase:

1. **Receiver Message:** $\mathcal{R}$ does the following:
   - Generate a PUF $\mathsf{PUF}_r : \{0,1\}^{3n} \to \{0,1\}^{3n}$.
   - **Test Queries :** For each $i \in [n]$, choose $\mathsf{TQ}_i \xleftarrow{\$} \{0,1\}^{3n}$, and compute $\mathsf{TR}_i = \mathsf{PUF}_r(\mathsf{TQ}_i)$. Store the pair $(\mathsf{TQ}_i, \mathsf{TR}_i)$. Send PUF_r to $\mathcal{C}$.
2. **Committer Message:** $\mathcal{C}$ does the following:
   - Generate a PUF $\mathsf{PUF}_s : \{0,1\}^n \to \{0,1\}^{3n}$.
   - For each $i \in [n]$, choose $\mathsf{s}_i \in \{0,1\}^n$. Compute $\sigma_{s_i} = \mathsf{PUF}_s(\mathsf{s}_i)$ and $\sigma_{r_i} = \mathsf{PUF}_r(\sigma_{s_i})$. Send $\mathsf{PUF}_s, \mathsf{PUF}_r$ to $\mathcal{R}$.
3. **Receiver Message:** $\mathcal{R}$ does the following:
   - **Verification :** For each $i \in [n]$, if $\mathsf{TR}_i \neq \mathsf{PUF}_r(\mathsf{TQ}_i)$, abort.
   - For each $i \in [n]$, choose a random string $\mathsf{r}_i \xleftarrow{\$} \{0,1\}^{3n}$ and send r_i to $\mathcal{C}$.
4. **Committer Message:** $\mathcal{C}$ does the following: If $\mathsf{b} = 0$, set $\mathsf{c}_i = \sigma_{s_i}$ for $i \in [n]$, else set $\mathsf{c}_i = (\sigma_{s_i} \oplus \mathsf{r}_i)$ for $i \in [n]$. Send c_i to $\mathcal{R}$.
5. **Receiver Message:** $\mathcal{R}$ does the following:
   - Generate a PUF $\mathsf{PUF}_R : \{0,1\}^{3n} \to \{0,1\}^{3n}$.
   - For $i \in [n]$, set $\mathsf{y}_i = \mathsf{PUF}_R(\mathsf{PUF}_r(\mathsf{c}_i)), \mathsf{z}_i = \mathsf{PUF}_R(\mathsf{PUF}_r(\mathsf{c}_i \oplus \mathsf{r}_i))$, send PUF_R to $\mathcal{C}$.
6. **Committer Message:** For each $i \in [n]$, $\mathcal{C}$ computes $\mathsf{x}_i = \mathsf{PUF}_R(\sigma_{r_i})$. $\mathcal{C}$ computes and sends $\mathsf{t}_i = \mathsf{IdealCom.Commit}(\mathsf{x}_i)$ to $\mathcal{R}$.

Decommitment Phase:

1. **Committer Message:** $\mathcal{C}$ does the following:
   - Send b to $\mathcal{R}$ and for each $i \in [n]$, send s_i, $\mathsf{IdealCom.Decommit}(\mathsf{x}_i)$ to $\mathcal{R}$.
2. **Receiver** $\mathcal{R}$ does the following:
   - For any $i \in [n]$, if $\mathsf{IdealCom.Decommit}(\mathsf{x}_i)$ does not verify, output $\perp$.
   - If $\mathsf{b} = 0$, $\mathsf{c}_i = \mathsf{PUF}_s(\mathsf{s}_i)$ and $\mathsf{x}_i = \mathsf{y}_i$ for all $i \in [n]$, output 0, else $\perp$.
   - If $\mathsf{b} = 1$, $\mathsf{c}_i = (\mathsf{PUF}_s(\mathsf{s}_i) \oplus \mathsf{r}_i)$ and $\mathsf{x}_i = \mathsf{z}_i$ for all $i \in [n]$, output 1, else $\perp$.

Figure 6: Protocol for Extractable Commitment in the malicious stateless PUF model with encapsulation.

phers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9610, pp. 271–287. Springer (2016)

[3] Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proceedings of the twentieth annual ACM symposium on Theory of computing. pp. 1–10. ACM (1988)

[4] Boureanu, I., Ohkubo, M., Vaudenay, S.: The limits of composable crypto with transferable setup devices. In: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15, Singapore, April 14-17, 2015. pp. 381–392. ACM (2015)

[5] Brzuska, C., Fischlin, M., Schröder, H., Katzenbeisser, S.: Physically uncloneable functions in the universal composition framework. In: Rogaway, P. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 6841, pp. 51–70. Springer (2011)

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

[7] Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4392, pp. 61–85. Springer (2007)

[8] Canetti, R., Fischlin, M.: Universally composable commitments. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 19–40. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2001)

[9] Chandran, N., Goyal, V., Sahai, A.: New constructions for UC secure computation using tamper-proof hardware. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 545–562. Springer, Heidelberg, Germany, Istanbul, Turkey (2008)

[10] Dachman-Soled, D., Fleischhacker, N., Katz, J., Lysyanskaya, A., Schröder, D.: Feasibility and infeasibility of secure computation with malicious pufs. In: Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II. Lecture Notes in Computer Science, vol. 8617, pp. 405–420. Springer (2014)

[11] Damgård, I., Scafuro, A.: Unconditionally secure and universally composable commitments from physical assumptions. In: Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II. Lecture Notes in Computer Science, vol. 8270, pp. 100–119. Springer (2013)

[12] Dodis, Y., Ostrovsky, R., Reyzin, L., Smith, A.: Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. SIAM J. Comput. 38(1), 97–139 (2008)

[13] Döttling, N., Kraschewski, D., Müller-Quade, J., Nilges, T.: General statistically secure computation with bounded-resettable hardware tokens. In: Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9014, pp. 319–344. Springer (2015)

[14] Döttling, N., Mie, T., Müller-Quade, J., Nilges, T.: Implementing resettable uc-functionalities with untrusted tamper-proof hardware-tokens. In: TCC. pp. 642–661 (2013)

[15] Dvir, Z., Kopparty, S., Saraf, S., Sudan, M.: Extensions to the method of multiplicities, with applications to kakeya sets and mergers. SIAM J. Comput. 42(6), 2305–2328 (2013)

[16] Eichhorn, I., Koeberl, P., van der Leest, V.: Logically reconfigurable pufs: memory-based secure key storage. In: Proceedings of the sixth ACM workshop on Scalable trusted computing. pp. 59–64. STC '11, ACM, New York, NY, USA (2011)

[17] Goldreich, O.: Foundations of Cryptography: Basic Tools. Cambridge University Press, Cambridge, UK (2001)

[18] Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on computing 18(1), 186–208 (1989)

[19] Goldwasser, S., Micali, S., Wigderson, A.: How to play any mental game, or a completeness theorem for protocols with an honest majority. In: Proc. of the Nienteenth Annual ACM STOC. vol. 87, pp. 218–229 (1987)

[20] Goyal, V., Ishai, Y., Sahai, A., Venkatesan, R., Wadia, A.: Founding cryptography on tamper-proof hardware tokens. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 308–326. Springer, Heidelberg, Germany, Zurich, Switzerland (Feb 9–11, 2010)

[21] Goyal, V., Maji, H.K.: Stateless cryptographic protocols. In: Ostrovsky, R. (ed.) IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011. pp. 678–687. IEEE Computer Society (2011)

[22] Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.: Fpga intrinsic pufs and their use for ip protection. In: CHES. pp. 63–80 (2007)

[23] Gupta, D., Ishai, Y., Maji, H.K., Sahai, A.: Secure computation from leaky correlated randomness. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 701–720. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015)

[24] Guruswami, V., Umans, C., Vadhan, S.P.: Unbalanced expanders and randomness extractors from parvaresh–vardy codes. J. ACM 56(4) (2009)

[25] Hazay, C., Lindell, Y.: Constructions of truly practical secure protocols using standards-smartcards. In: Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008. pp. 491–500 (2008)

[26] Hazay, C., Polychroniadou, A., Venkitasubramaniam, M.: Composable security in the tamper proof hardware model under minimal complexity. In: Theory of Cryptography Conference (TCC'16-B), to appear. LNCS, Springer, Heidelberg, Germany (2016)

[27] Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Extracting correlations. In: 50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA. pp. 261–270. IEEE Computer Society (2009)

[28] Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer - efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2008)

[29] Järvinen, K., Kolesnikov, V., Sadeghi, A., Schneider, T.: Efficient secure two-party computation with untrusted hardware tokens (full version). In: Towards Hardware-Intrinsic Security - Foundations and Practice, pp. 367–386 (2010)

[30] Järvinen, K., Kolesnikov, V., Sadeghi, A., Schneider, T.: Embedded SFE: offloading server and network using hardware tokens. In: Financial Cryptography and Data Security, 14th International Conference, FC 2010, Tenerife, Canary Islands, January 25-28, 2010, Revised Selected Papers. pp. 207–221 (2010)

[31] Katz, J.: Universally composable multi-party computation using tamper-proof hardware. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 115–128. Barcelona, Spain (May 20–24, 2007)

[32] Koçabas, Ü., Sadeghi, A.R., Wachsmann, C., Schulz, S.: Poster: practical embedded remote attestation using physically unclonable functions. In: ACM Conference on Computer and Communications Security. pp. 797–800 (2011)

[33] Kolesnikov, V.: Truly efficient string oblivious transfer using resettable tamper-proof tokens. In: Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings. pp. 327–342 (2010)

[34] Lindell, Y., Pinkas, B.: A proof of security of yaos protocol for two-party computation. Journal of Cryptology 22(2), 161–188 (2009)

[35] Ostrovsky, R., Scafuro, A., Visconti, I., Wadia, A.: Universally composable secure computation with (malicious) physically uncloneable functions. In: Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7881, pp. 702–718. Springer (2013)

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

[37] Pappu, R.S.: Physical One-Way Functions. Ph.D. thesis, MIT (2001)

[38] Rührmair, U.: On the security of puf protocols under bad pufs and pufs-inside-pufs attacks. Cryptology ePrint Archive, Report 2016/322 (2016), http://eprint.iacr.org/

[39] Sadeghi, A.R., Visconti, I., Wachsmann, C.: Enhancing rfid security and privacy by physically unclonable functions. In: Sadeghi, A.R., Naccache, D. (eds.) Towards Hardware-Intrinsic Security, pp. 281–305. Information Security and Cryptography, Springer Berlin Heidelberg (2010)

[40] Sadeghi, A.R., Visconti, I., Wachsmann, C.: Puf-enhanced rfid security and privacy. In: Workshop on Secure Component and System Identification (SECSI) (2010)

[41] Standaert, F.X., Malkin, T.G., Yung, M.: Does physical security of cryptographic devices need a formal study?(invited talk). In: International Conference on Information Theoretic Security. pp. 70–70. Springer (2008)

[42] Ta-Shma, A., Umans, C.: Better condensers and new extractors from parvaresh-vardy codes. In: Proceedings of the 27th Conference on Computational Complexity, CCC 2012, Porto, Portugal, June 26-29, 2012. pp. 309–315. IEEE (2012)

[43] Tuyls, P., Batina, L.: Rfid-tags for anti-counterfeiting. In: CT-RSA. pp. 115–131 (2006)

[44] Vadhan, S.P.: Constructing locally computable extractors and cryptosystems in the bounded-storage model. Journal of Cryptology 17(1), 43–77 (Jan 2004)

[45] Wolf, S., Wullschleger, J.: Oblivious transfer is symmetric. In: Vaudenay, S. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 4004, pp. 222–232. Springer (2006)

A Formal Models for PUFs

While we discuss the physical behaviour of PUFs, and their various properties in detail in Appendix B, here, we describe the formal modelling of various honest, malicious and encapsulating PUFs.

We model honest PUFs similar to prior work. The ideal functionality for honest PUFs is described in Figure 7. We assume that in situations where P_i is required to send a message of the form $(\ldots, P_i, \ldots)$, the ideal functionality checks that the message is indeed coming from party P_i, if not the ideal functionality $\mathcal{F}_{\mathsf{HPUF}}$ turns into waiting state.

Modeling Malicious PUFs. We model malicious PUFs as in [35]. Their ideal functionality is parameterized by two PUF families in order to handle honestly and maliciously generated PUFs: The honestly generated family is a pair $(\mathsf{Sample}_{\mathsf{normal}}, \mathsf{Eval}_{\mathsf{normal}})$ and the malicious one is $(\mathsf{Sample}_{\mathsf{mal}}, \mathsf{Eval}_{\mathsf{mal}})$. Whenever a party P_i initializes a PUF, then it specifies if it is an honest or a malicious PUF by sending $\mathsf{mode} \in \{\mathsf{nor}, \mathsf{mal}\}$ to the functionality $\mathcal{F}_{\mathsf{PUF}}$. The ideal functionality then initialises the appropriate PUF family and it also stores a tag nor or mal representing this family. Whenever the PUF is evaluated, the ideal functionality uses the evaluation algorithm that corresponds to the tag.

The handover procedure is identical to the original formulation of Brzuska et al., where each PUF has a status $\mathsf{flag} \in \{\mathsf{trans}(\mathcal{R}), \mathsf{notrans}\}$ that indicates if a PUF is in transit or not. A PUF that is in transit can be queried by the adversary. Thus, whenever a party P_i sends a PUF to P_j, then the status flag is changed from $\mathsf{notrans}$ to trans and the attacker can evaluate the PUF. At some point, the attacker sends $\mathsf{ready}_{\mathsf{PUF}}$ to the ideal functionality to indicate that it is not querying the PUF anymore. The ideal functionality then hands the PUF over to P_j and changes the status flag back to $\mathsf{notrans}$. The party P_j may evaluate the PUF. Finally, when the attacker sends the message $\mathsf{received}_{\mathsf{PUF}}$ to the ideal functionality, then $\mathcal{F}_{\mathsf{PUF}}$ sends $\mathsf{received}_{\mathsf{PUF}}$ to P_i in order to notify P_i that the handover is over. The ideal functionality for malicious PUFs is shown in Figure 8. We refer the reader to [35] for more details on the different properties of malicious PUFs.

We additionally allow malicious PUFs to maintain $\mathsf{poly}(n)$ a-prior bounded memory. This is done by allowing $\mathsf{Eval}_{\mathsf{mal}}$ to be a stateful procedure.

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

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

Figure 7: The ideal functionality $\mathcal{F}_{\mathsf{HPUF}}$ for honest PUFs.

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

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

Figure 8: The ideal functionality $\mathcal{F}_{\mathsf{MPUF}}$ for malicious PUFs.

Modeling Encapsulating PUFs. We model malicious PUFs that can encapsulate functionalities as in [35, 9]. This functionality formalizes the intuition that an honest user can create a PUF implementing a random function, but an adversary given the PUF can only observe its input/output characteristics.

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

- When a party $P_i \in \mathbb{P} \cup \{\mathcal{S}\}$ writes $(\mathsf{init}_{\mathsf{PUF}}, \mathsf{sid}, \mathtt{mode}, P_i)$ on the input tape of $\mathcal{F}_{\mathsf{E\text{-}PUF}}$, where $\mathtt{mode} \in \{\mathtt{normal}, \mathtt{mal}\}$, then $\mathcal{F}_{\mathsf{E\text{-}PUF}}$ checks whether $\mathcal{L}$ already contains a tuple $(\mathsf{sid}, \mathsf{id}, *, *, *, *)$ for some id. If it does, turn to waiting state. Else, draw $\mathsf{id} \leftarrow \mathsf{Sample}_{\mathtt{mode}}(1^K)$ 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 input tape of P_i. If any of the checks failed, turn to waiting state.

- When the adversary P_i writes $\mathsf{reassign}(\mathsf{sid}, \mathsf{sid}', P_i)$ on the input tape of $\mathcal{F}_{\mathsf{E\text{-}PUF}}$, check if there exists a tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, P_i, \mathsf{notrans})$, and check that $\mathcal{L}$ does not already contains a tuple $(\mathsf{sid}, \mathsf{id}, *, *, *, *)$ for some id. If either of the conditions are not met, turn to waiting state. Else, replace the first tuple with $(\mathsf{sid}', \mathsf{id}, \mathtt{mode}, P_i, \mathsf{notrans})$.

- When the adversary P_i writes $(\mathsf{encap}_{\mathsf{PUF}}, \mathsf{sid}, \mathsf{sid}', P_i)$ on the input tape of $\mathcal{F}_{\mathsf{E\text{-}PUF}}$, check if there exist tuples $(\mathsf{sid}, *, *, P_i, \mathsf{notrans})$ and $(\mathsf{sid}', *, *, P_i, \mathsf{notrans})$. If such tuples exist, set $\mathsf{owner}(\mathsf{sid}) = \mathsf{sid}'$ [a].

- When party P_i sends $(\mathsf{handover}_{\mathsf{PUF}}, \mathsf{sid}, P_i, P_j)$ to $\mathcal{F}_{\mathsf{E\text{-}PUF}}$, 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 $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \perp, \mathsf{trans}(P_j))$. Write $(\mathsf{invoke}_{\mathsf{PUF}}, \mathsf{sid}, P_i, P_j)$ on P_i's input tape [b].

- When a party $P_i \in \mathbb{P} \cup \{\mathcal{S}\}$ writes $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{sid}, P_i, q)$ on $\mathcal{F}_{\mathsf{E\text{-}PUF}}$'s input tape, check if there exists a tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, P_i, \mathsf{notrans})$ or $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \perp, \mathsf{trans}(*))$ in $\mathcal{L}$. If not, then turn into waiting state. Else, run $a \leftarrow \mathsf{Eval}_{\mathtt{mode}}(1^K, \mathsf{id}, q)$. Write $(\mathsf{response}_{\mathsf{PUF}}, \mathsf{sid}, q, a)$ on P_i's input tape.

- The $\mathsf{Eval}_{\mathtt{mal}}$ procedure can either makes calls to $\mathsf{Eval}_{\mathtt{normal}}$, or can write $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{sid}*, \mathsf{sid}, q*)$ on $\mathcal{F}_{\mathsf{E\text{-}PUF}}$'s input tape. If $\mathsf{Eval}_{\mathtt{mal}}$ writes $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{sid}*, \mathsf{sid}, q*)$ on $\mathcal{F}_{\mathcal{F}_{\mathsf{E\text{-}PUF}}}$'s input tape, check if $\mathsf{owner}(\mathsf{sid}*) = \mathsf{sid}$. If not, turn to waiting state. Else, like the previous bullet, check if there exists a tuple $(\mathsf{sid}*, \mathsf{id}, \mathtt{mode}, P_i, \mathsf{notrans})$ or $(\mathsf{sid}*, \mathsf{id}, \mathtt{mode}, \perp, \mathsf{trans}(*))$ in $\mathcal{L}$. If not, then turn into waiting state. Else, run $a \leftarrow \mathsf{Eval}_{\mathtt{mode}}(1^K, \mathsf{id}, q)$ and return $(\mathsf{response}_{\mathsf{PUF}}, \mathsf{sid}*, q, a)$ as output to sid.

- When the adversary sends $(\mathsf{ready}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ to $\mathcal{F}_{\mathsf{E\text{-}PUF}}$, check if $\mathcal{L}$ contains $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \perp, \mathsf{trans}(P_j))$. If not, turn into waiting state. Else, change $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \perp, \mathsf{trans}(P_j))$ to $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, P_j, \mathsf{notrans})$, write $(\mathsf{handover}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ on P_j's input tape and store $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$.

- When the adversary sends $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ to $\mathcal{F}_{\mathsf{E\text{-}PUF}}$, check if $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ has been stored. If not, return to waiting state. Else, write this tuple to the input tape of P_i.

[a] Intuitively, when a (malicious) party encapsulates a PUF, this sets the outer PUF as owner of the inner PUF. Even the adversary can access the inner PUF via evaluation queries to outer PUF. This step permits multiple iterative encapsulations.

[b] Handover does not change the owner (outer PUF) of an (inner) encapsulated PUF.

Figure 9: The ideal functionality $\mathcal{F}_{\mathsf{E\text{-}PUF}}$ for malicious PUFs that may *encapsulate* PUFs.

$\mathcal{F}_{\mathsf{E\text{-}PUF}}$ models the PUF (sent by party P_i to party P_j) encapsulating some functionality M_{ij}. The changes from the previous definition [35] that we make is that M_{ij} is now an oracle machine (instead of a functionality) which can make evaluation calls to other PUFs itself. The ideal functionality for malicious PUFs that could possibly encapsulate honest PUFs, is described in Figure 9. $\mathcal{F}_{\mathsf{E\text{-}PUF}}$ models the following sequence of events: (1) a party P_i samples a random PUF from the challenge space, (2) P_i then gives this PUF to another party P_j (the receiver) who can use the PUF as a black-box implementing M_{ij}, (3) On giving M_{ij}, P_i loses oracle access to all PUFs of which it was previously the owner but which M_{ij} has oracle access to. Figure 9 has the formal description of $\mathcal{F}_{\mathsf{E\text{-}PUF}}$ based on such an algorithm M_{ij}.

We assume that every PUF has a *single* calling procedure known as its *owner*. This owner can either be a party, or another PUF (in the case of adversarially generated PUFs). This models (refer to the first bullet in Figure 9) the fact that an adversary that receives a PUF implementing M_{xy} can either keep the PUF to make calls later or incorporate the functionality of this PUF in a black-box manner into another (maliciously created) PUF, but cannot do both. The evaluation procedure for a malicious encapsulating outer PUF, carefully checks that the outer PUF has ownership of inner PUFs (refer the second bullet in Figure 9), before allowing the malicious outer evaluation procedure oracle access to any inner PUF. The handover operation (described in the third bullet in Figure 9) is similarly carefully modified to ensure that the party that receives an encapsulated PUF can only access the inner PUF via evaluation queries to the outer PUF. Each PUF is uniquely identified by an identifier known as id.

Finally, we note that our model may also allow an adversary to "dismount" a PUF, i.e., separate out its inner component PUFs. For simplicity, we choose to not formalize this requirement. Our protocols trivially remain secure in this model since we never require the honest parties to hand over any "encap"-PUFs back to the adversary, where an "encap"-PUF is a malicious PUF that may be encapsulating honest PUFs.

B Physically Unclonable Functions: Modeling

In this section we describe PUFs, mostly following definitions given in [5]. A PUF is a noisy physical source of randomness, whose randomness property comes from an uncontrollable manufacturing process. A PUF is evaluated with a physical stimulus, called the *challenge*, and its physical output, called the *response*, is measured. Since 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 [5] is the following.

A PUF-family $\mathcal{P}$ is a pair of (not necessarily efficient) algorithms Sample and Eval, and is parameterized by the bound on the noise of PUF's response d_{noise} and the range of the PUF's output rg. Algorithm Sample abstracts the PUF fabrication process and works as follows. On input the security parameter, it outputs a PUF-index id from the PUF-family satisfying the security property (that we define soon) according to the security parameter. Algorithm Eval abstracts the PUF-evaluation process. On input a challenge q, it evaluates the PUF on q and outputs the response a of length rg. The output is guaranteed to have bounded noise d_{noise}, meaning that, when running $\mathsf{Eval}(1^K, \mathsf{id}, q)$ twice, the Hamming distance of any two responses a_1, a_2 is smaller than $d_{\mathsf{noise}}(K)$. Without loss of generality, we assume that the challenge space of a PUF is a full set of strings of a

certain length.

Definition 3 (Physically Unclonable Functions)**.** *Let rg denote the size of the range of the PUF responses of a PUF-family and d_{noise} denote a bound of the PUF's noise. $\mathcal{P} = (\mathsf{Sample}, \mathsf{Eval})$ is a family of (rg, d_{noise})-PUF if it satisfies the following properties.*

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

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

Security of PUFs. We assume that PUFs enjoy the properties of *unclonability* 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. Notice that unpredictability as defined here implies mild forms of unclonability [5]. In the following we recall the concept of average min-entropy.

Definition 4 (Average min-entropy)**.** *The average min-entropy of the measurement $\mathsf{PUF}(q)$ conditioned on the measurements of challenges $\mathcal{Q} = (q_1, \ldots, q_{\mathsf{poly}(n)})$ is defined by*

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

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

Unpredictability. A (rg, d_{noise})-PUF family $\mathcal{P} = (\mathsf{Sample}, \mathsf{Eval})$ for security parameter K is $(d_{\mathsf{min}}(K), \mathsf{m}(K))$-unpredictable if for any $q \in \{0,1\}^K$ and challenge list $\mathcal{Q} = (q_1, \ldots, q_{\mathsf{poly}(n)})$, one has that, if for all $1 \leq k \leq \mathsf{poly}(n)$ the Hamming distance satisfies $\mathsf{dis}_{\mathsf{ham}}(q, q_k) \geq d_{\mathsf{min}}(K)$, then the average min-entropy satisfies $\tilde{H}_\infty(\mathsf{PUF}(q)|\mathsf{PUF}(\mathcal{Q})) \geq \mathsf{m}(K)$, where $\mathsf{PUF}(\mathcal{Q})$ denotes a sequence of random variables $\mathsf{PUF}(q_1), \ldots, \mathsf{PUF}(q_{\mathsf{poly}(n)})$ each corresponding to an evaluation of the PUF on challenge q_k. Such a PUF-family is called a $(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, \mathsf{m})$-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. [12] 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 5 (Fuzzy Extractors)**.** *A $(\mathsf{m}, \ell, t, \epsilon)$-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 $\mathsf{p} \in \{0,1\}^*$ is a helper data string. The algorithm FuzRep, on input an element $w' \in \mathcal{M}$ and a helper data string $\mathsf{p} \in \{0,1\}^*$ 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 $(st, \mathsf{p}) \xleftarrow{\$} \mathsf{FuzGen}$, then $\mathsf{FuzRep}(w', \mathsf{p}) = st$.*
- **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., $SD((st, \mathsf{p}), (U_\ell, \mathsf{p})) \leq \epsilon$.*

Given a $(rg(K), d_{\mathsf{noise}}(K), d_{\mathsf{min}}(K), \mathsf{m}(K))$-PUF family with $d_{\mathsf{min}}(K) = o(K/\log K)$, a *matching* fuzzy extractor has as parameters $\ell(K) = K$ and $t(K) = d_{\mathsf{noise}}(K)$. 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 $p(K)$ and all set of challenges $q_1, \ldots, q_{p(K)}$, the probability that a randomly chosen challenge is within distance smaller than d_{min} with any q_k, for $1 \leq k \leq K$ is negligible.
- **Extraction Independence.** For all challenges $q_1, \ldots, q_{p(K)}$, and for a challenge q such that $\mathsf{dis}(q, q_k) > d_{\mathsf{min}}$ for $1 \leq k \leq p(K)$, 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 $(st, \mathsf{p}) \xleftarrow{\$} \mathsf{FuzGen}(a)$ it holds that $st \leftarrow \mathsf{FuzRep}(a', \mathsf{p})$.

Unclonability. We define unclonability as in [5]. Consider an adversary $\mathcal{A}$ that wishes to clone a PUF PUF_1 sampled from a PUF family PUF while having access to several other PUFs $\mathsf{PUF}_2, \ldots, \mathsf{PUF}_n$ from the same family. A family PUF is said to be *clonable* if there exists a probabilistic polynomial time algorithm (PPT) $\mathcal{A}$ such that for all PPT distinguishers $\mathbf{D}$, the advantage of the distinguisher in the following game is negligible in λ.
$\mathsf{Unc}(\mathsf{PUF}, \mathcal{A}, \lambda)$

- **Learning Phase:** Proceeding adaptively, the adversary $\mathcal{A}$ has access to an oracle Sample which, given an index i, draws a PUF PUF_i from the family according to Sample as well as to Eval oracles that evaluate PUF_i. The index of the first sampling query is denoted by i_0.

- **Challenge Phase:** Eventually, we withdraw the adversarys possibility to query PUF_{i_0} by withdrawing $\mathcal{A}$'s access to the Eval oracle of PUF_{i_0}. $\mathcal{A}$ can now be queried with input challenge values c. For all challenges c, $\mathcal{A}$ is required to output answers r that are (computationally) indistinguishable from answers given by PUF_{i_0}. That is, for any distinguisher $\mathbf{D}$ that is given two oracles that are either both equal to the Eval algorithm of PUF_{i_0}, or one oracle is $\mathcal{A}$ and one oracle is the Eval algorithm of PUF_{i_0}, the advantage of $\mathbf{D}$ is equal to:

$$\Pr[\ \mathbf{D}^{\mathsf{PUF}_{i_0},\mathcal{A}} = 1] - \Pr[\ \mathbf{D}^{\mathsf{PUF}_{i_0},\mathsf{PUF}_{i_0}} = 1].$$

Definition 6. *(Unclonability) A family of PUFs* $\mathsf{PUF} = (\mathsf{Sample}, \mathsf{Eval})$ *is said to be unclonable if it is not clonable.*

C 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 [6].

Protocol syntax. Following [18] and [17], 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 7. *(UC-Realizing an Ideal Functionality) Let $\mathcal{F}$ be an ideal functionality, and Π be a protocol. We say that Π UC-**realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model** if for any hybrid-model PPT adversary $\mathcal{A}$, there exists an ideal process expected PPT adversary Sim such that for every PPT environment $\mathcal{Z}$:*

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

Note that the above equation, says that in the ideal world, the simulator Sim has no access to the ideal functionality $\mathcal{G}$. However, when $\mathcal{G}$ is a set-up assumption, this is not necessarily true and the simulator may have access to $\mathcal{G}$ even in the ideal world. Indeed, there exist different formulations of the UC framework, capturing different requirements on the set-assumptions (e.g., [7, 5]). In [7] for example, the set-up assumption is global, which means that the environment has direct access to the set-up functionality $\mathcal{G}$. Hence, the simulator Sim needs to have oracle access to $\mathcal{G}$ as well. In [5] they assume that Sim cannot simulate (*program*) a PUF, and thus it needs access to the ideal functionality $\mathcal{F}_{\mathsf{PUF}}$. [5] however restricts the access of the environment to $\mathcal{F}_{\mathsf{PUF}}$. $\mathcal{Z}$ has not permanent access to $\mathcal{F}_{\mathsf{PUF}}$.

Oblivious Transfer Functionality. Oblivious Transfer (OT) is a two-party game in which a sender holds a pair of strings (s_0, s_1), and a receiver needs to obtain one string according to its input bit b. The transfer of the desired string is oblivious in the sense that the sender does not know the string obtained by the receiver, while the receiver obtaining one string gains no information about the other one. The OT Functionality $\mathcal{F}_{\mathsf{ot}}$ is shown in Fig. 10.

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 $s_0, s_1 \in \{0,1\}^K$, 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 $(\mathsf{send}, \cdot)$ message was previously sent do nothing).

Figure 10: The Oblivious Transfer Functionality $\mathcal{F}_{\mathsf{ot}}$.

Commitment Functionality. The ideal functionality for a commitment scheme as presented in [8], is depicted in Fig. 11.

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 ignore the message.

Figure 11: The Commitment Functionality $\mathcal{F}_{\mathsf{com}}$.

D UC-Secure Commitments

We denote by $\mathcal{F}_{\mathsf{aux}}$ an auxiliary setup functionality accessed by the real world parties and the extractor.

Definition 8. *(Ideal Commitment Scheme in the $\mathcal{F}_{\mathsf{aux}}$-hybrid model). A commitment scheme run by two parties - the sender $\mathcal{S}$ and receiver $\mathcal{R}$ is a tuple of PPT algorithms $\mathsf{Com} = (\mathsf{Commit}, \mathsf{Decommit})$ (which may be interactive) having access to an ideal setup functionality $\mathcal{F}_{\mathsf{aux}}$, implementing the following two-phase functionality. Given an input $\mathsf{b} \in \{0,1\}$ to $\mathcal{S}$, in the first phase called commitment phase, $\mathcal{S}$ runs the algorithm Commit on input b to produce the commitment c and sends this to $\mathcal{R}$. In the second phase, called the decommitment phase, $\mathcal{S}$ runs the algorithm $\mathsf{Decommit}$ on inputs (b, c) to produce an output (d, b) that is sent to $\mathcal{R}$. $\mathcal{R}$ finally outputs "accept" or "reject" depending on the values of $(\mathsf{c}, d, \mathsf{b})$. Com is an ideal commitment scheme if it satisfies the following.*

1. ***Completeness:*** *For any $\mathsf{b} \in \{0,1\}$, if $\mathcal{S}$ and $\mathcal{R}$ are honest, $\mathcal{R}$ accepts the commitment c and the decommitment (d, b) with probability 1.*

2. ***Statistical Hiding:*** *For any malicious receiver $\mathcal{R}^*$, after the commitment phase, the view of $\mathcal{R}^*$ when $\mathcal{S}$ commits to bit 0 is statistically indistinguishable from the view of $\mathcal{R}^*$ when $\mathcal{S}$ commits to bit 1.*

3. ***Statistical Binding:*** *For any malicious committer $\mathcal{S}^*$, there exists a negligible function ϵ, such that $\mathcal{S}^*$ succeeds in the following game with probability at most $\epsilon(n)$ where n is the security parameter: On security parameter n, $\mathcal{S}^*$ interacts with $\mathcal{R}$ in the commitment phase to obtain the commitment c. Then, in the decommitment phase, in one case $\mathcal{S}^*$ runs $\mathsf{Decommit}(0, \mathsf{c})$ to produce $(d_0, 0)$ and in the other case runs $\mathsf{Decommit}(1, \mathsf{c})$ to produce $(d_1, 1)$. $\mathcal{S}^*$ succeeds if $\mathcal{R}$ outputs "accept" in both cases.*

Definition 9 (Interface Access to an Ideal Functionality $\mathcal{F}_{\mathsf{aux}}$)**.** *Let $\pi = (\mathcal{P}_1, \mathcal{P}_2)$ be a two party protocol in the $\mathcal{F}_{\mathsf{aux}}$-hybrid model. That is, parties $\mathcal{P}_1$ and $\mathcal{P}_2$ need to query the ideal functionality $\mathcal{F}_{\mathsf{aux}}$ in order to carry out the protocol. An algorithm $\mathcal{M}$ has interface access to the ideal functionality $\mathcal{F}_{\mathsf{aux}}$ with respect to protocol π if all queries made by either party $\mathcal{P}_1$ or $\mathcal{P}_2$ to $\mathcal{F}_{\mathsf{aux}}$ during the protocol execution can be observed (but not answered) by $\mathcal{M}$ and $\mathcal{M}$ has oracle access to $\mathcal{F}_{\mathsf{aux}}$. Consequently, $\mathcal{F}_{\mathsf{aux}}$ can be a non programmable and non PPT functionality.*

Definition 10 (Ideal Extractable Commitment Scheme in the $\mathcal{F}_{\mathsf{aux}}$ Model)**.** $\mathsf{ExtCom} = (\mathsf{ExtCom.Commit}, \mathsf{ExtCom.Decommit}, \mathsf{ExtCom.Extract})$ *is an Ideal Extractable Commitment Scheme in the $\mathcal{F}_{\mathsf{aux}}$ model if* $(\mathsf{ExtCom.Commit}, \mathsf{ExtCom.Decommit})$ *is an ideal commitment scheme and there exists a straight line strict polynomial time extractor* E *having interface access to $\mathcal{F}_{\mathsf{aux}}$, that runs the commitment phase only and outputs a value $\mathsf{b}^* \in \{0, 1, \perp\}$ such that, for all malicious committers $\mathcal{S}^*$, the following properties are satisfied:*

1. ***Simulation:*** *The view generated by the interaction between E and $\mathcal{S}^*$ is statistically indistinguishable from the view generated when $\mathcal{S}^*$ interacts with the honest receiver $\mathcal{R}$.*
2. ***Extraction:*** *Let c be a valid output of the commitment phase run between $\mathcal{S}^*$ and E. If E outputs $\perp$, then the probability that $\mathcal{S}^*$ will provide an accepting decommitment is negligible.*
3. ***Binding:*** *If $\mathsf{b}^* \neq \perp$, then the probability that $\mathcal{S}^*$ decommits to a bit $\mathsf{b} \neq \mathsf{b}^*$ is negligible.*

Definition 11 (Ideal Equivocal Commitment Scheme in the $\mathcal{F}_{\mathsf{aux}}$ Model)**.** $\mathsf{EqvCom} = (\mathsf{EqvCom.Commit}, \mathsf{EqvCom.Decommit}, \mathsf{EqvCom.Equivocate})$ *is an Ideal Equivocal Commitment Scheme in the $\mathcal{F}_{\mathsf{aux}}$ model if* $(\mathsf{EqvCom.Commit}, \mathsf{EqvCom.Decommit})$ *is an ideal commitment scheme and there exists a straight line strict polynomial time simulator* Sim *having interface access to $\mathcal{F}_{\mathsf{aux}}$, that runs the commitment phase only such that, for all malicious receivers $\mathcal{R}^*$, the following properties are satisfied:*

1. ***Simulation:*** *The view generated by the interaction between Sim and $\mathcal{R}^*$ is statistically indistinguishable from the view generated when $\mathcal{R}^*$ interacts with the honest committer $\mathcal{S}$.*
2. ***Equivocation:*** *Let c be a valid output of the commitment phase run between Sim and $\mathcal{R}^*$. For all $\mathsf{b} \in \{0, 1\}$, probability that Sim generates an invalid decommitment of c with respect to b is negligible.*
3. ***Hiding:*** *Let c be a valid output of the commitment phase. The probability that $\mathcal{R}^*$ can compute $\mathsf{b} \in \{0, 1\}$ just given c such that there exists a decommitment of c with respect to input b is ϵ-close to $\frac{1}{2}$ where ϵ is a negligible function.*

Definition 12 (UC-Secure Commitment Scheme in the $\mathcal{F}_{\mathsf{aux}}$ Model)**.** $\mathsf{UC\text{-}Com} = (\mathsf{UC\text{-}Com.Commit}, \mathsf{UC\text{-}Com.Decommit}, \mathsf{U}$
is a UC-Secure Commitment Scheme in the $\mathcal{F}_{\mathsf{aux}}$ model if:
$(\mathsf{UC\text{-}Com.Commit}, \mathsf{UC\text{-}Com.Decommit}, \mathsf{UC\text{-}Com.Extract})$ *is an ideal extractable commitment scheme and,* $(\mathsf{UC\text{-}Com.Commit}, \mathsf{UC\text{-}Com.Decommit}, \mathsf{UC\text{-}Com.Equivocate})$ *is an ideal equivocal commitment scheme according to the above definitions.*

E Proof of Security for OT in Malicious Bounded Stateless PUF Model

In this section, we give the formal proof of Theorem 1.

Claim 2 (Correctness)**.** *For all* $(\mathsf{m}_0, \mathsf{m}_1) \in \{0,1\}^{2n}$ *and* $\mathsf{b} \in \{0,1\}$*, the output of* $\mathcal{R}$ *equals* m_b*.*

Proof. The honest receiver computes $\mathsf{S}_\mathsf{b} \oplus \mathsf{r} = (\mathsf{m}_\mathsf{b} \oplus \mathsf{PUF}_s(\mathsf{val} \oplus \mathsf{x}_\mathsf{b})) \oplus \mathsf{r} = (\mathsf{m}_\mathsf{b} \oplus \mathsf{PUF}_s(\mathsf{c} \oplus \mathsf{x}_\mathsf{b} \oplus \mathsf{x}_\mathsf{b})) \oplus \mathsf{r} = (m_b \oplus \mathsf{PUF}_s(\mathsf{c})) \oplus \mathsf{r} = \mathsf{m}_\mathsf{b} \oplus \mathsf{r} \oplus \mathsf{r} = \mathsf{m}_\mathsf{b}$. □

Receiver Security. Let the environment be denoted by $\mathcal{Z}_\mathcal{S}$. Initially, the environment chooses a bit $\mathsf{b} \in \{0,1\}$ and sends it to the honest receiver $\mathcal{R}$ as his input. The strategy for the simulator $\mathsf{Sim}_\mathcal{S}$ against a malicious sender is described in Figure 12.

1. **Sender Message:**
   - $\mathsf{Sim}_\mathcal{S}$ obtains $(\mathsf{t}_0, \mathsf{t}_1) = \mathsf{UC\text{-}Com.Commit}(\mathsf{x}_0, \mathsf{x}_1)$ along with PUF_s from $\mathcal{Z}_\mathcal{S}$.
2. **Receiver Message:**
   $\mathsf{Sim}_\mathcal{S}$ does the following:
   - Run $\mathsf{UC\text{-}Com.Extract}(\mathsf{t}_0, \mathsf{t}_1)$ to obtain $(\mathsf{x}'_0, \mathsf{x}'_1)$.
   - Choose a random string $\mathsf{c} \xleftarrow{\$} \{0,1\}^n$.
   - Compute $\mathsf{r} = \mathsf{PUF}_s(\mathsf{c})$ and $\mathsf{r}' = \mathsf{PUF}_s(\mathsf{c} \oplus \mathsf{x}'_0 \oplus \mathsf{x}'_1)$.
   - Store the pairs (c, r) and $(\mathsf{c} \oplus \mathsf{x}'_0 \oplus \mathsf{x}'_1, \mathsf{r}')$ and send PUF_s to $\mathcal{Z}_\mathcal{S}$.
3. **Sender Message:**
   - $\mathcal{Z}_\mathcal{S}$ sends $(\mathsf{x}_0, \mathsf{x}_1) = \mathsf{UC\text{-}Com.Decommit}(\mathsf{t}_0, \mathsf{t}_1)$ to $\mathsf{Sim}_\mathcal{S}$.
4. **Receiver Message:**
   $\mathsf{Sim}_\mathcal{S}$ does the following:
   - If $(\mathsf{x}'_0, \mathsf{x}'_1) \neq (\mathsf{x}_0, \mathsf{x}_1)$, abort.
   - Compute and send $\mathsf{val} = \mathsf{c} \oplus \mathsf{x}_0$ to $\mathcal{Z}_\mathcal{S}$.
5. **Sender Message:**
   - $\mathcal{Z}_\mathcal{S}$ sends $(\mathsf{S}_0, \mathsf{S}_1)$ to $\mathsf{Sim}_\mathcal{S}$.
6. **Message to $\mathcal{F}_\mathsf{ot}$:**
   - $\mathsf{Sim}_\mathcal{S}$ computes $\mathsf{m}_0 = (\mathsf{S}_0 \oplus \mathsf{r})$ and $\mathsf{m}_1 = (\mathsf{S}_1 \oplus \mathsf{r}')$.
   - Send $(\mathsf{m}_0, \mathsf{m}_1)$ to $\mathcal{F}_\mathsf{ot}$ which then outputs m_b to the receiver.

Figure 12: Simulation strategy against a malicious sender.

Claim 3. *The simulation against a malicious sender is statistically secure for the strategy given in*

Figure 12.

Proof. Let random variable abort denote whether the protocol is aborted in step 4. The view of $\mathcal{Z}_{\mathcal{S}}$ is the joint distribution $\left(\mathsf{PUF}_s, \mathsf{abort}, \mathsf{val}, \mathsf{m_b}\right)$. We show that this view is statistically indistinguishable in the real and ideal worlds.

PUF_s is created by $\mathcal{Z}_{\mathcal{S}}$. Note that in step 2, $\mathsf{Sim}_{\mathcal{S}}$ uses the honest receiver strategy to return PUF_s. Therefore, PUF_s received by $\mathcal{Z}_{\mathcal{S}}$ is identical in both worlds. In the real world, the protocol aborts in step 4 only if the decommitment does not verify. In the ideal world, $\mathsf{Sim}_{\mathcal{S}}$ aborts in step 4 only if the values $(\mathsf{x}'_0, \mathsf{x}'_1) \neq (\mathsf{x}_0, \mathsf{x}_1)$. The extraction property of the commitment scheme guarantees that if the real world does not abort, then in the ideal world $(\mathsf{x}'_0, \mathsf{x}'_1) = (\mathsf{x}_0, \mathsf{x}_1)$ with probability at least $(1 - 2^{-n})$, and vice versa. Hence, the value of the abort variable is $(1 - 2^{-n})$-close in the real and ideal worlds even conditioned on the previous view. The real and ideal views are clearly statistically indistinguishable conditioned on aborting. Next, we also show that the views in the real and ideal world are statistically indistinguishable conditioned on not aborting.

Observe that $\mathsf{Sim}_{\mathcal{S}}$ uses honest receiver strategy to choose a random string c. Moreover, since the sender cannot predict prior receiver queries to PUF_s even given PUF_s, c is identically distributed in both worlds. Since PUF_s is stateless and independent of $(\mathsf{x}_0, \mathsf{x}_1)$, the distribution of simulator queries is identically distributed to honest receiver queries. Additionally, observe that in the case of the honest receiver, he queries the PUF on two random strings and picks one of them as his desired string c based on the value of p. Therefore, the PUF can not self-destruct after the first query it receives and thus in the ideal world, the simulator can query it on both c and $\mathsf{c} \oplus \mathsf{x}'_0 \oplus \mathsf{x}'_1$.

Now, $\mathsf{val} = (\mathsf{c} \oplus \mathsf{x}_0)$ in the ideal world and $\mathsf{val} = (\mathsf{c} \oplus \mathsf{x}_\mathsf{b})$ in the real world, but c acts as a random mask (i.e., there exists $\mathsf{c}' = (\mathsf{c} \oplus \mathsf{x}_0 \oplus \mathsf{x}_1)$ that is statistically indistinguishable from c). Thus, val is statistically indistinguishable in both worlds. For the same reason, $(\mathsf{c} \oplus \mathsf{x}_1)$ in the ideal world is also statistically indistinguishable from the random variable val in the real world.

It remains to show that $\mathsf{m_b}$ (forwarded to $\mathcal{Z}_{\mathcal{S}}$ by the honest receiver) is identical in both worlds. Equivalently, we must prove that both m_0 and m_1 are extracted correctly by the simulator. It is straightforward to see that $\mathsf{m}_0 = \mathsf{S}_0 \oplus \mathsf{PUF}_s(\mathsf{val} \oplus x_0)$ is correct. Moreover, because of the sender simulation strategy, $\mathsf{m}_1 = \mathsf{S}_1 \oplus \mathsf{PUF}_s(\mathsf{val} \oplus x_1) = \mathsf{S}_1 \oplus \mathsf{PUF}_s(c \oplus x_0 \oplus x_1)$ is correctly extracted. This completes the proof.

□

Sender Security. Let the environment be denoted by $\mathcal{Z}_{\mathcal{R}}$. Initially, the environment chooses a pair of messages $\{\mathsf{m}_0, \mathsf{m}_1\} \in \{0,1\}^{2n}$ and sends it to the honest sender $\mathcal{S}$ as his input. The strategy for the simulator $\mathsf{Sim}_{\mathcal{R}}$ against a malicious receiver is described in Figure 13.

1. **Sender Message:**
   $\mathsf{Sim}_{\mathcal{R}}$ does the following:
   - Generate a PUF PUF_s.
   - Choose a pair of random strings $(\overline{\mathsf{x}}_0, \overline{\mathsf{x}}_1) \xleftarrow{\$} \{0,1\}^{2n}$.
   - Send $(\mathsf{t}_0, \mathsf{t}_1) = \mathsf{UC\text{-}Com.Commit}(\overline{\mathsf{x}}_0, \overline{\mathsf{x}}_1)$ along with PUF_s to $\mathcal{Z}_{\mathcal{R}}$.
2. **Receiver Message:**
   - $\mathcal{Z}_{\mathcal{R}}$ returns PUF_s to $\mathsf{Sim}_{\mathcal{R}}$.
3. **Sender Message:**
   $\mathsf{Sim}_{\mathcal{R}}$ does the following:
   - Choose a pair of random strings $(\mathsf{x}_0, \mathsf{x}_1) \xleftarrow{\$} \{0,1\}^{2n}$.
   - Send $(\mathsf{x}_0, \mathsf{x}_1) = \mathsf{UC\text{-}Com.Equivocate}(\mathsf{t}_0, \mathsf{t}_1)$ to $\mathcal{Z}_{\mathcal{R}}$.
4. **Receiver Message:**
   - $\mathcal{Z}_{\mathcal{R}}$ sends val to $\mathsf{Sim}_{\mathcal{R}}$.
5. **Message to $\mathcal{F}_{\mathsf{ot}}$:**
   $\mathsf{Sim}_{\mathcal{R}}$ does the following:
   - Record $\mathcal{C}$ where $\mathcal{C} = \{\mathsf{c}_1, \mathsf{c}_2, \ldots, \mathsf{c}_{\mathsf{poly}(n)}\}$ is the set of queries made by $\mathcal{Z}_{\mathcal{R}}$ to PUF_s.
   - If $\exists\, \mathsf{c} \in \mathcal{C}, \mathsf{b}' \in \{0,1\}$ such that $\mathsf{val} = \mathsf{c} \oplus \mathsf{x}_{\mathsf{b}'}$, set $\mathsf{b} = \mathsf{b}'$. Else, set $\mathsf{b} \xleftarrow{\$} \{0,1\}$.
   - Send b to $\mathcal{F}_{\mathsf{ot}}$ and obtain m_{b}.
6. **Sender Message:**
   $\mathsf{Sim}_{\mathcal{R}}$ does the following:
   - Compute $\mathsf{S}_{\mathsf{b}} = \mathsf{m}_{\mathsf{b}} \oplus \mathsf{PUF}_s(\mathsf{val} \oplus \mathsf{x}_{\mathsf{b}})$.
   - Compute $\mathsf{S}_{\overline{\mathsf{b}}} = \mathsf{y} \oplus \mathsf{PUF}_s(\mathsf{val} \oplus \mathsf{x}_{\overline{\mathsf{b}}})$ where $\mathsf{y} \xleftarrow{\$} \{0,1\}^n$.
   - Send $(\mathsf{S}_0, \mathsf{S}_1)$ to $\mathcal{Z}_{\mathcal{R}}$.

Figure 13: Simulation strategy against a malicious receiver.

Claim 4. *The simulation against a malicious receiver is statistically secure for the strategy given in Figure 13.*

Proof. The view of $\mathcal{Z}_{\mathcal{R}}$ is the joint distribution $\Big(\mathsf{PUF}_s, (\mathsf{t}_0, \mathsf{t}_1), (\mathsf{x}_0, \mathsf{x}_1), (\mathsf{S}_0, \mathsf{S}_1)\Big)$. We show that

this view is statistically indistinguishable in the real and ideal worlds. First, note that $\mathsf{Sim}_{\mathcal{R}}$ uses the honest sender's strategy to generate PUF_s. Therefore, PUF_s received by $\mathcal{Z}_{\mathcal{R}}$ is identical in both worlds. Also, observe that $\mathsf{Sim}_{\mathcal{R}}$ uses the honest sender's strategy to choose and commit to a pair of uniform random strings $(\bar{\mathsf{x}}_0, \bar{\mathsf{x}}_1)$. Since $(\mathsf{t}_0, \mathsf{t}_1)$ are commitments to this pair of random strings, $(\mathsf{t}_0, \mathsf{t}_1)$ are statistically indistinguishable in both worlds even given PUF_s. The decommitment $(\mathsf{x}_0, \mathsf{x}_1)$ in step 3 is statistically indistinguishable in the real and ideal worlds because of the equivocation property of $\mathsf{UC\text{-}Com}$.

Recall that $\mathcal{C} = \{\mathsf{c}_1, \mathsf{c}_2, \ldots, \mathsf{c}_{\mathsf{poly}(n)}\}$ is the set of queries made by $\mathcal{Z}_{\mathcal{R}}$ to PUF_s. If $(\mathsf{val} \oplus \mathsf{x}_0), (\mathsf{val} \oplus \mathsf{x}_1) \in \mathcal{C}$, then there exists $\mathsf{c}_i, \mathsf{c}_j \in \mathcal{C}$ such that $(\mathsf{c}_i \oplus \mathsf{c}_j) = (\mathsf{x}_0 \oplus \mathsf{x}_1)$. However, since $(\mathsf{x}_0, \mathsf{x}_1)$ are chosen by $\mathsf{Sim}_{\mathcal{R}}$ uniformly at random and independent of all previous messages sent or received, $\mathsf{Pr}\left[\ \exists\ \mathsf{c}_i, \mathsf{c}_j \in \mathcal{C}\ \mathsf{s.t}\ (\mathsf{c}_i \oplus \mathsf{c}_j) = (\mathsf{x}_0 \oplus \mathsf{x}_1)\right] = \frac{|C|^2}{2^n} \leq \frac{1}{2^{n/2}}$.

Thus, there exists at most one b such that $(\mathsf{val} \oplus \mathsf{x}_{\mathsf{b}}) \in \mathcal{C}$ except with probability $\frac{1}{2^{n/2}}$. Note that $\mathsf{Sim}_{\mathcal{R}}$ extracts b by comparing $(\mathsf{val} \oplus \mathsf{x}_0)$ and $(\mathsf{val} \oplus \mathsf{x}_1)$ with every $\mathsf{c}_i \in \mathcal{C}$. Therefore, S_{b}, computed as $\mathsf{S}_{\mathsf{b}} = \mathsf{m}_{\mathsf{b}} \oplus \mathsf{PUF}_s(\mathsf{val} \oplus \mathsf{x}_{\mathsf{b}})$ in both worlds is at most $\frac{1}{2^{n/2}}$-far even given the previous view.

It remains to show that $\mathsf{S}_{\bar{\mathsf{b}}}$ is close in the real and ideal worlds even given the previous view. Note that $(\mathsf{val} \oplus \mathsf{x}_{\bar{\mathsf{b}}}) \notin \mathcal{C}$. Since PUF_s is unpredictable, $\mathcal{Z}_{\mathcal{R}}$ cannot learn the output of PUF_s on $(\mathsf{val} \oplus \mathsf{x}_{\bar{\mathsf{b}}})$ without having queried it and since PUF_s is unclonable, $\mathcal{Z}_{\mathcal{R}}$ cannot create a clone of PUF_s and query it on $(\mathsf{val} \oplus \mathsf{x}_{\bar{\mathsf{b}}})$ after having sent back PUF_s in step 2. Therefore, $\mathsf{PUF}_s(\mathsf{val} \oplus \mathsf{x}_{\bar{\mathsf{b}}})$ is $(1 - 2^{-n/2})$-close to uniform, and $\mathsf{S}_{\bar{\mathsf{b}}}$ is $(1 - 2^{-n/2})$-close to uniform even given the previous view. Thus, $\mathsf{S}_{\bar{\mathsf{b}}}$ is statistically independent from the rest of $\mathcal{Z}_{\mathcal{R}}$'s view, unless $\mathcal{Z}_{\mathcal{R}}$ queried PUF_s on a value which it can guess with probability only 2^{-n}. This completes our proof. □

F Proof of Security for OT in Malicious Bounded Stateful PUF Model

In this section, we give the formal proof of Theorem 2.

Claim 5 (Correctness)**.** *For all* $(\mathsf{m}_0, \mathsf{m}_1) \in \{0,1\}^{2n}$ *and* $\mathsf{b} \in \{0,1\}$*, the output of* $\mathcal{R}$ *equals* m_{b}.

Proof. $\mathcal{R}$ outputs $(\mathsf{S}_{\mathsf{b}} \oplus \mathsf{r})$, where $\mathsf{S}_{\mathsf{b}} = \mathsf{m}_{\mathsf{b}} \oplus \mathsf{PUF}_s(\mathsf{val} \oplus \mathsf{x}_{\mathsf{b}} \oplus \hat{\mathsf{x}}_{\mathsf{b}}) = \mathsf{m}_{\mathsf{b}} \oplus \mathsf{PUF}_s(\mathsf{c}) = \mathsf{m}_{\mathsf{b}} \oplus \mathsf{r}$. Thus, $\mathcal{R}$ outputs m_{b}. □

Receiver Security. Let the environment be denoted by $\mathcal{Z}_{\mathcal{S}}$. The environment chooses a bit $b \in \{0,1\}$ and sends it to the honest receiver as his input. The strategy for the simulator $\mathsf{Sim}_{\mathcal{S}}$ against a malicious sender is described in Figure 14.

Repeat the following protocol K times in parallel for fresh inputs.
Given: Malicious stateful PUFs.

1. **Sender Message:** Obtain PUF_s and (t_0, t_1) from $\mathcal{Z}_{\mathcal{S}}$.
2. **Receiver Message:** $\mathsf{Sim}_{\mathcal{S}}$ does the following.
   - Obtain (x_0, x_1) using $\mathsf{UC\text{-}Com.Extract}(t_0, t_1)$.
   - Pick $(\hat{\mathsf{x}}_0, \hat{\mathsf{x}}_1) \xleftarrow{\$} \{0,1\}^{2n}$ (following honest strategy).
   - Choose a pair of random strings $(\mathsf{c}_0, \mathsf{c}_1) \xleftarrow{\$} \{0,1\}^{2n}$ such that $(\mathsf{c}_0 \oplus \mathsf{c}_1) = (\mathsf{x}_0 \oplus \mathsf{x}_1 \oplus \hat{\mathsf{x}}_0 \oplus \hat{\mathsf{x}}_1)$.
   - Compute $\mathsf{r}_0 = \mathsf{PUF}_s(\mathsf{c}_0), \mathsf{r}_1 = \mathsf{PUF}_s(\mathsf{c}_1)$.
   - Set $\mathsf{c} = \mathsf{c}_{\mathsf{p}}$ and $\mathsf{r} = \mathsf{r}_{\mathsf{p}}$ for $\mathsf{p} \xleftarrow{\$} \{0,1\}$ and store the pair (c, r).
   - Send $(\hat{\mathsf{x}}_0, \hat{\mathsf{x}}_1)$ with PUF_s, to $\mathcal{Z}_{\mathcal{S}}$.
3. **Sender Message:** $\mathcal{S}^a$ sends $(\mathsf{x}_0, \mathsf{x}_1) = \mathsf{UC\text{-}Com.Decommit}(\mathsf{t}_0, \mathsf{t}_1)$ to $\mathsf{Sim}_{\mathcal{S}}$.
4. **Receiver Message:** $\mathsf{Sim}_{\mathcal{S}}$ does the following.
   - If $\mathsf{UC\text{-}Com.Decommit}(\mathsf{t}_0, \mathsf{t}_1)$ does not verify, abort. Else, pick $\mathsf{b} \xleftarrow{\$} \{0,1\}$ and compute $\mathsf{val} = \mathsf{c} \oplus \mathsf{x}_{\mathsf{b}} \oplus \hat{\mathsf{x}}_{\mathsf{b}}$.
   - Send (val) to $\mathcal{Z}_{\mathcal{S}}$.
5. **Simulator Message to $\mathcal{F}_{\mathsf{ot}}$:** Obtain $(\mathsf{S}_0, \mathsf{S}_1)$ from $\mathcal{Z}_{\mathcal{S}}$.
   - Compute $\mathsf{m}_0 = \mathsf{S}_0 \oplus \mathsf{PUF}_s(\mathsf{val} \oplus \mathsf{x}_0 \oplus \hat{\mathsf{x}}_0)$.
   - Compute $\mathsf{m}_1 = \mathsf{S}_1 \oplus \mathsf{PUF}_s(\mathsf{val} \oplus \mathsf{x}^1 \oplus \hat{\mathsf{x}}_1)$.
   - Send (m_0, m_1) to $\mathcal{F}_{\mathsf{ot}}$, such that $\mathcal{F}_{\mathsf{ot}}$ sends m_b to the honest receiver.

[a] We abuse notation in some places and use $\mathcal{S}$ and $\mathcal{Z}_{\mathcal{S}}$ interchangeably. Similarly, for $\mathcal{R}$ and $\mathcal{Z}_{\mathcal{R}}$.

Figure 14: Sender simulation for K 2-choose-1 OTs (with at most ℓ leakage) in the malicious stateful PUF model.

Claim 6. *The sender simulation is statistically secure, for the strategy in Figure 14.*

Proof. Since the pairs $(\hat{x}_0, \hat{x}_1)$ are chosen independently of PUF_s, in the view of a (possibly stateful) PUF_s, the distribution of simulator queries are statistically indistinguishable from the distribution of honest sender queries, thus the probability of aborting in Step 2 (owing to a malicious aborting stateful PUF_s) is statistically close in both the worlds.

The simulator follows honest receiver strategy for all the next messages. By correctness of $\mathsf{UC\text{-}Com.Extract}(\cdot)$ and because the distribution of PUF_s queries are statistically indistinguishable

in the real and ideal worlds; it follows (using a similar argument as Section 3) that the simulator extracts (m_0, m_1) correctly, conditioned on not aborting in Step 2. This proves that the view of the malicious sender is close in the real and ideal worlds, except the receiver queries to PUF_s (which may be recorded by a malicious stateful PUF_s). However, since PUF_s can have at most ℓ bits of state, it obtains at most ℓ bits joint leakage over all K OTs. □

Repeat the following protocol K times in parallel for fresh inputs.
Given: Malicious stateful PUFs.

1. **Sender Message:** The simulator $\mathsf{Sim}_{\mathcal{R}}$ follows honest sender strategy to:
   - Generate a PUF $\mathsf{PUF}_s : \{0,1\}^n \to \{0,1\}^n$.
   - Choose a pair of random strings $(\mathsf{x}_0, \mathsf{x}_1) \xleftarrow{\$} \{0,1\}^{2n}$.
   - Compute $(\mathsf{t}_0, \mathsf{t}_1) = \mathsf{UC\text{-}Com.Commit}(\mathsf{x}_0, \mathsf{x}_1)$ and send $(\mathsf{t}_0, \mathsf{t}_1)$ to $\mathcal{Z}_{\mathcal{R}}$.
   - Send PUF_s to $\mathcal{Z}_{\mathcal{R}}$.
2. **Receiver Message:**
   $\mathcal{Z}_{\mathcal{R}}$ sends $(\hat{\mathsf{x}}_0, \hat{\mathsf{x}}_1) \xleftarrow{\$} \{0,1\}^{2n}$ along with PUF_s, to $\mathsf{Sim}_{\mathcal{R}}$.
3. **Sender Message:** $\mathsf{Sim}_{\mathcal{R}}$ follows honest sender strategy to send $(\mathsf{x}_0, \mathsf{x}_1) = \mathsf{UC\text{-}Com.Decommit}(\mathsf{t}_0, \mathsf{t}_1)$ to $\mathcal{Z}_{\mathcal{R}}$.
4. **Receiver Message:** Obtain val from $\mathcal{Z}_{\mathcal{R}}$.
5. **Simulator Message to Ideal OT:** $\mathsf{Sim}_{\mathcal{R}}$ does the following.
   - If there exists b' such that for each query to PUF_s by $\mathcal{Z}_{\mathcal{R}}$, val and $x \in \{(x_0 \oplus \hat{x}_0), (x_1 \oplus \hat{x}_1)\}, \mathsf{val} = c \oplus x_{b'} \oplus \hat{x}_{b'})$, set $b = b'$. Else set $b = 1$.
   - Send b to $\mathcal{F}_{\mathsf{ot}}$ and obtain output m_b. Pick $m_{\bar{b}} \xleftarrow{\$} \{0,1\}$.
6. **Sender Message:**
   - Compute: $\mathsf{S}_0 = \mathsf{m}_0 \oplus \mathsf{PUF}_s(\mathsf{val} \oplus \mathsf{x}_0 \oplus \hat{\mathsf{x}}_0)$, and $\mathsf{S}_1 = \mathsf{m}_1 \oplus \mathsf{PUF}_s(\mathsf{val} \oplus \mathsf{x}_1 \oplus \hat{\mathsf{x}}_1)$.
   - Send $(\mathsf{S}_0, \mathsf{S}_1)$ to $\mathcal{Z}_{\mathcal{R}}$.

Figure 15: Receiver Simulation for K OTs (with $\leq \ell$ leakage).

Sender Security. Let the environment be denoted by $\mathcal{Z}_{\mathcal{R}}$. The environment chooses a pair of messages $\{m_0, m_1\}$ and sends them to the honest sender as his input. The simulation strategy $\mathsf{Sim}_{\mathcal{R}}$ against a malicious receiver is described in Figure 15.

Claim 7. *The receiver simulation is statistically secure for the strategy in Figure 15.*

Proof. Recall that the only modification in this section is to add security against a sender that maliciously creates stateful PUFs. The argument against a malicious receiver remains essentially the same.

Following the same arguments as Section 3 , we can now show that the simulator extracts the correct input bit b in Step 5, such that the distribution of $(S_b, S_{\bar{b}})$ is at most $2^{-\frac{n}{2}}$-far in the real and ideal worlds. □

This completes the proof of Theorem 2.

G One-Sided Correlation Extractors with Malicious Security: Proofs

In this section, we give the proof of security of the one-sided malicious OT-extractor described in Figure 4.

Correctness.

Claim 8. *For all* $(m_0, m_1) \in \{0,1\}^2$ *and* $b \in \{0,1\}$*, the output of* $\mathcal{R}$ *equals* x_b*.*

Proof. The receiver computes: $y_b \oplus \mathcal{E}.\mathsf{Ext}(m_b^1||m_b^2||m_b^3 \dots m_b^K, s)$
$= x_b \oplus \mathcal{E}.\mathsf{Ext}(m_b^1||m_b^2||m_b^3 \dots m_b^K, s) \oplus \mathcal{E}.\mathsf{Ext}(m_b^1||m_b^2||m_b^3 \dots m_b^K, s) = x_b.$ □

Receiver Security. Let the environment be denoted by $\mathcal{Z}_{\mathcal{S}}$. The environment chooses a bit $b \in \{0,1\}$ and sends it to the honest receiver as his input. The strategy for the simulator $\mathsf{Sim}_{\mathcal{S}}$ against a malicious sender is described in Figure 16.

Let $\mathcal{E} : \{0,1\}^K \times \{0,1\}^n \to \{0,1\}$ be a strong randomness $(K, 2^{-n})$-extractor for seed length $d = O(n)$.

1. **Invoking OT Correlations:**
   - For $i \in [K]$, $\mathcal{S}$ sends input m_0^i, m_1^i to the i^{th} OT.
   - For $i \in [K]$, $\mathsf{Sim}_{\mathcal{S}}$ honestly emulates the i^{th} OT, and sets receiver input (the same) choice bit b.
2. **Sender Message:**
   - $\mathsf{Sim}_{\mathcal{S}}$ obtains (y_0, y_1, s) from $\mathcal{S}$.
   - For $b \in \{0,1\}$, $\mathsf{Sim}_{\mathcal{S}}$ computes $x_{\mathsf{b}} = y_{\mathsf{b}} \oplus \mathcal{E}.\mathsf{Ext}(m_{\mathsf{b}}^1||m_{\mathsf{b}}^2||m_{\mathsf{b}}^3 \dots ||m_{\mathsf{b}}^K, s)$.
   - $\mathsf{Sim}_{\mathcal{S}}$ sends (x_0, x_1) to $\mathcal{F}_{\mathsf{ot}}$, which generates the output for $\mathcal{R}$.

Figure 16: Sender Simulator for Malicious Correlation Extractor.

Claim 9. *The simulation against a malicious receiver is perfectly secure for the strategy given in Figure 16.*

Proof. Since the sender obtains no message from the receiver, it suffices to show that the simulator correctly extracts the sender's input. Note that the simulator extracts $y_0 \oplus \mathcal{E}.\mathsf{Ext}(m_0^1||m_0^2||m_0^3 \dots m_0^K, s)$ which equals x_0, and $y_0 \oplus \mathcal{E}.\mathsf{Ext}(m_1^1||m_1^2||m_1^3 \dots m_1^K$ which equals x_1. Thus, extraction occurs correctly, and the joint distribution (x_0, x_1, b, x_b) is identical in the real and ideal worlds. □

Sender Security. Let the environment be denoted by $\mathcal{Z}_{\mathcal{R}}$. The environment chooses inputs $(x_0, x_1) \in \{0,1\}^2$ and sends them to the honest sender as his input. The strategy for the simulator $\mathsf{Sim}_{\mathcal{R}}$ against a malicious receiver is described in Figure 17.

Let $\mathcal{E} : \{0,1\}^K \times \{0,1\}^n \to \{0,1\}$ be a strong randomness $(K, 2^{-n})$-extractor for seed length $d = O(n)$.

1. **Invoking OT Correlations:**
   - For $i \in [K]$, $\mathsf{Sim}_{\mathcal{R}}$ picks inputs $m_0^i, m_1^i \xleftarrow{\$} \{0,1\}^2$ and sends (m_0^i, m_1^i) as input to the i^{th} OT.
   - **Receiver Leakage.** $\mathcal{R}$ requests L as an ℓ-bit leakage function on the joint distribution of $(m_0^1, m_1^1, m_0^2, m_1^2, \dots m_0^K, m_1^K)$. $\mathsf{Sim}_{\mathcal{R}}$ sends $L(m_0^1, m_1^1, m_0^2, m_1^2, \dots m_0^K, m_1^K)$ to $\mathcal{R}$.
   - For $i \in [K]$, $\mathsf{Sim}_{\mathcal{R}}$ obtains inputs b_i (not necessarily all equal) from $\mathcal{R}$. $\mathsf{Sim}_{\mathcal{R}}$ honestly emulates the i^{th} OT and outputs $m_{b_i}^i$ to $\mathcal{R}$.
2. **Sender Message:**
   - $\mathsf{Sim}_{\mathcal{R}}$ finds the value of $\mathsf{bit} \in \{0,1\}$ such that $|\{i : b_i = \mathsf{bit}\}| \leq \ell + n$.
   - $\mathsf{Sim}_{\mathcal{R}}$ sends $\mathsf{bit}' = 1 - \mathsf{bit}$ to $\mathcal{F}_{\mathsf{ot}}$ and obtains $x_{\mathsf{bit}'}$. It picks random seed s and compute $y_{\mathsf{bit}'} = \mathcal{E}.\mathsf{Ext}(m_{\mathsf{bit}'}^1||m_{\mathsf{bit}'}^2||m_{\mathsf{bit}'}^3 \dots ||m_{\mathsf{bit}'}^K, s) \oplus x_{\mathsf{bit}'}$. It sets $y_{\mathsf{bit}} \xleftarrow{\$} \{0,1\}$, and sends (s, y_0, y_1) to $\mathcal{R}$.

Figure 17: Receiver Simulator for Malicious Correlation Extractor.

Claim 10. *The simulation against a malicious sender is statistically secure for the strategy given in Figure 17.*

Proof. Until Step 1, the view of the receiver is identical in the real and ideal worlds, because the simulator follows honest sender strategy. Note that for any receiver strategy $\mathcal{Z}_{\mathcal{R}}$, there must exist some $\mathsf{bit} \in \{0,1\}$ such that $|\{i : b_i = \mathsf{bit}\}| \leq \ell + n$. $\mathsf{Sim}_{\mathcal{R}}$ sends $\mathsf{bit}' = (1 - \mathsf{bit})$ to $\mathcal{F}_{\mathsf{ot}}$ and obtains $x_{\mathsf{bit}'}$. It uses honest sender strategy to compute $y_{\mathsf{bit}'}$. Thus, $y_{\mathsf{bit}'}$ is identically distributed in the real and ideal worlds.

Moreover, with ℓ bits of additional universal leakage over all pairs of sender inputs $(m_0^1, m_1^1, m_0^2, m_1^2, \ldots m_0^K, m_1^K)$, the receiver obtains $\leq 2\ell + n$ bits of entropy over the $K = (2\ell + 2n)$ bit string $(m_{\mathsf{bit}}^1 || m_{\mathsf{bit}}^2 || \ldots m_{\mathsf{bit}}^K)$.

The distribution of the seed s is identical in the real and simulated worlds, because $\mathsf{Sim}_{\mathcal{R}}$ follows honest sender strategy. Therefore, $(\mathcal{E}.\mathsf{Ext}((m_{\mathsf{bit}}^1 || m_{\mathsf{bit}}^2 || \ldots m_{\mathsf{bit}}^K), s), s)$ is at most 2^{-n} far from (U_n, s), where U_n denotes the uniform distribution over $\{0,1\}$ (which is the distribution from which y_{bit} is picked by $\mathsf{Sim}_{\mathcal{R}}$). Therefore, y_{bit} is at most 2^{-n}-far in the real and ideal worlds. □

H High Production Rate

Here, we show how to obtain an improved production rate at the cost of higher simulation error. We sample small disjoint subsets with sufficiently high entropy, and then run the protocol in Figure 4 independently on these subsets to obtain a single OT from each subset.

In our case, we use the trivial sub-sampling technique of picking random indices with suitable probability. In case a sample repeats itself, we discard it and re-sample. We work in the setting where $N/2 - \ell \geq K$. Our main technical lemma is the following.

Imported Lemma 1 (Sub-sampling [44, 23])**.** *Let $(A_{[n]}, L)$ be a joint distribution such that, there exists a constant $\mu \in \{0,1\}$ such that, $\tilde{H}_\infty(A_{[n]}, L) \geq \mu n$. For every constant $\epsilon \in (0, \mu)$ and $\rho = \omega(\log n)$, there exists an efficient algorithm which outputs $(S_1, \ldots S_m) \in \left(2^{[n]}\right)^{[m]}$ such that $m = n/\rho$ and with probability $1 - \mathsf{negl}(n)$, the following holds:*

1. *Large and Distinct: There exists a constant $\lambda \in \{0,1\}$ such that $|S_i| = \lambda\rho$. We have $S_i \cap S_j = \phi$, for all $i, j \in [m]$ and $i \neq j$.*
2. *High Entropy: $\tilde{H}_\infty(S_{i+1} | S_{[i]}, L) \geq (\mu - \epsilon)|S_{i+1}$.*

Now, the result of Theorem 4 can be obtained as a direct application of Imported Lemma 1. We will be working in the setting when $\frac{N}{2} - \ell \geq n$. Now, we apply Imported Lemma 1 to obtain the disjoint sets $S_1, S_2, \ldots S_m$ for $m = n/log^2 n$ and $\rho = \log^2 n$. Next, we apply the protocol in Figure 4 to each of the sets independently for the following choice of parameters: $n' = |S_i| = \lambda\rho, t'_R = (\frac{t_R}{n} + \epsilon)|S_i|$. Then, the simulation error is bounded by $2^{-\theta(\log^2 n)} = \mathsf{negl}(n)$.

I From UC Oblivious Transfer to UC Two-Party Computation

In this section, we show that given UC oblivious transfer, we can obtain UC secure two party computation of any functionality. This section is taken almost verbatim from [10]. The main idea is to first construct a semi-honest secure two-party computation protocol using Yaos garbled-circuit protocol, and to then apply the compiler of Ishai, Prabhakaran, and Sahai [28].

Semi-honest secure two-party computation. Lindell and Pinkas presented a proof for Yao's two-party secure-computation protocol[34]. They show how to instantiate the garbling part of the protocol with a private-key encryption scheme having certain properties. In addition, the authors show that any pseudorandom function is sufficient to instantiate such a private-key encryption scheme. Our main observation is that we can replace the pseudorandom function with a PUF. This

has already been observed before by Brzuska et al. [5] in a different context. With this observation, we can apply the result of [34] to obtain a protocol for semi-honest secure two-party computation based on PUFs only (and no computational assumptions).

Imported Theorem 1. *Let f be any functionality. Then there is a (constant-round) protocol that securely computes f for semi-honest adversaries in the $(\mathcal{F}_{\mathsf{PUF}}, \mathcal{F}_{\mathsf{ot}})$- hybrid model.*

We omit the proof since it follows easily from prior work.

Universally composable two-party computation. In the next step we apply the IPS compiler[28], a black-box compiler that takes as input

- an "outer" MPC protocol π with security against a constant fraction of malicious parties.
- an "inner" two-party protocol ρ, in the $\mathcal{F}_{\mathsf{ot}}$-hybrid model, where the security of ρ only needs to hold against semi-honest parties.

and outputs a two-party protocol $\phi_{\pi,\rho}$ which is secure in the $\mathcal{F}_{\mathsf{ot}}$-hybrid model against malicious corruptions.
In our setting, we must be careful to give information-theoretic instantiations of the outer and inner protocols so that our final protocol $\phi_{\pi,\rho}$ will be unconditionally secure in the $\mathcal{F}_{\mathsf{ot}}$-hybrid model. Fortunately, we may instantiate the outer protocol, π, with the seminal BGW protocol [3] and may instantiate the inner protocol, ρ, with the semi-honest version of the two-party GMW protocol [19] in the $\mathcal{F}_{\mathsf{ot}}$-hybrid model.
Let ψ denote any OT-protocol described in this paper and let $\psi^{\phi}_{\pi,\rho}(f)$ denote the IPS-compiled protocol which makes subroutine calls to ψ instead of $\mathcal{F}_{\mathsf{ot}}$ and computes the functionality f. Therefore, using the above theorem, along with the UC composition theorem, we obtain the following result:

Imported Theorem 2. *For any functionality f, protocol $\psi^{\phi}_{\pi,\rho}(f)$ securely computes f in the $(\mathcal{F}_{\mathsf{PUF}}, \mathcal{F}_{\mathsf{aux}})$-hybrid model.*

J UC Computation with Encapsulated Malicious PUFs: Proofs

In this section, we describe the proof of security of the UC secure OT protocol in the malicious encapsulated stateless PUF model shown in 5.

Receiver Security. Let the environment be denoted by $\mathcal{Z}_{\mathcal{S}}$. The environment chooses bit $\mathsf{b} \in \{0,1\}$ and sends it to the honest receiver as his input. The strategy for simulator $\mathsf{Sim}_{\mathcal{S}}$ against a malicious sender is described in Figure 18.

Claim 11. *The sender simulation is statistically secure, for the strategy in Figure 18.*

Proof. Since the pairs $(\hat{\mathsf{x}}^i_0, \hat{\mathsf{x}}^i_1)$ are chosen independently of PUF_s and the simulator follows honest receiver strategy for all other messages, the view of the sender is statistically indistinguishable in both worlds. By the correctness of $\mathsf{UC\text{-}Com.Extract}(\cdot)$ and because the distribution of PUF_s queries is statistically indistinguishable in the real and ideal worlds; it follows (using a similar argument as Section 3) that the simulator extracts (m_0, m_1) correctly, conditioned on not aborting in Step 2. □

1. **Coin Flipping:** For each $i \in [n]$, $\mathsf{Sim}_{\mathcal{S}}$ does the following:
   - Choose $\mathsf{c}_1^i \xleftarrow{\$} \{0,1\}^n$ and send $\mathsf{d}^i = \mathsf{UC\text{-}Com.Commit}(\mathsf{c}_1^i)$ to $\mathcal{Z}_{\mathcal{S}}$.
   - Obtain c_2^i from $\mathcal{Z}_{\mathcal{S}}$ and compute $\mathsf{c}^i = \mathsf{c}_1^i \oplus \mathsf{c}_2^i$.
2. **Sender Message:** $\mathsf{Sim}_{\mathcal{S}}$ obtains PUF_s and (t_0^i, t_1^i) for $i \in [K]$, from $\mathcal{Z}_{\mathcal{S}}$.
3. **Receiver Message:** $\mathsf{Sim}_{\mathcal{S}}$ does the following.
   - Obtain $(\mathsf{x}_0^i, \mathsf{x}_1^i)$ using $\mathsf{UC\text{-}Com.Commit}(t_0^i, t_1^i)$. For all $i \in [n]$, pick $(\hat{\mathsf{x}}_0^i, \hat{\mathsf{x}}_1^i) \xleftarrow{\$} \{0,1\}^{2n^2}$.
   - For each $i \in [n]$, set $\mathsf{c}_0^i = (\mathsf{c}^i \oplus x_0^i \oplus x_1^i \oplus \hat{\mathsf{x}}_0^i \oplus \hat{\mathsf{x}}_1^i)$. Compute $rr_0^i = \mathsf{PUF}_s(\mathsf{c}_0^i), rr^i = \mathsf{PUF}_s(\mathsf{c}^i)$. Send PUF_s to $\mathcal{Z}_{\mathcal{S}}$.
4. **Sender Message:** Obtain $(\mathsf{x}_0^i, \mathsf{x}_1^i) = \mathsf{UC\text{-}Com.Decommit}(\mathsf{t}_0^i, \mathsf{t}_1^i)$ from $\mathcal{Z}_{\mathcal{S}}$.
5. **Receiver Message:** $\mathsf{Sim}_{\mathcal{S}}$ does the following using honest receiver strategy.
   - For each $i \in [n]$, if $\mathsf{UC\text{-}Com.Decommit}(\mathsf{t}_0^i, \mathsf{t}_1^i)$ does not verify, abort. Else pick $\mathsf{b}_i \xleftarrow{\$} \{0,1\}$, compute $\mathsf{val}^i = \mathsf{c}^i \oplus \mathsf{x}_{\mathsf{b}_i}^i$ Send $(\mathsf{val}^1, \mathsf{val}^2, \ldots, \mathsf{val}^n)$ to $\mathcal{Z}_{\mathcal{S}}$.
6. **Cut-and-choose:** $\mathsf{Sim}_{\mathcal{S}}$ follows honest receiver strategy.
   - Obtains $t_{\mathcal{S}}$ from $\mathcal{Z}_{\mathcal{S}}$. $\mathsf{Sim}_{\mathcal{S}}$ picks and sends $r_{\mathcal{R}} \xleftarrow{\$} \{0,1\}^{2K}$. $\mathsf{Sim}_{\mathcal{S}}$ obtains $r_{\mathcal{S}} = \mathsf{UC\text{-}Com.Decommit}(t_{\mathcal{S}})$ from $\mathcal{S}$ and $(\mathcal{S}, \mathsf{Sim}_{\mathcal{S}})$ use $(r_{\mathcal{S}} \oplus r_{\mathcal{R}})$ to pick a subset I of indices $i \in [n]$, of size $\frac{K}{2}$.
   - For $i \in [I]$, $\mathsf{Sim}_{\mathcal{S}}$ sends $\mathsf{c}_1^i = \mathsf{UC\text{-}Com.Decommit}(\mathsf{d}^i)$.
7. **Receiver Message:** $\mathsf{Sim}_{\mathcal{S}}$ picks $\mathsf{b}' \xleftarrow{\$} \{0,1\}$, and for all $i \in [n] \setminus I$, sends $\mathsf{bc}_i = \mathsf{b}_i \oplus \mathsf{b}'$.
8. **Sender Message:** Obtain $(\mathsf{S}_0, \mathsf{S}_1)$ from $\mathcal{Z}_{\mathcal{S}}$.
   - Compute $\mathsf{m}_0 = \mathsf{S}_0 \bigoplus_{i \in n \setminus I} \mathsf{PUF}_s(\mathsf{val}^i \oplus \mathsf{x}_{\mathsf{bc}_i}^i \oplus \hat{\mathsf{x}}_{\mathsf{bc}_i}^i)$ and $m_1 = S_1 \bigoplus_{i \in n \setminus I} \mathsf{PUF}_s(\mathsf{val}^i \oplus \mathsf{x}_{1-\mathsf{bc}_i}^i \oplus \hat{\mathsf{x}}_{1-\mathsf{bc}_i}^i)$. Note that $\mathsf{Sim}_{\mathcal{S}}$ knows, for each i, $\mathsf{PUF}_s(\mathsf{val}^i \oplus \mathsf{x}_0^i \oplus \hat{\mathsf{x}}_0^i)$ and $\mathsf{PUF}_s(\mathsf{val}^i \oplus \mathsf{x}_1^i \oplus \hat{\mathsf{x}}_1^i)$.
   - Send (m_0, m_1) to $\mathcal{F}_{\mathsf{ot}}$, such that $\mathcal{F}_{\mathsf{ot}}$ sends m_b to the honest receiver.

Figure 18: Sender simulation for malicious stateless PUFs with encapsulation.

Sender Security. Let the environment be denoted by $\mathcal{Z}_{\mathcal{R}}$. The environment chooses inputs $(m_0, m_1) \in \{0,1\}^K$ and sends them to the honest sender as his input. The strategy for simulator $\mathsf{Sim}_{\mathcal{R}}$ against a malicious receiver is described in Figure 19.

Claim 12. *The receiver simulation is statistically secure, for the strategy in Figure 19.*

Proof. Let random variable abort denote whether the protocol is aborted in step 4. The view of $\mathcal{Z}_{\mathcal{R}}$ is $\left(\mathsf{PUF}_s, (\mathsf{t}_0^i, \mathsf{t}_1^i), \mathsf{abort}, (\mathsf{x}_0^i, \mathsf{x}_1^i), (\hat{\mathsf{x}}_0^i, \hat{\mathsf{x}}_1^i), (\mathsf{S}_0, \mathsf{S}_1)\right)$ for all $i \in [n]$. We show that this view is statistically indistinguishable in the real and ideal worlds. First, note that $\mathsf{Sim}_{\mathcal{R}}$ uses the honest sender's strategy to generate PUF_s. Therefore, PUF_s received by $\mathcal{Z}_{\mathcal{R}}$ is identical in both worlds. Also, observe that, for all $i \in [n]$, $\mathsf{Sim}_{\mathcal{R}}$ uses the honest sender strategy to choose and commit to pair of uniform random strings $(\mathsf{x}_0^i, \mathsf{x}_1^i)$. Since $(\mathsf{t}_0^i, \mathsf{t}_1^i)$ are commitments to this pair of random strings, $(\mathsf{t}_0^i, \mathsf{t}_1^i)$ are statistically indistinguishable in both worlds even given PUF_s. Observe that the protocol aborts only if any of the test queries do not verify. Since all the test queries are picked from an identical distribution in both worlds (independent of sender input) and the entire view of the receiver is statistically indistinguishable so far, the probability of verification is statistically close in both worlds. Hence, the value of the abort variable is statistically close. Conditioned on aborting, the view is statistically indistinguishable in both worlds and it remains to show that the view is close even conditioned on not aborting in Step 4. For every $i \in [n]$, the decommitment $(\mathsf{x}_0^i, \mathsf{x}_1^i)$ in step 3 is statistically indistinguishable in the real and ideal worlds because of the equivocation property of $\mathsf{UC\text{-}Com}$.

Recall that $\mathcal{C} = \{\mathsf{c}_1, \mathsf{c}_2, \ldots, \mathsf{c}_{\mathsf{poly}(n)}\}$ is the set of queries made by $\mathcal{Z}_{\mathcal{R}}$ to PUF_s. If there exists $i \in [n]$ such that $(\mathsf{val}^i \oplus \mathsf{x}_0^i), (\mathsf{val}^i \oplus \mathsf{x}_1^i) \in \mathcal{C}$, then there exists $\mathsf{c}_i, \mathsf{c}_j \in \mathcal{C}$ such that $(\mathsf{c}_i \oplus \mathsf{c}_j) = (\mathsf{x}_0^i \oplus \mathsf{x}_1^i)$. However, since $(\mathsf{x}_0^i, \mathsf{x}_1^i)$ are chosen by $\mathsf{Sim}_{\mathcal{R}}$ uniformly at random and independent of all previous messages sent or received, $\mathsf{Pr}\ [\ \exists\ \mathsf{c}_i, \mathsf{c}_j \in \mathcal{C}\ \mathsf{s.t}\ (\mathsf{c}_i \oplus \mathsf{c}_j) = (\mathsf{x}_0^i \oplus \mathsf{x}_1^i)] = \frac{|C|^2}{2^n} \leq \frac{1}{2^{n/2}}$.

Thus, there exists at most one b such that $(\mathsf{val}^i \oplus \mathsf{x}_{\mathsf{b}}^i) \in \mathcal{C}$ for all i except with probability $\frac{1}{2^{n/2}}$. Note that $\mathsf{Sim}_{\mathcal{R}}$ extracts b by comparing, for every $i \in [n]$, $(\mathsf{val}^i \oplus \mathsf{x}_0^i)$ and $(\mathsf{val}^i \oplus \mathsf{x}_1^i)$ with every $\mathsf{c}_i \in \mathcal{C}$. Therefore, S_{b}, computed as $\mathsf{S}_{\mathsf{b}} = \mathsf{m}_{\mathsf{b}} \oplus \mathsf{PUF}_s(\mathsf{val}^1 \oplus \mathsf{x}_{\mathsf{b}}^1) \oplus \ldots \oplus \mathsf{PUF}_s(\mathsf{val}^n \oplus \mathsf{x}_{\mathsf{b}}^n)$ in both worlds is at most $\frac{1}{2^{n/2}}$-far even given the previous view.

It remains to show that $\mathsf{S}_{\overline{\mathsf{b}}}$ is close in the real and ideal worlds even given the previous view. Note that for all $i \in [n]$, $(\mathsf{val}^i \oplus \mathsf{x}_{\overline{\mathsf{b}}}^i) \notin \mathcal{C}$ except with probability $\frac{1}{2^{n/2}}$. We split the rest of our analysis into two cases.

Case 1 : Suppose $\mathcal{Z}_{\mathcal{R}}$ returned the original PUF_s in step 2. Since PUF_s is unpredictable, $\mathcal{Z}_{\mathcal{R}}$ cannot learn the output of PUF_s on $(\mathsf{val}^i \oplus \mathsf{x}_{\overline{\mathsf{b}}}^i)$ without having queried it and since PUF_s is unclonable, $\mathcal{Z}_{\mathcal{R}}$ cannot create a clone of PUF_s and query it on $(\mathsf{val}^i \oplus \mathsf{x}_{\overline{\mathsf{b}}}^i)$ after having sent back PUF_s in step 2. Therefore, $\mathsf{PUF}_s(\mathsf{val}^i \oplus \mathsf{x}_{\overline{\mathsf{b}}}^i)$ is at least $(1 - 2^{-n/2})$-close to uniform and hence $\mathsf{S}_{\overline{\mathsf{b}}}$ is at least $(1 - 2^{-n/2})$-close to uniform even given the previous view.

Case 2 : Suppose $\mathcal{Z}_{\mathcal{R}}$ returned a different malicious stateless PUF $\widehat{PUF}_s$ in step 2 that possibly internally encapsulated PUF_s. In step 6, $\mathsf{Sim}_{\mathcal{R}}$ makes n queries to $\widehat{PUF}_s$ to compute $\mathsf{S}_{\overline{\mathsf{b}}}$. These queries are $(\mathsf{val}^i \oplus \mathsf{x}_{\overline{\mathsf{b}}}^i)$ for each $i \in [n]$. Equivalently, these queries are $(c^i, c^i \oplus x_0^i \oplus x_1^i)$. We note that c^i is chosen to be an independent random variable via coin flipping (and therefore, $(c^i \oplus x_0^i \oplus x_1^i)$ is also an independent random variable). Furthermore, because of the cut-and-choose test, $\mathcal{R}$ must use this correctly generated c^i in most parallel executions (i.e., for most indices i). Therefore even though val^i is known to (and can be selected by) $\mathcal{Z}_{\mathcal{R}}$, the distribution of the evaluation queries is uniformly random even from the point of view of $\widehat{PUF}_s$.

1. **Coin Flipping:** For each $i \in [n]$, $\mathsf{Sim}_{\mathcal{R}}$ obtains d^i from $\mathcal{R}$, extract $\mathsf{c}_1^i = \mathsf{UC\text{-}Com.Extract}(\mathsf{d}^i)$. It picks and sends $\mathsf{c}_2^i \xleftarrow{\$} \{0,1\}^n$ to $\mathcal{Z}_{\mathcal{R}}$.

2. **Sender Message:** $\mathsf{Sim}_{\mathcal{R}}$ follows honest sender strategy to do the following.

   - Generate a PUF $\mathsf{PUF}_s : \{0,1\}^n \to \{0,1\}^n$.
   - **Test Queries :** For each $i \in [n]$, choose $\mathsf{TQ}_i \xleftarrow{\$} \{0,1\}^n$ and compute $\mathsf{TR}_i = \mathsf{PUF}_s(\mathsf{TQ}_i)$. Store the pair $(\mathsf{TQ}_i, \mathsf{TR}_i)$.
   - For $i \in [n]$, choose a pair of random strings $(\mathsf{x}_0^i, \mathsf{x}_1^i) \xleftarrow{\$} \{0,1\}^{2n}$. Compute $(\mathsf{t}_0^i, \mathsf{t}_1^i) = \mathsf{UC\text{-}Com.Commit}(\mathsf{x}_0^i, \mathsf{x}_1^i)$. Send $(\mathsf{t}_0^i, \mathsf{t}_1^i)$ and PUF_s to $\mathcal{Z}_{\mathcal{R}}$.

3. **Receiver Message:** Obtain $\widehat{PUF_s}$ and $(\hat{\mathsf{x}}_0^i, \hat{\mathsf{x}}_0^i)$ for $i \in [n]$ from $\mathcal{Z}_{\mathcal{R}}$.

4. **Sender Message:** $\mathsf{Sim}_{\mathcal{R}}$ follows honest sender strategy to do the following.

   - **Verification:** For each $i \in [n]$, if $\mathsf{TR}_i \neq \widehat{\mathsf{PUF}}_s(\mathsf{TQ}_i)$, abort.
   - For each $i \in [n]$, send $(\mathsf{x}_0^i, \mathsf{x}_1^i) = \mathsf{UC\text{-}Com.Decommit}(\mathsf{t}_0^i, \mathsf{t}_1^i)$ to $\mathcal{Z}_{\mathcal{R}}$.

5. **Receiver Message:** Obtain $(\mathsf{val}^1, \mathsf{val}^2, \ldots, \mathsf{val}^n)$ from $\mathcal{Z}_{\mathcal{R}}$.

6. **Cut-and-choose:** $\mathsf{Sim}_{\mathcal{R}}$ follows honest sender strategy to do the following.

   - $\mathsf{Sim}_{\mathcal{R}}$ picks $r_{\mathcal{S}} \xleftarrow{\$} \{0,1\}^{2K}$ and sends $t_{\mathcal{S}} = \mathsf{UC\text{-}Com.Commit}(r_{\mathcal{S}})$ to $\mathcal{Z}_{\mathcal{R}}$. $\mathsf{Sim}_{\mathcal{R}}$ obtains $r_{\mathcal{R}}$ from $\mathcal{Z}_{\mathcal{R}}$. $\mathsf{Sim}_{\mathcal{R}}$ sends $r_{\mathcal{S}} = \mathsf{UC\text{-}Com.Decommit}(t_S)$ to $\mathcal{Z}_{\mathcal{R}}$, and $(\mathsf{Sim}_{\mathcal{R}}, \mathcal{Z}_{\mathcal{R}})$ use $(r_{\mathcal{S}} \oplus r_{\mathcal{R}})$ to pick a subset I of indices of size $\frac{K}{2}$.
   - For all $i \in [I]$, obtain c_1^i from $\mathcal{Z}_{\mathcal{R}}$ and check that the decommitment is correct.
   - $\mathsf{Sim}_{\mathcal{R}}$ uses honest strategy to compute $\mathsf{c}^i = \mathsf{c}_1^i \oplus \mathsf{c}_2^i$, and check if either $\mathsf{val}^i = \mathsf{c}^i \oplus \mathsf{x}_0^i \oplus \hat{\mathsf{x}}_0^i$ or $\mathsf{val}^i = \mathsf{c}^i \oplus \mathsf{x}_1^i \oplus \hat{\mathsf{x}}_1^i$. If not, $\mathsf{Sim}_{\mathcal{R}}$ aborts.

7. **Receiver Message:** For each $i \in [n] \setminus I$, obtain bc_i from $\mathcal{Z}_{\mathcal{R}}$.

8. **Sender Message:** $\mathsf{Sim}_{\mathcal{R}}$ does the following.

   - For each $i \in [n]$, recall that $\mathsf{Sim}_{\mathcal{R}}$ already extracted c_1^i, and can compute $\mathsf{c}^i = (\mathsf{c}_1^i \oplus \mathsf{c}_2^i)$. Then $\mathsf{Sim}_{\mathcal{R}}$ computes $\overline{\mathsf{x}}^i = (\mathsf{c}^i \oplus \mathsf{val}^i)$, and if $\overline{\mathsf{x}}^i = \mathsf{x}_0^i$, sets $\mathsf{b}^i = 0$ else sets $\mathsf{b}^i = 1$. Set $b = 0$ if $\sum_{i \in [n] \setminus I} (\mathsf{bc}_i \oplus \mathsf{b}_i) \leq \frac{n}{2}$ else set $b = 1$.
   - Send b to $\mathcal{F}_{\mathsf{ot}}$ and obtain m_b. Set $m_{1-b} \leftarrow \{0,1\}^K$. Compute $\mathsf{bc}^i = \mathsf{b}^i \oplus \mathsf{b}$.
   - Compute $\mathsf{S}_0 = \mathsf{m}_0 \bigoplus_{i \in n \setminus I} \mathsf{PUF}_s(\mathsf{val}^i \oplus \mathsf{x}_{\mathsf{bc}_i}^i \oplus \hat{\mathsf{x}}_{\mathsf{bc}_i}^i)$, $\mathsf{S}_1 = \mathsf{m}_1 \bigoplus_{i \in n \setminus I} \mathsf{PUF}_s(\mathsf{val}^i \oplus \mathsf{x}_{1-\mathsf{bc}_i}^i \oplus \hat{\mathsf{x}}_{1-\mathsf{bc}_i}^i)$. Send $(\mathsf{S}_0, \mathsf{S}_1)$ to $\mathcal{Z}_{\mathcal{R}}$.

Figure 19: Receiver Simulation for malicious stateless PUFs with encapsulation.

$\mathsf{Sim}_{\mathcal{R}}$ makes n test queries in the verification phase in step 3, each of which is chosen independently of all other variables, and uniformly at random. These queries appear statistically indistinguishable from random to $\mathcal{Z}_{\mathcal{R}}$, since by the unclonability property, a (malicious) party cannot predict previous parties' queries to a PUF that it did not create. Conditioned on the protocol not aborting, the output of PUF_r must be identical to the output of PUF_s on all n test queries. We will show that conditioned on the receiver not aborting after the cut-and-choose in the real execution, the simulator is able to extract the correct bit b.

First of all, conditioned on not aborting, with overwhelming probability, in all but $\sqrt{n}$ of the indices in $[n] \setminus I$, the receiver did send $\mathsf{val}^i = \mathsf{c}^i \oplus \mathsf{x}_0^i \oplus \hat{\mathsf{x}}_0^i$ or $\mathsf{val}^i = \mathsf{c}^i \oplus \mathsf{x}_1^i \oplus \hat{\mathsf{x}}_1^i$. In all these instances, the simulator is able to extract a bit b. We now show that the output of $\widehat{PUF}_s$ on $\mathsf{val}^i \oplus (\mathsf{x}_0^i \oplus \hat{\mathsf{x}}_0^i) \oplus (\mathsf{x}_1^i \oplus \hat{\mathsf{x}}_1^i)$ is statistically unpredictable to the receiver in all but $\sqrt{n}$ of these remaining instances.

Now, because of statistical hiding of the sender's commitment to $(\mathsf{x}_0^i, \mathsf{x}_1^i)$, the encapsulated PUF $\widehat{PUF}_s$ is independent of $\mathsf{val}^i \oplus (\mathsf{x}_0^i \oplus \hat{\mathsf{x}}_0^i) \oplus (\mathsf{x}_1^i \oplus \hat{\mathsf{x}}_1^i)$. Therefore, $\mathsf{val}^i \oplus (\mathsf{x}_0^i \oplus \hat{\mathsf{x}}_0^i) \oplus (\mathsf{x}_1^i \oplus \hat{\mathsf{x}}_1^i)$ is identically distributed as a test query and either the sender will thus abort with overwhelming probability, or the receiver is only able to learn the output of $\widehat{PUF}_s$ on $\mathsf{val}^i \oplus (\mathsf{x}_0^i \oplus \hat{\mathsf{x}}_0^i) \oplus (\mathsf{x}_1^i \oplus \hat{\mathsf{x}}_1^i)$ in at most $\sqrt{n}$ executions.

Then the bit b, which is extracted as the maximum of all possible values of b over various indices i, can be wrongly extracted from at most $O(\sqrt{n})$ indices, with overwhelming probability. Therefore, b is extracted correctly by the simulator with overwhelming probability. At this point, since the simulator uses honest sender strategy to create sets (S_0, S_1) the view of the receiver remains statistically close. □

K UC Commitments with Encapsulated Malicious PUFs: Full Proofs

Recall that Damgård and Scafuro [11] construct UC commitments using stateless PUFs, secure in the malicious stateful PUF model. Their scheme becomes insecure when parties are allowed to create malicious encapsulated PUFs. In this section, we construct UC commitments using stateless PUFs, that are secure in an incomparable model, where an adversary can encapsulate honest PUFs (see Section 2.1), to create malicious *stateless encapsulated* PUFs. Note that the protocol does not require any party to have the ability to encapsulate PUFs, but is secure against parties that do have this ability.

We begin by observing that it suffices to construct an extractable commitment scheme that is secure against encapsulation. Given such a scheme, Damgård and Scafuro [11] show that it is possible to compile the extractable commitment scheme using an additional ideal commitment scheme, to obtain a UC commitment scheme that is secure in the malicious stateless PUF model. Since the compiler in [11] doesn't require any additional PUFs at all, if the extractable commitment and the ideal commitment are secure against encapsulation attacks, then so is the resulting UC commitment. We also observe that the unconditional ideal commitment scheme in [35] is already secure against malicious unbounded stateful PUFs that may be encapsulating honest PUFs. We describe this ideal commitment scheme next.

Ideal Commitments. Let n denote the security parameter. We describe an ideal bit commitment scheme $\mathsf{IdealCom} = (\mathsf{IdealCom.Commit}, \mathsf{IdealCom.Decommit})$ from [35] in Figure 20, that is statistically hiding and binding even in the malicious fully stateful PUF model.

Inputs: Committer $\mathcal{C}$ has private input $\mathsf{b} \in \{0,1\}$ and Receiver $\mathcal{R}$ has no input.

Commitment Phase

1. **Committer Message:** $\mathcal{C}$ does the following:
   - Generate a PUF $\mathsf{PUF}_s : \{0,1\}^n \to \{0,1\}^{3n}$.
   - Choose a random string $\mathsf{s} \xleftarrow{\$} \{0,1\}^n$.
   - Compute $\sigma_s = \mathsf{PUF}_s(\mathsf{s})$.
   - Send PUF_s to $\mathcal{R}$.
2. **Receiver Message:** $\mathcal{R}$ picks and sends a random string $\mathsf{r} \xleftarrow{\$} \{0,1\}^{3n}$ to $\mathcal{C}$.
3. **Committer Message:** $\mathcal{C}$ does the following:
   - If $\mathsf{b} = 0$, set $\mathsf{c} = \sigma_s$. Else if $\mathsf{b} = 1$, set $\mathsf{c} = (\sigma_s \oplus \mathsf{r})$.
   - Send c to $\mathcal{R}$.

Decommitment Phase

1. **Committer Message:** $\mathcal{C}$ sends (s, b) to $\mathcal{R}$.
2. **Receiver** $\mathcal{R}$ does the following:
   - If $\mathsf{b} = 0$ and $\mathsf{c} = \mathsf{PUF}_s(\mathsf{s})$, output 0.
   - Else if $\mathsf{b} = 1$ and $\mathsf{c} = \mathsf{PUF}_s(\mathsf{s}) \oplus \mathsf{r}$, output 1.
   - Else, output $\perp$.

Figure 20: Protocol for Ideal Commitment in the malicious stateless PUF model with encapsulation.

Claim 13. *The protocol in Figure 20 is an ideal bit commitment scheme in the encapsulated malicious stateless PUF model.*

In this scheme, only a single PUF is used that is created by the committer and sent to the receiver. Since the receiver never needs to send any PUF to the committer, creating an encapsulated PUF does not help the receiver in the protocol. Moreover, since the committer already itself creates the (possibly malicious) PUF it sends to the receiver, the committer binding argument holds even if $\mathcal{C}$ creates an encapsulated PUF.

Thus, this scheme is secure against PUF encapsulation attacks.

Now, we prove correctness and security of the extractable commitment scheme described in Figure 6.

Completeness: It can be observed from the protocol description that for any $\mathsf{b} \in \{0,1\}$, if $\mathcal{C}$ and $\mathcal{R}$ are honest, the receiver $\mathcal{R}$ accepts the decommitment with probability 1.

Statistical Binding: Consider a malicious committer $\mathcal{C}^*$. We will show that the probability of $\mathcal{C}^*$ breaking the binding property of the commitment is negligible. In order to break binding, after committing to a bit b, $\mathcal{C}^*$ must provide a decommitment to $\overline{\mathsf{b}}$ such that $\mathcal{R}$ accepts it. In order to provide a valid decommitment for $\overline{\mathsf{b}} = 1$, for all $i \in [n]$, $\mathcal{C}^*$ must find $\mathsf{s}_i, \mathsf{s}'_i \in \{0,1\}^n$ such that $\mathsf{PUF}_s(\mathsf{s}_i) = \mathsf{PUF}_s(\mathsf{s}'_i) \oplus \mathsf{r}_i$, where $\mathsf{r}_i \xleftarrow{\$} \{0,1\}^{3n}$ is chosen randomly by the receiver. That is, $\mathsf{PUF}_s(\mathsf{s}_i) \oplus \mathsf{PUF}_s(\mathsf{s}'_i) = \mathsf{r}_i$. The total number of possible pairs of strings $(\mathsf{s}_i, \mathsf{s}'_i)$ is 2^{2n}. So, the number of possible values for $\mathsf{PUF}_s(\mathsf{s}_i) \oplus \mathsf{PUF}_s(\mathsf{s}'_i)$ is 2^{2n}. However, the number of possible choices for the random string r_i is 2^{3n}. Therefore, for each index i, the probability that $\mathcal{C}^*$ can cheat is at most 2^{-n}, and therefore the probability that the committer violates binding, is negligible.

Statistical Hiding: We want to argue that the view of any malicious receiver $\mathcal{R}^*$ (that may possibly encapsulate PUFs), the view of $\mathcal{R}^*$ at the end of the commitment phase is statistically close when the honest committer $\mathcal{C}$ commits to 0 versus 1. The view of the receiver is $\{\mathsf{PUF}_s, \mathsf{PUF}_r, \mathsf{c}_1, \ldots, \mathsf{c}_n, \mathsf{t}_1, \ldots, \mathsf{t}_n\}$.

In both cases, $\mathcal{C}$ generates PUF_s honestly, and therefore PUF_s is identically distributed. $\mathcal{C}$ returns the same PUF_r that he got from $\mathcal{R}^*$ in step 1 and since PUFs are stateless, PUF_r is identically distributed in both cases even given PUF_s.

Since s_i is chosen uniformly at random by $\mathcal{C}$ for all i, $\mathsf{PUF}_s(\mathsf{s}_i)$ and $(\mathsf{PUF}_s(\mathsf{s}_i) \oplus \mathsf{r}_i)$ are identically distributed irrespective of $\mathcal{R}^*$'s choice of r_i. Hence, c_i is identically distributed in both cases for all $i \in [n]$ even given the previous view.

By the hiding property of $\mathsf{IdealCom}$, $\mathsf{t}_i = \mathsf{IdealCom.Commit}(\mathsf{x}_i)$ is statistically indistinguishable in both cases for all $i \in [n]$ even given the previous view.

Extraction: Consider a malicious committer $\mathcal{C}^*$ with input some bit $\mathsf{b} \in \{0,1\}$. The strategy for the straight line polynomial time extractor E against a malicious committer is described in Figure 21. Since E follows the honest receiver's strategy, the view of the malicious committer $\mathcal{C}^*$ when interacting with E is identical to the view of $\mathcal{C}^*$ when interacting with the honest receiver.

First, we prove that if E outputs $\perp$, then the probability that $\mathcal{C}^*$ will provide an accepting decommitment is negligible. Let $\mathcal{Q} = \{\mathsf{Q}_1, \ldots, \mathsf{Q}_{\mathsf{poly}(n)}\}$ be the set of queries made by $\mathcal{C}^*$ to PUF_R.

Suppose $\mathcal{C}^*$ provides an accepting decommitment of $\mathsf{b} = 0$. Since the decommitment accepts, we know that for all $i \in [n]$, $\mathsf{x}_i = \mathsf{y}_i$. However, since y_i is the output of an honest PUF generated by E and y_i is already known to E, by the unpredictability of PUFs, except with negligible probability, unless $\mathcal{C}^*$ queries PUF_R, y_i is distributed uniformly at random in the view of $\mathcal{C}^*$. Observe that encapsulating or changing PUF_R doesn't help since PUF_R is never sent back to E. Now, since E outputs $\perp$, there does not exist $(q_1, \ldots, q_n) \in \mathcal{Q}$ such that for all $i \in [n]$, $\mathsf{PUF}_R(q_i) = \mathsf{y}_i$. Therefore, there exists at least $i \in [n]$ such that $\mathsf{x}_i \neq \mathsf{y}_i$ except with negligible probability which is a contradiction.

Suppose $\mathcal{C}^*$ provides an accepting decommitment of $\mathsf{b} = 1$. Since the decommitment accepts, we know that for all $i \in [n]$, $\mathsf{x}_i = \mathsf{z}_i$. However, since z_i is the output of an honest PUF generated by E and z_i is already known to E, by the unpredictability of PUFs, except with negligible probability, unless $\mathcal{C}^*$ queries PUF_R, z_i is distributed uniformly at random in the view of $\mathcal{C}^*$. Observe that encapsulating or changing PUF_R doesn't help since PUF_R is never sent back to E. Now, since E outputs $\perp$, there does not exist $(q_1, \ldots, q_n) \in \mathcal{Q}$ such that for all $i \in [n]$, $\mathsf{PUF}_R(q_i) = \mathsf{z}_i$. Therefore, there exists at least $i \in [n]$ such that $\mathsf{x}_i \neq \mathsf{z}_i$ except with negligible probability which is a contradiction.

Next, we prove that if E outputs $\mathsf{b}^* \neq \perp$, then the probability that $\mathcal{C}^*$ decommits to a bit $\mathsf{b} \neq \mathsf{b}^*$ is negligible. First, observe that encapsulating or changing PUF_R doesn't help since PUF_R is never sent back to E.

- Case 1: $\mathsf{b}^* = 0$. Suppose $\mathcal{C}^*$ decommits to $\mathsf{b} = 1$. Since the decommitment is accepting, for all $i \in [n]$, $\mathsf{IdealCom.Decommit}(\mathsf{x}_i)$ verifies correctly, $\mathsf{c}_i = (\mathsf{PUF}_s(\mathsf{s}_i) \oplus \mathsf{r}_i)$ and $\mathsf{x}_i = \mathsf{z}_i$. Now, we know that for all $i \in [n]$, $\mathsf{z}_i = \mathsf{PUF}_R(\widehat{\mathsf{PUF}}_r(\mathsf{c}_i \oplus \mathsf{r}_i))$. Therefore, there exists $(\overline{\mathsf{q}}_1, \ldots, \overline{\mathsf{q}}_n) \in \mathcal{Q}$ such that $\overline{\mathsf{q}}_i = \widehat{\mathsf{PUF}}_r(\mathsf{c}_i \oplus \mathsf{r}_i) = \widehat{\mathsf{PUF}}_r(\mathsf{PUF}_s(\mathsf{s}_i))$. Also, since $\mathsf{b}^* = 0$, there exists $(q_1, \ldots, q_n) \in \mathcal{Q}$ such that for all $i \in [n]$, $\mathsf{PUF}_R(q_i) = \mathsf{y}_i$. We know that $\mathsf{z}_i = \mathsf{PUF}_R(\widehat{\mathsf{PUF}}_r(\mathsf{c}_i))$. Therefore, $q_i = \widehat{\mathsf{PUF}}_r(\mathsf{c}_i) = \widehat{\mathsf{PUF}}_r(\mathsf{PUF}_s(\mathsf{s}_i) \oplus \mathsf{r}_i)$.
- Case 2: $\mathsf{b}^* = 1$. Suppose $\mathcal{C}^*$ decommits to $\mathsf{b} = 0$. Since the decommitment is accepting, for all $i \in [n]$, $\mathsf{IdealCom.Decommit}(\mathsf{x}_i)$ verifies correctly, $\mathsf{c}_i = \mathsf{PUF}_s(\mathsf{s}_i)$ and $\mathsf{x}_i = \mathsf{y}_i$. Now, we know that for all $i \in [n]$, $\mathsf{y}_i = \mathsf{PUF}_R(\widehat{\mathsf{PUF}}_r(\mathsf{c}_i))$. Therefore, there exists $(\overline{\mathsf{q}}_1, \ldots, \overline{\mathsf{q}}_n) \in \mathcal{Q}$ such that $\overline{\mathsf{q}}_i = \widehat{\mathsf{PUF}}_r(\mathsf{c}_i) = \widehat{\mathsf{PUF}}_r(\mathsf{PUF}_s(\mathsf{s}_i))$. Also, since $\mathsf{b}^* = 1$, there exists $(q_1, \ldots, q_n) \in \mathcal{Q}$ such that for all $i \in [n]$, $\mathsf{PUF}_R(q_i) = \mathsf{z}_i$. We know that $\mathsf{z}_i = \mathsf{PUF}_R(\widehat{\mathsf{PUF}}_r(\mathsf{c}_i \oplus \mathsf{r}_i))$. Therefore, $q_i = \widehat{\mathsf{PUF}}_r(\mathsf{c}_i \oplus \mathsf{r}_i) = \widehat{\mathsf{PUF}}_r(\mathsf{PUF}_s(\mathsf{s}_i) \oplus \mathsf{r}_i)$.

In both cases, there exists $(q_1, \ldots, q_n) \in \mathcal{Q}$ such that $q_i = \widehat{\mathsf{PUF}}_r(\mathsf{PUF}_s(\mathsf{s}_i) \oplus \mathsf{r}_i)$. Since r_i is chosen uniformly at random by E after $\widehat{\mathsf{PUF}}_r$ was sent by $\mathcal{C}^*$, $\mathcal{C}^*$ could not have queried $\widehat{\mathsf{PUF}}_r$ on $(\mathsf{PUF}_s(\mathsf{s}_i) \oplus \mathsf{r}_i)$ before sending $\widehat{\mathsf{PUF}}_r$. So, the only way $\mathcal{C}^*$ might be able to know the output of $\widehat{\mathsf{PUF}}_r$ on $(\mathsf{PUF}_s(\mathsf{s}_i) \oplus \mathsf{r}_i)$ is if $\mathcal{C}^*$ sends an encapsulated PUF $\widehat{\mathsf{PUF}}_r$ in step 2 that encapsulates PUF_r. Note that E makes n test queries in the verification phase in step 3, each of which is chosen independently of all other variables, and uniformly at random. These queries are not known to $\mathcal{C}^*$ since we assume that a (malicious) party cannot predict previous parties' queries to a PUF that it did not create. Conditioned on the protocol not aborting, the output of $\widehat{\mathsf{PUF}}_r$ must be identical to the output of PUF_r on all n test queries.

Also, since r_i is chosen uniformly at random by E after $\widehat{\mathsf{PUF}}_r$ was sent by $\mathcal{C}^*$, the queries $(\mathsf{PUF}_s(\mathsf{s}_i) \oplus \mathsf{r}_i)$ for $i \in [n]$ are distributed uniformly at random, and identically as the test queries, from the point of view of $\widehat{\mathsf{PUF}}_r$. Given a set of $2n$ queries each of which appear to be uniformly random to $\widehat{\mathsf{PUF}}_r$, the probability that it correctly picks the answers to the n test queries to all be equal to the output of PUF_r, and the answers to the n evaluation queries to all be different, is $\frac{1}{\binom{2n}{n}} \leq \frac{1}{2^n}$. Therefore, the probability that $\mathcal{C}^*$ decommits to $\mathsf{b} \neq \mathsf{b}^*$ is $\frac{1}{\binom{2n}{n}} \leq \frac{1}{2^n}$ which is negligible.

UC-Secure Commitments from Extractable Commitments. We import the following theorem from [11], which allows us to obtain UC-secure commitments in the malicious encapsulated stateless PUF model, based on the ideal commitment and the extractable commitment described above.

1. **Receiver Message:** E does the following:
   - Generate a PUF $\mathsf{PUF}_r : \{0,1\}^{3n} \to \{0,1\}^{3n}$.
   - **Test Queries :** For each $i \in [n]$,
     - Choose $\mathsf{TQ}_i \xleftarrow{\$} \{0,1\}^{3n}$.
     - Compute $\mathsf{TR}_i = \mathsf{PUF}_r(\mathsf{TQ}_i)$.
     - Store the pair $(\mathsf{TQ}_i, \mathsf{TR}_i)$.
   - Send PUF_r to $\mathcal{C}^*$.
2. **Committer Message:** $\mathcal{C}^*$ sends $\mathsf{PUF}_s, \widehat{\mathsf{PUF}}_r$ to E.
3. **Receiver Message:** E does the following:
   - **Verification :** For each $i \in [n]$, if $\mathsf{TR}_i \neq \widehat{\mathsf{PUF}}_r(\mathsf{TQ}_i)$, abort.
   - For each $i \in [n]$, choose a random string $\mathsf{r}_i \xleftarrow{\$} \{0,1\}^{3n}$ and send r_i to $\mathcal{C}^*$.
4. **Committer Message:** $\mathcal{C}^*$ sends $(\mathsf{c}_1, \ldots, \mathsf{c}_n)$ to E.
5. **Receiver Message:** E does the following:
   - Generate a PUF $\mathsf{PUF}_R : \{0,1\}^{3n} \to \{0,1\}^{3n}$.
   - For each $i \in [n]$, compute $\mathsf{y}_i = \mathsf{PUF}_R(\widehat{\mathsf{PUF}}_r(\mathsf{c}_i)), \mathsf{z}_i = \mathsf{PUF}_R(\widehat{\mathsf{PUF}}_r(\mathsf{c}_i \oplus \mathsf{r}_i))$.
   - Send PUF_R to $\mathcal{C}^*$.
6. **Committer Message:** $\mathcal{C}^*$ sends $(\mathsf{t}_1, \ldots, \mathsf{t}_n)$ to E.
7. **Extraction:** E does the following:
   - Let $\mathcal{Q} = \{\mathsf{Q}_1, \ldots, \mathsf{Q}_{\mathsf{poly}(n)}\}$ be the set of queries made by $\mathcal{C}^*$ to PUF_R.
   - If there exists $(q_1, \ldots, q_n) \in \mathcal{Q}$ such that $\mathsf{PUF}_R(q_i) = \mathsf{y}_i$ for all $i \in [n]$, output $b^* = 0$.
   - Else, if there exists $(q_1, \ldots, q_n) \in \mathcal{Q}$ such that $\mathsf{PUF}_R(q_i) = \mathsf{z}_i$ for all $i \in [n]$, output $b^* = 1$.
   - Else output $\perp$.

Figure 21: Extraction Strategy E for $\mathsf{UC\text{-}Com}$.

Imported Theorem 3. *[11] If* UC-Com *is an ideal extractable commitment scheme in the* $\mathcal{F}_{\mathsf{aux}}$*-hybrid model, and* IdealCom *is an ideal commitment scheme in the* $\mathcal{F}_{\mathsf{aux}}$*-hybrid model, there exists an unconditional UC-secure commitment scheme in the* $\mathcal{F}_{\mathsf{aux}}$*-hybrid model, where* $\mathcal{F}_{\mathsf{aux}}$ *is an auxiliary setup functionality accessed by the real world parties and the simulator or extractor.*

L Bounded Stateful PUFs with a Non-Rewinding Simulator

In this section, we modify the protocol from Section 4 so that the simulator does not have to rewind the stateful PUF. In order to achieve this, we will use an (n, k) threshold secret sharing scheme, and we now define that below before describing the protocol.

Definition 13 ((n, k) **Threshold Secret Sharing**). *An* (n, k) *threshold secret sharing scheme consists of two algorithms* (Share, Recon)*. The algorithm* Share *takes as input a secret message* m *and outputs a set of* n *shares. The algorithm* Recon *takes as input a set of shares and outputs the secret message* m *if the size of the set is at least* k*. The security guarantee is that the secret message* m *is impossible to recover from a set of shares with size less than* k*.*

Let (Share, Recon) be a $(K/2, K/2)$ threshold secret sharing scheme. The protocol Π_K in Figure 22 allows us to use an ℓ-bounded stateful PUF to obtain K secure (but leaky) oblivious transfers, such that a malicious sender can obtain at most ℓ bits of additional universal leakage on the joint distribution of the receiver's choice input bits $(b_1, b_2, \ldots b_K)$. This scheme removes the additional assumption on the simulator, in particular, the simulator is no longer required to have the ability to rewind a (malicious, stateful) PUF. This is done by combining standard secret sharing with cut-and-choose techniques. For clarity, we omit the cut-and-choose from the description of the protocol in Figure 22, and only prove the security of the protocol against any fully malicious sender, and a special-malicious receiver. The special-malicious receiver is allowed to behave maliciously everywhere except in Step 4 of the experiment, where the receiver is required to be semi-honest. It is easy to see that this semi-honest behavior can be enforced via a standard cut-and-choose on the views of the receiver.

Theorem 5. *The protocol* Π_K *unconditionally UC-securely realizes* $\mathcal{F}_{\mathsf{ot}}^{[\otimes K]}$ *in an* ℓ*-bounded-stateful PUF model, except that a malicious sender can obtain at most* ℓ *bits of additional universal leakage on joint distribution of the receiver's choice bits over all* $\mathcal{F}_{\mathsf{ot}}^{[\otimes K]}$*.*

Correctness.

Claim 14. *For all* $(\mathsf{m}_0, \mathsf{m}_1) \in \{0,1\}^{2n}$ *and* $\mathsf{b} \in \{0,1\}$*, the output of* $\mathcal{R}$ *equals* m_b*.*

Proof. $\mathcal{R}$ outputs $(\mathsf{S}_\mathsf{b} \oplus \mathsf{Recon}(\mathsf{r}^1, \ldots, \mathsf{r}^{K/2}))$, where: $\mathsf{S}_\mathsf{b} = \mathsf{m}_\mathsf{b} \oplus \mathsf{Recon}(\mathsf{PUF}_s(\mathsf{val}^1 \oplus \mathsf{x}_\mathsf{b}^1 \oplus \hat{\mathsf{x}}_\mathsf{b}^1), \ldots, \mathsf{PUF}_s(\mathsf{val}^{K/2} \oplus \mathsf{x}_\mathsf{b}^{K/2} \oplus \hat{\mathsf{x}}_\mathsf{b}^{K/2})) = \mathsf{m}_\mathsf{b} \oplus \mathsf{Recon}(\mathsf{r}^1, \ldots, \mathsf{r}^{K/2})$. Thus, $\mathcal{R}$ outputs m_b. □

Receiver Security. Let the environment be denoted by $\mathcal{Z}_\mathcal{S}$. The environment chooses a bit $b \in \{0,1\}$ and sends it to the honest receiver as his input. The strategy for the simulator $\mathsf{Sim}_\mathcal{S}$ against a malicious sender is described in Figure 23.

Claim 15. *The sender simulation is statistically secure, for the strategy in Figure 23.*

Inputs: Sender $\mathcal{S}$ has private inputs $(\mathsf{m}_0, \mathsf{m}_1) \in \{0,1\}^{2n}$ and $\mathcal{R}$ has private input $\mathsf{b} \in \{0,1\}$.

1. **Sender Message:** $\mathcal{S}$ does the following.
   - Generate a PUF $\mathsf{PUF}_s : \{0,1\}^n \to \{0,1\}^n$.
   - For each $i \in [K]$:
     - Choose a pair of random strings $(\mathsf{x}_0^i, \mathsf{x}_1^i) \xleftarrow{\$} \{0,1\}^{2n}$.
     - Send $(\mathsf{t}_0^i, \mathsf{t}_1^i) = \mathsf{UC\text{-}Com.Commit}(\mathsf{x}_0^i, \mathsf{x}_1^i)$ to $\mathcal{R}$.
   - Send PUF_s to $\mathcal{R}$.
2. **Receiver Message:**
   - For each $i \in [K]$, $\mathcal{R}$ does the following: Choose a random string $(\mathsf{c}^i) \xleftarrow{\$} \{0,1\}^n$ and compute $\mathsf{r}^i = \mathsf{PUF}_s(\mathsf{c}^i)$. Store the pair $(\mathsf{c}^i, \mathsf{r}^i)$. Pick and send $(\hat{\mathsf{x}}_0^i, \hat{\mathsf{x}}_1^i) \xleftarrow{\$} \{0,1\}^{2n}$.
   - Send PUF_s to $\mathcal{S}$.
3. **Sender Message:**
   For each $i \in [K]$, $\mathcal{S}$ sends $(\mathsf{x}_0^i, \mathsf{x}_1^i) = \mathsf{UC\text{-}Com.Decommit}(\mathsf{t}_0^i, \mathsf{t}_1^i)$ to $\mathcal{R}$.
4. **Receiver Message:** For each $i \in [K]$, $\mathcal{R}$ does the following :
   - If $\mathsf{UC\text{-}Com.Decommit}(\mathsf{t}_0, \mathsf{t}_1)$ does not verify, abort.
   - Else, compute and send $\mathsf{val}^i = \mathsf{c}^i \oplus \mathsf{x}_{\mathsf{b}}^i \oplus \hat{\mathsf{x}}_{\mathsf{b}}^i$ to $\mathcal{S}$.
5. **Sender Message:** $\mathcal{S}$ does the following:
   - Consider the values $\{\mathsf{PUF}_s(\mathsf{val}^i \oplus \mathsf{x}_0^i \oplus \hat{\mathsf{x}}_0^i)\}_{i=1}^{K/2}$ as a set of shares and compute $s_0 = \mathsf{Recon}(\mathsf{PUF}_s(\mathsf{val}^1 \oplus \mathsf{x}_0^1 \oplus \hat{\mathsf{x}}_0^1), \ldots, \mathsf{PUF}_s(\mathsf{val}^{K/2} \oplus \mathsf{x}_0^{K/2} \oplus \hat{\mathsf{x}}_0^{K/2}))$.
   - Compute $s_1 = \mathsf{Recon}(\mathsf{PUF}_s(\mathsf{val}^1 \oplus \mathsf{x}_1^1 \oplus \hat{\mathsf{x}}_1^1), \ldots, \mathsf{PUF}_s(\mathsf{val}^{K/2} \oplus \mathsf{x}_1^{K/2} \oplus \hat{\mathsf{x}}_1^{K/2}))$.
   - Compute $\mathsf{S}_0 = (\mathsf{m}_0 \oplus s_0)$, $\mathsf{S}_1 = (\mathsf{m}_1 \oplus s_1)$ and send $(\mathsf{S}_0, \mathsf{S}_1)$ to $\mathcal{R}$.

Outputs: $\mathcal{S}$ has no output. $\mathcal{R}$ outputs $\mathsf{m}_{\mathsf{b}} := (\mathsf{S}_{\mathsf{b}} \oplus \mathsf{Recon}(\mathsf{r}^1, \ldots, \mathsf{r}^{K/2}))$.

Figure 22: Protocol Π_K for K 2-choose-1 OTs (with at most ℓ leakage) in the malicious stateful PUF model.

1. **Sender Message:** Obtain PUF_s and $(\mathsf{t}_0^i, \mathsf{t}_1^i)$ for each $i \in [K]$ from $\mathcal{S}$.
2. **Receiver Message:** For each $i \in [K]$, $\mathsf{Sim}_{\mathcal{S}}$ does the following.
   - Obtain $(\mathsf{x}_0^i, \mathsf{x}_1^i)$ using $\mathsf{UC\text{-}Com.Extract}(t_0^i, t_1^i)$.
   - Pick $(\hat{\mathsf{x}}_0^i, \hat{\mathsf{x}}_1^i) \xleftarrow{\$} \{0,1\}^{2n}$ (following honest strategy).
   - Choose a random string $(\mathsf{c}^i) \xleftarrow{\$} \{0,1\}^n$.
   - If $i \leq K/2$, compute $\mathsf{r}^i = \mathsf{PUF}_s(\mathsf{c}^i)$. Else, compute $\mathsf{r}^i = \mathsf{PUF}_s(\mathsf{c}^i \oplus \mathsf{x}_0^i \oplus \hat{\mathsf{x}}_0^i \oplus \mathsf{x}_1^i \oplus \hat{\mathsf{x}}_1^i)$.
   - Store the pair $(\mathsf{c}^i, \mathsf{r}^i)$ and send $(\hat{\mathsf{x}}_0, \hat{\mathsf{x}}_1)$ to $\mathcal{S}$.

   Send PUF_s to $\mathcal{S}$.
3. **Sender Message:** For each $i \in [K]$, $\mathsf{Sim}_{\mathcal{S}}$ obtains $(\mathsf{x}_0^i, \mathsf{x}_1^i) = \mathsf{UC\text{-}Com.Decommit}(\mathsf{t}_0^i, \mathsf{t}_1^i)$ from $\mathcal{S}$.
4. **Receiver Message:** $\mathsf{Sim}_{\mathcal{S}}$ picks $\mathsf{b} \xleftarrow{\$} \{0,1\}$ and then, for each $i \in [K]$, $\mathsf{Sim}_{\mathcal{S}}$ does the following.
   - If $\mathsf{UC\text{-}Com.Decommit}(\mathsf{t}_0^i, \mathsf{t}_1^i)$ does not verify, abort.
   - Else, compute $\mathsf{val}^i = \mathsf{c}^i \oplus \mathsf{x}_{\mathsf{b}}^i \oplus \hat{\mathsf{x}}_{\mathsf{b}}^i$.
   - Send $(\mathsf{val})^i$ to $\mathcal{S}$.
5. **Simulator Message to $\mathcal{F}_{\mathsf{ot}}$:** Obtain $(\mathsf{S}_0, \mathsf{S}_1)$ from $\mathcal{S}$.
   - Compute $\mathsf{m}_{\mathsf{b}} = \mathsf{S}_{\mathsf{b}} \oplus \mathsf{Recon}(\mathsf{r}^1, \ldots, \mathsf{r}^{K/2})$.
   - Compute $\mathsf{m}_{1-\mathsf{b}} = \mathsf{S}_{1-\mathsf{b}} \oplus \mathsf{Recon}(\mathsf{r}^{K/2+1}, \ldots, \mathsf{r}^{K})$.
   - Send (m_0, m_1) to $\mathcal{F}_{\mathsf{ot}}$, such that $\mathcal{F}_{\mathsf{ot}}$ sends m_{b} to the honest receiver.

Figure 23: Sender simulation for K 2-choose-1 OTs (with at most ℓ leakage) in the malicious stateful PUF model.

Proof. First, observe that for each i, the pairs $(\hat{x}_0^i, \hat{x}_1^i)$ and the string c^i are chosen uniformly at random and independently of PUF_s. Therefore, the simulator's queries are independent of PUF_s and not correlated. Thus, in the view of a (possibly stateful) PUF_s, the distribution of simulator queries are statistically indistinguishable from the distribution of honest sender queries, and the probability of aborting in Step 2 (owing to a malicious aborting stateful PUF_s) is statistically close in both the worlds.

The simulator follows honest receiver strategy for all the next messages. By correctness of $\mathsf{UC\text{-}Com.Extract}(\cdot)$, correctness of the secret sharing scheme and because the distribution of PUF_s queries are statistically indistinguishable in the real and ideal worlds; it follows (using a similar argument as Section 3) that the simulator extracts (m_0, m_1) correctly, conditioned on not aborting in Step 4. This proves that the view of the malicious sender is close in the real and ideal worlds, except the receiver queries to PUF_s (which may be recorded by a malicious stateful PUF_s). However, since PUF_s can have at most ℓ bits of state, it obtains at most ℓ bits joint leakage over all K OTs. □

Sender Security. Let the environment be denoted by $\mathcal{Z}_{\mathcal{R}}$. The environment chooses a pair of messages $\{m_0, m_1\}$ and sends them to the honest sender as his input. We consider a special-malicious receiver, i.e., a receiver that is malicious in all steps except Step 4. The simulation strategy $\mathsf{Sim}_{\mathcal{R}}$ against a malicious receiver is described in Figure 24.

Claim 16. *Assuming that the receiver is semi-honest in step 4, the receiver simulation is statistically secure for the strategy in Figure 15.*

Proof. Following the same arguments as in Appendix F , we can show that the simulator extracts the correct input bit b in Step 5, such that the distribution of $(S_b, S_{\bar{b}})$ is at most $2^{-\frac{n}{2}}$-far in the real and ideal worlds. □

Given: Malicious stateful PUFs.

1. **Sender Message:** The simulator $\mathsf{Sim}_{\mathcal{R}}$ follows honest sender strategy to:
   - Generate a PUF $\mathsf{PUF}_s : \{0,1\}^n \to \{0,1\}^n$.
   - For each $i \in [K]$:
     - Choose a pair of random strings $(\mathsf{x}_0^i, \mathsf{x}_1^i) \xleftarrow{\$} \{0,1\}^{2n}$.
     - Compute $(\mathsf{t}_0^i, \mathsf{t}_1^i) = \mathsf{UC\text{-}Com.Commit}(\mathsf{x}_0^i, \mathsf{x}_1^i)$ and send $(\mathsf{t}_0^i, \mathsf{t}_1^i)$ to $\mathcal{Z}_{\mathcal{R}}$.
   - Send PUF_s to $\mathcal{Z}_{\mathcal{R}}$.
2. **Receiver Message:**
   For each $i \in [K]$, $\mathcal{Z}_{\mathcal{R}}$ sends $(\hat{\mathsf{x}}_0^i, \hat{\mathsf{x}}_1^i) \xleftarrow{\$} \{0,1\}^{2n}$ to $\mathsf{Sim}_{\mathcal{R}}$. $\mathcal{Z}_{\mathcal{R}}$ also sends PUF_s to $\mathsf{Sim}_{\mathcal{R}}$.
3. **Sender Message:** $\mathsf{Sim}_{\mathcal{R}}$ follows honest sender strategy to do the following: For each $i \in [K]$, send $(\mathsf{x}_0^i, \mathsf{x}_1^i) = \mathsf{UC\text{-}Com.Decommit}(\mathsf{t}_0^i, \mathsf{t}_1^i)$ to $\mathcal{Z}_{\mathcal{R}}$.
4. **Receiver Message:** For each $i \in [K]$, obtain val^i from $\mathcal{Z}_{\mathcal{R}}$.
5. **Simulator Message to Ideal OT:** $\mathsf{Sim}_{\mathcal{R}}$ does the following.
   - If there exists b' such that for each query to PUF_s by $\mathcal{Z}_{\mathcal{R}}$, val^i and $x^i \in \{(x_0^i \oplus \hat{x}_0^i), (x_1^i \oplus \hat{x}_1^i)\}, \mathsf{val}^i = c^i \oplus x_{b'}^i \oplus \hat{x}_{b'}^i$, set $b = b'$. Else set $b = 1$.
   - Send b to $\mathcal{F}_{\mathsf{ot}}$ and obtain output m_b. Pick $m_{\bar{b}} \xleftarrow{\$} \{0,1\}$.
6. **Sender Message:**
   - Compute: $s_1 = \mathsf{Recon}(\mathsf{PUF}_s(\mathsf{val}^1 \oplus \mathsf{x}_1^1 \oplus \hat{\mathsf{x}}_1^1), \ldots, \mathsf{PUF}_s(\mathsf{val}^K \oplus \mathsf{x}_1^K \oplus \hat{\mathsf{x}}_1^K))$.
   - Compute $\mathsf{S}_0 = (\mathsf{m}_0 \oplus s_0)$, $\mathsf{S}_1 = (\mathsf{m}_1 \oplus s_1)$ and send $(\mathsf{S}_0, \mathsf{S}_1)$ to $\mathcal{Z}_{\mathcal{R}}$.

Figure 24: Receiver Simulation for K OTs (with $\leq \ell$ leakage).