Round-Optimal Token-Based Secure Computation

Carmit Hazay* Antigoni Polychroniadou† Muthuramakrishnan Venkitasubramaniam‡

Abstract

Secure computation in the presence of tamper-proof hardware tokens is proven under the assumption that the holder of the token is only given black-box access to the functionality of the token. Starting with the work of Goldreich and Ostrovsky [GO96], a long series of works studied tamper-proof hardware for realizing two-party functionalities in a variety of settings.

In this work we focus our attention on two important complexity measures of token-based secure computation: *round complexity* and *hardness assumptions* and present the following results in the two-party setting:

- A round optimal generic secure protocol in the plain model assuming one-way functions, where the tokens are created by a single party.
- A round optimal generic UC secure protocol assuming one-way functions.

Our constructions only make black-box use of the underlying primitives and are proven in the real/ideal paradigm with security in the presence of static malicious adversaries.

Keywords: Secure Computation, Tamper-Proof Hardware, Round Complexity, Minimal Assumptions

*Bar-Ilan University, Israel. Email: `carmit.hazay@biu.ac.il`.
†Aarhus University, Denmark. Email: `antigoni@cs.au.dk`.
‡University of Rochester, Rochester, NY 14611, NY. Email: `muthuv@cs.rochester.edu`.

1 Introduction

A central goal in cryptography is to understand the complexity measures of interactive cryptographic protocols, where two such important measures are *round complexity* and *hardness assumptions*. Secure computation is a fundamental cryptographic research area for which understanding the round complexity is important from both theoretical and practical standpoints [Yao86, GMW87]. The seminal work of Katz and Ostrovsky [KO04] establishes that secure computation of most two-party functionalities that admit black-box proofs of security in the *plain model* requires at least five rounds. They further complemented their lower bound by constructing a five-round secure two-party computation protocol for any two-party functionality. Recently, it was demonstrated by Ostrovsky, Richelson and Scafuro [ORS15] how to strengthen this result by constructing a protocol that only makes *black-box* usage of the underlying cryptographic primitives.

While these results demonstrate upper and lower bounds in the plain model, a stronger form of security considers the concurrent setting where an unbounded number of protocols run concurrently in an arbitrary and adversarially controlled network environment. In the concurrent setting, *Universally Composable* or UC security, introduced by Canetti [Can01], provides the strongest guarantees. Unfortunately, stand-alone secure protocols typically fail to remain secure in the UC setting. In fact, without assuming some *trusted help*, UC security is impossible to achieve for most tasks [CF01, CKL06, Lin03]. Consequently, UC secure protocols have been constructed under various *trusted setup* assumptions in a long series of works; see [BCNP04, CDPW06, KLP07, CPS07, LPV09, DMRV13] for few examples.

In a series of works regarding *software obfuscation*, starting with the work of Goldreich and Ostrovsky [GO96], tamper-proof hardware tokens have been explored for realizing a variety of cryptographic tasks. In the context of constructing UC secure protocols, Katz in [Kat07] demonstrated the feasibility of achieving UC secure protocols for arbitrary functionalities assuming tamper-proof tokens. In his formulation, the parties can create a token that compute arbitrary functionalities such that any adversary that is given access to the token can *only* observe the input/output behavior of the token. In the UC framework, Katz described an ideal functionality $\mathcal{F}_{\mathrm{WRAP}}$ that captures this model. Note that tokens can either be stateful or stateless, depending on whether the tokens are allowed to maintain some state between invocations (where stateless tokens are easier to implement). In this paper, we focus on constructing protocols that only rely on stateless tokens and yet are secure against adversaries that create malicious stateful tokens.

In this paper the main question we are interested in, is understanding the round complexity measure of achieving static secure computation assuming tamper-proof tokens, both in the plain model as well as in the UC framework. In the Common Reference String (CRS) model, much effort has already been put in providing round efficient UC secure protocols. To name a few results, the work of Horvitz and Katz [HK07] shows how to obtain a two-round two-party secure computation, whereas the work of Garg et al. [GGHR14] gives a two-round multi-party protocol assuming the existence of indistinguishability obfuscation for polynomial circuits. While considering adaptive corruptions, the following works [GP15, CGP15, DKR15] show how to achieve adaptive UC security in $O(1)$ rounds in the CRS model based on indistinguishability obfuscation. Notably, among these results, Garg and Polychroniadou [GP15] show how to obtain a two-round protocol. This stands in contrast to the plain model, where it is unknown whether constant round protocols with security against adaptive adversaries tolerating any arbitrary number of corruptions exist.

In the literature that studies secure computation with stateless tokens, the work of Chandran et al. [CGS08] gives an $O(\kappa)$ rounds protocol where κ is the security parameter. Following that, Goyal et al. [GIS$^+$10] provided a $O(1)$ rounds construction based on collision-resistant hash functions (CRH) and a linear number of rounds assuming one-way functions (OWF), however, as we demonstrate in Appendix B, there is an issue with this particular protocol of theirs. In more detail, Goyal et al. first provided a "Quasi-

OT" protocol based on tokens that admits one-sided simulation and one-sided indistinguishability. Next, they provided a transformation from Quasi-OT to full OT. We demonstrate that this transformation is insecure by providing a concrete counter example that demonstrates that the real and ideal views can be distinguished.[1] The work of Choi et al. [CKS$^+$14] improves this result and provides a five-round UC secure protocol for arbitrary functionalities assuming CRH and verifiable random functions (VRF).[2] Finally, Agrawal et al. [AAG$^+$14] studied the complexity of constructing secure protocols with stateless tokens. The previous works thus leave the following question open:

> *Can we construct round optimal secure computation protocols with UC security assuming stateless tokens?*

We remark that it is easy to see that non-interactive secure computation is impossible when relying on stateless tokens, as a malicious receiver can repetitively query the tokens to recompute the function on multiple inputs. Thus, the best round complexity we can hope for assuming tamper-proof tokens is two.

Moreover, the work of [CKS$^+$14] shows that under a restricted setting where only one of the parties is allowed to create tokens, UC security is impossible to achieve with stateless tokens when the simulation only makes "black-box" usage of the code of the token. In other words, if we wish the tokens to be created by only one party then we need to rely on non-black-box simulation to achieve UC security. We stress that the result of [CKS$^+$14] holds only if we require UC security, and achieving the same in the plain model is unknown. Note that such constructions are desirable as they support applications in scenarios where only one party has the capability to create tokens. Secondly, we know that without any trusted setup, five-rounds are necessary and sufficient to construct secure protocols for arbitrary functionalities in the plain model [KO04, ORS15]. Thus, we are interested in the following question:

> *Can we construct round optimal black-box simulation secure protocols assuming stateless tokens in the plain model, where tokens are created by only one party?*

Towards minimizing the underlying hardness assumptions, the work of [GIS$^+$10] proves that information theoretic security is impossible to achieve using stateless tokens. They complemented this result by providing a protocol that is information-theoretically secure using stateful tokens.[3] Moreover, they showed that it is possible to UC realize any functionality in the computational setting assuming the existence of one-way functions, when additionally relying on stateless tokens. As mentioned before, we demonstrate that this result no longer holds. Other constructions include the work of Chandran et al. [CGS08] that constructs an $O(\kappa)$ rounds OT protocol based on enhanced trapdoor permutations (ETDP). The work of Choi et al. [CKS$^+$14], extending the techniques of [GIS$^+$10] and [DKM11], provides an $O(1)$ rounds construction based on CRH and an $O(\kappa)$ rounds construction based on OWF. This leaves the following question open:

> *Can we construct an O(1) rounds UC secure protocols based on stateless tokens assuming one-way functions?*

Common to all the above questions is whether it is possible to achieve a construction that makes only "black-box" usage of the underlying cryptographic primitives.[4] Additional related works regarding stateless tokens are the works of Moran and Segev [MS08] and Kolesnikov [Kol10]. The former work shows how to realize statistically secure UC commitments while the latter gives efficient string oblivious-transfer with covert security in the plain model.

---

[1] In private communication, the authors have acknowledged this issue and are in the process of updating their version.

[2] We believe that the work of Choi et al. [CKS$^+$14] is not vulnerable to the attack presented in our counter-example.

[3] Goyal et al. [GIS$^+$10] further showed that if *token-encapsulation* is allowed then unconditional security is achievable even with stateless tokens.

[4] A protocol makes black-box use of a cryptographic primitive if it only needs to access the input/output interface of the primitive.

1.1 Our Results

We answer all questions in the affirmative. Concretely, we provide *round optimal* constructions under *minimal assumption* of secure two-party computation both in the plain model and in the UC framework, where in the plain model only a single party needs to create tokens. Our protocols are comprised of two phases: (1) tokens exchange phase and (2) two communicated messages. We further show how to reuse the tokens for multiple parallel and sequential executions by employing Lamport's one-time secure signature scheme [Lam79]. More concretely, our first result gives a round optimal construction in the plain model:

Theorem 1.1 (Informal)**.** *Assuming the existence of one-way functions, there exists a two-round protocol that securely realizes any two-party functionality in the plain model assuming tokens, where the tokens are created by a single party.*

More formally, we show how to obtain a two-round protocol implementing the oblivious-transfer (OT) functionality where only the sender transmits tokens. Moreover, our protocol requires only black-box usage of the code in the token. Our second result provides a two-round UC secure protocol that realizes any arbitrary "well-formed" functionality under minimal assumptions. More formally,

Theorem 1.2 (Informal)**.** *Assuming the existence of one-way functions, there exists a two-round protocol that UC realizes any two-party functionality assuming tokens.*

As a warm-up, we first provide a three-round UC construction that extends our protocol from Theorem 1.1. This already demonstrates the feasibility of UC secure protocols under minimal assumptions. We next show how to construct a two-round UC protocol by employing recent techniques introduced by Ostrovsky, Richelson and Scafuro in [ORS15], which constructed a round optimal secure computation protocol in the plain model without any setup.

Finally, we remark that all constructions presented in this work only make black-box use of the underlying primitives (i.e. one-way functions).

1.2 Our Techniques

In the following, we present the high-level ideas incorporated into our protocols. Our constructions are proven in the real/ideal paradigm with security in the presence of static malicious adversaries.

Obtaining round optimal secure computation in the plain model. We focus our attention on constructing an OT protocol executed between a receiver and a sender, which is sufficient for general secure two-party computation [Kil88, IPS08]. In the malicious setting, security follows by constructing a simulator that produces an indistinguishable view while extracting the adversary's input. In order to achieve input extraction, our starting point is a technique presented in [GIS$^+$10]. Roughly speaking, in order to extract the receiver's input, the sender chooses a function F from a pseudorandom function family that maps $\{0,1\}^m$ to $\{0,1\}^n$ bits where $m >> n$, and incorporates it into a token that it sends to the receiver. Next, the receiver commits to its input b by first sampling a random string $u \in \{0,1\}^m$ and querying the PRF token on u, receiving back a value v. Finally, it sends $\mathsf{com}_b = (\mathsf{Ext}(u;r) \oplus b, r, v)$ where $\mathsf{Ext}(\cdot,\cdot)$ is a randomness strong extractor. Now, since the PRF is highly compressing, it holds with high probability that conditioned on v, u has very high min-entropy and therefore $\mathsf{Ext}(u;r) \oplus b, r$ statistically hides b.[5] Furthermore, since the simulator

[5]We remark that this does not contradict the result Haitner et al. [HHRS15] who show that statistically hiding commitments cannot be achieved via a fully black-box construction based on one-way functions in fewer than $\frac{\kappa}{\log \kappa}$ rounds, where κ is the

monitors the queries made by the receiver to the PRF token, it can directly extract the receiver's input with the knowledge u upon given the receiver's commitment. This is because by the pseudorandomness property of F, it is computationally infeasible for an adversarial receiver to obtain two values u, u' that map to v. Given this tool in our arsenal, a simple (incorrect) protocol to realize OT can be constructed as follows. In the token exchange phase, the parties exchange two sets of PRF tokens, respectively denoted by $\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF}}$ and $\mathsf{TK}_{\mathrm{R}}^{\mathsf{PRF}}$. Next, the receiver commits to its bit com_b using the approach described above, followed by the sender committing to its input $(\mathsf{com}_{s_0}, \mathsf{com}_{s_1})$ along with a one-time memory (OTM), where an OTM token implements the one-out-of-two string OT functionality. More specifically, it stores two strings s_0 and s_1, and given a single bit b outputs s_b and self-destructs. The OTM of our protocol behaves as follows:

- On input b^*, u^*, the token outputs $(s_b, \mathsf{decom}_{s_b})$ only if $\mathsf{com}_b = (\mathsf{Ext}(u^*; r) \oplus b^*, r, v)$ and $\mathsf{PRF}(u^*) = v$. Otherwise, the token aborts.

The receiver then runs the token to obtain s_b and verifies if decom_{s_b} correctly decommits com_{s_b} to s_b. This simple idea is vulnerable to an input-dependent abort attack, where the token aborts depending on the value b^*. To prevent this, analogous to [GIS$^+$10], a second (still incorrect) idea is to have κ parallel independent instances of the above protocol where the sender sets as input $(z_i, z_i \oplus \Delta)$ to the ith instance, where $z_1, \ldots, z_\kappa, \Delta$ are chosen at random. Along with its message, the sender also sends $C_0 = s_0 \oplus w, C_1 = s_1 \oplus w \oplus \Delta$ where $w = \bigoplus_{i=1}^{\kappa} z_i$. Then, the receiver samples random bits $b_1, \ldots, b_\kappa$ conditioned on $\bigoplus_{i=1}^{\kappa} b_i = b$, where b is its input. Finally, to reconstruct s_b, the receiver simply adds all the outputs of the κ OTM calls to C_b. Note that extraction can be achieved by simply extracting all of the sender's and receiver's inputs to the tokens, by monitoring the PRF queries. More precisely, the simulation in [GIS$^+$10] extracts the sender's input by first sampling two sets of random bit-vectors $\{b_i\}_{i\in[\kappa]}$ and $\{b'_i\}_{i\in[\kappa]}$ that add up to 0 and 1, respectively, and then running the receiver's strategy with these vectors as inputs, i.e. add the outputs the tokens would have revealed on input b_i (or b'_i) and then unmasking C_0 (resp., C_1) to obtain s_0 (resp., s_1).

While this strategy seems to work, a subtle issue arises since the values are extracted directly instead of running the token on the actual $\{b_i\}_{i\in[\kappa]}$ and $\{b'_i\}_{i\in[\kappa]}$ values. In fact, as we demonstrate in our counter example, by making the tokens abort on selective inputs we can show that the value extracted by the simulator and the value obtained by the real receiver can be distinguished. Intuitively, this is because no check is made by the receiver to ensure that each of the inputs to the individual tokens add up to Δ; see our concrete counter example in Appendix B. This is precisely why the simulation and the protocol presented in [GIS$^+$10] fail.

In this work we rectify this issue by using an alternative mechanism to extract the sender's inputs where the simulator can extract both inputs by running the token twice. The high-level idea is that the receiver commits to its input via the look ahead trapdoor commitment scheme of Pass and Wee [PW09]. In their construction, the trapdoor is committed to by the sender in a first message, and is revealed after the receiver commits to its input. Now, since the simulator can extract the sender's commitment, it will be able to equivocate the receiver's input using the trapdoor while the honest receiver will not be able to do so. Specifically, it will be able to run the tokens twice, once with $\{b_i\}_{i\in[\kappa]}$ adding up to 0 and another time with new $\{b'_i\}_{i\in[\kappa]}$ adding up to 1. Nevertheless, this yields only a three message protocol as the sender needs to commit to the trapdoor in a first message. In the plain model, we are able to reduce the number of messages into two by relying on rewinding for extracting the trapdoor and then committing to the receiver's input.

security parameter. This is because the tokens themselves cannot be replaced by arbitrary one-way functions, or even a random permutation, as we specifically require the functionality implemented in the token to be compressing. Second, we cannot obtain a construction in the plain model without any setup even though the PRF function implemented in the tokens makes only black-box access to an underlying one-way function, since our proof crucially relies on the fact that the PRF key is hidden from the receiver via the token.

Obtaining round optimal UC secure computation. In the UC model, however, this approach does not work. Instead, we rely on the beautiful work of Ostrovsky, Richelson and Scafuro [ORS15] which considers an alternative combiner that allows the simulator to directly extract the sender's input without requiring to run the token twice. As a warmup consider the following sender's algorithm that chooses first two random stringss x_0 and x_1 and computes their shares $[x_b] = (x_b^1, \ldots, x_b^{2\kappa})$ for $b \in \{0,1\}$ using the $\kappa+1$-out-of-2κ Shamir secret-sharing scheme. Next, for each $b \in \{0,1\}$, the sender commits to $[x_b]$ by first generating two vectors α_b and β_b such that $\alpha_b \oplus \beta_b = [x_b]$, and then committing to these vectors. Finally, the parties engage in 2κ parallel OT executions where the sender's input to the jth instance are the decommitments to $(\alpha_0[j], \beta_0[j])$ and $(\alpha_1[j], \beta_1[j])$. The sender further sends $(s_0 \oplus x_0, s_1 \oplus x_1)$. Thus, to learn s_b, the receiver needs to learn x_b. For this, it enters the bit b for $\kappa+1$ or more OT executions and then reconstructs the shares for x_b, followed by reconstructing s_b using these shares. Nevertheless, this reconstruction procedure works only if there is a mechanism that verifies whether the shares are consistent.

To resolve this issue, Ostrovsky et al. made the observation that the Shamir secret-sharing scheme has the property for which there exists a linear function ϕ such that any vector of shares $[x_b]$ is valid if and only if $\phi(x_b) = 0$. Moreover, since the function ϕ is linear, it suffices to check whether $\phi(\alpha_b) + \phi(\beta_b) = 0$. Nevertheless, this check requires from the receiver to know the entire vectors α_b and β_b for its input b. This means it would have to use b as the input to all the 2κ OT executions, which may lead to an input-dependent abort attack. Instead, Ostrovsky et al. introduced a mechanism for checking consistency indirectly via a *cut-and-choose* mechanism. More formally, the sender chooses κ pairs of vectors that add up to $[x_b]$. It is instructive to view them as matrices $A_0, B_0, A_1, B_1 \in \mathbb{Z}_p^{\kappa \times 2\kappa}$ where for every row $i \in [\kappa]$ and $b \in \{0,1\}$, it holds that $A_b[i,\cdot] \oplus B_b[i,\cdot] = [x_b]$. Next, the sender commits to each entry of each matrix separately and sets as input to the jth OT the decommitment information of the entire column $((A_0[\cdot,j], B_0[\cdot,j]), (A_1[\cdot,j], B_1[\cdot,j]))$. Upon receiving the information for a particular column j, the receiver checks if for all i, $A_b[i,j] \oplus B_b[i,j]$ agree on the same value. We refer to this as the *shares consistency check*.

Next, to check the validity of the shares, the sender additionally sends vectors $[z_1^b], \ldots, [z_\kappa^b]$ in the clear along with the sender's message where it commits to the entries of A_0, A_1, B_0 and B_1 such that $[z_i^b]$ is set to $\phi(A_0[i,\cdot])$. Depending on the challenge message, the sender decommits to $A_0[i,\cdot]$ and $A_1[i,\cdot]$ if $c_i = 0$ and $B_0[i,\cdot]$ and $B_1[i,\cdot]$ if $c_i = 1$. If $c_i = 0$, then the receiver checks whether $\phi(A_b[i,\cdot]) = [z_i^b]$, and if $c_i = 1$ it checks whether $\phi(B_b[i,\cdot]) + z_i^b = 0$. This check ensures that except for at most $s \in \omega(\log \kappa)$ of the rows $(A_b[i,\cdot], B_b[i,\cdot])$ satisfy the condition that $\phi(A_b[i,\cdot]) + \phi(B_b[i,\cdot]) = 0$ and for each such row i, $A_b[i,\cdot] + B_b[i,\cdot]$ represents a valid set of shares for both $b = 0$ and $b = 1$. This check is denoted by the *shares validity check*. In the final protocol, the sender sets as input in the jth parallel OT, the decommitment to the entire jth columns of A_0 and B_0 corresponding to the receiver's input 0 and A_1 and B_1 for input 1. Upon receiving the decommitment information on input b_j, the receiver considers a column "good" only if $A_{b_j}[i,j] + B_{b_j}[i,j]$ add up to the same value for every i. Using another cut-and-choose mechanism, the receiver ensures that there are sufficiently many good columns which consequently prevents any input-independent behavior. We refer this to the shares-validity check.

We adapt the idea introduced in [ORS15] to obtain a two-round UC protocol as follows. The receiver commits to its input bits $b_1, \ldots, b_{2\kappa}$ and the challenge bits for the share consistency check $c_1, \ldots, c_\kappa$ using the PRF tokens. Then, the sender sends all the commitments *a la* [ORS15] and $2\kappa + \kappa$ tokens, where the first 2κ tokens provide the decommitments to the columns as in the OT, and the second set of κ tokens give the decommitments of the rows for the shares consistency check. The simulator now extracts the sender's inputs by monitoring its queries and we are able to show that there cannot be any input independent behavior of the token if it passes both the shares consistency check and the shares validity check.

2 Preliminaries

Basic notations. We denote the security parameter by κ. We say that a function $\mu : \mathbb{N} \to \mathbb{N}$ is *negligible* if for every positive polynomial $p(\cdot)$ and all sufficiently large κ's it holds that $\mu(\kappa) < \frac{1}{p(\kappa)}$. We use the abbreviation PPT to denote probabilistic polynomial-time. We specify next the definition of computationally indistinguishable and statistical distance.

Definition 2.1. *Let $X = \{X(a, \kappa)\}_{a\in\{0,1\}^*,\kappa\in\mathbb{N}}$ and $Y = \{Y(a, \kappa)\}_{a\in\{0,1\}^*,\kappa\in\mathbb{N}}$ be two distribution ensembles. We say that X and Y are* computationally indistinguishable*, denoted $X \stackrel{c}{\approx} Y$, if for every* PPT *machine D, every $a \in \{0,1\}^*$, every positive polynomial $p(\cdot)$ and all sufficiently large κ's,*

$$\left|\Pr\left[D(X(a,\kappa), 1^\kappa) = 1\right] - \Pr\left[D(Y(a,\kappa), 1^\kappa) = 1\right]\right| < \frac{1}{p(\kappa)}.$$

Definition 2.2. *Let X_κ and Y_κ be random variables accepting values taken from a finite domain $\Omega \subseteq \{0,1\}^\kappa$. The* statistical distance *between X_κ and Y_κ is*

$$SD(X_\kappa, Y_\kappa) = \frac{1}{2}\sum_{\omega\in\Omega} \big|\Pr[X_\kappa = \omega] - \Pr[Y_\kappa = \omega]\big|.$$

*We say that X_κ and Y_κ are ε-*close *if their statistical distance is at most $SD(X_\kappa, Y_\kappa) \leq \varepsilon(\kappa)$. We say that X_κ and Y_κ are* statistically close*, denoted $X_\kappa \approx_s Y_\kappa$, if $\varepsilon(\kappa)$ is negligible in κ.*

2.1 Pseudorandom Functions

Informally speaking, a pseudorandom function (PRF) is an efficiently computable function that looks like a truly random function to any PPT observer. Namely,

Definition 2.3 (Pseudorandom function ensemble)**.** *Let $F = \{\mathsf{PRF}_\kappa\}_{\kappa\in\mathbb{N}}$ where for every κ, $\mathsf{PRF}_\kappa : \{0,1\}^\kappa \times \{0,1\}^m \to \{0,1\}^l$ is an efficiently computable ensemble of keyed functions. We say that $F = \{\mathsf{PRF}_\kappa\}_{\kappa\in\mathbb{N}}$ is* a pseudorandom function ensemble *if for every* PPT *machine D, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all sufficiently large κ's,*

$$|\Pr[D^{\mathsf{PRF}_\kappa(k,\cdot)}(1^\kappa)] = 1 - \Pr[D^{f_\kappa}(1^\kappa) = 1]| \leq \mathsf{negl}(\kappa),$$

where k is picked uniformly from $\{0,1\}^\kappa$ and f_κ is chosen uniformly at random from the set of functions mapping m-bit strings into l-bit strings. We sometimes omit κ from our notation when it is clear from the context.

2.2 Commitment Schemes

Commitment schemes are used to enable a party, known as the *sender* S, to commit itself to a value while keeping it secret from the *receiver* R (this property is called *hiding*). Furthermore, in a later stage when the commitment is opened, it is guaranteed that the "opening" can yield only a single value determined in the committing phase (this property is called *binding*). In this work, we consider commitment schemes that are *statistically binding*, namely while the hiding property only holds against computationally bounded (non-uniform) adversaries, the binding property is required to hold against unbounded adversaries. Formally,

Definition 2.4 (Commitment schemes)**.** *A* PPT *machine $\mathsf{Com} = \langle S, R\rangle$ is said to be a non-interactive commitment scheme if the following two properties hold.*

Computational hiding: *For every (expected)* PPT *machine* R^**, it holds that the following ensembles are computationally indistinguishable.*

- $\{\mathbf{View}_{\mathsf{Com}}^{\mathrm{R}^*}(m_1, z)\}_{\kappa \in N, m_1, m_2 \in \{0,1\}^\kappa, z \in \{0,1\}^*}$
- $\{\mathbf{View}_{\mathsf{Com}}^{\mathrm{R}^*}(m_2, z)\}_{\kappa \in N, m_1, m_2 \in \{0,1\}^\kappa, z \in \{0,1\}^*}$

where $\mathbf{View}_{\mathsf{Com}}^{R^*}(m, z)$ *denotes the random variable describing the output of* R^* *after receiving a commitment to* m *using* Com.

Statistical binding: *For any (computationally unbounded) malicious sender* S^* *and auxiliary input* z*, it holds that the probability that there exist valid decommitments to two different values for a view* v*, generated with an honest receiver while interacting with* $\mathrm{S}^*(z)$ *using* Com*, is negligible.*

We refer the reader to [Gol01] for more details. We recall that non-interactive perfectly binding commitment schemes can be constructed based on one-way permutation, whereas two-round statistically binding commitment schemes can be constructed based on one-way functions [Nao91]. To set up some notations, we let $\mathsf{com}_m \leftarrow \mathsf{Com}(m; r_m)$ denote a commitment to a message m, where the sender uses uniform random coins r_m. The decommitment phase consists of the sender sending the decommitment information $\mathsf{decom}_m = (m, r_m)$ which contains the message m together with the randomness r_m. This enables the receiver to verify whether decom_m is consistent with the transcript com_m. If so, it outputs m; otherwise it outputs $\perp$. For simplicity of exposition, in the sequel, we will assume that random coins are an implicit input to the commitment functions, unless specified explicitly.

Definition 2.5 (Trapdoor commitment schemes)**.** *Let* $\mathsf{Com} = (\mathrm{S}, \mathrm{R})$ *be a statistically binding commitment scheme. We say that* Com *is a trapdoor commitment scheme is there exists an expected* PPT *oracle machine* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *such that for any* PPT R^* *and all* $m \in \{0,1\}^\kappa$*, the output* (τ, w) *of the following experiments is computationally indistinguishable:*

- *an honest sender* S *interacts with* R^* *to commit to* m*, and then opens the commitment:* τ *is the view of* R^* *in the commit phase, and* w *is the message* S *sends in the open phase.*

- *the simulator* $\mathcal{S}$ *generates a simulated view* τ *for the commit phase, and then opens the commitment to* m *in the open phase: formally* $(\tau, state) \leftarrow \mathcal{S}_1^{\mathrm{R}^*}(1^\kappa)$*,* $w \leftarrow \mathcal{S}_2(state, m)$*.*

2.3 Randomness Extractors

The min-entropy of a random variable X is $H_\infty(X) = -\log(\max_x \Pr[X = x])$.

Definition 2.6 (Extractors)**.** *A function* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$ *is a* (k, ε)*-strong extractor if for all pairs of random variables* (X, I) *such that* $X \in \{0,1\}^n$ *and* $H_\infty(X|I) \geq k$ *it holds that*

$$SD((\mathsf{Ext}(X, S), S, I), (U_m, S, I)) \leq \varepsilon,$$

where S *is uniform over* $\{0,1\}^t$ *and* U_m *is the uniform distribution over* $\{0,1\}^m$.

The Leftover Hash Lemma shows how to explicitly construct an extractor from a family of pairwise independent functions $\mathcal{H}$. The extractor uses a random hash function $h \leftarrow \mathcal{H}$ as its seed and keeps this seed in the output of the extractor.

Theorem 2.7 (Leftover Hash Lemma)**.** *If* $\mathcal{H} = \{h : \{0,1\}^n \to \{0,1\}^m\}$ *is a pairwise independent family where* $m = n - 2\log\frac{1}{\varepsilon}$*, then* $\mathsf{Ext}(x,h) = (h, h(x))$ *is a strong* (n,ε)*-extractor.*

In this work we will consider the case where $m = 1$ and $n \geq 2\kappa + 1$ where κ is the security parameter. This yields $\varepsilon = 2^{-\frac{2\kappa+1-1}{2}} = 2^{-\kappa}$.

2.4 Hardcore Predicates

Definition 2.8 (Hardcore predicate)**.** *Let* $f : \{0,1\}^\kappa \to \{0,1\}^*$ *and* $\mathsf{H} : \{0,1\}^\kappa \to \{0,1\}$ *be a polynomial-time computable functions. We say* H *is a hardcore predicate of* f*, if for every* PPT *machine* A*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr[x \leftarrow \{0,1\}^\kappa; y = f(x) : A(1^\kappa, y) = \mathsf{H}(x)] \leq \frac{1}{2} + \mathsf{negl}(\kappa).$$

An important theorem by Goldreich and Levin [GL89] states that if f is a one-way function over $\{0,1\}^\kappa$ then the one-way function f' over $\{0,1\}^{2\kappa}$, defined by $f'(x,r) = (f(x), r)$, admits the following hardcore predicate $b(x,r) = \langle x, r\rangle = \Sigma x_i r_i \bmod 2$, where x_i, r_i is the ith bit of x, r respectively. In the following, we refer to this predicate as the GL bit of f. We will use the following theorem that establishes the list-decoding property of the GL bit.

Theorem 2.9 ([GL89])**.** *There exists a* PPT *oracle machine* Inv *that on input* (κ, ε) *and oracle access to a predictor* PPT B*, runs in time poly*$(\kappa, \frac{1}{\varepsilon})$*, makes at most* $O(\frac{\kappa^2}{\varepsilon^2})$ *queries to* B *and outputs a list* L *with* $|L| \leq \frac{4\kappa}{\varepsilon^2}$ *such that if*

$$\Pr[r \leftarrow \{0,1\}^\kappa : B(r) = \langle x, r\rangle] \geq \frac{1}{2} + \frac{\varepsilon}{2}$$

then

$$\Pr[L \leftarrow \mathsf{Inv}^B(\kappa, \varepsilon) : x \in L] \geq \frac{1}{2}.$$

2.5 Secret-Sharing

A secret-sharing scheme allows distribution of a secret among a group of n players, each of whom in a *sharing phase* receive a share (or piece) of the secret. In its simplest form, the goal of secret-sharing is to allow only subsets of players of size at least $t+1$ to reconstruct the secret. More formally a $t+1$-out-of-n secret sharing scheme comes with a sharing algorithm that on input a secret s outputs n shares $s_1, \ldots, s_n$ and a reconstruction algorithm that takes as input $((s_i)_{i\in S}, S)$ where $|S| > t$ and outputs either a secret s' or $\perp$. In this work, we will use the Shamir's secret sharing scheme [Sha79] with secrets in $\mathbb{F} = GF(2^\kappa)$. We present the sharing and reconstruction algorithms below:

Sharing algorithm: For any input $s \in \mathbb{F}$, pick a random polynomial $f(\cdot)$ of degree t in the polynomial-field $\mathbb{F}[x]$ with the condition that $f(0) = s$ and output $f(1), \ldots, f(n)$.

Reconstruction algorithm: For any input $(s'_i)_{i\in S}$ where none of the s'_i are $\perp$ and $|S| > t$, compute a polynomial $g(x)$ such that $g(i) = s'_i$ for every $i \in S$. This is possible using Lagrange interpolation where g is given by

$$g(x) = \sum_{i\in S} s'_i \prod_{j\in S/\{i\}} \frac{x-j}{i-j}\ .$$

Finally the reconstruction algorithm outputs $g(0)$.

We will additionally rely on the following property of secret-sharing schemes. To this end, we view the Shamir secret-sharing scheme as a linear code generated by the following $n \times (t+1)$ Vandermonde matrix

$$A = \begin{pmatrix} 1 & 1^2 & \cdots & 1^t \\ 1 & 2^2 & \cdots & 2^t \\ \vdots & \vdots & \vdots & \vdots \\ 1 & n^2 & \cdots & n^t \end{pmatrix}$$

More formally, the shares of a secret s that are obtained via a polynomial f in the Shamir scheme, can be obtained by computing $A\mathbf{c}$ where $\mathbf{c}$ is the vector containing the coefficients of f. Next, we recall that for any linear code A, there exists a parity check matrix H of dimension $(n-t-1) \times n$ which satisfies the equation $HA = \mathbf{0}_{(n-t-1)\times(t+1)}$, i.e. the all 0's matrix. We thus define the linear operator $\phi(v) = Hv$ for any vector v. Then it holds that any set of shares $\mathbf{s}$ is valid if and only if it satisfies the equation $\phi(\mathbf{s}) = \mathbf{0}_{n-t-1}$.

The authors in [DZ13] were the first to propose an algorithm for verifying membership in (binary) codes, i.e., verifying the product of Boolean matrices in quadratic time with exponentially small error probability, while previous methods only achieved constant error.

3 Modeling Tamper-Proof Hardware

Our modeling for tamper-proof hardware is based on the modeling of [Kat07, GIS+10] for stateless tokens by defining an ideal functionality $\mathcal{F}_{\text{WRAP}}^{\text{Stateless}}$ that models a real world functionality where a sender S sends a stateless token M (specified by the code of a Turing machine), to a receiver R. Each such machine is uniquely identified by a machine identifier mid. Note that the receiver may run M multiple times on inputs of its choice as the token is stateless, and thus the functionality must save the description of the code it gets from the sender. Nevertheless, our protocols prevent R from gaining an additional information when reusing M multiple times using standard techniques. Moreover, our protocols are secure even in the case where S provides a maliciously created *stateful* token. The description of $\mathcal{F}_{\text{WRAP}}^{\text{Stateless}}$ is given in Figure 1.

Functionality $\mathcal{F}_{\text{WRAP}}^{\text{Stateless}}$

Functionality $\mathcal{F}_{\text{WRAP}}^{\text{Stateless}}$ is parameterized by a polynomial $p(\cdot)$ and an implicit security parameter κ.

Create. Upon receiving $(\mathsf{Create}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}, M)$ from S, where M is a Turing machine, do:

1. Send $(\mathsf{Create}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid})$ to R.
2. Store $(\mathrm{S}, \mathrm{R}, \mathsf{mid}, M)$.

Execute. Upon receiving $(\mathsf{Run}, \mathsf{sid}, \mathrm{S}, \mathsf{mid}, x)$ from R, find the unique stored tuple $(\mathrm{S}, \mathrm{R}, \mathsf{mid}, M)$. If no such tuple exists, do nothing. Run $M(x)$ for at most $p(\kappa)$ steps, and let out be the response ($\mathsf{out} = \perp$ if M does not halt in $p(k)$ steps). Send $(\mathsf{sid}, \mathrm{R}, \mathsf{mid}, \mathsf{out})$ to R.

Figure 1: The ideal functionality for single-use stateless tokens.

4 Static MPC from Stateless Tokens

4.1 Two-Round OT in the Plain Model

4.1.1 Building Blocks: Commitment Schemes

Trapdoor commitment schemes. A core building block of our protocol is a trapdoor commitment scheme TCom (cf. Definition 2.5) introduced by Pass and Wee in [PW09]. In Figure 2 we describe their 4-round trapdoor commitment scheme that is based on one-way permutations. In particular, the protocol comprises a 4-round challenge-response protocol where the receiver commits to its challenge in the first message (using a non-interactive perfectly binding commitment scheme). The knowledge of the receiver's challenge enables the simulator to cheat in the commit phase and equivocate the committed message into any bit (this notion of "look ahead" trapdoor commitment is borrowed from the area of zero-knowledge proofs).

More specifically, the trapdoor commitment scheme TCom, described in Figure 2, proceeds as follows. In order to commit to a bit m the sender commits to a matrix M of size 2×2, so that m is split into two shares which are committed within the two rows of M. Next, the receiver sends a challenge bit e where the sender must open the two commitments that lie in the eth column (and must correspond to the same share of m, thus it is easy to verify correctness). Later, in the decommit phase the sender opens the values to a row of his choice enabling the receiver to reconstruct m. Note that if the sender knows the challenge bit in advance it can commit to two distinct bits by making sure that one of the columns has different bits. In order to decrease the soundness error this protocol is repeated multiple times in parallel. In this paper we implement the internal commitment of Pass and Wee using a statistical hiding commitment scheme that is based on pseudorandom functions; see details below.

Non-interactive commitment schemes. Our construction further relies on a non-interactive perfectly binding commitment scheme that is incorporated inside the sender's token $\mathsf{TK}_{\mathrm{S}}^{\mathsf{com}}$. Such commitments can be build based on the existence of one-way permutations. Importantly, it is possible to relax our assumptions to one-way functions by relying on a two-round statistically binding commitment scheme [Nao91], and allowing the token $\mathsf{TK}_{\mathrm{S}}^{\mathsf{com}}$ to take an additional input that will serve as the first message of the commitment scheme. Overall, that implies that we only need to assume one-way functions. For clarity of presentation, we use a non-interactive commitment scheme that is based on one-way permutations; see Section 4.1.3 for more details.

4.1.2 Our Protocol

We are now ready to introduce our first protocol that securely computes the functionality $\mathcal{F}_{\mathrm{OT}} : ((s_0, s_1), b) \mapsto (\perp, s_b)$ in the plain model, using only two rounds (see Figure 4) and a one-way token transfer phase that involves sending a set of tokens from the sender to the receiver in *one direction* (see Figure 3). For simplicity of exposition, in the sequel we will assume that the random coins are an implicit input to the commitments and the extractor, unless specified explicitly. Informally, in the one-way token transfer phase the sender sends two types of tokens. The PRF tokens $\{\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}\}_{l\in[4\kappa^2]}$ are used by the receiver to commit to its input b using the shares $\{b_i\}_{i\in[\kappa]}$. Namely, the number of tokens equals 4κ (which denote the number of tokens per Pass-Wee commitment), times κ which is the number of the receiver's input shares. Whereas, the commitment token $\mathsf{TK}_{\mathrm{S}}^{\mathsf{Com}}$ is used by the receiver to obtain the commitments of the sender in order to mask the values $\{(s_0^i, s_1^i)\}_{i\in[\kappa]}$ which are later used to conceal the sender's real inputs to the oblivious transfer. Next, the receiver shares its bit b into $b_1, \ldots, b_\kappa$ such that $b = \bigoplus_{i=1}^{\kappa} b_i$ and commits to these shares using the Pass-Wee trapdoor commitment scheme. Importantly, we consider a slightly variant of the Pass-Wee

Trapdoor commitment scheme TCom **[PW09]**

The commitment scheme TCom uses a statistically binding commitment scheme Com and runs between sender S and receiver R.

Input: S holds a message $m \in \{0,1\}$.

Commit Phase:

R $\rightarrow$ S: R chooses a challenge $e = e_1, \ldots, e_\kappa \leftarrow \{0,1\}$ and sends the commitment $\mathsf{com}_e \leftarrow \mathsf{Com}(e)$ to S.

S $\rightarrow$ R: S proceeds as follows:

1. S chooses $\eta_1, \ldots, \eta_\kappa \leftarrow \{0,1\}^\kappa$.
2. For all $i \in [\kappa]$, S commits to the following matrix:

$$\begin{pmatrix} \mathsf{com}^{00}_{\eta_i} & \mathsf{com}^{01}_{m\oplus\eta_i} \\ \\ \mathsf{com}^{10}_{\eta_i} & \mathsf{com}^{11}_{m\oplus\eta_i} \end{pmatrix} = \begin{pmatrix} \mathsf{Com}(\eta_i) & \mathsf{Com}(m\oplus\eta_i) \\ \\ \mathsf{Com}(\eta_i) & \mathsf{Com}(m\oplus\eta_i) \end{pmatrix}$$

R $\rightarrow$ S: R sends decom_e of the challenge $e = e_1, \ldots, e_\kappa \leftarrow \{0,1\}$ to S.

S $\rightarrow$ R: S proceeds as follows:

1. For all $i \in [\kappa]$, S sends the decommitments of the column $(\mathsf{decom}^{0e_i}_{(e_i\cdot m)\oplus\eta_i}, \mathsf{decom}^{1e_i}_{(e_i\cdot m)\oplus\eta_i})$.
2. For all $i \in [\kappa]$, R checks that the decommitments are valid and that $\mathsf{decom}^{0e_i}_{(e_i\cdot m)\oplus\eta_i} = \mathsf{decom}^{1e_i}_{(e_i\cdot m)\oplus\eta_i}$.

Decommit Phase:

1. For all $i \in [\kappa]$, S chooses $r = r_1, \ldots, r_\kappa \leftarrow \{0,1\}$ and sends the bit m and the decommitments of the row $(\mathsf{decom}^{r_i0}_{\eta_i}, \mathsf{decom}^{r_i1}_{r_i\oplus\eta_i})$.
2. For $i \in [\kappa]$, R checks that the decommitments are valid and that $m = \mathsf{decom}^{r_i0}_{\eta_i} \oplus \mathsf{decom}^{r_i1}_{r_i\oplus\eta_i}$.

Figure 2: Trapdoor commitment scheme

commitment scheme where we combine the last two steps of the commit phase with the decommit phase. In particular, the final verification in the commit phase is included as part of the decommitment phase and incorporated into the sender's tokens $\{\mathsf{TK}_i\}$ that are forwarded in the second round. The sender further sends the commitments to its inputs s_0, s_1 computed based on hardcore predicates for the (s_i^0, s_i^1) values and a combiner specified as follows. The sender chooses $z_1, \ldots, z_\kappa$ and Δ at random, where $\bigoplus_{i=1}^{\kappa} z_i$ masks s_0 and $\bigoplus_{i=1}^{\kappa} z_i \oplus \Delta$ masks s_1. Finally, the sender respectively commits to each z_i and $z_i \oplus \Delta$ using the hardcore bits computed on the (s_i^0, s_i^1) values. More precisely, it sends

$$s_0' = w \oplus s_0 \text{ and } s_1' = w \oplus \Delta \oplus s_1$$
$$\forall\, i \in [\kappa]\ w_i^0 = z_i \oplus \mathsf{H}(s_i^0) \text{ and } w_i^1 = z_i \oplus \Delta \oplus \mathsf{H}(s_i^1)$$

where $w = \bigoplus_{i=1}^{\kappa} z_i$. If none of the tokens abort, the receiver obtains $s_i^{b_i}$ for all $i \in [\kappa]$ and computes $s_b = s'_b \oplus (w_1^{b_1} \oplus H(s_1^{b_1})) \cdots \oplus (w_\kappa^{b_\kappa} \oplus H(s_\kappa^{b_\kappa}))$. If any of the OT tokens, i.e. TK_i, aborts then the receiver assumes a default value for s_b.

Remark 4.1. *In [GIS+10], it is pointed out by Goyal et al. in Footnote 12 that assuming a default value in case the token aborts might cause an input-dependent abort. However, this problem arises only in their protocol as a result of the faulty simulation. In particular, our protocol is not vulnerable to this since the simulator for a corrupted sender follows the honest receiver's strategy to extract both the inputs via (statistical) equivocation. In contrast, the simulation in [GIS+10] runs the honest receiver's strategy for a randomly chosen input in a main execution to obtain the (adversarially corrupted) sender's view and uses a "receiver-independent" strategy to extract the sender's inputs. For more details, see Appendix B.*

Remark 4.2. *In Footnote 10 of [GIS+10], Goyal et al. explain why it is necessary that the receiver run the token implementing the one-time memory functionality (OTM) in the prescribed round. More precisely, they provide a scenario where the receiver can violate the security of a larger protocol in the OT-hybrid by delaying when the token implementing the OTM is executed. Crucial to this attack is the ability of the receiver to run the OTM token on different inputs. In order to prevent such an attack, the same work incorporates a mechanism where the receiver is forced to run the token in the prescribed round. We remark here that our protocol is not vulnerable to such an attack. We ensure that there is only one input on which the receiver can query the OTM token and this invalidates the attack presented in [GIS+10].*

One-Way Token Transfer Phase $\mathsf{TK}_{\mathrm{TP}}$

Let (1) Com be a non-interactive perfectly binding commitment scheme and (2) F, F' be two families of pseudorandom functions that map $\{0,1\}^{5\kappa} \to \{0,1\}^{\kappa}$ and $\{0,1\}^{\kappa} \to \{0,1\}^{p(\kappa)}$, respectively.

One-Way Token Transfer Phase:

1. $\{\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}\}_{l\in[4\kappa^2]}$: S chooses $4\kappa^2$ random PRF keys $\{\gamma_l\}_{l\in[4\kappa^2]}$ for family F. Let $\mathsf{PRF}_{\gamma_l} : \{0,1\}^{5\kappa} \to \{0,1\}^{\kappa}$ denote the pseudorandom function. For all $l \in [4\kappa^2]$, S creates a token $\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}$ by sending $(\mathsf{Create}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_l, M_1)$ to R, that on input x outputs $\mathsf{PRF}_{\gamma_l}(x)$, where M_1 is the functionality.
2. $\mathsf{TK}_{\mathrm{S}}^{\mathsf{Com}}$: S chooses a random PRF' key γ' for family F'. Let $\mathsf{PRF}'_{\gamma'} : \{0,1\}^{\kappa} \to \{0,1\}^{p(\kappa)}$ denote the pseudorandom function. S creates token $\mathsf{TK}_{\mathrm{S}}^{\mathsf{Com}}$ by sending $(\mathsf{Create}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{l+1}, M_2)$ to R where M_2 is the functionality that on input (tcom_{b_i}, i) proceeds as follows:
   - If $i = 0$: compute $V = \mathsf{PRF}'_{\gamma'}(0^{\kappa})$, parse V as $e||r$ and output $\mathsf{com}_e \leftarrow \mathsf{Com}(e; r)$.
   - Otherwise: compute $V = \mathsf{PRF}'_{\gamma'}(\mathsf{tcom}_{b_i}||i)$, parse V as $s_0^i||s_1^i||r_0||r_1$, compute $\mathsf{com}_{s_b^i} \leftarrow \mathsf{Com}(s_b^i; r_b)$ for $b = \{0,1\}$, and output $\mathsf{com}_{s_0^i}, \mathsf{com}_{s_1^i}$.

   We remark that if V is longer than what is required in either case, we simply truncate it to the appropriate length.

Figure 3: OT in the plain model - one-way token transfer phase

Theorem 4.1. *Assume the existence of one-way permutations, then the protocol presented in Figure 4 securely realizes $\mathcal{F}_{\mathrm{OT}}$ in the $\mathcal{F}_{\mathrm{WRAP}}^{\mathrm{Stateless}}$-hybrid model.*

Protocol Π_{OT}

Protocol Π_{OT} is presented in the $\mathcal{F}_{\mathrm{WRAP}}^{\mathrm{Stateless}}$-hybrid model with sender S and receiver R. Let (1) Com be a non-interactive perfectly binding commitment scheme, (2) $\mathsf{TCom} = \{\mathsf{TCmsg}_1, \mathsf{TCmsg}_2, \mathsf{TCmsg}_3\}$ denote the three messages exchanged in the commit phase of the trapdoor commitment scheme, (3) F, F' be two PRF families that map $\{0,1\}^{5\kappa} \to \{0,1\}^{\kappa}$ and $\{0,1\}^{\kappa} \to \{0,1\}^{p(\kappa)}$, respectively (4) H denote a hardcore bit function and (5) $\mathsf{Ext} : \{0,1\}^{5\kappa} \times \{0,1\}^{d} \to \{0,1\}$ denote a randomness extractor where the source has length 5κ and the seed has length d (for simpler exposition we drop the randomness in description below).

One-Way Token Transfer Phase: S creates tokens $\{\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}\}_{l \in [4\kappa^2]}$ and $\mathsf{TK}_{\mathrm{S}}^{\mathsf{Com}}$ as per Figure 3.

Input: S holds two strings $s_0, s_1 \in \{0,1\}^{\kappa}$ and R holds a bit b.

The Protocol:

$\mathrm{R} \to \mathrm{S}$:

1. R sends $(\mathsf{Run}, \mathsf{sid}, \mathrm{S}, \mathsf{mid}_{l+1}, (0^{\kappa}, 0))$ and receives com_e and interprets it as TCmsg_1.
2. For all $i \in [\kappa]$ and $j \in [4\kappa]$, R sends $(\mathsf{Run}, \mathsf{sid}, \mathrm{S}, \mathsf{mid}_{1_l}, u_i^j)$ where $u_i^j \leftarrow \{0,1\}^{5\kappa}$ and receives $v_i^j = \mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}(u_i^j)$ (where $l \in [4\kappa^2]$ is an encoding of the pair (i,j)). If the token aborts the receiver aborts.
3. R chooses $\kappa - 1$ random bits $b_1, \ldots, b_{\kappa-1}$ and sets b_κ such that $b = \bigoplus_{i=1}^{\kappa} b_i$. For all $i \in [\kappa]$, it commits to b_i be setting $\mathsf{tcom}_{b_i} = (M_1^i, \ldots, M_\kappa^i)$. In particular, $\forall j \in [\kappa]$ as per Figure 2:

$$M_j^i = \begin{pmatrix} (\mathsf{Ext}(u_i^{4j-3}) \oplus \eta_{i,j}, v_i^{4j-3}) & (\mathsf{Ext}(u_i^{4j-1}) \oplus b_i \oplus \eta_{i,j}, v_i^{4j-1}) \\ (\mathsf{Ext}(u_i^{4j-2}) \oplus \eta_{i,j}, v_i^{4j-2}) & (\mathsf{Ext}(u_i^{4j}) \oplus b_i \oplus \eta_{i,j}, v_i^{4j}) \end{pmatrix}.$$

4. For all $i \in [\kappa]$, R sends tcom_{b_i}.

$\mathrm{S} \to \mathrm{R}$:

1. S chooses $z_1, \ldots, z_\kappa, \Delta \leftarrow \{0,1\}$, computes $w = \bigoplus_{i=1}^{\kappa} z_i$ and sends $s_0' = w \oplus s_0$, $s_1' = w \oplus \Delta \oplus s_1$ and $\{w_i^0 = z_i \oplus \mathsf{H}(s_i^0), w_i^1 = z_i \oplus \Delta \oplus \mathsf{H}(s_i^1)\}_{i \in [\kappa]}$ where (s_i^0, s_i^1) are computed by running the code of the token $\mathsf{TK}_S^{\mathsf{Com}}$ on input $\mathsf{tcom}_{b_i} || i$.
2. S sends $\mathsf{TCmsg}_3 = (e, \mathsf{decom}_e)$.
3. For all $i \in [\kappa]$, S creates a token TK_i by sending $(\mathsf{Create}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{l+1+i}, M_3)$ to R where M_3 implements the following functionality:
   - On input $(b_i, \mathsf{TCdecom}_{b_i})$:
     If $\mathsf{TCdecom}_{b_i}$ is verified correctly then output $(s_b^i, \mathsf{decom}_{s_b^i})$, else output $(\perp, \perp)$

Output Phase:

1. For all $i \in [\kappa]$, R sends $(\mathsf{Run}, \mathsf{sid}, \mathrm{S}, \mathsf{mid}_{l+1}, (\mathsf{com}_{b_i}, i))$ and receives $\mathsf{com}_{s_0^i}, \mathsf{com}_{s_0^i}$.
2. For all $i \in [\kappa]$, R sends $(\mathsf{Run}, \mathsf{sid}, \mathrm{S}, \mathsf{mid}_{l+1+i}, (b_i, \mathsf{TCdecom}_{b_i}))$ and receives $(s_b^i, \mathsf{decom}_{s_b^i})$. If the decommitments $\mathsf{decom}_{s_b^i}$ and decom_e are valid, R computes $\tilde{z}_i = \mathsf{H}(s_b^i) \oplus w_i^{b_i}$ and $s_b = \bigoplus_{i=1}^{\kappa} \tilde{z}_i \oplus s_b'$. If any of the tokens abort, the receiver sets $s_b = \perp$, where $\perp$ is a default value.

Figure 4: OT in the plain model

Proof overview. On a high-level, when the sender is corrupted the simulator rewinds the adversary in order to extract both S's inputs to the OT. Namely, in the first execution simulator $\mathcal{S}$ plays the role of the honest receiver with input 0 and learns the challenge e. It then rewinds the adversary and changes the receiver's commitments b_i's in a way that allows equivocating these commitments into both $b = 0$ and $b = 1$. Finally, $\mathcal{S}$ runs tokens $\{\mathsf{TK}_i\}_{i\in[\kappa]}$ twice by decommiting into two different sets of bit-vectors, which allows $\mathcal{S}$ to extract both inputs s_0 and s_1. The security proof follows by exploiting the trapdoor commitment property, which allows in the simulation to open the commitments of the receiver's input shares $\{b_i\}_{i\in[\kappa]}$ into two distinct bit-vectors that correspond to distinct bits. The indistinguishability argument asserts that the simulated and real views are statistically close, due to the statistical hiding property of the commitment scheme that we use within the Pass-Wee trapdoor commitment scheme.

On the other hand, when the receiver is corrupted the simulator extracts its input b based on the first message and the queries to the tokens. We note that extraction must be carried out carefully, as the receiver commits to each bit b_i using κ matrices and may commit to different bits within each set of matrices (specifically, there may be commitment for which the committed bit is not even well defined). Upon extracting b, the proof continues by considering a sequence of hybrids where we replace the hardcore bits for the positions $\{b_i \oplus 1\}_{i\in[\kappa]}$. Specifically, these are the positions in which the receiver cannot ask for decommitments and hence does not learn $\{s^i_{b_i\oplus 1}\}_{i\in[\kappa]}$. Our proof of indistinguishability relies on the list-decoding ability of the Goldreich-Levin hardcore predicate (cf. Theorem 2.9), that allows extraction of the input from an adversary that can guess the hardcore predicate on the input with probability significantly better than a half.

Proof: We consider each corruption case separately.

Simulating the corrupted S. Let $\mathcal{A}$ be a PPT adversary that corrupts S then we construct a simulator $\mathcal{S}$ as follows,

1. $\mathcal{S}$ invokes $\mathcal{A}$ on its input and a random string of the appropriate length.
2. $\mathcal{S}$ emulates the role of $\mathcal{F}^{\rm Stateless}_{\rm WRAP}$ where upon receiving from $\mathcal{A}$ the messages $\{(\mathsf{Create}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_l, M_1)\}_{l\in[4\kappa^2]}$ and $(\mathsf{Create}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{l+1}, M_2)$, $\mathcal{S}$ stores these codes.
3. Next, $\mathcal{S}$ emulates the role of the honest receiver using an input bit $b = 0$. If com_e is decommitted correctly, $\mathcal{S}$ stores this value and rewinds the adversary to the first message. Otherwise, $\mathcal{S}$ halts and outputs $\mathcal{A}$'s view thus far, sending $(\perp, \perp)$ to the ideal functionality.
4. $\mathcal{S}$ picks two random bit-vectors $(b_1, \ldots, b_\kappa)$ and $(b'_1, \ldots, b'_\kappa)$ such that $\bigoplus_{i=1}^{\kappa} b_i = 0$ and $\bigoplus_{i=1}^{\kappa} b'_i = 1$. Let $e = e_1, \ldots, e_\kappa$ denote the decommitment of com_e obained from the previous step. Then, for all $i, j \in [\kappa]$, $\mathcal{S}$ sends matrix M^j_i where the e_ith column is defined by

$$\begin{pmatrix} (\mathsf{Ext}(u_i^{4j-3}) \oplus \eta_{i,j}, & v_i^{4j-3}) \\ (\mathsf{Ext}(u_i^{4j-2}) \oplus \eta_{i,j}, & v_i^{4j-2}) \end{pmatrix}$$

whereas the $(1-e_i)$th column is set

$$\text{w.p. } \frac{1}{2} \text{ to } \begin{pmatrix} (\mathsf{Ext}(u_i^{4j-1}) \oplus \eta_{i,j}, & v_i^{4j-1}) \\ (\mathsf{Ext}(u_i^{4j}) \oplus 1 \oplus \eta_{i,j}, & v_i^{4j}) \end{pmatrix}, \text{ and}$$
$$\text{w.p. } \frac{1}{2} \text{ to } \begin{pmatrix} (\mathsf{Ext}(u_i^{4j-1}) \oplus 1 \oplus \eta_{i,j}, & v_i^{4j-1}) \\ (\mathsf{Ext}(u_i^{4j}) \oplus \eta_{i,j}, & v_i^{4j}) \end{pmatrix}.$$

5. Upon receiving the sender's message the simulator checks if com_e is decommitted correctly. Otherwise, $\mathcal{S}$ rewinds the adversary to before the first message was sent and returns to Step 4. In each rewinding $\mathcal{S}$ uses fresh randomness to generate the receiver's message. It repeatedly rewinds until the malicious sender successfully decommits e. If it tries to make more than $2^{\kappa/2}$ attempts, it simply halts outputting fail.

   Next, to extract s_0, it decommits to $b_1, \ldots, b_n$ (and to extract s_1, it decommits to $b'_1, \ldots, b'_n$). Recall that to reveal a commitment to a value b_i the simulator decommits that row of the matrix that adds up to b_i. Notice that by out construction, such a row always exists and is either the first row or the second row with the probability $1/2$. We remark here that the simulator $\mathcal{S}$ emulates the code of the actual Turing Machine incorporated in the token as opposed to running the token itself. Furthermore, each of the two extractions start with the Turing Machine in the same start (as opposed to running the machine in sequence). This is because the code in the malicious token can be stateful and rewinding it back to the start state prevents stateful behavior. More precisely, the simulator needs to proceed exactly as the honest receiver would in either case. If for any $b \in \{0,1\}$ extraction fails for s_b, then following the honest receiver's strategy the simulator sets s_b to the default value $\perp$.

6. Finally, $\mathcal{S}$ sends (s_0, s_1) to the trusted party that computes $\mathcal{F}_{\mathrm{OT}}$ and halts, outputting whatever $\mathcal{A}$ does.

The next claim establishes that the commitments made by the receiver are statistically hiding. We remark that this claim is analogous to Claim 20 from [GIS+10]. For completeness, we present it below.

Lemma 4.3. *For any $i \in [\kappa]$, let D_b denote the distribution obtained by sampling a random com_{b_i} with $b_i = b$. Then D_0 and D_1 are $2^{-\kappa+1}$-close.*

Proof: Informally, the proof follows from the fact that u_i has high min-entropy conditioned on v_i and therefore $(\mathsf{Ext}(u_i, h), h)$ hides u_i information theoretically as it is statistically close to the uniform distribution. More formally, consider a possibly maliciously generated token M_1 that incorporates an *arbitrary functionality* from 5κ bits to κ. It is possible to think of M_1 as a function even if the token is stateful since we only consider the min-entropy of the input with respect to the output when M_1 is invoked from the same state.

Let S_v denote the subset of $\{0,1\}^{5\kappa}$ that contains all $x \in \{0,1\}^{5k}$ such that $M_1(x) = v$. First, we claim that for a randomly chosen $x \leftarrow \{0,1\}^{5\kappa}$, $S_{M_1(x)}$ is of size at least $2^{3\kappa}$ with probability at least $1 - 2^{-\kappa}$. Towards proving this we calculate the number of x's for which $|S_{M_1(x)}| < 2^{3\kappa}$ and denote such an x by *bad*. Now, since there are at most 2^k possible values that M_1 may output, then the number of bad x's is:

$$\sum_{v:|S_v|<2^{3\kappa}} |S_v| < 2^{\kappa} \times 2^{3\kappa} = 2^{4\kappa}.$$

Therefore, the probability that a uniformly chosen x is bad is at most $2^{4k}/2^{5k} = 2^{-k}$. Let U and V denote random variables such that V is the response of M_1 on U. It now holds that

$$\Pr[u \leftarrow \{0,1\}^{5\kappa} : H_\infty(U|V = M_1(u)) \geq 3\kappa] > 1 - 2^{-\kappa}.$$

In other words, the min-entropy of U is at least 3κ with very high probability. Now, whenever this is the case, using the Leftover Hash Lemma (cf. Definition 2.7) with $\epsilon = 2^{-\kappa}$, $m = 1$ and $k = 3\kappa$ implies that $(\mathsf{Ext}(U, h), h)$ is $2^{-\kappa}$-close to the uniform distribution. Combining the facts that $\mathsf{com}_b = (\mathsf{Ext}(U, h) \oplus b, h, V)$ and that U has high min-entropy at least with probability $1 - 2^{-\kappa}$, we obtain that D_0 and D_1 are $2^{-\kappa} + 2^{-\kappa}$-close. $\square$

We now prove that the sender's view in both the simulated and real executions is computationally indistinguishable via a sequence of hybrid executions. More formally,

Lemma 4.4. *The following two ensembles are computationally indistinguishable,*

$$\left\{\mathbf{IDEAL}_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}(z),I}(\kappa,(s_0,s_1),b)\right\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*} \stackrel{\mathrm{c}}{\approx} \left\{\mathbf{REAL}^{\mathcal{F}^{\mathrm{Stateless}}_{\mathrm{WRAP}}}_{\Pi_{\mathrm{OT}},\mathcal{A}(z),I}(\kappa,(s_0,s_1),b)\right\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}$$

Proof: Roughly speaking, we prove that the joint output distribution of both the receiver and the sender is computationally indistinguishable. Our proof follows by a sequence of hybrid executions defined below. We denote by $\mathbf{Hybrid}^i_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_i(z),I}(\kappa,(s_0,s_1),b)$ the random variable that corresponds to the simulator's output in hybrid execution $\mathrm{H_i}$ when running against party $\mathcal{S}_i$ that plays the role of the receiver according to the specifications in this hybrid (where $\mathcal{S}_0$ refers to the honest real receiver).

Hybrid H_0: In the first hybrid, we consider a simulator S_0 that receives the real input b of the receiver and simply follows the protocol as the honest receiver would. Finally, it outputs the view of the adversary and the receiver's output as computed in the emulation. It follows from construction that the distribution of the output of the first hybrid is identical to the real execution.

Hybrid H_1: In this hybrid, the simulator $\mathcal{S}_1$ receives the real input of the receiver and proceeds as follows. It first interacts with the adversary with the actual receiver's input and checks if it successfully decommits e. If it does not, then the simulator simply outputs the view of the adversary and $\perp$ as the receiver's output. Otherwise, it proceeds to a rewinding phase. In this phase, it repeatedly rewinds the adversary to the first message and then samples a new first message by committing to b using fresh randomness. Specifically, $\mathcal{S}_1$ invokes token $\mathsf{TK}_{\mathsf{S}}^{\mathsf{PRF},l}$ each time on new random inputs u_i^j (for all $i \in [\kappa]$, $j \in [4\kappa]$ where l encodes (i,j)), and continues rewinding $\mathcal{A}$ until it obtains an interaction in which the adversary successfully decommits to e again. If the simulation makes more than $2^{\kappa/2}$ rewinding attempts, then it aborts.

We now argue that the view produced in this hybrid is statistically close to the view produced within the previous hybrid. Observe that if the simulation does not cut off after $2^{\kappa/2}$ attempts, then the view is identically distributed to the view in H_0. Therefore to show that the views are statistically close, it suffices to prove that the simulation aborts with negligible probability. Let p be the probability with which the adversary decommits e correctly when the receiver honestly generates a commitment to b. We consider two cases:

- If $p > \frac{\kappa}{2^{\kappa/2}}$, then the probability that the simulation takes more than $2^{\kappa/2}$ steps can be computed as $\left(1-\frac{\kappa}{2^{\kappa/2}}\right)^{2^{\kappa/2}} = e^{-\kappa}$ and which is negligible in κ.
- If $p < \frac{\kappa}{2^{\kappa/2}}$, then the probability that the simulation aborts is bounded by the probability that it proceeds to the rewinding phase which is at most p and hence negligible in κ.

It only remains to argue that the expected running time of the simulation is polynomial. We remark that this follows from Lemma 4.5 proven in the next hybrid by setting p_0 and p_b to p.

Hybrid H_2: In this hybrid, the simulator $\mathcal{S}_2$ proceeds identically to $\mathcal{S}_1$ with the exception that in the first run where the simulator looks for the decommitment of e, it follows the honest receiver's strategy with input 0 instead of its real input. Now, since each commitment is generated honestly, it follows from Lemma 4.3 using an union bound that the first message generated by the receiver with input b and input 0 are $4\kappa^2 2^{-\kappa-1}$-close. Moreover, since the only difference in the two hybrids is within the first message sent by the receiver in the first execution, the following distributions are statistically close.

- $\{\mathbf{Hybrid}^1_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_1(z),I}(\kappa,(s_0,s_1),b)\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}$

- $\{\mathbf{Hybrid}^2_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_2(z),I}(\kappa,(s_0,s_1),b)\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}$

It only remains to argue that the running time of the simulation is still polynomial.

Lemma 4.5. *The expected running time of $\mathcal{S}_2$ is polynomial-time and the probability that $\mathcal{S}_2$ aborts is negligible.*

Proof: Let p_0 be the probability that adversary successfully decommits to e in the main execution of hybrid H_2 and p_b be the probability that the adversary successfully decommits when the receiver's commitments are made to the real input b. Now, since the first message of the receiver when the commitment is made to 0 or b is $2^{-O(\kappa)}$-close, we have that $|p_0 - p_b| < 2^{-O(\kappa)}$.

Next, we prove that the expected number of times the simulator runs the execution is

$$(1-p_0)\times 1 + p_0 \times \min\left\{2^{\kappa/2},\frac{1}{p_b}\right\}.$$

We consider two cases and argue both regarding the running time and abort probability in each case.

- $p_0 > 2\kappa 2^{-\kappa/2}$: Since $|p_b - p_0| < 2^{-O(\kappa)}$, it follows that,

  $$p_b > p_0 - 2^{-O(\kappa)} = 2\kappa 2^{-\kappa/2} - 2^{-O(\kappa)} > \kappa 2^{-\kappa/2} = \frac{p_0}{2}$$

  Therefore, $p_0/p_b < 2$. Now, since $\min\left\{2^{\kappa/2},\frac{1}{p_b}\right\} = \frac{1}{p_b}$, the expected number of rewinding attempts is

  $$(1-p_0) + p_0 \times \frac{1}{p_b} < 3$$

  which is polynomial.

  Next, we argue regarding the abort probability. Specifically, the probability that the number of attempts exceeds $2^{\kappa/2}$ is given by

  $$(1-p_b)^{2^{\kappa/2}} < \left(1 - \frac{\kappa}{2^{\kappa/2}}\right)^{2^{\kappa/2}} = O(e^{-\kappa}).$$

  Therefore, the probability that the simulator aborts is negligible.

- $p_0 < 2\kappa 2^{-\kappa/2}$: Since $\min\left\{2^{\kappa/2},\frac{1}{p_b}\right\} = 2^{\kappa/2}$, the expected number of rewinding attempts is

  $$(1-p_0) + p_0 \times 2^{\kappa/2} < 1 + 2\kappa^2$$

  which is polynomial.

  The abort probability in this case is bounded by p_0 which is negligible.

$\square$

Hybrids $\mathrm{H}_{3,0}\ldots,\mathrm{H}_{3,\kappa}$: We define a collection of hybrid executions such that for every $i \in [\kappa]$ hybrid $\mathrm{H}_{3,\mathrm{i}}$ is defined as follows. Assume that $(b_1,\ldots,b_\kappa)$ correspond to the bit-vector for the real input of the receiver b. Then in $\mathrm{H}_{3,\mathrm{i}}$, the first i commitments are computed as in the simulation (i.e. equivocated using the trapdoor e), whereas the remaining $\kappa - i$ commitments are set as commitments of

$b_{i+1},\ldots,b_\kappa$ as in the real execution. Note that hybrid $\mathrm{H}_{3,0}$ is identical to hybrid H_2 and that the difference between every two consecutive hybrids $\mathrm{H}_{3,\mathrm{i}-1}$ and $\mathrm{H}_{3,\mathrm{i}}$ is regarding the way the ith commitment is computed, which is either a commitment to b_i computed honestly in the former hybrid, or equivocated using the trapdoor in the latter hybrid. Indistinguishability of $\mathrm{H}_{3,\mathrm{i}}$ and $\mathrm{H}_{3,\mathrm{i}-1}$ follows similarly to the indistinguishability argument of H_1 and H_2, as the only difference is in how the unopened commitments are generated. Therefore, we have the following lemma.

Claim 4.6. *For every $i \in [\kappa]$,*

$$\{\mathbf{Hybrid}^{1,i-1}_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_{1,i-1}(z),I}(\kappa,(s_0,s_1),b)\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}$$
$$\stackrel{\mathrm{s}}{\approx} \quad \{\mathbf{Hybrid}^{1,i}_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_{1,i}(z),I}(\kappa,(s_0,s_1),b)\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}$$

Note that the proof regarding the expected running time of the simulator is identical to the proof of Lemma 4.5.

IDEAL: In this hybrid, we consider the actual simulator. First, we observe that the view of the adversary output by $\mathcal{S}_{3,\kappa}$ in $\mathrm{H}_{3,\kappa}$ is independent of the receiver's real input b. This is because in $\mathrm{H}_{3,\kappa}$, all commitments are computed in an equivocation mode, where the real input b of the receiver is used only after the view of the adversary is generated. More precisely, only after $\mathcal{S}_{3,\kappa}$ obtains a second view on which the adversary successfully decommits to e, does it use the tokens to extract s_b by decommitting the equivocal commitments to $b_1,\ldots,b_n$ such that $\bigoplus_i b_i = b$. In fact, since in the rewinding phase all the commitments are equivocated, the b_i's themselves can also be sampled after the view of the adversary is generated.

Next, we observe that the actual simulator proceeds exactly as $\mathcal{S}_{3,\kappa}$ with the exception that it runs the tokens twice after the adversary's view is obtained and the rewinding phase is completed. Namely, it runs the token once with a vector of b_i's that add up to 0 in order to obtain s_0, then rewinds the tokens back to the original state and runs them another time with a vector of b'_i's that add up to 1 in order to extract s_1. (s_0,s_1) are then fed to the ideal functionality. Recall that $\mathcal{S}_{3,\kappa}$ on the other hand, runs the tokens only once for the actual receiver's input b. Now, since the view of the adversary in $\mathrm{H}_{3,\kappa}$ and **IDEAL** are identically distributed, it follows that the value extracted for s_b in $\mathrm{H}_{3,\kappa}$ is identically distributed to s_b in the ideal execution for both $b=0$ and $b=1$. Therefore, we can conclude that the output of the simulator in $\mathrm{H}_{3,\kappa}$ and the joint output of the simulator and honest receiver the ideal execution, are identically distributed.

Claim 4.7. *The following two ensembles are identical,*

$$\{\mathbf{Hybrid}^{3,\kappa}_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_{1,i}(z),I}(\kappa,(s_0,s_1),b)\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}$$
$$\approx \quad \left\{\mathbf{IDEAL}_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}(z),I}(\kappa,(s_0,s_1),b)\right\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}$$

■

Simulating the corrupted R. Let $\mathcal{A}$ be a PPT adversary that corrupts R then we construct a simulator $\mathcal{S}$ as follows,

1. $\mathcal{S}$ invokes $\mathcal{A}$ on its input and a random string of the appropriate length.

2. $\mathcal{S}$ emulates the role of $\mathcal{F}_{\rm WRAP}^{\rm Stateless}$ for tokens $\{\mathsf{TK}_{\rm S}^{\mathsf{PRF},l}\}_{l\in[4\kappa^2]}$ using truly random functions, where for each query $u \in \{0,1\}^{5\kappa}$ made by $\mathcal{A}$ to token $\mathsf{TK}_{\rm S}^{\mathsf{PRF},l}$, $\mathcal{S}$ returns a random v from $\{0,1\}^{\kappa}$. $\mathcal{S}$ further stores the pair (u,v) in the query/answer list of $\mathsf{TK}_{\rm S}^{\mathsf{PRF},l}$.

   Finally, $\mathcal{S}$ emulates $\mathcal{F}_{\rm WRAP}^{\rm Stateless}$ for $\mathsf{TK}_{\rm S}^{\mathsf{Com}}$ using a truly random function that maps elements from $\{0,1\}^{\kappa} \to \{0,1\}^{p(\kappa)}$.

3. Next, $\mathcal{S}$ splits the set of receiver's queries (tcom_b, i^*) to the token $\mathsf{TK}_{\rm S}^{\mathsf{Com}}$ (that were further part of the adversary's message), and adds them either to the "valid" set $\mathcal{I}_{\mathsf{Com}}$ or "invalid" set $\mathcal{J}_{\mathsf{Com}}$. More formally, let $T = q(\kappa)$ denote the number of times the token $\mathsf{TK}_{\rm S}^{\mathsf{Com}}$ is queried by R for some polynomial q. For each query (tcom_b, i^*), we say that the query is valid if and only if there exist values $\{(\beta_i^t, u_i^t, v_i^t)\}_{i\in[\kappa],t\in[4\kappa]}$ such that $\mathsf{tcom}_{b_i} = (M_1^i, \ldots, M_\kappa^i)$, $\forall i,j \in [\kappa]$,

$$M_j^i = \begin{pmatrix} \beta_i^{4j-3}, v_i^{4j-3} & \beta_i^{4j-1}, v_i^{4j-1} \\ \\ \beta_i^{4j-2}, v_i^{4j-2} & \beta_i^{4j}, v_i^{4j} \end{pmatrix}$$

   and, for every $i \in [\kappa], t \in [4\kappa]$, the query/answer pair (u_i^t, v_i^t) has already been recorded as a query to the corresponding PRF token. Next, for every valid query, the simulator tries to extract the committed value. This it done by first computing

$$\gamma_{00}^j = \beta_i^{4j-3} \oplus \mathsf{Ext}(u_i^{4j-3}) \qquad \gamma_{01}^j = \beta_i^{4j-1} \oplus \mathsf{Ext}(u_i^{4j-1})$$
$$\gamma_{10}^j = \beta_i^{4j-2} \oplus \mathsf{Ext}(u_i^{4j-2}) \qquad \gamma_{11}^j = \beta_i^{4j} \oplus \mathsf{Ext}(u_i^{4j}).$$

   Next it marks the indices j for which $\gamma_{00}^j = \gamma_{10}^j$ and $\gamma_{01}^j = \gamma_{11}^j$. Moreover, for the marked indices it computes $\gamma^j = \gamma_{00}^j \oplus \gamma_{01}^j$. If there are at least more than half the indices that are marked and are commitments to the same value, say γ then $(\mathsf{tcom}_b, i^*, \gamma)$ is added to $\mathcal{I}_{\mathsf{Com}}$. Otherwise $(\mathsf{tcom}_b, i^*, \perp)$ is added to $\mathcal{J}_{\mathsf{Com}}$.

   Next, $\mathcal{S}$ computes $b = \bigoplus_{i=1}^{\kappa} b_i$ and sends b to the trusted party that computes $\mathcal{F}_{\rm OT}$. Upon receiving s_b, $\mathcal{S}$ picks a random $s_{b\oplus 1}$ from the appropriate domain and completes the execution by playing the role of the honest sender on these two inputs.

We now prove that the receiver's view in both the simulated and real executions is computationally indistinguishable via a sequence of hybrid executions. More formally,

Lemma 4.8. *The following two ensembles are computationally indistinguishable,*

$$\left\{\mathbf{IDEAL}_{\mathcal{F}_{\rm OT},\mathcal{S}(z),I}(\kappa,(s_0,s_1),b)\right\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*} \stackrel{\rm c}{\approx} \left\{\mathbf{REAL}_{\Pi,\mathcal{A}(z),I}^{\mathcal{F}_{\rm WRAP}^{\rm Stateless}}(\kappa,(s_0,s_1),b)\right\}_{\kappa\in\mathbb{N},s_0,s_1,,bz\in\{0,1\}^*}$$

Proof: Roughly speaking, we prove that the join output distribution of both the receiver and the sender is computationally indistinguishable. Now, since only the receiver (which is the corrupted party) has an input, the proof boils down to proving that the receiver's view is indistinguishable in both executions. Our proof follows by a sequence of hybrid executions defined below. We denote by $\mathbf{Hybrid}^i_{\mathcal{F}_{\rm OT},\mathcal{S}_i(z),I}(\kappa,(s_0,s_1),b)$ the random variable that corresponds to the adversary's view in hybrid execution $\rm H_i$ when running against party $\mathcal{S}_i$ that plays the role of the sender according to the specifications in this hybrid (where $\mathcal{S}_0$ refers to the honest real sender).

Hybrid H_0: The first hybrid execution is the real execution.

Hybrids $\mathrm{H}_{1,0}\ldots,\mathrm{H}_{1,4\kappa^2}$: We define a collection of hybrid executions such that for every $l \in [4\kappa^2]$ hybrid $\mathrm{H}_{1,l}$ is defined as follows. We modify the code of token $\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}$ by replacing the function PRF_{γ_l} with a truly random function f_l. In particular, given a query u the token responds with a randomly chosen κ bit string v, rather than running the original code of M_1. We maintain a list of $\mathcal{A}$'s queries and responses so that repeated queries will be consistently answered. In addition, the code of token TK_i is modified so that it verifies the decommitment against the random functions f_l as opposed to the PRF functions previously embedded in $\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}$. It is simple to verify that the adversary's view in every two consecutive hybrid executions is computationally indistinguishable due to the security of the pseudorandom function PRF_{γ_l}. Moreover, since the PRF key is hidden from the receiver, it follows from the pseudorandomness property that the views in every two consecutive hybrid executions are computationally indistinguishable. More formally, we have the following lemma.

Claim 4.9. *For every $l \in [4\kappa^2]$,*

$$\begin{aligned}&\{\mathbf{Hybrid}^{1,l-1}_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_0(z),I}(\kappa,(s_0,s_1),b)\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}\\ \stackrel{\mathrm{c}}{\approx}\quad&\{\mathbf{Hybrid}^{1,l}_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_1(z),I}(\kappa,(s_0,s_1),b)\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}\end{aligned}$$

Hybrid H_2: Similarly, we consider a hybrid execution for which the code of token $\mathsf{TK}_{\mathrm{S}}^{\mathsf{Com}}$ is modified so that it makes use of a truly random function f' rather than a pseudorandom function $\mathsf{PRF}_{\gamma'}$. Just as in the previous hybrid, we have the following Lemma.

Claim 4.10.

$$\begin{aligned}&\{\mathbf{Hybrid}^{1}_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_1(z),I}(\kappa,(s_0,s_1),b)\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}\\ \stackrel{\mathrm{c}}{\approx}\quad&\{\mathbf{Hybrid}^{2}_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_2(z),I}(\kappa,(s_0,s_1),b)\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}\end{aligned}$$

Hybrids $\mathrm{H}_{3,0}\ldots,\mathrm{H}_{3,4\kappa^2}$: This sequence of hybrids executions is identical to hybrid H_2 except that here we ensure that no two queries made by $\mathcal{A}$ to the token $\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}$ have the same response. Specifically, in case of a collision simulator $\mathcal{S}_{3,l}$ aborts.

Claim 4.11. *For every $l \in [4\kappa^2]$,*

$$\begin{aligned}&\{\mathbf{Hybrid}^{3,l-1}_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_2(z),I}(\kappa,(s_0,s_1),b)\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}\\ \stackrel{\mathrm{s}}{\approx}\quad&\{\mathbf{Hybrid}^{3,l}_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_3(z),I}(\kappa,(s_0,s_1),b)\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}\end{aligned}$$

Proof: As we replaced PRF functions to truly random functions, we have that the probability the simulation aborts in $\mathrm{H}_{3,l}$ is at most the probability of finding a collision for a random function. To prove statistical indistinguishability it suffices to show that this probability is negligible. More formally, if the adversary makes a total of Q queries to both tokens, then the probability that any pair of queries yields a collision can be bounded by $\binom{Q}{2}2^{-\ell}$ where ℓ is the minimum length of the outputs of all random functions. In our case this is κ and hence the probability that the simulator aborts in every hybrid is negligible. $\square$

Hybrid H_4: In this hybrid execution, simulator $\mathcal{S}_4$ plays the role of the sender as in hybrid H_3 except that it extracts the adversary's input bit b as carried out in the simulation by $\mathcal{S}$. First, we observe that for any $i \in [\kappa]$ and $t \in [4\kappa]$, the probability that the receiver reveals a valid pre-image u_i^t for v_i^t for which there does not exists a query/answer pair (u_i^t, v_i^t) collected by the simulator is exponentially small since we rely on truly random functions in this hybrid. Therefore, except with negligible probability, the receiver will be able to decommit only to $\gamma_{00}^j, \gamma_{01}^j, \gamma_{10}^j, \gamma_{11}^j$ as extracted by the simulator. Consequently, using the soundness of the Pass-Wee trapdoor commitment scheme, it follows that the receiver can only decommit to b_i and b as extracted by the simulator. Therefore, we can conclude that the probability that a malicious receiver can equivocate the commitment tcom_{b_i} is negligible. The above does not make any difference to the receiver's view which implies that,

Claim 4.12.

$$\{\mathbf{Hybrid}^3_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_3(z),I}(\kappa,(s_0,s_1),b)\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}$$
$$\approx \quad \{\mathbf{Hybrid}^4_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_4(z),I}(\kappa,(s_0,s_1),b)\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}$$

Moreover, recall that extraction is straight-line, thus the simulator still runs in strict polynomial-time.

Hybrids $\mathrm{H}_{5,0}, \widetilde{\mathrm{H}}_{5,0}, \ldots, \mathrm{H}_{5,\kappa}, \widetilde{\mathrm{H}}_{5,\kappa}$: Let tcom_{b_i} be the ith commitment sent to S in the first message. Then $\mathrm{H}_{5,\mathrm{i}}$ proceeds identically to $\widetilde{\mathrm{H}}_{5,i-1}$, whereas $\widetilde{\mathrm{H}}_{5,i}$ proceeds identically to $\mathrm{H}_{5,\mathrm{i}}$, with the following exceptions:

- If there exists a tuple $(\mathsf{tcom}_{b_i}, i, \gamma)$ in $\mathcal{I}_{\mathsf{Com}}$, then in experiment $\mathrm{H}_{5,\mathrm{i}}$, $\mathsf{H}(s^i_{\gamma\oplus 1})$ is replaced by a random bit in the second message fed to the adversary.
- If there exists a tuple $(\mathsf{tcom}_{b_i}, i, \perp)$ in $\mathcal{J}_{\mathsf{Com}}$, then in experiment $\mathrm{H}_{5,\mathrm{i}}$, $\mathsf{H}(s^i_0)$ is replaced by a random bit in the second message fed to the adversary.
- If there exists a tuple $(\mathsf{tcom}_{b_i}, i, \perp)$ in $\mathcal{J}_{\mathsf{Com}}$, then in experiment $\widetilde{\mathrm{H}}_{5,i}$, $\mathsf{H}(s^i_1)$ is replaced by a random bit in the second message fed to the adversary.

Note that hybrid $\mathrm{H}_{5,0}$ is identical to hybrid H_4 and that the difference between every pair of consecutive hybrids $\mathrm{H}_{5,\mathrm{i}-1}$ and $\mathrm{H}_{5,\mathrm{i}}$ is with respect to $\mathsf{H}(s^i_{b_i\oplus 1})$ in case $i \in [|\mathcal{I}_{\mathsf{Com}}|]$ or $(\mathsf{H}(s^i_0), \mathsf{H}(s^i_1))$ in case $i \in [|\mathcal{J}_{\mathsf{Com}}|]$, that are replaced with a random bit in $\mathrm{H}_{5,\mathrm{i}}$. We now prove the following.

Claim 4.13. *For every* $i \in [\kappa]$,

$$\{\widetilde{\mathbf{Hybrid}}^{5,i-1}_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_{5,i-1}(z),I}(\kappa,(s_0,s_1),b)\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}$$
$$\stackrel{\mathrm{c}}{\approx} \quad \{\mathbf{Hybrid}^{5,i}_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_{5,i}(z),I}(\kappa,(s_0,s_1),b)\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}$$

and

$$\{\mathbf{Hybrid}^{5,i}_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_{5,i-1}(z),I}(\kappa,(s_0,s_1),b)\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}$$
$$\stackrel{\mathrm{c}}{\approx} \quad \{\widetilde{\mathbf{Hybrid}}^{5,i}_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_{5,i}(z),I}(\kappa,(s_0,s_1),b)\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}$$

Proof: Intuitively, the indistinguishability of any pair of hybrids follows from the computational hiding property of the commitment scheme Com and the binding property of tcom_{b_i}. Assume for contradiction, that there exists $i \in [\kappa]$ for which hybrids $\widetilde{\mathbf{Hybrid}}^{5,i-1}$ and $\mathbf{Hybrid}^{5,i}$ are distinguishable by a PPT distinguisher D with probability ε.

If there exists a tuple $(\mathsf{tcom}_{b_i}, i, \gamma) \in \mathcal{I}_{\mathsf{Com}}$ then define $b^* = 1 \oplus \gamma$, otherwise define $b^* = 0$. Then, it follows that the only difference between hybrids $\widetilde{\mathrm{H}}_{5,i-1}$ and $\mathrm{H}_{5,\mathrm{i}}$ is that $\mathsf{H}(s^i_{b^*})$ is computed correctly in $\widetilde{\mathrm{H}}_{5,i-1}$ while replaced with a random bit in $\mathrm{H}_{5,\mathrm{i}}$. Next, we show how to build an adversary $\mathcal{A}_{\mathsf{Com}}$ that on input a commitment $\mathsf{Com}(s)$ identifies $\mathsf{H}(s)$ with probability non-negligibly better than $\frac{1}{2} + \frac{\varepsilon}{2}$. Then using the Goldreich-Levin Theorem (Theorem 2.9), it follows that we can extract value $s^i_{b^*}$ and this violates the hiding property of the commitment scheme Com.

More formally, consider $\mathcal{A}_{\mathsf{Com}}$ that receives as input a commitment to a randomly chosen string s, namely $\mathsf{Com}(s)$. $\mathcal{A}_{\mathsf{Com}}$ internally incorporates the adversary $\mathcal{A}_{\mathsf{Com}}$ and emulates the experiment $\widetilde{\mathrm{H}}_{5,i}$ with the exception that in place of $\mathsf{Com}(s^i_{b^*})$, $\mathcal{A}_{\mathsf{Com}}$ instead feeds $\mathsf{Com}(s)$ and replaces $\mathsf{H}(s^i_{b^*})$ with a uniformly chosen bit, say $\tilde{b}$. Finally, it feeds the output of the hybrid experiment conducted internally, namely, the view of the adversary to D, and computes an output based on D's output g as follows:

- If $g = 1$, then $\mathcal{A}_{\mathsf{Com}}$ outputs the value for $\tilde{b}$ as the prediction for $\mathsf{H}(s)$, and outputs $1 - \tilde{b}$ otherwise.

Denote by $\overline{\mathrm{H}}_{5,i}$ the experiment that proceeds identically to $\widetilde{\mathrm{H}}_{5,i}$ with the exception that, in place of $\mathsf{H}(s^i_{b^*})$ we feed $1 \oplus \mathsf{H}(s^i_{b^*})$, namely the complement of the value of the hardcore predicate. Let $\overline{\mathbf{Hybrid}}^{5,i}$ denote the distribution of the view of the adversary in this hybrid. It now follows that

$$\begin{aligned}
\varepsilon &< \Big| \Pr[(v, s_b) \leftarrow \widetilde{\mathbf{Hybrid}}^{5,i} : D(v) = 1] - \Pr[(v, s_b) \leftarrow \mathbf{Hybrid}^{5,i} : D(v) = 1] \Big| \\
&= \Big| \Pr[(v, s_b) \leftarrow \widetilde{\mathbf{Hybrid}}^{5,i} : D(v) = 1] \\
&\qquad - \frac{1}{2}\Big(\Pr[(v, s_b) \leftarrow \widetilde{\mathbf{Hybrid}}^{5,i} : D(v) = 1] + \Pr[(v, s_b) \leftarrow \overline{\mathbf{Hybrid}}^{5,i} : D(v) = 1] \Big) \Big| \\
&= \frac{1}{2} \Big| \Pr[(v, s_b) \leftarrow \widetilde{\mathbf{Hybrid}}^{5,i} : D(v) = 1] - \Pr[(v, s_b) \leftarrow \overline{\mathbf{Hybrid}}^{5,i} : D(v) = 1] \Big|.
\end{aligned}$$

Without loss of generality we can assume that,[6]

$$\frac{1}{2}\Big(\Pr[(v, s_b) \leftarrow \widetilde{\mathbf{Hybrid}}^{5,i} : D(v) = 1] - \Pr[(v, s_b) \leftarrow \overline{\mathbf{Hybrid}}^{5,i} : D(v) = 1] \Big) > \varepsilon$$

Therefore,

$$\begin{aligned}
&\frac{1}{2}\Big(\Pr[(v, s_b) \leftarrow \widetilde{\mathbf{Hybrid}}^{5,i} : D(v) = 1] - (1 - \Pr[(v, s_b) \leftarrow \overline{\mathbf{Hybrid}}^{5,i} : D(v) = 0)] \Big) > \varepsilon \\
&\textit{i.e.}, \frac{1}{2} \Pr[(v, s_b) \leftarrow \widetilde{\mathbf{Hybrid}}^{5,i} : D(v) = 1] + \frac{1}{2} \Pr[(v, s_b) \leftarrow \overline{\mathbf{Hybrid}}^{5,i} : D(v) = 0)] > \frac{1}{2} + \varepsilon \\
&\qquad\qquad \textit{i.e.}, \Pr[\beta \leftarrow \{0,1\} : (v, s) \leftarrow H^b : D(v) = b] > \frac{1}{2} + \varepsilon
\end{aligned}$$

where $H^0 = \overline{\mathbf{Hybrid}}^{5,i}$ and $H^1 = \widetilde{\mathbf{Hybrid}}^{5,i}$. We now observe that sampling from H^b where b is uniformly chosen is equivalent to sampling from $\mathrm{H}_{5,\mathrm{i}}$. Therefore, since $\mathcal{A}_{\mathsf{Com}}$ internally emulates

[6] Otherwise, we can replace D with another distinguisher that flips D's output.

$\mathrm{H}_{5,\mathrm{i}}$ by selecting $\tilde{b}$ at random and the distinguisher identifies precisely if this bit $\tilde{b}$ came from H^0 or H^1 correctly, we can conclude that $\tilde{b}$ is the value of the hardcore bit when it comes from H^0 and the complement of $\tilde{b}$ when it comes from H^1. Therefore, $\mathcal{A}_{\mathsf{Com}}$ guesses $\mathsf{H}(s)$ correctly with probability $\frac{1}{2}+\varepsilon$. Using the list-decoding algorithm of Goldreich-Leving hardcore-predicate (cf. Theorem 2.9), it follows the such an adversary can be used to extract s thereby contradicting the computational hiding property of the Com scheme.

We remark that proving indistinguishability of $\mathbf{Hybrid}^{5,i}$ and $\widetilde{\mathbf{Hybrid}}^{5,i}$ follows analogously and this concludes the proof of the Lemma. □

Hybrids H_6: In this hybrid execution simulator $\mathcal{S}_5$ does not know the sender's inputs (s_0, s_1), but rather communicates with a trusted party that computes $\mathcal{F}_{\mathrm{OT}}$. $\mathcal{S}_6$ behaves exactly as $\mathcal{S}_{5,\kappa}$ except that when extracting the bit b it sends it to the trusted party, which returns s_b. Moreover, $\mathcal{S}_6$ uses a random value for $s_{b\oplus 1}$. We argue that hybrids $\mathcal{S}_{5,\kappa}$ and $\mathcal{S}_6$ are identically distributed as the set $\{w^i_{b\oplus 1}\}_{i\in[\kappa]}$ is independent of $\{s^i_{b_i\oplus 1}\}_{i\in[\kappa]}$.

Claim 4.14.

$$\begin{aligned}&\{\mathbf{Hybrid}^{5,\kappa}_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_{5,\kappa}(z),I}(\kappa,(s_0,s_1),b)\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}\\ \approx\ &\{\mathbf{Hybrid}^{5}_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_{6}(z),I}(\kappa,(s_0,s_1),b)\}_{\kappa\in\mathbb{N},s_0,s_1,b,z\in\{0,1\}^*}\end{aligned}$$

Proof: Following from the fact that $\mathsf{H}(s^i_{b_i\oplus 1})$ is replaced with a random bit for all $i\in[\kappa]$, it must hold that the values $w_i^{1\oplus b_i}$ are random as well as these values are masked using random independent bits instead of the set $\{\mathsf{H}(s^i_{b_i\oplus 1})\}_{i\in[\kappa]}$. As a result, these values contribute to a random value $s_{b\oplus 1}$. In addition, we claim that the adversary can only learn s_b where b is the bit extracted by $\mathcal{S}_6$. This is because $\mathcal{A}$ can only invoke token TK_i on the commitment com_{b_i}, for which is can only open in a single specific way. □

Finally, note that hybrid H_6 is identical to the simulation described above, which concludes the proof.

■

■

4.1.3 Relaxing to One-Way Functions

Recall that in our construction we rely on one-way permutations for a non-interactive perfectly binding commitment scheme. To relax this assumption to one-way functions, we make the following modification to the $\mathsf{TK}^{\mathsf{Com}}_{\mathsf{S}}$ token. More formally, let $\widehat{\mathsf{Com}}$ denote the Naor two-round commitment scheme [Nao91] that is based on one-way functions, where, given a random tape r and the first message from the receiver R_0, denote by $\widehat{\mathsf{Com}}(m;r,R_0)$ the sender's message. Then, $\mathsf{TK}^{\mathsf{Com}}_{\mathsf{S}}$ incorporates the following functionality, that given input $(\mathsf{tcom}_{b_i}, i, R_0, R_1)$ proceeds as follows:

- If $i=0$: compute $V=\mathsf{PRF}'_{\gamma'}(0^\kappa||R_0||R_1)$, parse V as $e||r$ and output $\mathsf{com}_e\leftarrow\widehat{\mathsf{Com}}(e;r,R_0)$.
- Otherwise: compute $V=\mathsf{PRF}'_{\gamma'}(\mathsf{tcom}_{b_i}||i||R_0||R_1)$, parse V as $s^i_0||s^i_1||r_0||r_1$, compute $\mathsf{com}_{s^i_b}\leftarrow\widehat{\mathsf{Com}}(s^i_b;r_b,R_b)$ for $b=\{0,1\}$, and output $\mathsf{com}_{s^i_0},\mathsf{com}_{s^i_1}$.

Finally, along with the first message, the receiver also produces R_0, R_1 for each of the commitments made so that the sender can reconstruct the values being committed to. Two issues arise when proving security using the modified token.

1. The first messages for the Naor commitment used when querying the token might not be the same as the one produced in the first message by the receiver. In this case, by the pseudorandomness property of the PRF it follows that the values for these commitments computed by the sender will be independent of the commitments received from the token by the receiver. Hence, the statistically-hiding property of the values used by the sender will not be violated.

2. The binding property of the commitment scheme is only statistical (as opposed to perfect). This will affect the failure probability of the simulator when extracting the sender's input only by a negligible amount and can be bounded overall by incorporating a union bound argument.

4.1.4 On Concurrency and Reusability

Following the work of Choi et al. [CKS$^+$14], we investigate the possibility of exchanging tokens just once and (re-)using the tokens for an *unbounded* number of oblivious transfers. In this section we argue how to extend our protocol to achieve *reusability* and a restricted form of concurrency. More precisely, we show how the same set of tokens can be used to execute an arbitrary number of oblivious transfers where the tokens can be executed either sequentially and/or in parallel. To formalize this, Choi et al. [CKS$^+$14] defined the $\mathcal{F}_{\mathrm{multi-OT}}$ functionality which obtains from both the sender and receiver vectors of inputs, to be used in the parallel sessions of the OT. Suppose that there are at most $q_2(n)$ sequential sessions identified with sequence number sid and in each sequential session there are at most $q_1(n)$ parallel sessions identified by session identifier ssid. Towards achieving this, we show that it is possible to exchange all tokens at the beginning of the protocol. Recall that the sender sends two sets of tokens, one as part of the token exchange phase, and the other as part of the second message whose code depends on the first message of the receiver.

To reduce the number of PRF tokens we must ensure that the sender cannot create stateful tokens that encode information about the PRF queries. Indeed, such as attack can be carried out once the PRF tokens are being reused. For instance, the token can split two queries into five strings and return them as the responses for 10 subsequent queries. Since the output of the PRF queries are relayed to sender, the sender will be able to recover the first query and this violates the min-entropy argument required to prove that the commitments are statistically hiding. On a high-level, we get around this by requiring the sender send 2κ (identical) tokens that compute the same PRF functionality. Then, for each query, the receiver picks a subset of κ tokens to be queried and verifies that it received the same outcome from any token in that subset. In case any one of the tokens abort or the outcomes are not identical, the receiver aborts. Security is then shown by proving that for any query u, the high min entropy of its outcome v is maintained even conditioned on subsequent queries. Namely, we extend the proof of Lemma 4.3, and prove that with overwhelming probability, there are sufficiently many preimages for v (namely $2^{2\kappa+1}$) even conditioned on all subsequent queries. Intuitively, this follows because with overwhelming probability any two token queries are associated with two distinct tokens subsets.[7] The rest of the proof against a corrupted sender follows similarly to the proof of Theorem 4.1.

Lemma 4.15 (Lemma 4.3 restated)**.** *For any $i \in [\kappa]$, let D_b denote the distribution obtained by sampling a random com_{b_i} with $b_i = b$. Then D_0 and D_1 are $2^{-\kappa+1}$-close.*

[7] We wish to acknowledge that our approach is inspired by the communication exchanged with the authors of the [GIS$^+$10] paper while communicating their fix to the issue we found in their paper.

Proof: We continue with the formal proof of the extended Lemma 4.3. Specifically, for each of the 4κ tokens sent by the receiver, instead of sending independent tokens to implement the PRF, we will require the sender to send 2κ tokens implementing the same PRF functionality in order to implement a simple cut-and-choose strategy. In all the sender sends $4\kappa \times 2\kappa = 8\kappa^2$ tokens.

For each of the 4κ commitments, we modify the receiver's algorithm as follows: Consider the PRF to be used for a particular commitment. Let the 2κ tokens (allegedly) implementing a particular PRF to be used for this commitment be $\mathsf{TK}_1^0, \mathsf{TK}_1^1, \ldots, \mathsf{TK}_\kappa^0, \mathsf{TK}_\kappa^1$. Then the receiver picks a uniformly sampled u and proceeds as follows:

1. Pick κ random bits $h_1, \ldots, h_k$.
2. For every $i \in [\kappa]$, run $\mathsf{TK}_i^{h_i}$ on input u. If all tokens do not output the same value the receiver halts.
3. Commit by running an extractor on u as described in our protocol.

For a given malicious sender S^*, let there be at most $T = \mathrm{poly}(k)$ sequential sessions in all and let $(U_1, V_1), \ldots, (U_T, V_T)$ be the random variables representing the (query/answer) pair for this PRF in the T sessions, where V_i (and subsequent values) is set to $\perp$ if the token aborts or all tokens do not give consistent answers on query U_i. In Lemma 4.3, it suffices to prove that for any i, with high probability the min-entropy of U_i conditioned on $(U_1, \ldots, U_{i-1}, U_{i+1}, \ldots, U_T, V_1, , V_T)$ is at least $2\kappa + 1$. Since U_j for $j \neq i$ are each independently sampled by the receiver, we can fix their values to arbitrary strings. Therefore, it suffices to show that for any sequence of values $u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_T$ with high probability the min-entropy of U_i conditioned on ($V_1, \ldots, V_T$ and $U_j = u_j$ for $j \neq i$) is at least $2\kappa + 1$.

Denote by the event Good if there exist no two queries picked by the receiver for which the same values for $h_1, \ldots, h_k$ in Step 1 are chosen. Using a union bound, except with probability $\binom{T}{2}\frac{1}{2^\kappa}$, and therefore with negligible probability, Good holds. Since Good holds except with negligible probability, it suffices to prove our claim when Good holds. Let u^* be such that some token in session i returns v_i where the input sequence is $u_1, \ldots, u_{i-1}, u^*$. Then the two sequences $u_1, \ldots, u_{i-1}, u_i, \ldots, u_T$ and $u_1, \ldots, u_{i-1}, u^*, \ldots, u_T$ will result in the same sequence of responses (until either some token aborts or some token gives an inconsistent answer). This is because, since Good holds, for every $j > i$, u_j is queried on at least one token (among the 2κ tokens implementing the PRF) that was not queried in session i and therefore will behave independently from the query made in session i. In particular this means that a consistent response for u_j for any $j > i$ must be identical for sequences $u_1, \ldots, u_{i-1}, u_i, \ldots, u_T$ and $u_1, \ldots, u_{i-1}, u^*, \ldots, u_T$ or must result in a premature abort in one of the sequences. It therefore suffices to show that there are sufficiently many u^* values for which the same sequence of responses are obtained and if the receiver aborts, it aborts in the same session for the sequences corresponding to u_i and u^*.

Following Lemma 4.3 we have that except with probability $1/2^\kappa$, there is a set S_{V_i} of size at least $2^{3\kappa}$ possible values for u such that on input sequence beginning with $u_1, \ldots, u_{i-1}, u$, the token will respond with V_i in session i. Let $A^i, A^{i+1}, \ldots, A^T$ be subsets of S_{V_i} such that $u \in A^j$ if on input sequence $u_1, \ldots, u_{i-1}, u^*, \ldots, u_T$, the receiver aborts during session j. Let $A^{T+1} \subseteq S_{V_i}$ be those u on which the receiver does not abort at all. Note that the A^j's form a partition of S_{V_i}. To argue the claim, it suffices to show that with high probability U_i belongs to A^j such that $|A^j| > 2^{2k+1}$. More formally, we bound the number of "bad" u's, namely $u \in S_{V_i}$ such that u belongs to A^j and $|A^j| < 2^{2k+1}$. Since there are at most $T + 1$ sets in all, the number of such queries is at most $2^{2k+1} \times (T + 1)$. Furthermore, one these queries will be chosen with probability at most $2^{2\kappa+1} \times T/|S_{V_j}|$ which is at most $T/2^{\kappa-1}$ since $|S_{V_j}| > 2^{3\kappa}$. This is negligible. Therefore, it holds that, except with negligible probability, U_i belongs to A^j such that $|A^j| > 2^{2k+1}$ and this concludes the proof of the Lemma. ■

Next, to make the latter set of commitment tokens independent of the first message of the protocol, we will rely on *digital signatures*. A similar approach was pursued in the work of [CKS+14] where digital signatures, which require an additional property of ***unique*** signatures, are employed. Recall that a signature scheme $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ is said to be *unique* if for every verification key vk and every message m, there exists only one signature σ for which $\mathsf{Ver}_{\mathsf{vk}}(m, \sigma) = 1$. Such signature schemes can be constructed based on specific number theoretic assumptions [DY05]. In this paper we take a different approach and rely only on one-way functions. For simplicity we provide a construction based on non-interactive perfectly binding commitment schemes that, in turn, can be based on one-way permutations. By relying on techniques described in Section 4.1.3 we can relax this assumption to one-way functions.

Our main idea is to rely on an instantiation of Lamport's one-time signature scheme [Lam79] with Com (instead of one-way functions), which additionally has the uniqueness property as in [CKS+14]. In the following, we consider the setting where a sender S and a receiver R engage in several oblivious transfer instantiations, both in parallel and in sequentially. As mentioned above, we will identify every session by two quantifiers $(\mathsf{sid}, \mathsf{ssid})$ where sid is the sequential session identifier and ssid is the parallel session identifier. On a high-level, for every OT instance identified by $(\mathsf{sid}, \mathsf{ssid}, i)$ we generate a one-time signature/verification key pair by applying a PRF on $\tau = \mathsf{sid}||\mathsf{ssid}||\mathsf{tcom}||i$ where tcom is the commitment to the receiver's input for that instance generated using the 4κ sets of tokens specified above. Then the receiver is allowed to query the OTM token TK_i only if it possesses a valid signature corresponding to its commitment tcom for that instance. Now, since the signatures are unique, the signatures themselves do not carry any additional information. In more detail, we modify the tokens as follows: As in the previous protocol Π_{OT}, we additionally rely on a non-interactive perfectly binding commitment scheme Com and PRFs F, F'.

1. $\{\mathsf{TK}^0_{1_j}, \mathsf{TK}^1_{1_j}, \ldots, \mathsf{TK}^0_{\kappa_j}, \mathsf{TK}^1_{\kappa_j}\}_{j\in[4\kappa]}$: S chooses $8\kappa^2$ random PRF keys $\{\gamma_l\}_{l\in[8\kappa^2]}$ for family F. Let $\mathsf{PRF}_{\gamma_l} : \{0,1\}^{5\kappa} \to \{0,1\}^{\kappa}$ denote the pseudorandom function. S creates these tokens by sending $\{(\mathsf{Create}, \mathsf{ssid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_l, M_1)\}_{l\in[8\kappa^2]}$ to R where M_1 is the machine that on input (sid, x), outputs $\mathsf{PRF}_{\gamma_l}(\mathsf{sid}||x)$.

2. $\mathsf{TK}^{\mathsf{Com}}_{\mathrm{S}}$: S chooses a random PRF' key γ' for family F'. Let $\mathsf{PRF}'_{\gamma'} : \{0,1\}^{\kappa} \to \{0,1\}^{p(\kappa)}$ denote the pseudorandom function. S creates token $\mathsf{TK}^{\mathsf{Com}}_{\mathrm{S}}$ by sending $(\mathsf{Create}, \mathsf{ssid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{l+1}, M_2)$ to R where M_2 is the machine that on input $(\mathsf{sid}, \mathsf{ssid}, \mathsf{tcom}_{b_i}, i)$ does the following:

   - If $i = 0$: Compute $V = \mathsf{PRF}'_{\gamma'}(\mathsf{sid}||0^{\kappa})$. Then, parse V as $e||r$ and output $\mathsf{com}_e \leftarrow \mathsf{Com}(e; r)$.
   - Otherwise: Let $\tau = \mathsf{sid}||\mathsf{ssid}||\mathsf{tcom}_{b_i}||i$. Compute $V = \mathsf{PRF}'_{\gamma'}(\tau)$. Then output

$$(\mathsf{com}_{s^i_0}, \mathsf{com}_{s^i_1}), \mathsf{vk}_\tau = \begin{pmatrix} \mathsf{Com}(x^0_1; r^0_1) & \cdots & \mathsf{Com}(x^0_{\kappa+1}; r^0_{\kappa+1}) \\ \mathsf{Com}(x^0_1, r^0_1) & \cdots & \mathsf{Com}(x^0_{\kappa+1};, r^1_{\kappa+1}) \end{pmatrix}$$

   where V is parsed as $s^i_0||s^i_1||r_0||r_1||(x^b_\ell)_{b\in\{0,1\},\ell\in[\kappa+1]}||(r^b_\ell)_{b\in\{0,1\},\ell\in[\kappa+1]}$ and $\mathsf{com}_{s^i_b} \leftarrow \mathsf{Com}(s^i_b; r_b)$ for $b = \{0,1\}$.

3. TK_i: For all $i \in [\kappa]$, S creates a token TK_i by sending $(\mathsf{Create}, \mathsf{ssid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{l+1+i}, M_3)$ to R where M_3 is defined as follows given input $(\sigma, \mathsf{sid}, \mathsf{ssid}, i, b_i, \mathsf{tcom}_{b_i}, \mathsf{TCdecom}_{b_i})$:

   Check 1: $\mathsf{TCdecom}_{b_i}$ is a valid decommitment of tcom_{b_i} to b_i.

   Check 2: σ is (the unique) valid signature of tcom_{b_i} corresponding to the verification key vk_τ, where vk_τ is the key generated by querying $\mathsf{TK}^{\mathsf{Com}}_{\mathrm{S}}$ where $\tau = \mathsf{sid}||\mathsf{ssid}||\mathsf{tcom}_{b_i}||i$.

If both the checks pass then output $(s_b^i, \mathsf{decom}_{s_b^i})$, otherwise output $(\perp, \perp)$.

Furthermore, we use the same protocol as described in the previous section with the following two changes:

- R sends all the verification keys vk_τ along with its first message.
- S verifies whether the verification keys correctly correspond to the commitments tcom_{b_i} and then provides a signature of tcom_{b_i} for every i, under key sk_τ for $\tau = \mathsf{sid}||\mathsf{ssid}||\mathsf{tcom}_{b_i}||i$, along with its second message to the receiver.

Proof Sketch: We briefly highlight the differences in the proof for the modified protocol.

Sender corruption. The simulator proceeds in stages, a stage for each $\mathsf{sid} \in [q_2(n)]$. In Stage sid, the simulation proceeds as in the previous protocol Π_{OT}. Recall first that the simulation for that protocol extracts e in a first run and then uses e to equivocate the receiver's commitments. We will employ the same strategy here, with the exception that the simulator extracts $e_{\mathsf{sid},\mathsf{ssid}}$ simultaneously in the first run for every $\mathsf{ssid} \in [q_1(n)]$ (namely, for all parallel sessions), as there are $q_1(n)$ parallel sessions. We remark that in the extraction phase we rely heavily on the fact that these sessions are run in parallel. Then in the rewound executions, the simulator equivocates the receiver's commitments accordingly. In addition, the simulator produces all signatures for the second message honestly. Indistinguishability follows essentially as before. We remark that since the signatures are unique given the verification key, the signatures do not carry any additional information beyond the message.

On a high-level, we rely on the same sequence of hybrids as in the previous protocol Π_{OT}, once for each simulation stage. Below are the changes to the hybrids for each stage:

1. In Hybrid H_1, the simulator proceeds exactly as $\mathcal{S}_1$, with the exception that it extracts all the trapdoors $e_{\mathsf{sid},\mathsf{ssid}}$ simultaneously.
2. In Hybrid H_2, the simulation proceeds as the previous hybrid with the exception that it honestly commits to the receiver's input as 0 in *all* parallel sessions in the first run.
3. Instead of the $\kappa+1$ hybrids $\mathrm{H}_{3,0}, \ldots, \mathrm{H}_{3,\kappa}$ used in the previous protocol Π_{OT} in order to replace the receiver's commitments from being honestly generated to equivocal commitments using the trapdoor, we consider $q_1(n) \times \kappa$ hybrids for the $q_1(n)$ parallel OT sessions.
4. Finally, in Hybrid H_4, the simulation proceeds analogously with the exception that it extracts the sender's input in all $q_1(n)$ sessions simultaneously.

Receiver corruption. The simulator proceeds in stages, a stage for each $\mathsf{sid} \in [q_2(n)]$, where in each stage the simulation proceeds similarly to the simulation of the previous protocol Π_{OT}. Recall first that the simulation for Π_{OT} extracts all the queries made to both the tokens and records these values. Then upon receiving the first message, these queries are used for extracting the receiver's input. We will employ the same strategy here, with the exception that for every parallel session with identifier $\mathsf{ssid} \in [q_1(n)]$ the simulator extracts the receiver's input simultaneously. As for the second message, the simulator acts analogously to our previous simulation for each parallel session.

To argue indistinguishability, we consider a sequence of hybrids executions, once for each sequential session analogous to the modifications for the sender corruption. First, from the unforgeability of the one-time signature scheme we conclude that the malicious receiver cannot make bad queries to the

OTM token. More formally, in Hybrid H_4 corresponding to every sequential session, we argue from the unforgeability of the signature scheme that a receiver queries the OT token on an input

$$(\sigma, \mathsf{sid}, \mathsf{ssid}, i, b_i, \mathsf{tcom}_{b_i}, \mathsf{TCdecom}_{b_i})$$

only if it requested a query $(\mathsf{sid}, \mathsf{ssid}, \mathsf{tcom}_{b_i}, i)$ to $\mathsf{TK}_{\mathrm{S}}^{\mathsf{Com}}$ and sent tcom_{b_i} in the ith coordinate for that session as part of its first message to the sender. This will allow us to combine with the argument made in Hybrid 4 of Lemma 4.8 to conclude that there is at most one value of b_i for which it can make the query $(\sigma, \mathsf{sid}, \mathsf{ssid}, i, b_i, \mathsf{tcom}_{b_i}, \mathsf{TCdecom}_{b_i})$ to tokens $\{\mathsf{TK}_{i_j}\}_{j\in[4\kappa]}$. Next, for each parallel session, we include hybrids between H_5 and H_6, one for each of the sender's inputs in each parallel sessions that the receiver cannot obtain and indistinguishability follows analogous to proof of Lemma 4.11.

4.2 Three-Round UC OT

In this section we present a UC secure OT protocol that realizes functionality $\mathcal{F}_{\mathrm{OT}}$, presented in Figure 5, that requires a token exchange phase and runs in three rounds. Our protocol is similar to the protocol presented in Figures 3 and 4 except for the following two differences. (1) First, in order to enable the input extraction of the sender's inputs using a straight-line simulator, we add one more round of interaction for which the sender commits to the challenge e. As shown below, this implies that the commitment used by the sender is straight-line extractable, which allows the simulator for a corrupted sender to learn e *before* it needs to commit to the receiver's input. More specifically, based on the "look ahead" property of the Pass and Wee commitment scheme [PW09], the scheme we employ in Figure 4 becomes a trapdoor commitment given the knowledge of the challenge prior to committing. Relying on the fact that the Pass-Wee scheme is a trapdoor commitment, we obtain the same security proof as in our plain model OT but without rewinding the sender. More concretely, the simulator invokes the set of tokens $\{\mathsf{TK}_i\}_{i\in\kappa}$ twice (with two distinct sets of bit-vectors) in order to extract both s_0 and s_1. Note that such token resetting does not violate the UC simulation framework as the simulator is given the code of these tokens and can locally employ them multiple times without letting the environment observe it. (2) In addition, in order to support such an extractable commitment, we use an additional set of tokens $\{\mathsf{TK}_{\mathrm{R}}^{\mathsf{PRF},i}\}_{i\in[\kappa]}$ that is sent by the receiver in the token exchange phase, and allows to extract the sender's committed challenge e by monitoring the queries for this token. This further implies that we need the tokens to be exchanged in both directions. Note that sending the tokens in both directions is crucial for obtaining straight-line extraction since it allows the simulator the power of observing the sender's queries to the tokens. Without adding these tokens it is not clear how to extract the sender's input with no rewinding. We recall that input extraction of the receiver is already achieved in a straight-line manner due to the same reasoning of monitoring the tokens queries.

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

Functionality $\mathcal{F}_{\mathrm{OT}}$ communicates with with sender S and receiver R, and adversary $\mathcal{S}$.

1. Upon receiving input $(\mathsf{sender}, \mathsf{sid}, s_0, s_1)$ from S where $s_0, s_1 \in \{0,1\}^\kappa$, record (sid, s_0, s_1).
2. Upon receiving $(\mathsf{receiver}, \mathsf{sid}, b)$ from R, where a tuple (sid, s_0, s_1) is recorded and $b \in \{0,1\}$, send (sid, s_b) to R and sid to $\mathcal{S}$. Otherwise, abort.

Figure 5: The oblivious transfer functionality.

In the following, we formally describe our protocol and then sketch the proof outline which follows the same outline as the proof of the plain model OT. As before, we split the protocol description into two figures, one for the token exchange phase in Figure 6 and one of the three round protocol in Figure 7.

Token Exchange Phase $\mathsf{TK}_{\text{UCTP}}$

Let (1) Com be a non-interactive perfectly binding commitment scheme and (2) F, F' be two families of pseudorandom functions that map $\{0,1\}^{5\kappa} \to \{0,1\}^{\kappa}$ and $\{0,1\}^{\kappa} \to \{0,1\}^{p(\kappa)}$, respectively.

Token Exchange Phase:

1. $\{\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}\}_{l\in[4\kappa^2]}$: S chooses $4\kappa^2$ random PRF keys $\{\gamma_l\}_{l\in[4\kappa^2]}$ for family F. Let $\mathsf{PRF}_{\gamma_l} : \{0,1\}^{5\kappa} \to \{0,1\}^{\kappa}$ denote the pseudorandom function. For all $l \in [4\kappa^2]$, S creates a token $\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}$ by sending $(\mathsf{Create}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_l, M_1)$ to R, that on input x outputs $\mathsf{PRF}_{\gamma_l}(x)$, where M_1 is the functionality.
2. $\mathsf{TK}_{\mathrm{S}}^{\mathsf{Com}}$: S chooses a random PRF' key γ' for family F'. Let $\mathsf{PRF}'_{\gamma'} : \{0,1\}^{\kappa} \to \{0,1\}^{p(\kappa)}$ denote the pseudorandom function. S creates token $\mathsf{TK}_{\mathrm{S}}^{\mathsf{Com}}$ by sending $(\mathsf{Create}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{l+1}, M_2)$ to R where M_2 implements the following functionality:
   - On input (tcom_{b_i}):
     compute $V = \mathsf{PRF}'_{\gamma'}(\mathsf{tcom}_{b_i})$,
     parse V as $s_0^i||s_1^i||r_0||r_1$,
     compute $\mathsf{com}_{s_b^i} \leftarrow \mathsf{Com}(s_b^i; r_b)$ for $b = \{0,1\}$,
     and output $\mathsf{com}_{s_0^i}, \mathsf{com}_{s_1^i}$.
3. Similarly, R generates the tokens $\{\mathsf{TK}_{\mathrm{R}}^{\mathsf{PRF},i}\}_{i\in[\kappa]}$ which are analogous to the PRF tokens $\{\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}\}_{l\in[4\kappa^2]}$ generated by S by sending $(\mathsf{Create}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}_i, M_3)$ to S for all $i \in [\kappa]$.

Figure 6: UC OT - Tokens Exchange Phase

Theorem 4.2. *Assume the existence of one-way permutations, then the protocol presented in Figure 7 UC realizes $\mathcal{F}_{\mathrm{OT}}$ in the $\mathcal{F}_{\mathrm{WRAP}}^{\mathrm{Stateless}}$-hybrid model.*

Proof: Let $\mathcal{A}$ be a malicious PPT real adversary attacking protocol $\Pi_{\mathrm{OT}}^{\mathrm{UC}}$ from Figure 7 in the $\mathcal{F}_{\mathrm{WRAP}}^{\mathrm{Stateless}}$-hybrid model. We construct an ideal adversary $\mathcal{S}$ with access to $\mathcal{F}_{\mathrm{OT}}$ which simulates a real execution of $\Pi_{\mathrm{OT}}^{\mathrm{UC}}$ with $\mathcal{A}$ such that no environment $\mathcal{Z}$ can distinguish the ideal process with $\mathcal{S}$ and $\mathcal{F}_{\mathrm{OT}}$ from a real execution of $\Pi_{\mathrm{OT}}^{\mathrm{UC}}$ with $\mathcal{A}$. $\mathcal{S}$ starts by invoking a copy of $\mathcal{A}$ and running a simulated interaction of $\mathcal{A}$ with environment $\mathcal{Z}$, emulating the honest party. We describe the actions of $\mathcal{S}$ for every corruption case.

Simulating the communication with $\mathcal{Z}$: Every message that $\mathcal{S}$ receives from $\mathcal{Z}$ it internally feeds to $\mathcal{A}$ and every output written by $\mathcal{A}$ is relayed back to $\mathcal{Z}$.

Simulating the corrupted S. Similarly to the proof of Theorem 4.1, when the sender is corrupted, the simulator needs to extract the challenge first in order to extract both S's inputs to the OT. The simulation follows the same outlines as the simulation in the former proof except that here the simulator extracts e at the onset of the simulation, as it is not allowed to rewind the adversary. Next, it proceeds to extract the sender's inputs by exploiting the equivocation of the trapdoor commitment scheme. We briefly sketch the simulation and the proof below.

3-Round UC Protocol $\Pi_{\mathrm{OT}}^{\mathrm{UC}}$

Protocol $\Pi_{\mathrm{OT}}^{\mathrm{UC}}$ is presented in the $\mathcal{F}_{\mathrm{WRAP}}^{\mathrm{Stateless}}$-hybrid model with sender S and receiver R. Let (1) Com be a non-interactive perfectly binding commitment scheme, (2) $\mathsf{TCom} = \{\mathsf{TCmsg}_1, \mathsf{TCmsg}_2, \mathsf{TCmsg}_3\}$ denote the three messages exchanged in the commit phase of the trapdoor commitment scheme, (3) F, F' be two families of pseudorandom functions that map $\{0,1\}^{5\kappa} \to \{0,1\}^{\kappa}$ and $\{0,1\}^{\kappa} \to \{0,1\}^{p(\kappa)}$, respectively (4) H denote a hardcore bit function and (5) $\mathsf{Ext} : \{0,1\}^{5\kappa} \times \{0,1\}^{d} \to \{0,1\}$ denote a randomness extractor where the source has length 5κ and the seed has length d.

Tokens Exchange Phase: S creates tokens $\{\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}\}_{l\in[4\kappa^2]}$ and $\mathsf{TK}_{\mathrm{S}}^{\mathsf{Com}}$ and R creates tokens $\{\mathsf{TK}_{\mathrm{R}}^{\mathsf{PRF},i}\}_{i\in[\kappa]}$ as per Figure 6.

Input: S holds two strings $s_0, s_1 \in \{0,1\}^{\kappa}$ and R holds a bit b.

The Protocol:

$\mathrm{S} \to \mathrm{R}$:

1. S picks κ random bits $e_1, \ldots, e_\kappa$ and κ strings $u_1, \ldots, u_\kappa \leftarrow \{0,1\}^{5\kappa}$, sends $(\mathsf{Run}, \mathsf{sid}, \mathrm{S}, \mathsf{mid}_{3_i}, u_i)$ and receives $v_i = \mathsf{TK}_{\mathrm{R}}^{\mathsf{PRF},i}(u_i)$ for all $i \in [\kappa]$. If any of the tokens abort, the sender aborts.
2. S sends the receiver the commitments $\mathsf{com}_{e_i} = (\mathsf{Ext}(u_i) \oplus e_i, v_i)$ for all $i \in [\kappa]$.

$\mathrm{R} \to \mathrm{S}$:

1. For all $i \in [\kappa]$ and $j \in [4\kappa]$, R chooses $u_i^j \leftarrow \{0,1\}^{5\kappa}$, sends $(\mathsf{Run}, \mathsf{sid}, \mathrm{S}, \mathsf{mid}_{1_l}, u_i^j)$ and receives $v_i^j = \mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}(u_i^j)$ (where $l \in [4\kappa^2]$ is an encoding of the pair (i,j)). If any of the tokens abort, the receiver aborts.
2. R chooses $\kappa - 1$ random bits $b_1, \ldots, b_{\kappa-1}$ and sets b_κ such that $b = \bigoplus_{i=1}^{\kappa} b_i$. For all $i \in [\kappa]$, it commits to b_i be setting $\mathsf{tcom}_{b_i} = (M_1^i, \ldots, M_\kappa^i)$. In particular, $\forall j \in [\kappa]$ as per Figure 2:
$$M_j^i, = \begin{pmatrix} (\mathsf{Ext}(u_i^{4j-3}) \oplus \eta_{i,j}, v_i^{4j-3}) & (\mathsf{Ext}(u_i^{4j-1}) \oplus b_i \oplus \eta_{i,j}, v_i^{4j-1}) \\ (\mathsf{Ext}(u_i^{4j-2}) \oplus \eta_{i,j}, v_i^{4j-2}) & (\mathsf{Ext}(u_i^{4j}) \oplus b_i \oplus \eta_{i,j}, v_i^{4j}) \end{pmatrix}.$$
3. For all $i \in [\kappa]$, R sends tcom_{b_i}.

$\mathrm{S} \to \mathrm{R}$:

1. S chooses $z_1, \ldots, z_\kappa, \Delta \leftarrow \{0,1\}$ and sends $\{w_i^0 = z_i \oplus \mathsf{H}(s_i^0), w_i^1 = z_i \oplus \Delta \oplus \mathsf{H}(s_i^1)\}_{i\in[\kappa]}$, $s_0' = w \oplus s_0$ and $s_1' = w \oplus \Delta \oplus s_1$ where $w = \bigoplus_{j=1}^{\kappa} z_j$.
2. S sends $\mathsf{TCmsg}_3 = (e, \mathsf{decom}_e)$.
3. For all $i \in [\kappa]$, S creates a token TK_i by sending $(\mathsf{Create}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{l+1+i}, M_4)$ to R where M_4 implements the following functionality:
   - On input $(b_i, \mathsf{TCdecom}_{b_i})$:
     If $\mathsf{TCdecom}_{b_i}$ is verified correctly then output $(s_b^i, \mathsf{decom}_{s_b^i})$,
     else output $(\perp, \perp)$

Output Phase:

1. For all $i \in [\kappa]$, R sends $(\mathsf{Run}, \mathsf{sid}, \mathrm{S}, \mathsf{mid}_{l+1}, (\mathsf{com}_{b_i}, i))$ receiving back $\mathsf{com}_{s_0^i}, \mathsf{com}_{s_0^i}$.
2. For all $i \in [\kappa]$, R sends $(\mathsf{Run}, \mathsf{sid}, \mathrm{S}, \mathsf{mid}_{i+2}, (b_i, \mathsf{TCdecom}_{b_i}))$ and receives $(s_b^i, \mathsf{decom}_{s_b^i})$. If the decommitments $\mathsf{decom}_{s_b^i}$ and decom_e are valid, R computes $\tilde{z}_i = \mathsf{H}(s_b^i) \oplus w_i^{b_i}$ and $s_b = \bigoplus_{i=1}^{\kappa} \tilde{z}_i \oplus s_b'$.

Figure 7: 3-Round UC OT

1. $\mathcal{S}$ emulates the role of $\mathcal{F}_{\rm WRAP}^{\rm Stateless}$ as follows, where upon receiving from $\mathcal{A}$ the messages $\{(\mathsf{Create}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_l, M_1)\}_{l\in[4\kappa^2]}$ and $(\mathsf{Create}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{l+1}, M_2)$, $\mathcal{S}$ stores these codes.

2. Next, $\mathcal{S}$ emulates the role of $\mathcal{F}_{\rm WRAP}^{\rm Stateless}$ for tokens $\{\mathsf{TK}_{\rm R}^{\mathsf{PRF},i}\}_{i\in[\kappa]}$, where for each query $u \in \{0,1\}^{5\kappa}$ made by $\mathcal{A}$ to token $\mathsf{TK}_{\rm R}^{\mathsf{PRF},i}$, $\mathcal{S}$ returns a random v from $\{0,1\}^{\kappa}$. $\mathcal{S}$ further stores the pair (u, v) in the query/answer list of $\mathsf{TK}_{\rm R}^{\mathsf{PRF},i}$.

3. Upon receiving the set of commitments $\{\mathsf{com}_{e_i} = (\mathsf{com}_{e_i}^0, \mathsf{com}_{e_i}^1)\}_i$ from $\mathcal{A}$, $\mathcal{S}$ extracts e as follows. For each $i \in [\kappa]$, $\mathcal{S}$ verifies first whether there is a recorded pair of the form (u_i, v_i) in the ith query/answer list, where $v_i = \mathsf{com}_{e_i}^1$. If so, $\mathcal{S}$ records the bit $e_i = \mathsf{Ext}(u_i) \oplus \mathsf{com}_{e_i}^0$. Else, $\mathcal{S}$ sets e_i as a random bit.

4. Next, $\mathcal{S}$ picks two random bit-vectors $(b_1, \ldots, b_\kappa)$ and $(b'_1, \ldots, b'_\kappa)$ such that $\bigoplus_{i=1}^{\kappa} b_i = 0$ and $\bigoplus_{i=1}^{\kappa} b'_i = 1$ and commits to these values as follows. Let $e = e_1, \ldots, e_\kappa$ denote the challenge being properly decommitted by the sender. Then, for all $i, j \in [\kappa]$, $\mathcal{S}$ sends matrix M_i^j where the e_ith column is defined by
$$\begin{matrix}(\mathsf{Ext}(u_i^{4j-3}) \oplus \eta_{i,j}, v_i^{4j-3})\\ (\mathsf{Ext}(u_i^{4j-2}) \oplus \eta_{i,j}, v_i^{4j-2})\end{matrix}$$
whereas the $(1-e_i)$th column is defined by
$$\begin{matrix}(\mathsf{Ext}(u_i^{4j-1}) \oplus b'_i \oplus \eta_{i,j}, v_i^{4j-1})\\ (\mathsf{Ext}(u_i^{4j}) \oplus b'_i \oplus \eta_{i,j}, v_i^{4j}).\end{matrix}$$

5. To extract s_0, $\mathcal{S}$ simply follows the honest receiver's code with the exception that it appropriately decommits the equivocal commitment to b_i. Next, upon extracting s_0, $\mathcal{S}$ rewinds the set of tokens $\{\mathsf{TK}_i\}_{i\in[\kappa]}$ where this time $\mathcal{S}$ inputs the decommitments that correspond to the bit-vector $\bigoplus_{i=1}^{\kappa} b'_i = 1$. If the execution is completed successfully (as would have been verified by the honest receiver), then the simulator records s_1. Otherwise, $\mathcal{S}$ records $s_1 = \perp$.

6. $\mathcal{S}$ sends (s_0, s_1) to the trusted party that computes $\mathcal{F}_{\rm OT}$ and halts, outputting whatever $\mathcal{A}$ does.

Note that the real and simulated executions differ due to the commitment sent by the receiver in the second step. Specifically, the simulator commits to both bit-vectors $(b_1, \ldots, b_\kappa)$ and $(b'_1, \ldots, b'_\kappa)$, whereas the real receiver only commits the bit-vector that corresponds to its input b. Statistical indistinguishability follows immediately from the proof of Lemma 4.4. It is just left to argue that in case the simulator fails to extract any bit e_i from com_{e_i}, then $\mathcal{A}$ will not be able to produce a valid decommitment for this commitment. Roughly speaking, this follows from the pseudorandomness of the functions embedded inside tokens $\{\mathsf{TK}_{\rm R}^{\mathsf{PRF},i}\}_{i\in[\kappa]}$. Namely, as $\mathcal{A}$ does not query the token $\mathsf{TK}_{\rm R}^{\mathsf{PRF},i}$ on u_i for some i, the probability that it correctly guesses v_i is negligible. More concretely, assume by contradiction that $\mathcal{A}$ successfully produces a valid decommitment for some $\mathsf{com}_{e_i} = (a_i, b_i)$ for which b_i is not recorded by the simulator as a token query, with non-negligible probability. Then, it is possible to construct a distinguisher between the pseudorandom function and a random function (as for a random function this even can only occur with negligible probability).

Simulating the corrupted R**.** Simulating the receiver follows similarly to the simulation in the proof of Theorem 4.1 with the exception that the receiver here also sends the PRF tokens. Loosely speaking, the simulator extracts the adversary's input b based on the second message and the queries to the sender's

PRF tokens. Now, since the commitment scheme used by the sender is statistically hiding, using a similar argument to Lemma 4.3, the proof follows similarly as in Theorem 4.1. ■

4.3 Two-Round UC OT

In this section we present another UC OT protocol with improved round complexity. Our third protocol is inspired by the recently developed protocol from [ORS15] and requires only two rounds (see Figure 9) and a token exchange phase (see Figure 8, we note that here the number of PRF tokens forwarded by the sender adds up to 3κ which is the overall challenge length picked by the receiver, whereas the number of tokens sent by the receiver is twice $4\kappa^2$). We begin by recalling that in both of our prior protocols in the plain model (see Section 4.1.2) as well as the 3-round protocol in the UC model (see Section 4.2) we relied on the fact that the receiver's commitments were made via a trapdoor commitment scheme. This is needed to make the sender's inputs independent of the receiver's input. In the plain model, the trapdoor e could be committed via a token sent at the token exchange phase to the receiver, as it could be extracted using rewinding. However, for the UC protocol we needed to add an extra message from S to R in order to provide a straight-line extraction strategy. We stress that it seems unlikely to reduce the number of rounds to two if we pursue this approach.

We consider a new approach here for which the sender's inputs are extracted directly by monitoring the queries it makes to a PRF token (just as the receiver's input was extracted in our two other protocols). Nevertheless, as explained in the proof of protocol Π_{OT}, doing so does not necessarily allow us to extract the inputs correctly as the tokens can behave maliciously. We therefore need additional checks to ensure that the sender's inputs can be verified. For instance, if in our previous protocols the receiver could have ensured that the sender's inputs to each of the OT add up to Δ, then we could rely on such an extraction here. Note that while we are not able to use this combiner, we do show how to rely on the combiner of [ORS15] to reduce the number of rounds. An overview of their approach is given in the introduction.

We are now ready to describe our protocol $\Pi_{\mathrm{OT}}^{\mathrm{UC}}$ that is presented in the $\mathcal{F}_{\mathrm{WRAP}}^{\mathrm{Stateless}}$-hybrid model with sender S and receiver R using the following building blocks: let (1) Com be a non-interactive perfectly binding commitment scheme, (2) let $\mathcal{SS} = (\mathsf{Share}, \mathsf{Recon})$ be a $(\kappa+1)$-out-of-2κ Shamir secret-sharing scheme over $\mathbb{Z}_p$, together with a linear map $\phi : \mathbb{Z}_p^{2\kappa} \to \mathbb{Z}_p^{\kappa-1}$ such that $\phi(v) = 0$ iff v is a valid sharing of some secret, (3) F, F' be two families of pseudorandom functions that map $\{0,1\}^{5\kappa} \to \{0,1\}^{\kappa}$ and $\{0,1\}^{\kappa} \to \{0,1\}^{p(\kappa)}$, respectively (4) H denote a hardcore bit function and (5) $\mathsf{Ext} : \{0,1\}^{5\kappa} \times \{0,1\}^{d} \to \{0,1\}$ denote a randomness extractor where the source has length 5κ and the seed has length d.

Token Exchange Phase $\mathsf{TK}_{\mathrm{UCTP}}$

Let F be a family of pseudorandom functions that maps $\{0,1\}^{5\kappa} \to \{0,1\}^{\kappa}$.

Token Exchange Phase:

1. $\{\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}\}_{l\in[3\kappa]}$: S chooses 3κ random PRF keys $\{\gamma_l\}_{[l\in 3\kappa]}$ for family F. Let $\mathsf{PRF}_{\gamma_l} : \{0,1\}^{5\kappa} \to \{0,1\}^{\kappa}$ denote the pseudorandom function. S creates token $\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}$ by sending $(\mathsf{Create}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_l, M_1)$ to R that on input x, outputs $\mathsf{PRF}_{\gamma_l}(x)$ for all $l \in [3\kappa]$, where M_1 is the functionality.
2. Similarly, R generates the tokens $\{\mathsf{TK}_{\mathrm{R}}^{\mathsf{PRF},l'}\}_{l'\in[8\kappa^2]}$ which are analogous to the PRF tokens $\{\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}\}_{l\in[3\kappa]}$ by sending $(\mathsf{Create}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}_{l'}, M_2)$ to S for all $l' \in [8\kappa^2]$.

Figure 8: UC OT - Tokens Exchange Phase

Theorem 4.3. *Assume the existence of one-way permutations, then the protocol presented in Figure 9 UC realizes $\mathcal{F}_{\mathrm{OT}}$ in the $\mathcal{F}_{\mathrm{WRAP}}^{\mathrm{Stateless}}$-hybrid model.*

Proof overview. On a high-level, when the sender is corrupted our simulation proceeds analogously to the simulation from [ORS15] where the simulator generates the view of the malicious sender by honestly generating the receiver's messages and then extracting all the values committed to by the sender. Nevertheless, while in [ORS15] the authors rely on extractable commitments and extract the sender's inputs via rewinding, we directly extract its inputs by monitoring the queries made by the malicious sender to the $\{\mathsf{TK}_{\mathrm{R}}^{\mathsf{PRF},i}\}_i$ tokens. The proof of correctness follows analogously. More explicitly, the share consistency check ensures that for any particular column that the receiver obtains, if the sum of the values agree on the same bit, then the receiver extracts the correct share of $[x_b]$ with high probability. Note that it suffices for the receiver to obtain $\kappa + 1$ good columns for its input b to extract enough shares to reconstruct x_b since the shares can be checked for validity. Namely, the receiver chooses $\kappa/2$ indices T_b and sets its input for these OT executions as b. For the rest of the OT executions, the receiver sets its input as $1 - b$. Denote this set of indices by T_{1-b}. Then, upon receiving the sender's response to its challenge and the OT responses, the receiver first performs the shares consistency check. If this check passes, it performs the shares validity check for all columns, both with indices in T_{1-b} and for the indices in a random subset of size $\kappa/2$ within T_b. If one of these checks do not pass, the receiver aborts. If both checks pass, it holds with high probability that the decommitment information for $b = 0$ and $b = 1$ are correct in all but $s \in \omega(\log n)$ indices. Therefore, the receiver will extract $[x_b]$ successfully both when its input $b = 0$ and $b = 1$. Furthermore, it is ensured that if the two checks performed by the receiver pass, then a simulator can extract both x_0 and x_1 correctly by simply extracting the sender's input to the OT protocol and following the receiver's strategy to extract.

On the other hand, when the receiver is corrupted, our simulation proceeds analogous to the simulation in [ORS15] where the simulator generates the view of the malicious receiver by first extracting the receiver's input b and then obtaining s_b from the ideal functionality. It then completes the execution following the honest sender's code with (s_0, s_1), where s_{1-b} is set to random. Moreover, while in [ORS15] the authors relies on a special type of interactive commitment that allows the extraction of the receiver's input via rewinding, we instead extract this input directly by monitoring the queries made by the malicious receiver to the $\{\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}\}_{l\in[3\kappa]}$ tokens. The proof of correctness follows analogously. Informally, the idea is to show that the receiver can learn $\kappa + 1$ or more shares for either x_0 or x_1 but not both. In other words there exists a bit b for which a corrupted receiver can learn at most κ shares relative to s_{1-b}. Thus, by replacing s_{1-b} with a random string, it follows from the secret-sharing property that obtaining at most κ shares keeps s_{1-b} information theoretically hidden.

Proof: Let $\mathcal{A}$ be a malicious PPT real adversary attacking protocol $\Pi_{\mathrm{OT}}^{\mathrm{UC}}$ from Figure 9 in the $\mathcal{F}_{\mathrm{WRAP}}^{\mathrm{Stateless}}$-hybrid model. We construct an ideal adversary $\mathcal{S}$ with access to $\mathcal{F}_{\mathrm{OT}}$ which simulates a real execution of $\Pi_{\mathrm{OT}}^{\mathrm{UC}}$ with $\mathcal{A}$ such that no environment $\mathcal{Z}$ can distinguish the ideal process with $\mathcal{S}$ and $\mathcal{F}_{\mathrm{OT}}$ from a real execution of $\Pi_{\mathrm{OT}}^{\mathrm{UC}}$ with $\mathcal{A}$. $\mathcal{S}$ starts by invoking a copy of $\mathcal{A}$ and running a simulated interaction of $\mathcal{A}$ with environment $\mathcal{Z}$, emulating the honest party. We describe the actions of $\mathcal{S}$ for every corruption case.

Simulating the communication with $\mathcal{Z}$: Every message that $\mathcal{S}$ receives from $\mathcal{Z}$ it internally feeds to $\mathcal{A}$ and every output written by $\mathcal{A}$ is relayed back to $\mathcal{Z}$.

Simulating the corrupted S. We begin with describing our simulation:

2-Round UC Protocol $\Pi^{\text{UC}}_{\text{OT}}$

Tokens Exchange Phase: S creates tokens $\{\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}\}_{l\in[3\kappa]}$ and R creates tokens $\{\mathsf{TK}_{\mathrm{R}}^{\mathsf{PRF},i}\}_{i\in[\kappa]}$ as per Figure 8.

Input: S holds two strings $s_0, s_1 \in \{0,1\}^\kappa$ and R holds a bit b.

The Protocol:

R $\rightarrow$ S:

1. R selects a random subset $T_{1-b} \subseteq [2\kappa]$ of size $\kappa/2$. Define $T_b = [2\kappa]/T_{1-b}$. For every $j \in [2\kappa]$, R sets $b_j = \beta$ if $j \in T_\beta$.
2. R samples uniformly at random $c_1, \ldots, c_\kappa \leftarrow \{0,1\}$.
3. Finally, R sends $(\{\mathsf{com}_{b_j}\}_{j\in[2\kappa]}, \{\mathsf{com}_{c_i}\}_{i\in[\kappa]})$ to S where

$$\forall\, i \in [\kappa], j \in [2\kappa] \quad \mathsf{com}_{b_j} = (\mathsf{Ext}(u_j) \oplus b_j, v_j) \quad \text{and} \quad \mathsf{com}_{c_i} = (\mathsf{Ext}(u'_i) \oplus c_i, v'_i)$$

$u_j, u'_i \leftarrow \{0,1\}^{5\kappa}$ and v_j, v'_i are obtained by sending $(\mathsf{Run}, \mathsf{sid}, \mathrm{S}, \mathsf{mid}_j, u_j)$ and $(\mathsf{Run}, \mathsf{sid}, \mathrm{S}, \mathsf{mid}_{2\kappa+i}, u'_i)$.

S $\rightarrow$ R:

1. S picks two random strings $x_0, x_1 \leftarrow \mathbb{Z}_p$ and secret shares them using $\mathcal{SS}$. In particular, S computes $[x_b] = (x_b^1, \ldots, x_b^{2\kappa}) \leftarrow \mathsf{Share}(x_b)$ for $b \in \{0,1\}$.
2. S commits to the shares $[x_0], [x_1]$ as follows. It picks random matrices $A_0, B_0 \leftarrow \mathbb{Z}_p^{\kappa\times 2\kappa}$ and $A_1, B_1 \leftarrow \mathbb{Z}_p^{\kappa\times 2\kappa}$ such that $\forall i \in [\kappa]$:

$$A_0[i,\cdot] + B_0[i,\cdot] = [x_0], \ \ A_1[i,\cdot] + B_1[i,\cdot] = [x_1].$$

S computes two matrices $Z_0, Z_1 \in \mathbb{Z}_p^{\kappa\times\kappa-1}$ and sends them in the clear such that:

$$Z_0[i,\cdot] = \phi(A_0[i,\cdot]), Z_1[i,\cdot] = \phi(A_1[i,\cdot]).$$

3. S sends matrices $(\mathsf{com}_{A_0}, \mathsf{com}_{B_0}, \mathsf{com}_{A_1}, \mathsf{com}_{B_1})$ to R, where,

$$\begin{aligned}\forall\, i \in [\kappa],\ j \in [2\kappa], \beta \in \{0,1\} \quad & \mathsf{com}_{A_\beta[i,j]} = (\mathsf{Ext}(u^{A_\beta[i,j]} \oplus A_\beta[i,j], v^{A_\beta[i,j]}) \\ & \mathsf{com}_{B_\beta[i,j]} = (\mathsf{Ext}(u^{B_\beta[i,j]} \oplus B_\beta[i,j], v^{B_\beta[i,j]})\end{aligned}$$

where $(u^{A_\beta[i,j]}, u^{B_\beta[i,j]}) \leftarrow \{0,1\}^{5\kappa}$ and $(v^{A_\beta[i,j]}, v^{B_\beta[i,j]})$ are obtained by sending $(\mathsf{Run}, \mathsf{sid}, \mathrm{S}, \mathsf{mid}_{[i,j,\beta]}, u^{A_\beta[i,j]})$ and $(\mathsf{Run}, \mathsf{sid}, \mathrm{S}, \mathsf{mid}_{2\kappa^2+[i,j,\beta]}, u^{B_\beta[i,j]})$, respectively, to the token $\mathsf{TK}_{\mathrm{R}}^{\mathsf{PRF},[i,j,\beta]}$ where $[i,j,\beta]$ is an encoding of the indices i, j, β into an integer in $[2\kappa^2]$.

4. S sends $C_0 = s_0 \oplus x_0$ and $C_1 = s_1 \oplus x_1$ to R.
5. For all $j \in [2\kappa]$, S creates a token TK_j by sending $(\mathsf{Create}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{3\kappa+j}, M_3)$ to R where M_3 is the functionality that on input $(b_j, \mathsf{decom}_{b_j})$, aborts if decom_{b_j} is not verified correctly. Otherwise it outputs $(A_{b_j}[\cdot,j], \mathsf{decom}_{A_{b_j}[\cdot,j]}, B_{b_j}[\cdot,j], \mathsf{decom}_{B_{b_j}[\cdot,j]})$.
6. For all $i \in [\kappa]$, S creates a token $\widehat{\mathsf{TK}}_i$ by sending $(\mathsf{Create}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{5\kappa+i}, M_4)$ to R where M_4 is the functionality that on input $(c_i, \mathsf{decom}_{c_i})$ aborts if decom_{c_i} is not verified correctly. Otherwise it outputs,

$$\begin{aligned}&(A_0[i,\cdot], \mathsf{decom}_{A_0[i,\cdot]}, A_1[i,\cdot], \mathsf{decom}_{A_1[i,\cdot]}), \text{ if } c = 0 \\ &(B_0[i,\cdot], \mathsf{decom}_{B_0[i,\cdot]}, B_1[i,\cdot], \mathsf{decom}_{B_1[i,\cdot]}), \text{ if } c = 1\end{aligned}$$

Output Phase: See Figure 10.

Figure 9: 2-Round UC OT

Output Phase for $\Pi_{\mathrm{OT}}^{\mathrm{UC}}$

Output Phase:

1. For all $j \in [2\kappa]$, R sends $(\mathsf{Run}, \mathsf{sid}, \mathrm{S}, \mathsf{mid}_{3\kappa+j}, (b_j, \mathsf{decom}_{b_j}))$ receiving back $(A_{b_j}[\cdot, j], \mathsf{decom}_{A_{b_j}[\cdot,j]}, B_{b_j}[\cdot, j], \mathsf{decom}_{B_{b_j}[\cdot,j]})$.
2. For all $i \in [\kappa]$, R sends $(\mathsf{Run}, \mathsf{sid}, \mathrm{S}, \mathsf{mid}_{5\kappa+i}, (c_i, \mathsf{decom}_{c_i}))$ receiving back $(A_0[\cdot, i], A_1[\cdot, i])$ or $(B_0[\cdot, i], B_1[\cdot, i])$.

Combiner:

Shares Validity Check Phase: For all $i \in [\kappa]$, if $c_i = 0$ check that $Z_0[i, \cdot] = \phi(A_0[i, \cdot])$ and $Z_1[i, \cdot] = \phi(A_1[i, \cdot])$. Otherwise, if $c_i = 1$ check that $\phi(B_0[i, \cdot]) + Z_0[i, \cdot] = 0$ and $\phi(B_1[i, \cdot]) + Z_1[i, \cdot] = 0$. If the tokens do not abort and all the checks pass, the receiver proceeds to the next phase.

Shares Consistency Check Phase: For each $b \in \{0, 1\}$, R randomly chooses a set T_b for which $b_j = b$ of $\kappa/2$ coordinates. For each $j \in T_b$, R checks that there exists a unique x_b^j such that $A_b[i, j] + B_b[i, j] = x_b^j$ for all $i \in [\kappa]$. If so, x_b^j is marked as consistent. If the tokens do not abort and all the shares obtained in this phase are consistent, R proceeds to the reconstruction phase. Else it abort.

Reconstruction Phase: For $j \in [2\kappa]/T_{1-b}$, if there exists a unique x_b^j such that $A_b[i, j]+B_b[i, j] = x_b^j$, mark share j as a good column. If R obtains less than $\kappa + 1$ good shares, it aborts. Otherwise, let $x_b^{j_1}, \ldots, x_b^{j_{\kappa+1}}$ be any set of $\kappa + 1$ consistent shares. R computes $x_b \leftarrow \mathsf{Recon}(x_b^{j_1}, \ldots, x_b^{j_{\kappa+1}})$ and outputs $s_b = C_b \oplus x_b$.

Figure 10: Output Phase for $\Pi_{\mathrm{OT}}^{\mathrm{UC}}$

1. Upon corrupting $\mathcal{A}$, $\mathcal{S}$ emulates the role of $\mathcal{F}_{\mathrm{WRAP}}^{\mathrm{Stateless}}$ where upon receiving from $\mathcal{A}$ the messages $\{(\mathsf{Create}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_l, M_1)\}_{l\in[3\kappa]}$, $\mathcal{S}$ stores the codes of these tokens.

2. Next, $\mathcal{S}$ emulates the role of $\mathcal{F}_{\mathrm{WRAP}}^{\mathrm{Stateless}}$ for tokens $\{\mathsf{TK}_{\mathrm{R}}^{\mathsf{PRF},l'}\}_{l'\in[8\kappa^2]}$, where for each query $u \in \{0, 1\}^{5\kappa}$ made by $\mathcal{A}$ to token $\mathsf{TK}_{\mathrm{R}}^{\mathsf{PRF},l'}$, $\mathcal{S}$ returns a random v from $\{0, 1\}^{\kappa}$. $\mathcal{S}$ further stores the pair (u, v) in the query/answer list of $\mathsf{TK}_{\mathrm{R}}^{\mathsf{PRF},l'}$.

3. $\mathcal{S}$ generates the first message by following the code of the honest receiver with input $b = 0$.

4. Upon receiving the second message from $\mathcal{A}$, i.e. the commitments $(\mathsf{com}_{A_0}, \mathsf{com}_{B_0}, \mathsf{com}_{A_1}, \mathsf{com}_{B_1})$ and (C_0, C_1), it completes the execution by following the honest receiver's code.

5. Next, $\mathcal{S}$ tries to extract s_0 and s_1. For this, it first extracts matrices A_0, B_0, A_1, B_1 from the respective commitments as described in the simulation for the proof of Π_{OT}. More precisely, given any commitment β, v, it first checks if there exists a query/answer pair (u, v) that has already been recorded with respect to that token. If there exists such a query then the simulator sets the decommitted value to be $\beta \oplus \mathsf{Ext}(u)$, and $\perp$ otherwise. Next, to extract s_b, $\mathcal{S}$ proceeds as follows: For every $i \in [\kappa]$, it computes $A_b[i, j] \oplus B_b[i, j]$ for all $j \in [2\kappa]$ and marks that column j good if they all agree to the same value, say, γ_j. If it finds more than $\kappa + 1$ good columns, it reconstructs the secret x_b by using share reconstruction algorithm on $\{\gamma_j\}_{j\in\mathsf{good}}$. Otherwise, it sets x_b to $\perp$.

6. $\mathcal{S}$ computes $s_0 = C_0 \oplus x_0$ and $s_1 = C_1 \oplus x_1$ and sends (s_0, s_1) to the trusted party that computes $\mathcal{F}_{\mathrm{OT}}$ and halts, outputting whatever $\mathcal{A}$ does.

Next, we prove the correctness of our simulation in the following lemma.

Lemma 4.16. $\left\{\mathbf{View}^{\mathcal{F}^{\mathrm{Stateless}}_{\mathrm{WRAP}}}_{\Pi^{\mathrm{UC}}_{\mathrm{OT}},\mathcal{A},\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}} \stackrel{\mathrm{c}}{\approx} \left\{\mathbf{View}^{\mathcal{F}_{\mathrm{OT}}}_{\pi_{\mathrm{IDEAL}},\mathcal{S},\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}}.$

Proof: Our proof follows by a sequence of hybrid executions defined below.

Hybrid H_0: In this hybrid game there is no trusted party that computes functionality $\mathcal{F}_{\mathrm{OT}}$. Instead, we define a simulator $\mathcal{S}_0$ that receives the real input of the receiver and internally emulates the protocol $\Pi^{\mathrm{UC}}_{\mathrm{OT}}$ with the adversary $\mathcal{A}$ by simply following the honest receiver's strategy. Finally, the output of the receiver in the internal emulation is just sent to the external honest receiver (as part of the protocol Π_{H_0}) that outputs it as its output. Now, since the execution in this hybrid proceeds identically to the real execution, we have the following claim,

Claim 4.17. $\left\{\mathbf{View}^{\mathcal{F}^{\mathrm{Stateless}}_{\mathrm{WRAP}}}_{\Pi^{\mathrm{UC}}_{\mathrm{OT}},\mathcal{A},\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}} \approx \left\{\mathbf{View}_{\Pi_{\mathrm{H}_0},\mathcal{S}_0,\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}}.$

Hybrids $\mathrm{H}_{1,0} \ldots, \mathrm{H}_{1,8\kappa^2}$: We define a collection of hybrid executions such that for every $l' \in [8\kappa^2]$ hybrid $\mathrm{H}_{1,l'}$ is defined as follows. We modify the code of token $\mathsf{TK}^{\mathsf{PRF},l'}_{\mathrm{R}}$ by replacing the function $\mathsf{PRF}_{\gamma_{l'}}$ with a truly random function $f_{l'}$. In particular, given a query u the token responds with a randomly chosen κ bit string v, rather than running the original code of M_2. We maintain a list of $\mathcal{A}$'s queries and responses so that repeated queries will be consistently answered. It is simple to verify that the adversary's view in every two consecutive hybrid executions is computationally indistinguishable due to the security of the pseudorandom function embedded within $\mathsf{TK}^{\mathsf{PRF},l'}_{\mathrm{R}}$. Moreover, since the PRF key is hidden from the sender, it follows from the pseudorandomness property that the views in every two consecutive hybrid are computationally indistinguishable. As in the previous hybrid, the simulator hands the output of the receiver in the internal emulation to the external receiver as part of the protocol $\Pi_{\mathrm{H}_{1,l'}}$. More formally, we have the following claim,

Claim 4.18. *For every* $l' \in [8\kappa^2]$, $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_{1,l'-1}},\mathcal{S}_{1,l'-1},\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}} \stackrel{\mathrm{c}}{\approx} \left\{\mathbf{View}_{\Pi_{\mathrm{H}_{1,l'}},\mathcal{S}_{1,l'},\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}}.$

Hybrids $\mathrm{H}_{2,0} \ldots, \mathrm{H}_{2,8\kappa^2}$: This sequence of hybrids executions is identical to hybrid $\mathrm{H}_{1,8\kappa^2}$ except that here $\mathcal{S}_2$ aborts if two queries made by $\mathcal{A}$ to the token $\mathsf{TK}^{\mathsf{PRF},l'}_{\mathrm{R}}$ results in the same response. Using a proof analogous to Lemma 4.11, we obtain the following claim.

Claim 4.19. *For every* $l' \in [8\kappa^2]$, $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_{2,l'-1}},\mathcal{S}_{2,l'-1},\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}} \stackrel{\mathrm{s}}{\approx} \left\{\mathbf{View}_{\Pi_{\mathrm{H}_{2,l'}},\mathcal{S}_{2,l'},\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}}.$

Hybrid H_3: In this hybrid, $\mathcal{S}_3$ proceeds identically to $\mathcal{S}_{2,8\kappa^2}$ using the honest receiver's input b with the exception that it does not report the output of the receiver as what is computed in the emulation by the simulator. Instead, $\mathcal{S}_3$ follows the code of the actual simulator to extract (s_0, s_1) and sets the receiver's output as s_b. Note that the view of the adversary is identical in both hybrids $\mathrm{H}_{2,8\kappa^2}$ and H_3. Therefore, to prove the indistinguishability of the joint output distribution, it suffices to show that the output of the honest receiver is the same. On a high-level, this will follow from the fact that if the honest receiver does not abort then the two checks performed by the receiver, namely, the shares validity check and the shares consistency check were successful, which would imply that there are at least $\kappa + 1$ good columns from which the simulator can extract the shares. Finally, we conclude that the reconstruction performed by the honest receiver and the simulator will yield the same value for s_b.

More formally, we argue indistinguishability conditioned on when the two consistency checks pass in the execution emulated by the simulator (in the event at least one of them do not pass, the receiver aborts and indistinguishability directly holds). Then, the following hold for any $s \in \omega(\log n)$:

Step 1: Since the shares validity check passed, following a standard cut-and-choose argument, it holds except with probability $2^{-O(s)}$ that there are at least $\kappa - s$ rows for which $\phi(A_b[i,\cdot]) + \phi(B_b[i,\cdot]) = 0$. In fact, it suffices if this holds at least for one row, say i^*. For $b \in \{0,1\}$, let the secret corresponding to $A_b[i^*,\cdot] + B_b[i^*,\cdot]$ be $\tilde{s}_b$.

Step 2: If for any column $j \in [2\kappa]$ and $b \in \{0,1\}$ there exists a value γ_j such that for all $i \in [\kappa]$

$$\gamma_b[j] = A_b[i,j] + B_b[i,j].$$

Combining with Step 1, we can conclude that $\gamma_b[j] = A_b[i^*,j] + B_b[i^*,j]$. Furthermore, if either the receiver or the simulator tries to extract the share corresponding to that column it will extract $\gamma_b[j]$ since the commitments made by the sender are binding. Therefore, we can conclude that if either the receiver or the simulator tries to reconstruct the secret for any $b \in \{0,1\}$, it will reconstruct only with shares in $\{\gamma_b[j]\}_{j \in J}$ which implies that they reconstruct only $\tilde{s}_b$.

Step 3: Now, since the shares consistency check passed, following another cut-and-choose argument, it holds except with probability $2^{-O(s)}$ that there is a set J of at least $2\kappa - s$ columns such that for any $j \in J$ the tokens do not abort on a valid input from the receiver and yield consistent values for both $b_j = 0$ and $b_j = 1$. This means that if the honest receiver selects $3\kappa/4$ columns with input as its real input b, the receiver is guaranteed to find at least $\kappa + 1$ indices in J. Furthermore, there will be $\kappa + 1$ columns in J for both inputs for the simulator to extract and when either of them extract they can only extract $\tilde{s}_b$.

Then to prove indistinguishability in this hybrid, it suffices to prove that the simulator reconstructs s_b if and only if the receiver extracts s_b and this follows directly from Step 3 in the proceeding argument, since there is a unique value $\tilde{s}_b$ that either of them can reconstruct and they will reconstruct that value with probability $1 - 2^{-O(s)}$ if the two checks pass. As the checks are independent of the real input of the receiver, indistinguishability of the hybrids follow.

Claim 4.20. $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_{2,8\kappa^2}},\mathcal{S}_{2,8\kappa^2},\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}} \stackrel{\mathrm{s}}{\approx} \left\{\mathbf{View}_{\Pi_{\mathrm{H}_3},\mathcal{S}_3,\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}}$.

Hybrid H_4: In this hybrid, $\mathcal{S}_4$ proceeds identically to $\mathcal{S}_3$ with the exception that the simulator sets the receiver's input in the main execution as 0 instead of the real input b. Finally, it reconstructs s_b and sets that as the honest receiver's output. It follows analogous to the proof of Lemma 4.3 that the output of H_3 and H_4 are statistically-close. Therefore, we have the following claim,

Claim 4.21. $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_3},\mathcal{S}_3,\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}} \stackrel{\mathrm{s}}{\approx} \left\{\mathbf{View}_{\Pi_{\mathrm{H}_4},\mathcal{S}_4,\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}}$.

Hybrid H_5: In this hybrid, we consider the simulation. Observe that our simulator proceeds identically to the simulation with $\mathcal{S}_4$ with the exception that it feeds the extracted values s_0 and s_1 to the ideal functionality while $\mathcal{S}_4$ instead just outputs s_b. Furthermore, the ideal simulator sends (s_0, s_1) to the $\mathcal{F}_{\rm OT}$ functionality. It follows from our simulation that the view of the adversary in $\rm H_5$ and the ideal execution are identically distributed. Furthermore, for both $b = 0$ and $b = 1$ we know that the value s_b extracted by the simulator and the value output by the honest receiver in the ideal execution are equal. Therefore, we can conclude that the output of $\rm H_4$ and the ideal execution are identically distributed.

Claim 4.22. $\left\{\mathbf{View}_{\Pi_{\rm H_4},\mathcal{S}_4,\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}} \approx \left\{\mathbf{View}^{\mathcal{F}_{\rm OT}}_{\pi_{\rm IDEAL},\mathcal{S},\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}}$.

■

Simulating the corrupted $\rm R$. We begin with describing our simulation:

1. Upon corrupting $\mathcal{A}$, $\mathcal{S}$ emulates the role of $\mathcal{F}^{\rm Stateless}_{\rm WRAP}$ for tokens $\{\mathsf{TK}^{\mathsf{PRF},l}_{\rm S}\}_{l\in[3\kappa]}$ using truly random functions, where for each query $u \in \{0,1\}^{5\kappa}$ made by $\mathcal{A}$ to token $\mathsf{TK}^{\mathsf{PRF},l}_{\rm S}$, $\mathcal{S}$ returns a random v from $\{0,1\}^{\kappa}$. $\mathcal{S}$ further stores the pair (u, v) in the query/answer list of $\mathsf{TK}^{\mathsf{PRF},l}_{\rm S}$.
2. Next, $\mathcal{S}$ emulates the role of $\mathcal{F}^{\rm Stateless}_{\rm WRAP}$ where upon receiving from $\mathcal{A}$ the message $\{(\mathsf{Create}, \mathsf{sid}, \rm R, S, \mathsf{mid}_{l'}, M_2)\}_{l'\in[8\kappa^2]}$, $\mathcal{S}$ stores the code of these tokens.
3. Upon receiving the first message from $\mathcal{A}$, i.e. the commitments com_{b_j} and com_{c_i} where $i \in [\kappa]$ and $j \in [2\kappa]$, $\mathcal{S}$ tries to extract b. For this, just as in previous simulations, it first extracts all the b_j values and then sets the receiver's input as that bit that occurs at least $\kappa + 1$ times among the b_j's. If no such bit exists, it sets b to be random. Next it sends b to the $\mathcal{F}_{\rm OT}$ functionality to obtain s_b, and completes the protocol following the honest sender's code with inputs (s_0, s_1) where s_{1-b} is set to random. In particular, it computes $C_b = x_b \oplus s_b$ and sets C_{1-b} to a random string.

Next, we sketch the correctness of our simulation in the following lemma.

Lemma 4.23. $\left\{\mathbf{View}^{\mathcal{F}^{\rm Stateless}_{\rm WRAP}}_{\Pi^{\rm UC}_{\rm OT},\mathcal{A},\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}} \stackrel{\rm c}{\approx} \left\{\mathbf{View}^{\mathcal{F}_{\rm OT}}_{\pi_{\rm IDEAL},\mathcal{S},\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}}$.

Proof: Our proof follows by a sequence of hybrid executions defined below.

Hybrid $\rm H_0$: In this hybrid game there is no trusted party that computes functionality $\mathcal{F}_{\rm OT}$. Instead, we define a simulator $\mathcal{S}_0$ that receives the real input of the sender and internally emulates the protocol $\Pi^{\rm UC}_{\rm OT}$ with the adversary $\mathcal{A}$ by simply following the honest sender's strategy. Finally, the output of the sender in the internal emulation is just sent to the external honest sender (as part of the protocol $\Pi_{\rm H_0}$) that outputs it as its output. Now, since the execution in this hybrid proceeds identically to the real execution, we have the following claim,

Claim 4.24. $\left\{\mathbf{View}^{\mathcal{F}^{\rm Stateless}_{\rm WRAP}}_{\Pi^{\rm UC}_{\rm OT},\mathcal{A},\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}} \approx \left\{\mathbf{View}_{\Pi_{\rm H_0},\mathcal{S}_0,\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}}$.

Hybrids $\rm H_{1,0}\ldots, H_{1,3\kappa}$: We define a collection of hybrid executions such that for every $l \in [3\kappa]$ hybrid $\rm H_{1,l}$ is defined as follows. We modify the code of token $\mathsf{TK}^{\mathsf{PRF},l}_{\rm S}$ by replacing the function PRF_{γ_l} with a truly random function f_l. In particular, given a query u the token responds with a randomly chosen κ bit string v, rather than running the original code of M_1. We maintain a list of $\mathcal{A}$'s queries and responses so that repeated queries will be consistently answered. In addition, the code of token TK_l (for $l \leq 2\kappa$) or $\widehat{\mathsf{TK}}_{l-2\kappa}$ (for $2\kappa+1 \leq l \leq 3\kappa$) is modified, as now this token does not run a check with respect to the PRF that is embedded within token $\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}$ but with respect to the random function f_l. It is simple to verify that the adversary's view in every two consecutive hybrid executions is computationally indistinguishable due to the security of the pseudorandom function PRF_{γ_l}. Moreover, since the PRF key is hidden from the receiver, it follows from the pseudorandomness property that the views in every two consecutive hybrid are computationally indistinguishable. As in the previous hybrid, the simulator hands the output of the sender in the internal emulation to the external receiver as part of the protocol $\Pi_{\mathrm{H}_{1,l}}$. More formally, we have the following claim,

Claim 4.25. *For every $l \in [3\kappa]$,* $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_{1,l-1}},\mathcal{S}_{1,l-1},\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}} \stackrel{c}{\approx} \left\{\mathbf{View}_{\Pi_{\mathrm{H}_{1,l}},\mathcal{S}_{1,l},\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}}$.

Hybrids $\mathrm{H}_{2,0}\ldots,\mathrm{H}_{2,3\kappa}$: This sequence of hybrids executions is identical to hybrid $\mathrm{H}_{1,3\kappa}$ except that here $\mathcal{S}_2$ aborts if two queries made by $\mathcal{A}$ to the token $\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}$ results in the same response. Using a proof analogous to Lemma 4.11, we obtain the following claim.

Claim 4.26. *For every $l \in [3\kappa]$,* $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_{2,l-1}},\mathcal{S}_{2,l-1},\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}} \stackrel{s}{\approx} \left\{\mathbf{View}_{\Pi_{\mathrm{H}_{2,l}},\mathcal{S}_{2,l},\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}}$.

Hybrid H_3: In this hybrid execution, simulator $\mathcal{S}_3$ plays the role of the sender as in hybrid $\mathrm{H}_{2,3\kappa}$ except that it extracts the adversary's input bit b as carried out in the simulation by $\mathcal{S}$ and the challenge string c. Clearly, this does not make any difference to the receiver's view which implies that,

Claim 4.27. $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_{2,3\kappa}},\mathcal{S}_{2,3\kappa},\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}} \stackrel{s}{\approx} \left\{\mathbf{View}_{\Pi_{\mathrm{H}_3},\mathcal{S}_3,\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}}$.

Hybrid H_4: In this hybrid execution, the simulator instead of emulating the original tokens $\{\mathsf{TK}_j\}_{j\in[2\kappa]}$, simulator $\mathcal{S}_4$ emulates functionalities $\{\widetilde{\mathsf{TK}}_j\}_{j\in[2\kappa]}$ in the following way. For all $j \in [2\kappa]$, if $\widetilde{\mathsf{TK}}_j$ is queried on $(b_j, \mathsf{decom}_{b_j})$ and decom_{b_j} is verified correctly, $\mathcal{S}_4$ outputs the column

$$(A_{b_j}[\cdot,j], \mathsf{decom}_{A_{b_j}[\cdot,j]}, B_{b_j}[\cdot,j], \mathsf{decom}_{B_{b_j}[\cdot,j]})$$

where b_j is the bit extracted by $\mathcal{S}_4$ as in the prior hybrid. Otherwise, if $\widetilde{\mathsf{TK}}_j$ is queried on $(1-b_j, \mathsf{decom}_{1-b_j})$ then $\mathcal{S}_4$ outputs $\perp$. Following the same argument as in Claim 4.12 it follows that the commitments made by the receiver are binding and thus a receiver will not be able to produce decommitments to obtain the value corresponding to $1-b_j$. Therefore, we have the following claim.

Claim 4.28. $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_3},\mathcal{S}_3,\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}} \stackrel{s}{\approx} \left\{\mathbf{View}_{\Pi_{\mathrm{H}_4},\mathcal{S}_4,\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}}$.

Hybrid H_5: In this hybrid execution, instead of emulating the original tokens $\{\widehat{\mathsf{TK}}_i\}_{i\in[\kappa]}$, simulator $\mathcal{S}_5$ emulates functionalities $\{\overline{\mathsf{TK}}_i\}_{i\in[\kappa]}$ in the following way. For all $i \in [\kappa]$, if $\overline{\mathsf{TK}}_i$ is queried on $(c_i, \mathsf{decom}_{c_i})$ and decom_{c_i} is verified correctly, $\mathcal{S}_5$ outputs the row

$$\begin{array}{ll}(A_0[i,\cdot], \mathsf{decom}_{A_0[i,\cdot]}, A_1[i,\cdot], \mathsf{decom}_{A_i[i,\cdot]}), & \text{if } c_i = 0\\ (B_0[i,\cdot], \mathsf{decom}_{B_0[i,\cdot]}, B_1[i,\cdot], \mathsf{decom}_{B_i[i,\cdot]}), & \text{if } c_i = 1\end{array}$$

where c_i is the bit extracted by $\mathcal{S}_5$ as in the prior hybrid. Otherwise, if $\overline{\mathsf{TK}}_i$ is queried on $(1-c_i, \mathsf{decom}_{1-c_i})$ then $\mathcal{S}_5$ outputs $\perp$. Indistinguishability follows using the same argument as in the previous hybrid. Therefore, we have the following claim.

Claim 4.29. $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_4},\mathcal{S}_4,\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}} \stackrel{s}{\approx} \left\{\mathbf{View}_{\Pi_{\mathrm{H}_5},\mathcal{S}_5,\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}}$.

Hybrid H_6: In this hybrid, the simulator $\mathcal{S}_6$ chooses an independent random string $x^* \leftarrow \mathbb{Z}_p$ instead of generating the matrices A_{1-b} and B_{1-b} according to the shares of x_{1-b}. We remark that $C_{1-b} = s_{1-b} \oplus x_{1-b}$ is still computed as in H_5 with x_{1-b}.

Claim 4.30. $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_5},\mathcal{S}_5,\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}} \stackrel{\mathrm{s}}{\approx} \left\{\mathbf{View}_{\Pi_{\mathrm{H}_6},\mathcal{S}_6,\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}}$.

Proof: Let $\tilde{A}_{1-b}, \tilde{B}_{1-b}$ contain the same entries as A_{1-b}, B_{1-b} in H_5 with the exception that the entries whose decommitments have been removed both in $\overline{\mathsf{TK}}$ and $\widetilde{\mathsf{TK}}$ as described in hybrids H_4 and H_5 are set to $\perp$. More precisely, given the extracted values for b_j's and c_i's, for every $j \in [2\kappa]$ such that $b_j = b$, $\tilde{A}_{1-b}(i,j) = \perp$ if $c_i = 1$ and $\tilde{B}_{1-b}(i,j) = \perp$ if $c_i = 0$ for all $i \in [\kappa]$.

Observe that, for every i, j, either $\tilde{A}_{1-b}[i,j] = A_{1-b}[i,j]$ or $\tilde{A}_{1-b}[i,j] = \perp$. The same holds for the $\tilde{B}_{1-b}$. We claim that the information of at most κ shares of x_{1-b} is present in matrices $\tilde{A}_{1-b}, \tilde{B}_{1-b}$. To this end, for every column j such that $b_j \neq 1-b$ and for every row i, depending on c_i, either $\tilde{A}_{1-b}[i,j] = \perp$, or $\tilde{B}_{1-b}[i,j] = \perp$. For every pair i, j, since $A_{1-b}[i,j]$ and $B_{1-b}[i,j]$ are both uniformly distributed, obtaining the value for at most one of them keeps $A_{1-b}[i,j] + B_{1-b}[i,j]$ statistically hidden. Now, since $b_j \neq 1-b$ for at least $\kappa+1$ shares, it follows that at least $\kappa+1$ shares of x_{1-b} are hidden. In other words, at most κ shares of x_{1-b} can be obtained by the receiver in H_5. Analogously at most κ shares of x^* are obtained in H_6. From our secret-sharing scheme, it follows that κ shares information theoretically hides the value. Therefore, the decommitments obtained by the receiver in H_5 and H_6 and identically distributed. The claim now follows from the fact that the commitments to the matrices $(\mathsf{com}_{A_0}, \mathsf{com}_{B_0}, \mathsf{com}_{A_1}, \mathsf{com}_{B_1})$ are statistically-hiding. □

Hybrid H_7: In this hybrid execution simulator $\mathcal{S}_7$ does not know the sender's inputs (s_0, s_1), but rather communicates with a trusted party that computes $\mathcal{F}_{\mathrm{OT}}$. $\mathcal{S}_7$ behaves exactly as $\mathcal{S}_6$, except that when extracting the bit b, it sends it to the trusted party which sends back s_b. Moreover, $\mathcal{S}_7$ uses random values for s_{1-b} and C_{1-b}. Note that since the value committed to in the matrices corresponding to $1-b$ is independent of x_{1-b}, this hybrid is identically distributed to the previous hybrid. We conclude with the following claim.

Claim 4.31. $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_6},\mathcal{S}_6,\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}} \equiv \left\{\mathbf{View}_{\Pi_{\mathrm{H}_7},\mathcal{S}_7,\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}}$.

Hybrid H_8: In this hybrid execution, tokens $\{\mathsf{TK}_j\}_{j\in[2\kappa]}$ are emulated instead of tokens $\{\widetilde{\mathsf{TK}}_j\}_{j\in[2\kappa]}$. In addition, tokens $\{\widehat{\mathsf{TK}}_i\}_{i\in[\kappa]}$ are emulated instead of tokens $\{\widetilde{\mathsf{TK}}_i\}_{i\in[\kappa]}$. Due to similar claims as above, it holds that

Claim 4.32. $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_7},\mathcal{S}_7,\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}} \stackrel{\mathrm{c}}{\approx} \left\{\mathbf{View}_{\Pi_{\mathrm{H}_8},\mathcal{S}_8,\mathcal{Z}}(n)\right\}_{n\in\mathbb{N}}$.

Finally, we note that hybrid H_8 is identical to the simulated execution which concludes the proof. ■

■

On relying on one-way functions. In this protocol the only place where one-way permutations are used is in the commitments made by the sender in the second round of the protocol via a non-interactive perfectly-binding commitment. This protocol can be easily modified to rely on statistically-binding commitments which have 2-round constructions based on one-way functions [Nao91]. Specifically, since the sender commits to its messages only in the second-round, the receiver can provide the first message of the 2-round commitment scheme along with the first message of the protocol.

5 Acknowledgements

We thank the anonymous reviewers of TCC who identified an issue in an earlier version of this work regarding the reusability of tokens. We further thank Yuval Ishai, Amit Sahai, and Vipul Goyal for fruitful discussions regarding token-based cryptography.

The first author acknowledges support from the Israel Ministry of Science and Technology (grant No. 3-10883). The second author acknowledges support from the Danish National Research Foundation and the National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation and from the Center for Research in Foundations of Electronic Markets (CFEM), supported by the Danish Strategic Research Council. In addition, this work was done in part while visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant CNS-1523467.

References

[AAG+14] Shashank Agrawal, Prabhanjan Ananth, Vipul Goyal, Manoj Prabhakaran, and Alon Rosen. Lower bounds in the hardware token model. In *TCC*, pages 663–687, 2014.

[BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195, 2004.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[CDPW06] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. *IACR Cryptology ePrint Archive*, 2006:432, 2006.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO*, pages 19–40, 2001.

[CGP15] Ran Canetti, Shafi Goldwasser, and Oxana Poburinnaya. Adaptively secure two-party computation from indistinguishability obfuscation. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 557–585, 2015.

[CGS08] Nishanth Chandran, Vipul Goyal, and Amit Sahai. New constructions for UC secure computation using tamper-proof hardware. In *EUROCRYPT*, pages 545–562, 2008.

[CKL06] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *J. Cryptology*, 19(2):135–167, 2006.

[CKS+14] Seung Geol Choi, Jonathan Katz, Dominique Schröder, Arkady Yerukhimovich, and Hong-Sheng Zhou. (efficient) universally composable oblivious transfer using a minimal number of stateless tokens. In *TCC*, pages 638–662, 2014.

[CPS07] Ran Canetti, Rafael Pass, and Abhi Shelat. Cryptography from sunspots: How to use an imperfect reference string. In *FOCS*, pages 249–259, 2007.

[DKM11] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. Unconditional and composable security using a single stateful tamper-proof hardware token. In *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings*, pages 164–181, 2011.

[DKR15] Dana Dachman-Soled, Jonathan Katz, and Vanishree Rao. Adaptively secure, universally composable, multiparty computation in constant rounds. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 586–613, 2015.

[DMRV13] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Muthuramakrishnan Venkitasubramaniam. Adaptive and concurrent secure computation from new adaptive, non-malleable commitments. In *ASIACRYPT*, pages 316–336, 2013.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *PKC*, pages 416–431, 2005.

[DZ13] Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In *TCC*, pages 621–641, 2013.

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In *TCC*, pages 74–94, 2014.

[GIMS10] Vipul Goyal, Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. Interactive locking, zero-knowledge pcps, and unconditional cryptography. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pages 173–190, 2010.

[GIS+10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, pages 308–326, 2010.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *STOC*, pages 25–32, 1989.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.

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

[Gol04] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[GP15] Sanjam Garg and Antigoni Polychroniadou. Two-round adaptively secure MPC from indistinguishability obfuscation. In *TCC*, pages 614–637, 2015.

[HHRS15] Iftach Haitner, Jonathan J. Hoch, Omer Reingold, and Gil Segev. Finding collisions in interactive protocols - tight lower bounds on the round and communication complexities of statistically hiding commitments. *SIAM J. Comput.*, 44(1):193–242, 2015.

[HK07] Omer Horvitz and Jonathan Katz. Universally-composable two-party computation in two rounds. In *CRYPTO*, pages 111–129, 2007.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591, 2008.

[Kat07] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In *EUROCRYPT*, pages 115–128, 2007.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

[KLP07] Yael Tauman Kalai, Yehuda Lindell, and Manoj Prabhakaran. Concurrent composition of secure protocols in the timing model. *J. Cryptology*, 20(4):431–492, 2007.

[KO04] Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In *CRYPTO*, pages 335–354, 2004.

[Kol10] Vladimir Kolesnikov. 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*, pages 327–342, 2010.

[Lam79] Leslie Lamport. Constructing digital signatures from a one-way function. *Technical Report CSL-98, SRI International*, 1979.

[Lin03] Yehuda Lindell. General composition and universal composability in secure multi-party computation. In *FOCS*, pages 394–403, 2003.

[LPV09] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. A unified framework for concurrent security: universal composability from stand-alone non-malleability. In *STOC*, pages 179–188, 2009.

[MS08] Tal Moran and Gil Segev. David and goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. In *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, pages 527–544, 2008.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991.

[ORS15] Rafail Ostrovsky, Silas Richelson, and Alessandra Scafuro. Round-optimal black-box two-party computation. *IACR Cryptology ePrint Archive*, 2015:553, 2015.

[PW09] Rafael Pass and Hoeteck Wee. Black-box constructions of two-party protocols from one-way functions. In *TCC*, pages 403–418, 2009.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FCOS*, pages 162–167, 1986.

A Secure Two-Party Computation

We briefly present the standard definition for secure multiparty computation in the plain model and refer to [Gol04, Chapter 7] for more details and motivating discussions. A two-party protocol problem is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a *functionality* and denote it $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$. That is, for every pair of inputs (x, y), the output-vector is a random variable $(f_1(x, y), f_2(x, y))$ ranging over pairs of strings where P_1 receives $f_1(x, y)$ and P_2 receives $f_2(x, y)$. We use the notation $(x, y) \mapsto (f_1(x, y), f_2(x, y))$ to describe a functionality. We prove the security of our protocols in the settings of *malicious* computationally bounded adversaries. Security is analyzed by comparing what an adversary can do in a *real* protocol execution to what it can do in an *ideal* scenario.

Execution in the ideal model. In an ideal execution, the parties submit inputs to a trusted party, that computes the output. An honest party receives its input for the computation and just directs it to the trusted party, whereas a corrupted party can replace its input with any other value of the same length. Since we do not consider fairness, the trusted party first sends the outputs of the corrupted parties to the adversary, and the adversary then decides whether the honest parties would receive their outputs from the trusted party or an *abort* symbol $\perp$. Let f be a two-party functionality where $f = (f_1, f_2)$, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine, and let $I \subset [2]$ be the set of corrupted parties (either P_1 is corrupted or P_2 is corrupted or neither). Then, the *ideal execution of* f on inputs (x, y), auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted $\mathbf{IDEAL}_{f,\mathcal{A}(z),I}(\kappa, x, y)$, is defined as the output pair of the honest party and the adversary $\mathcal{A}$ from the above ideal execution.

Execution in the real model. In the real model there is no trusted third party and the parties interact directly. The adversary $\mathcal{A}$ sends all messages in place of the corrupted party, and may follow an arbitrary polynomial-time strategy. The honest parties follow the instructions of the specified protocol π.

Let f be as above and let π be a two-party protocol for computing f. Furthermore, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine and let I be the set of corrupted parties. Then, the *real execution of* π on inputs (x, y), auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted $\mathbf{REAL}_{\pi,\mathcal{A}(z),I}(\kappa, x, y)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the real execution of π.

Security as emulation of a real execution in the ideal model. Having defined the ideal and real models, we can now define security of protocols. Loosely speaking, the definition asserts that a secure party protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated by saying that adversaries in the ideal model are able to simulate executions of the real-model protocol.

Definition A.1. *Let f and π be as above. Protocol π is said to* securely compute f with abort in the presence of malicious adversaries *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ for the ideal model, such that for every $I \subset [2]$,*

$$\left\{\mathbf{IDEAL}_{f,\mathcal{S}(z),I}(\kappa, x, y)\right\}_{\kappa\in\mathbb{N},x,y,z\in\{0,1\}^*} \stackrel{c}{\approx} \left\{\mathbf{REAL}_{\pi,\mathcal{A}(z),I}(\kappa, x, y)\right\}_{\kappa\in\mathbb{N},x,y,z\in\{0,1\}^*}$$

where κ is the security parameter.

The $\mathcal{F}$-hybrid model. In order to construct some of our protocols, we will use secure two-party protocols as subprotocols. The standard way of doing this is to work in a "*hybrid model*" where parties both interact with each other (as in the real model) and use trusted help (as in the ideal model). Specifically, when constructing a protocol π that uses a subprotocol for securely computing some functionality $\mathcal{F}$, we consider the case that the parties run π and use "ideal calls" to a trusted party for computing $\mathcal{F}$. Upon receiving the inputs from the parties, the trusted party computes $\mathcal{F}$ and sends all parties their output. Then, after receiving these outputs back from the trusted party the protocol π continues. Let $\mathcal{F}$ be a functionality and let π be a two-party protocol that uses ideal calls to a trusted party computing $\mathcal{F}$. Furthermore, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine. Then, the *$\mathcal{F}$-hybrid execution of* π on inputs (x, y), auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted $\mathbf{Hybrid}_{\pi^{\mathcal{F}},\mathcal{A}(z)}(\kappa, x, y)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the hybrid execution of π with a trusted party computing $\mathcal{F}$. By the composition theorem of [Can00] any protocol that securely implements $\mathcal{F}$ can replace the ideal calls to $\mathcal{F}$.

A.1 UC Security

Some of our proofs are given in the stronger Universal Composability (UC) setting. For completeness we briefly recall this framework, for more details we refer to [Can01].

Environment. The model of execution includes a special entity called the UC-environment (or environment) $\mathcal{Z}$. The environment "manages" the whole execution: it invokes all the parties at the beginning of the execution, generates all inputs and reads all outputs, and finally produces an output for the whole concurrent execution. Intuitively, the environment models the "larger world" in which the concurrent execution takes place (e.g., for a distributed computing task over the Internet, the environment models all the other activities occurring on the Internet at the same time).

Adversarial behavior. The model of execution also includes a special entity called the adversary, that represents adversarial activities that are directly aimed at the protocol execution under consideration. While honest parties only communicate with the environment through the input/output of the functions they compute, the adversary is also able to exchange messages with the environment in an arbitrary way throughout the computation.[8] Furthermore, the adversary controls the scheduling of the delivery of all messages exchanged between parties (where messages sent by the environment are directly delivered). Technically, this is modeled by letting the adversary read the outgoing message tapes of all parties and decide whether or not and when (if at all) to deliver the message to the recipient, therefore the communication is asynchronous and lossy. However, the adversary cannot insert messages and claim arbitrary sender identity. In other words, the communication is authenticated.

Protocol execution. The *execution of a protocol π with the environment $\mathcal{Z}$, adversary $\mathcal{A}$ and trusted party $\mathcal{G}$* proceeds as follows. The environment is the first entity activated in the execution, who then activates the adversary, and invokes other honest parties. At the time an honest party is invoked, the environment assigns it a unique identifier and inquiries the adversary whether it wants to corrupt the party or not. To start an execution of the protocol π, the environment initiates a *protocol execution session*, identified by a session identifier sid, and activates all the participants in that session. An activated honest party starts executing the protocol π thereafter and has access to the trusted party $\mathcal{G}$. We remark that in the UC model the environment only initiates one protocol execution session.

Invoking parties. The environment invokes an honest party by passing input (*invoke*, P_i) to it. P_i is the globally unique identity for the party and is picked dynamically by the environment at the time it is invoked. Immediately after that, the environment notifies the adversary of the invocation of P_i by sending the message (*invoke*, P_i) to it, who can then choose to corrupt this party by replying (*corrupt*, P_i). Note that here as the adversary is static, parties are corrupted only when they are "born" (invoked).

Session initiation. To start an execution of protocol π, the environment selects a subset U of parties that has been invoked so far. For each party $P_i \in U$, the environment activates P_i by sending a start-session message (*start-session*, P_i, sid, $c_{i,sid}$, $x_{i,sid}$) to it, where sid is a session id that identifies this execution. We remark that in the UC model, the environment starts only one session, and hence all the activated parties have the same session id.

Honest party execution. An honest party P_i, upon receiving (*start-session*, P_i, sid, $c_{i,sid}$, $x_{i,sid}$), starts executing its code $c_{i,sid}$ input $x_{i,sid}$. During the execution,

- The environment can read P_i's output tape and at any time may pass additional inputs to P_i;
- According to its code, P_i can send messages (delivered by the adversary) to the other parties in the session, in the format (P_i, P_j, s, content),[9] where P_j is the identity of the receiver;
- According to its code, P_i can send an input to the trusted party in the format (P_i, $\mathcal{F}$, s, input).

Adversary execution. Upon activation, the adversary may perform one of the following activities at any time during the execution.

[8] Through its interaction with the environment, the adversary is also able to influence the inputs to honest parties indirectly.

[9] The session id in the messages enables the receiver to correctly de-multiplexing a message to its corresponding session, even though the receiver may involve in multiple sessions simultaneously.

- The adversary can read the outgoing communication tapes of all honest parties and decides to deliver some of the messages.
- The adversary can exchange arbitrary messages with the environment.
- The adversary can read the inputs, outputs and incoming messages of a corrupted party, and instruct the corrupted party for any action.

Output. The environment outputs a final result for the whole execution in the end.

In the execution of protocol π with security parameter $\kappa \in \mathbb{N}$, environment $\mathcal{Z}$, adversary $\mathcal{A}$ and trusted party $\mathcal{G}$, we define $\mathbf{View}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa)$ to be the random variable describing the output of the environment $\mathcal{Z}$, resulting from the execution of the above procedure.

Let $\mathcal{F}$ be an ideal functionality; we denote by π_{IDEAL} the protocol accessing $\mathcal{F}$, called as the ideal protocol. In π_{IDEAL} parties simply interacts with $\mathcal{F}$ with their private inputs, and receive their corresponding outputs from the functionality at the end of the computation. Then the *ideal execution* of the functionality $\mathcal{F}$ is just the execution of the ideal protocol π_{IDEAL} with environment $\mathcal{Z}$, adversary $\mathcal{S}$ and trusted party $\mathcal{F}$. The output of the execution is thus $\mathbf{View}^{\mathcal{F}}_{\pi_{\mathrm{IDEAL}},\mathcal{S},\mathcal{Z}}(\kappa)$. On the other hand, the real model execution does not require the aid of any trusted party. Let π be a multi-party protocol implementing $\mathcal{F}$. Then, the *real execution* of π is the execution of π with security parameter κ, environment $\mathcal{Z}$ and adversary $\mathcal{A}$, whose output is the random variable $\mathbf{View}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa)$. Additionally, the *$\mathcal{G}$-Hybrid model execution* of a protocol π is the execution of π with security parameter κ, environment $\mathcal{Z}$ and adversary $\mathcal{A}$ and ideal functionality $\mathcal{G}$.

Security as emulation of a real model execution in the ideal model. Loosely speaking, a protocol *securely realizes* an ideal functionality if it *securely emulates* the ideal protocol π_{IDEAL}. This is formulated by saying that for every adversary $\mathcal{A}$ in the real model, there exists an adversary $\mathcal{S}$ (a.k.a. *simulator*) in the ideal model, such that no environment $\mathcal{Z}$ can tell apart if it is interacting with $\mathcal{A}$ and parties running the protocol, or $\mathcal{S}$ and parties running the ideal protocol π_{IDEAL}.

Definition A.2. (UC security) *Let $\mathcal{F}$ and π_{IDEAL} be defined as above, and π be a multi-party protocol in the $\mathcal{G}$-hybrid model. Then protocol π UC realizes $\mathcal{F}$ in $\mathcal{G}$-hybrid model, if for every uniform* PPT *adversary $\mathcal{A}$, there exists a uniform* PPT*simulator $\mathcal{S}$, such that for every non-uniform* PPT *environment $\mathcal{Z}$, the following two ensembles are computationally indistinguishable,*

$$\left\{\mathbf{View}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa)\right\}_{\kappa\in\mathbb{N}} \stackrel{\mathrm{c}}{\approx} \left\{\mathbf{View}^{\mathcal{F}}_{\pi_{\mathrm{IDEAL}},\mathcal{S},\mathcal{Z}}(\kappa)\right\}_{\kappa\in\mathbb{N}}.$$

B A Counter Example to [GIS+10]

In the following we provide a counter example to the sender simulation of the protocol in [GIS+10], Section 5.1.2, where a protocol with stateless tokens in the UC setting is presented. On a high-level, the issue is because the extraction procedure of the sender's inputs is achieved through monitoring the queries to the PRF token as opposed to running the actual OT token sent by the sender. This affects the distribution of the receiver's output as computed by the simulation. Below, we describe the counter example where the distribution of what the receiver learns is different in the real world and ideal. Consider the following sender's abort strategy:

- Pick $z_1, z_2, ..., z_{n-1}$ and Δ at random.

- The inputs of the first $n-1$ tokens are $z_1, z_1+\Delta, \ldots, z_{n-1}, z_{n-1}+\Delta$.
- Let $z_1+\ldots+z_{n-1}=a$ and $z_1+\ldots+z_{n-1}+\Delta=b$.
- The inputs to the nth token are some fixed values c (when $b_n=0$) and d (when $b_n=1$), where $c+d\neq\Delta$.

In addition, the sender defines the following functionality relative to the OT tokens. The first $n-1$ tokens never abort, yet the nth token aborts whenever the input $b_n=1$. Let $s_0=0$ and $s_1=1$ (we remark that we are not concerned about the actual inputs of the sender, but focus on the sum of the sender's inputs embedded within the OT tokens that the receiver learns). We next examine the honest receiver's output in both the real and ideal worlds. First, in the real world the honest receiver learns an output only if $b_n=0$ (since the nth token aborts whenever $b_n=1$). We consider two cases:

Case 1: The receiver's input is $b=0$. Then $b_n=0$ with probability $1/2$, and $b_n=1$ with probability $1/2$. Moreover, when $b_n=0$, the sum of the outputs from the tokens OT that is obtained by the receiver is $a+c$. This is because when $b_n=0$, then, $b_1+\ldots+b_{n-1}=0$, and the receiver learns a as the sum of the outputs of the first $n-1$ tokens and c from the nth token. On the other hand, when $b_n=1$ then the receiver aborts since the nth token aborts.

Case 2: The receiver's input $b=1$. Similarly, in this case the receiver will learn $b+c$ with probability $1/2$ and aborts with probability $1/2$.

In the ideal world, the simulator runs first with a random bit-vector and extracts the inputs to the tokens OT by monitoring the queries to the PRF token. Next, it generates two bit-vectors b_i's and b'_i's that add up to 0 and 1, respectively, and computes the sums of the sender's input that correspond to these bits. Then the distribution of these sums can be computed as follows:

Case 1: In case that $\sum b_i=0$, then $b_n=0$ with probability $1/2$, and $b_n=1$ with probability $1/2$. In the former case the receiver learns $a+c$, whereas in the latter case it learns $b+d$.

Case 2: In case that $\sum b'_i=1$, then with probability $1/2$, $b'_n=0$ and with probability $1/2$, $b'_n=1$. In the former case the receiver learns $b+c$, whereas in the latter it learns $a+d$.

Note that this distribution is different from the real distribution, where the receiver never learns $b+d$ or $a+d$ since the token will always abort and not reveal d. We remark that in our example the abort probability of the receiver is independent of its input as proven in Claim 17 in [GIS$^+$10], yet the distribution of what it learns is different.