Composable Security in the Tamper Proof Hardware Model under Minimal Complexity

Carmit Hazay[*] Antigoni Polychroniadou[†] Muthuramakrishnan Venkitasubramaniam[‡]

Abstract

We put forth a new formulation of tamper-proof hardware in the Global Universal Composable (GUC) framework introduced by Canetti et al. in TCC 2007. Almost all of the previous works rely on the formulation by Katz in Eurocrypt 2007 and this formulation does not fully capture tokens in a concurrent setting. We address these shortcomings by relying on the GUC framework where we make the following contributions:

1. We construct secure Two-Party Computation (2PC) protocols for general functionalities with optimal round complexity and computational assumptions using stateless tokens. More precisely, we show how to realize arbitrary functionalities with GUC security in two rounds under the minimal assumption of One-Way Functions (OWFs). Moreover, our construction relies on the underlying function in a black-box way. As a corollary, we obtain feasibility of Multi-Party Computation (MPC) with GUC-security under the minimal assumption of OWFs. As an independent contribution, we identify an issue with a claim in a previous work by Goyal, Ishai, Sahai, Venkatesan and Wadia in TCC 2010 regarding the feasibility of UC-secure computation with stateless tokens assuming collision-resistant hash-functions (and the extension based only on one-way functions).
2. We then construct a 3-round MPC protocol to securely realize arbitrary functionalities with GUC-security starting from any semi-honest secure MPC protocol. For this construction, we require the so-called one-many commit-and-prove primitive introduced in the original work of Canetti, Lindell, Ostrovsky and Sahai in STOC 2002 that is round-efficient and black-box in the underlying commitment. Using specially designed "input-delayed" protocols we realize this primitive (with a 3-round protocol in our framework) using stateless tokens and one-way functions (where the underlying one-way function is used in a black-box way).

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`.

Contents

1 Introduction

Secure Multi-Party Computation (MPC) enables a set of parties to mutually run a protocol that computes some function f on their private inputs, while preserving two important properties: *privacy* and *correctness*. The former implies data confidentiality, namely, nothing leaks by the protocol execution but the computed output, while, the later requirement implies that no corrupted party or parties can cause the output to deviate from the specified function. It is by now well known how to securely compute any efficient functionality [Yao86, GMW87, MR91, Bea91] under the stringent simulation-based definitions (following the ideal/real paradigm). These traditional results prove security in the stand-alone model, where a *single* set of parties run a *single* execution of the protocol. However, the security of most cryptographic protocols proven in the stand-alone setting does not remain intact if many instances of the protocol are executed concurrently [Can01, CF01, Lin03]. The strongest (but also the most realistic) setting for concurrent security, known as *Universally Composable* (UC) security [Can01] considers the execution of an unbounded number of concurrent protocols in an arbitrary and adversarially controlled network environment. 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, CDPW07, Kat07, KLP07, CPS07, LPV09, DMRV13] for few examples.

One such setup assumption and the focus of this work is the use of tamper-proof hardware tokens. The first work to model tokens in the UC framework was by Katz in [Kat07] who introduced the $\mathcal{F}_{\text{WRAP}}$-functionality to capture such tokens and demonstrated feasibility of realizing general functionalities with UC-security. Most of the previous works in the tamper proof hardware [Kat07, CGS08, LPV09, GIS$^+$10, DMRV13, CKS$^+$14, DKMN15b] rely on this formulation. As we explain next, this formulation does not provide adequate composability guarantees. We begin by mentioning that any notion of composable security in an interactive setting should allow for multiple protocols to co-exist in the same system and interact with each other. We revisit the following desiderata put forth by Canetti, Lin and Pass [CLP10] for any notion of composable security:

Concurrent multi-instance security: The security properties relating to local objects (including data and tokens) of the analyzed protocol itself should remain valid even when multiple instances of the protocol are executed concurrently and are susceptible to coordinated attacks against multiple instances. Almost all prior works in the tamper proof model do not specifically analyze their security in a concurrent setting. In other words, they only discuss UC-security of a single instance of the protocol. In particular, when executing protocols in the concurrent setting with tokens, an adversary could in fact transfer a token received from one execution to another and none of the previous works that are based on the $\mathcal{F}_{\text{WRAP}}$-functionality accommodate transfers.

Modular analysis: Security of the larger overall protocols must be deducible from the security properties of its components. In other words, composing protocols should preserve security in a modular way. One of the main motivations and features in the UC-framework is the ability to analyze a protocol locally in isolation while guaranteeing global security. This does not only enable easier design but identifies the required security properties. The current framework proposed by Katz [Kat07] does not allow for such a mechanism.

The state-of-affairs regarding tamper-proof tokens leads us to ask the following question.

Does there exist a UC-formulation of tamper-proof hardware tokens that guarantee strong composability guarantees and allows for modular design?

Since the work of [Kat07], the power of hardware tokens has been explored extensively in a long series of works, especially in the context of achieving UC-security (for example, [CGS08, MS08, GIS+10, DKM11, DMMN13, DKMN15b, CKS+14]). While the work of Katz [Kat07] assumed the stronger stateful tokens, the work of Chandran, Goyal and Sahai [CGS08] was the first to achieve UC-security using only stateless tokens. In this work we will focus only on the weaker stateless token model. In the tamper-proof model with stateless tokens, as we argue below, the issue of minimal assumptions and round-complexity have been largely unaddressed. The work of Chandran et al. [CGS08] gives an $O(\kappa)$-round protocol (where κ is the security parameter) based on enhanced trapdoor permutations. Following that, Goyal et al. [GIS+10] provided an (incorrect) $O(1)$-round construction based on Collision-Resistant Hash Functions (CRHFs). The work of Choi et al. [CKS+14], extending the techniques of [GIS+10] and [DKM11], establishes the same result and provide a five-round construction based on CRHFs.

All previous constructions require assumptions stronger than one-way functions (OWFs), namely either trapdoor permutations or CRHFs. Thus as a first question, we investigate the minimal assumptions required for token-based secure computation protocols. The works of [GIS+10] and [CKS+14] rely on CRHFs for realizing statistically-hiding commitment schemes. Towards minimizing assumptions, both these works, originally considered a variant of their respective protocols where they replace the construction of the statistically-hiding commitment scheme based on CRHFs to the one based on one-way functions [HHRS15] to obtain UC-secure protocols under minimal assumptions (See Theorem 3 in [GIS+10] and Footnote 7 in [CKS+14]). While analyzing the proof of this variant in the work of [GIS+10], we found an issue in the original construction based on CRHFs.[1] More recently, the authors of [CKS+14] have removed this observation in their updated version of the original work (see [CKS+13]).[2] Given the state of affairs, our starting point is to address the following fundamental question regarding tokens that remains open.

> *Can we construct tamper-proof UC-secure protocols using stateless tokens assuming only one-way functions?*

A second important question that we address here is:

> *What is the round complexity of UC-secure two-party protocols using stateless tokens assuming only one-way functions?*

We remark here that relying on black-box techniques, it would be impossible to achieve non-interactive secure computation even in the tamper proof model as any such approach would be vulnerable to a residual function attack.[3] This holds even if we allow an initial token exchange phase, where the two parties exchange tokens (that are independent of their inputs). Hence, the best we could hope for is two rounds.

(G)UC-secure protocols in the multi-party setting. In the UC framework, it is possible to obtain UC-secure protocols in the MPC setting by first realizing the UC-secure oblivious transfer functionality (UC OT) in the two-party setting and then combining it with general compilation techniques (e.g., [Kil88, CLOS02, IPS08, LPV12] to obtain UC-secure multi-party computation protocols. First, we remark that specifically in the stateless tamper-proof tokens model, prior works fail to consider multi-versions of the OT-functionality

[1]In fact, we present a concrete attack that breaks the security of their construction in Appendix A. We remark that our observation *only* affects one particular result in [GIS+10], namely, realizing the UC-secure oblivious transfer functionality based on CRHFs and stateless tokens.

[2]In private communication, the authors of [CKS+13] explained that the variant that naively replaces the commitment with one based on one-way functions might be vulnerable to covert attacks.

[3]Intuitively, this attack allows the recipient of the (only) message to repeatedly evaluate the function on different inputs for a fixed sender's input.

while allowing transferrability of tokens which is important in an MPC setting.[4] As such, none of the previous works explicitly study the round complexity of multi-party protocols in the tamper proof model (with stateless tokens), we thus initiate this study in this work and address the following question.

> *Can we obtain round-optimal multi-party computation protocols with GUC-security in the tamper proof model?*

Unidirectional token exchange. Consider the scenario where companies such as Amazon or Google wish to provide an *email spam-detection* service and users of this service want to keep their emails private (so as to not have unwanted advertisements posted based on the content of their emails). In such a scenario, it is quite reasonable to assume that Amazon or Google have the infrastructure to create tamper-proof hardware tokens in large scale while the clients cannot be expected to create tokens on their own. Most of the prior works assume (require) that both parties have the capability of constructing tokens. When relying on non-black-box techniques, the work of [CKS+14] shows how to construct UC-OT using a single stateless token and consequently requires only one of the parties to create the token. The work of Moran and Segev in [MS08] on the other hand shows how to construct UC-secure two-party computation via a black-box construction where tokens are required to be passed only in one direction, however, they require the stronger model of stateful tokens. It is desirable to obtain a black-box construction when relying on stateless tokens. Unfortunately, the work of [CKS+14] shows that this is impossible in the fully concurrent setting. More precisely, they show that UC-security is impossible to achieve for general functionalities via a black-box construction using stateless tokens if only one of the parties is expected to create tokens. In this work, we therefore wish to address the following question:

> *Is there a meaningful security notion that can be realized in a client-server setting relying on black-box techniques using stateless tokens where tokens are created only by the server?*

1.1 Our Results

As our first contribution, we put forth a formulation of the tamper-proof hardware as a "global" functionality that provides strong composability guarantees. Towards addressing the various shortcomings of the composability guarantees of the UC-framework, Canetti et al. [CDPW07] introduced the Global Universal Composability (GUC) framework which among other things allows to consider global setup functionalities such as the common reference string model, and more recently the global random oracle model [CJS14]. In this work, we put forth a new formulation of tokens in the GUC-framework that will satisfy all our desiderata for composition. Furthermore, in our formulation, we will be able to invoke the GUC composition theorem of [CDPW07] in a modular way. A formal description of the $\mathcal{F}_{\text{gWRAP}}$-functionality can be found in Figure 2 and more detailed discussion is presented in the next section.

In the two-party setting we resolve both the round complexity and computational complexity required to realize GUC-secure protocols in the stronger $\mathcal{F}_{\text{gWRAP}}$-hybrid stated in the following theorem:

Theorem 1.1 (Informal) *Assuming the existence of OWFs, there exists a two-round protocol that GUC realizes any (well-formed) two-party functionality in the global tamper proof model assuming stateless tokens. Moreover it only makes black-box use of the underlying OWF.*

[4] We remark that the work of [CKS+14] considers multiple sessions of OT between a single pair of parties. However, they do not consider multiple sessions between multiple pairs of parties which is required to realize UC-security in the multiparty setting.

As mentioned earlier, any (black-box) non-interactive secure computation protocol is vulnerable to a residual function attack assuming stateless tokens. Therefore, the best round complexity we can hope for assuming (stateless) tamper-proof tokens is two which our results shows is optimal. In concurrent work [DKMN15a], Dottling et al. show how to obtain UC-secure two-party computation protocol relying on one-way functions via non-black-box techniques.

In the multi-party setting, our first theorem follows as a corollary of our results from the two-party setting.

Theorem 1.2 *Assuming the existence of OWFs, there exists a $O(d_f)$-round protocol that GUC realizes any multi-party (well formed) functionality f in the global tamper proof model assuming stateless tokens, where d_f is the depth of any circuit implementing f.*

Furthermore, this construction relies on the underlying one-way function in a black-box manner. Next, we improve the round-complexity of our construction to obtain the following theorem:

Theorem 1.3 *Assuming the existence of OWFs and stand-alone semi-honest MPC, there exists a three-round protocol that GUC realizes any multi-party (well formed) functionality in the global tamper proof model assuming stateless tokens.*

We remark that our construction is black-box in the underlying one-way function but unlike our previous theorem it relies on the code of the MPC protocol in a non-black-box way. In particular, underlying MPC protocols typically rely on semi-honest oblivious-transfer and our construction is non-black-box in this assumptions.

Finally, in the client-server setting, we prove the following theorem in the full version [**?**]:

Theorem 1.4 (Informal) *Assuming the existence of one-way functions, there exists a two-round protocol that securely realizes any two-party functionality assuming stateless tokens in a client-server setting, where the tokens are created only by the server. We also provide an extension where we achieve UC-security against malicious clients and sequential and parallel composition security against malicious servers.*

In more detail, we provide straight-line (UC) simulation of malicious clients and standard rewinding-based simulation against malicious servers. Our protocols guarantee security of the servers against arbitrary malicious coordinating clients and protects every individual client executing sequentially or in parallel against a corrupted server. We believe that this is a reasonable model in comparison to the Common Reference String (CRS) model where both parties require a trusted entity to sample the CRS. Furthermore, it guarantees meaningful concurrent security that is otherwise not achievable in the plain model in two rounds.

1.2 Our Techniques

Our starting point for our round optimal secure two-party computation is the following technique from [GIS$^+$10] for an extractable commitment scheme.

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 to receive the value v. It sends as its commitment the string $\mathsf{com}_b = (\mathsf{Ext}(u;r) \oplus b, r, v)$ where $\mathsf{Ext}(\cdot,\cdot)$ is a strong randomness 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. Furthermore, it allows for extraction as the simulator can observe the queries made by the sender to the token and observe that queries that yields v to retrieve u. This commitment scheme is based on one-way functions but is only extractable. To obtain a full-fledged UC-commitment from an extractable commitment we can rely on standard techniques (See [PW09, HV15] for a few examples). Instead, in order to obtain round-optimal constructions for secure two-party computation, we extend this protocol directly to realize the UC oblivious transfer functionality. A first incorrect approach is the following protocol. The parties exchange two sets of PRF tokens. 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 an OT token that 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. Specifically, the code of that token 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^*. The work of [GIS+10] provides a combiner to handle this particular attack which we demonstrate is flawed. We describe the attack in Section A. We instead will rely on a combiner from the recent work of Ostrovsky, Richelson and Scafuro [ORS15] to obtain a two-round GUC-OT protocol.

GUC-secure multi-party computation protocols. In order to demonstrate feasibility, we simply rely on the work of [IPS08] who show how to achieve GUC-secure MPC protocols in the OT-hybrid. By instantiating the OT with our GUC-OT protocol, we obtain MPC protocols in the tamper proof model assuming only one-way functions. While this protocol minimizes the complexity assumptions, the round complexity would be high. In this work, we show how to construct a 3-round MPC protocol. Our starting point is to take any semi-honest MPC protocol in the stand-alone model and compile it into a malicious one using tokens following the paradigm in the original work of Canetti et al. [CLOS02] and subsequent works [Pas03, Lin03]. Roughly, the approach is to define a commit-and-prove GUC-functionality $\mathcal{F}_{\mathrm{CP}}$ and compile the semi-honest protocol using this functionality following a GMW-style compilation.

We will follow an analogous approach where we directly construct a full-fledged $\mathcal{F}_{\mathrm{CP}}^{1:\mathrm{M}}$-functionality that allows a single prover to commit to a string and then prove multiple statements on the commitment simultaneously to several parties. In the token model, realizing this primitive turns out to be non-trivial. This is because we need the commitment in this protocol to be straight-line extractable and the proof to be about the value committed. Recall that, the extractable commitment is based on a PRF token supplied by the receiver of the commitment (and the verifier in the zero-knowledge proof). The prover cannot attest the validity of its commitment (via an NP-statement) since it does not know the code (i.e. key) of the PRF. Therefore, any commit and prove scheme in the token model necessarily must rely on a zero-knowledge proof that is black-box in the underlying commitment scheme. In fact, in the seminal work of Ishai et al. [IKOS09] they showed how to construct such protocols that have been extensively used in several works where the goal is to obtain constructions that are black-box in the underlying primitives. Following this approach and solving its difficulties that appear in the tamper-proof hardwire model, we can compile a T-round semi-honest secure MPC protocol to a $O(T)$-round protocol. Next, to reduce the rounds of the computation we consider the approach of Garg et al. [GGHR14] who show how to compress the round complexity of any MPC protocol to a two-round GUC-secure MPC protocol in the CRS model using obfuscation primitives.

In more detail, in the first round of the protocol in [GGHR14], every party commits to its input along with its randomness. The key idea is the following compiler used in the second round: it takes any (interactive) underlying MPC protocol, and has each party obfuscate their "next-message" function in that protocol, providing one obfuscation for each round. To ensure correctness, zero-knowledge proofs are used to validate the actions of each party w.r.t the commitments made in the first step. Such a mechanism is also referred to as a commit-and-prove strategy. This enables each party to independently evaluate the obfuscation one by one, generating messages of the underlying MPC protocol and finally obtain the output. The observation here is that party P_i's next-message circuit for round j in the underlying MPC protocol depends on its private input x_i and randomness r_i (which are hard-coded in the obfuscation) and on input the transcript of the communication in the first $j-1$ rounds outputs its message for the next round.

To incorporate this approach in the token model, we can simply replace the obfuscation primitives with tokens. Next, to employ zero-knowledge proofs via a black-box construction, we require a zero-knowledge protocol that allows commitment of a witness via tokens at the beginning of the protocol and then in a later step prove a statement about this witness where the commitment scheme is used in a "black-box" way. A first idea here would be to compile using the zero-knowledge protocol of [IKOS09] that facilitate such a commit-and-prove paradigm. However, as we explain later this would cost us in round-complexity. Instead we will rely on so-called input-delayed proofs [LS90] that have recently received much attention [CPS$^+$16a, CPS$^+$16b, HV16]. In particular, we will rely on the recent work of [HV16] who shows how to construct the so-called "input-delay" commit-and-prove protocols which allow a prover to commit a string in an initial commit phase and then prove a statement regarding this string at a later stage where the input statement is determined later. However, their construction only allows for proving one statement regarding the commitment. One of our technical contributions is to extend this idea to allow multiple theorems and further extend it so that a single prover can prove several theorems to multiple parties simultaneously. This protocol will be 4-round and we show how to use this protocol in conjunction with the Garg et al.'s round collapsing technique.

1.3 Related Work

In recent and independent work, using the approach of [CJS14], Nilges [Nil15, MMN16] consider a GUC-like formulation of the tokens for the two-party setting where the parties have fixed roles. The focus in [Nil15, MMN16] was to obtain a formulation that accommodates reusability of a single token for several independent protocols in the UC-setting for the specific two-party case. In contrast to our work, they do not explicitly model or discuss adversarial transferability of the tokens. In particular they do not discuss in the multi-party case, which is the main motivation behind our work.

Another recent work by Boureanu, Ohkubo and Vaudenay [BOV15] studies the limit of composition when relying on tokens. In this work, they prove that EUC (or GUC)-security is impossible to achieve for most functionalities if tokens can be transferred in a restricted framework. More precisely, their impossibility holds, if the tokens themselves do not "encode" the session identifier in any way. Our work, circumvents this impossibility result by precisely allowing the tokens generated (by honest parties) to encode the session identifier in which they have to be used.

2 Modeling Tamper-Proof Hardware in the GUC Framework

In this section we describe our model and give our rationale for our approach. We provide a brief discussion on the Universal Composability (UC) framework [Can01], UC with *joint state* [CR03] (JUC) and Generalized UC [CDPW07] (GUC). For more details, we refer the reader to the original works and the discussion in [CJS14].

Basic UC. Introduced by Canetti in [Can01], the Universal Composability (UC) framework provides a framework to analyse security of protocols in complex network environments in a modular way. One of the fundamental contributions of this work was to give a definition that will allow to design protocols and demonstrate security by "locally" analyzing a protocol but guaranteeing security in a *concurrent* setting where security of the protocol needs to be intact even when it is run concurrently with many instances of arbitrary protocols. Slightly more technically, in the UC-framework, to demonstrate that a protocol Π securely realizes an ideal functionality $\mathcal{F}$, we need to show that for any adversary $\mathcal{A}$ in the real world interacting with protocol Π in the presence of arbitrary environments $\mathcal{Z}$, there exists an ideal adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ the view of an interaction with $\mathcal{A}$ is indistinguishable from the view of an interaction with the ideal functionality $\mathcal{F}$ and $\mathcal{S}$.

Unfortunately, soon after its inception, a series of impossibility results [CF01, CKL06, Lin03] demonstrated that most non-trivial functionalities cannot be realized in the UC-framework. Most feasibility results in the UC-framework relied on some sort of trusted setup such as the common reference string (CRS) model [CF01], tamper-proof model [Kat07] or relaxed security requirements such as super-polynomial simulation [Pas03, PS04, BS05]. When modeling trusted setup such as the CRS model, an extension of the UC-framework considers the $\mathcal{G}$-hybrid model where "all" real-world parties are given access to an ideal setup functionality $\mathcal{G}$. In order for the basic composition theorem to hold in such a $\mathcal{G}$-hybrid model, two restrictions have to be made. First, the environment $\mathcal{Z}$ cannot access the ideal setup functionality *directly*; it can only do so indirectly via the adversary. In some sense, the setup $\mathcal{G}$ is treated as "local" to a protocol instance. Second, two protocol instances of the same or different protocol cannot share "state" for the UC-composition theorem to hold. Therefore, a setup model such as the CRS in the UC-framework necessitates that each protocol uses its own local setup. In other words, an independently sampled reference string for every protocol instance. An alternative approach that was pursued in a later work was to realize a multi-version of a functionality and proved security of the multi-version using a single setup. For example, the original feasibility result of Canetti, Lindell, Ostrovsky and Sahai [CLOS02] realized the $\mathcal{F}_{\text{MCOM}}$-functionality which is the multi-version of the basic commitment functionality$\mathcal{F}_{\text{COM}}$ in the CRS model.

JUC. Towards accommodating a global setup such as the CRS for multiple protocol instances, Canetti and Rabin [CR03] introduced the Universal Composition with Joint State (JUC) framework. Suppose we want to analyze several instances of protocol Π with an instance $\mathcal{G}$ as common setup, then at the least, each instance of the protocol must share some state information regarding $\mathcal{G}$ (e.g., the reference string in the CRS model). The JUC-framework precisely accommodates such a scenario, where a new composition theorem is proven, that allows for composition of protocols that share some state. However, the JUC-model for the CRS setup would only allow the CRS to be accessible to a pre-determined set of protocols and in particular still does not allow the environment to directly access the CRS.

GUC. For most feasibility results in the (plain) CRS model both in the UC and JUC framework, the simulator $\mathcal{S}$ in the ideal world needed the ability to "program" the CRS. In particular, it is infeasible to allow the environment to access the setup reference string. As a consequence, we can prove security only if the reference string is privately transmitted to the protocols that we demand security of and cannot be made *publicly* accessible. The work of Canetti, Pass, Dodis and Walfish [CDPW07] introduced the Generalized UC-framework to overcome this shortcoming in order to model the CRS as a global setup that is publicly available. More formally, in the GUC-framework, a global setup $\mathcal{G}$ is accessible by any protocol running in the system and in particular allows direct access by the environment. This, in effect, renders all previous protocols constructed in the CRS model not secure in the GUC framework as the simulator loses the programmability of the CRS. In fact, it was shown in [CDPW07] that the CRS setup is insufficient to securely realize the ideal commitment functionality in the GUC-framework. More generally, they show that any setup that simply provides only "public" information is not sufficient to realize GUC-security for most non-trivial functionalities. They further demonstrated a feasibility in the Augmented CRS model, where the CRS contains signature keys, one for each party and a secret signing key that is not revealed to the parties, except if it is corrupt, in which case the secret signing key for that party is revealed.

As mentioned before, the popular framework to capture the tamper-proof hardware is the one due to [Kat07] who defined the $\mathcal{F}_{\rm WRAP}$-functionality in the UC-framework. In general, in the token model, the two basic advantages that the simulator has over the adversary is "observability" and "programmability". Observability refers to the ability of the simulator to monitor all queries made by an adversary to the token and programmability refers to the ability to program responses to the queries in an online manner. In the context of tokens, both these assumptions are realistic as tamper-proof tokens do provide both these abilities in a real-world. However, when modeling tamper proof hardware tokens in the UC-setting, both these properties can raise issues as we discuss next.

Apriori, it is not clear why one should model the tamper proof hardware as a global functionality. In fact, the tokens are local to the parties and it makes the case for it *not* to be globally accessible. Let us begin with the formulation by Katz [Kat07] who introduced the $\mathcal{F}_{\rm WRAP}$-functionality (see Figure 1 for the stateless variant). In the real world the creator or sender of a token specifies the code to be incorporated in a token by sending the description of a Turing machine M to the ideal functionality. The ideal functionality then emulates the code of M to the receiver of the token, only allowing black-box access to the input and output tapes of M. In the case of stateful tokens, M is modeled as an interactive Turing machine while for stateless tokens, standard Turing machines would suffice. Slightly more technically, in the UC-model, parties are assigned unique identifiers PID and sessions are assigned identifiers sid. In the tamper proof model, to distinguish tokens, the functionality accepts an identifier mid when a token is created. More formally, when one party PID_i creates a token with program M with token identifier mid and sends it to another party PID_j in session sid, then the $\mathcal{F}_{\rm WRAP}$ records the tuple $(\mathsf{PID}_i, \mathsf{PID}_j, \mathsf{mid}, M)$. Then whenever a party with identifier PID_j sends a query $(\mathsf{Run}, \mathsf{sid}, \mathsf{PID}_i, \mathsf{mid}, x)$ to the $\mathcal{F}_{\rm WRAP}$-functionality, it first checks whether there is a tuple of the form $(\cdot, \mathsf{PID}_j, \mathsf{mid}, \cdot)$ and then runs the machine M in this tuple if one exists.

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

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

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

1. Send $(\mathsf{Create}, \mathsf{PID}_i, \mathsf{PID}_j, \mathsf{mid})$ to R.
2. Store $(\mathsf{PID}_i, \mathsf{PID}_j, \mathsf{mid}, M)$.

Execute. Upon receiving $(\mathsf{Run}, \mathsf{sid}, \mathsf{PID}_i, \mathsf{mid}, x)$ from R, find the unique stored tuple $(\mathsf{PID}_i, \mathsf{PID}_j, \mathsf{mid}, M)$. 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 $(\mathsf{PID}_i, \mathsf{PID}_j, \mathsf{mid}, \mathsf{out})$ to R.

Figure 1: The ideal functionality for stateless tokens [Kat07].

In the UC-setting (or JUC), to achieve any composability guarantees, we need to realize the multi-use variants of the specified functionality and then analyze the designed protocol in a concurrent man-in-the-middle setting. In such a multi-instance setting, it is reasonable to assume that an adversary that receives a token from one honest party in a left interaction can forward the token to another party in a right interaction. Unfortunately, the $\mathcal{F}_{\mathrm{WRAP}}$-functionality does not facilitate such a transfer.

Let us modify $\mathcal{F}_{\mathrm{WRAP}}$ to accommodate transfer of tokens by adding a special "transfer" query that allows a token in the possession of one party to be transferred to another party. Since protocols designed in most works do not explicitly prove security in a concurrent man-in-the-middle setting, such a modification renders the previous protocols designed in $\mathcal{F}_{\mathrm{WRAP}}$ insecure. For instance, consider the commitment scheme discussed in the introduction based on PRF tokens. Such a scheme would be insecure as an adversary can simply forward the token from the receiver in a right interaction to the sender in a left interaction leading to a malleable commitment.

In order to achieve security while allowing transferability we need to modify the tokens themselves in such a way to be not useful in an execution different from where it is supposed to be used. If every honestly generated token admits only queries that are prefixed with the correct session identifier then transferring the tokens created by one honest party to another honest party will be useless as honest parties will prefix their queries with the right session and the honestly generated tokens will fail to answer on incorrect session prefixes. This is inspired by an idea in [CJS14], where they design GUC-secure protocols in the Global Random Oracle model [CJS14]. As such, introducing transferrability naturally requires protocols to address the issue of non-malleability.

While this modification allows us to model transferrability, it still requires us to analyze protocols in a concurrent man-in-the-middle setting. In order to obtain a more modular definition, where each protocol instance can be analyzed in isolation we need to allow the token to be transferred from the adversary to the environment. In essence, we require the token to be somewhat "globally" accessible and this is the approach we take.

2.1 The Global Tamper-Proof Model

A natural first approach would be to consider the same functionality in the GUC-framework and let the environment to access the $\mathcal{F}_{\mathrm{WRAP}}$-functionality. This is reasonable as an environment can have access to the tokens via auxiliary parties to whom the tokens were transferred to. However, naively incorporating this idea would deny "observability" and "programmability" to the simulator as all adversaries can simply transfer away their tokens the moment they receive them and let other parties make queries on their behalf. Indeed, one can show that the impossibility result of [CKS$^+$14] extends to this formulation of the tokens (at least if the code of the token is treated in a black-box manner).[5] A second approach would be to reveal to the simulator all queries made to the token received by the adversary even if transferred out to any party. However, such a formulation would be vulnerable to the following transferring attack. If an adversary received a token from one session, it can send it as its token to an honest party in another session and now observe all queries made by the honest party to the token. Therefore such a formulation of tokens is incorrect.

Our formulation will accommodate transferrability while still guaranteeing observability to the simulator. In more detail, we will modify the definition of $\mathcal{F}_{\mathrm{WRAP}}$ so that it will reveal to the simulator all "illegitimate" queries made to the token by any other party. This approach is analogous to the one taken by Canetti, Jain and Scafuro [CJS14] where they model the Global Random Oracle Model and are confronted by a similar issue; here queries made to a globally accessible random oracle via auxiliary parties by the environment must be made available to the simulator while protecting the queries made by the honest party.

[5]Informally, the only advantage that remains for the simulator is to see the code of the tokens created by the adversary. This essentially reduces to the case where tokens are sent only in one direction and is impossible due to a result of [CKS$^+$14] when the code is treated as a black-box.

In order to define "legitimate" queries we will require that all tokens created by an honest party, by default, will accept an input of the form (sid, x) and will respond with the evaluation of the embedded program M on input x, only if $\mathsf{sid} = \overline{\mathsf{sid}}$, where $\overline{\mathsf{sid}}$ corresponds to the session where the token is supposed to be used, i.e. the session where the honest party created the token. Furthermore, whenever an honest party in session $\overline{\mathsf{sid}}$ queries a token it received on input x, it will prefix the query with the correct session identifier, namely issue the query $(\overline{\mathsf{sid}}, x)$. An *illegitimate query* is one where the sid prefix in a query differs from the session identifier from which the party is querying from. Every illegitimate query will be recorded by our functionality and will be disclosed to the party whose session identifier is actually sid.

More formally, the $\mathcal{F}_{\mathrm{gWRAP}}$-functionality is parameterized by a polynomial $p(\cdot)$ which is the time bound that the functionality will exercise whenever it runs any program. The functionality admits the following queries:

Creation Query: This query allows one party S to create and send a token to another party R by sending the query $(\mathsf{Create}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}, M)$ where M is the description of the machine to be embedded in the token, mid is a unique identifier for the token and sid is the session identifier. The functionality records $(\mathrm{R}, \mathsf{sid}, \mathsf{mid}, M)$.[6]

Transfer Query: We explicitly provide the ability for parties to transfer tokens to other parties that were not created by them (eg, received from another session). Such a query will only be used by the adversary in our protocols as honest parties will always create their own tokens. When a transfer query of the form $(\mathsf{transfer}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid})$ is issued, the tuple $(\mathrm{S}, \overline{\mathsf{sid}}, \mathsf{mid}, M)$ is erased and a new tuple $(\mathrm{R}, \mathsf{sid}, \mathsf{mid}, M)$ is created where $\overline{\mathsf{sid}}$ is the identifier of the session where it was previously used.

Execute Query: To run a token the party needs to provide an input in a particular format. All honest parties will provide the input as $x = (\mathsf{sid}, x')$ and the functionality will run M on input x and supply the answer. In order to achieve non-malleability, we will make sure in all our constructions that tokens generated by honest parties will respond to a query only if it contains the correct sid.

Retrieve Query: This is the important addition to our functionality following the approach taken by [CJS14]. $\mathcal{F}_{\mathrm{gWRAP}}$-functionality will record all illegitimate queries made to a token. Namely for a token recorded as the tuple $(\mathrm{R}, \overline{\mathsf{sid}}, \mathsf{mid}, M)$ an illegitimate query is of the form (sid, x) where $\mathsf{sid} \neq \overline{\mathsf{sid}}$ and such a query will be recorded in a set $\mathcal{Q}_{\mathsf{sid}}$ that will be made accessible to the receiving party corresponding to sid.

A formal description of the ideal functionality $\mathcal{F}_{\mathrm{gWRAP}}$ is presented in Figure 2. We emphasize that our formulation of the tamper-proof model will now have the following benefits:

1. It overcomes the shortcomings of the $\mathcal{F}_{\mathrm{WRAP}}$-functionality as defined in [Kat07] and used in subsequent works. In particular, it allows for transferring tokens from one session to another while retaining "observability".

2. Our model allows for designing protocols in the UC-framework and enjoys the composition theorem as it allows the environment to access the token either directly or via other parties.

[6]We remark here that the functionality does not explicitly store the PID of the creator of the token. We made this choice since the simulator in the ideal world will create tokens for itself which will serve as a token created on behalf of an honest party.

3. Our model explicitly rules out "programmability" of tokens. We remark that it is (potentially) possible to explicitly provide a mechanism for programmability in the $\mathcal{F}_{\text{gWRAP}}$-functionality. We chose to not provide such a mechanism so as to provide stronger composability guarantees.

4. In our framework, we can analyze the security of a protocol in isolation and guarantee concurrent multi-instance security directly using the GUC-composition theorem. Moreover, it suffices to consider a "dummy" adversary that simply forwards the environment everything (including the token).

An immediate consequence of our formulation is that it renders prior works such as [Kat07, CGS08, DKM11, DKMN15a] that rely on the programmability of the token insecure in our model. The works of [GIS$^+$10, CKS$^+$14] on the other hand can be modified and proven secure in the $\mathcal{F}_{\text{gWRAP}}$-hybrid as they do not require the tokens to be programmed.

Functionality $\mathcal{F}_{\text{gWRAP}}$

Parameters: Polynomial $p(\cdot)$.

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

Execute. Upon receiving $(\mathsf{Run}, \mathsf{sid}, \mathsf{mid}, x)$ from R, find the unique stored tuple $(\mathrm{R}, \mathsf{sid}, \mathsf{mid}, M)$. If such a tuple does not exist, do nothing. Otherwise, interpret $x = (\overline{\mathsf{sid}}, x')$ and 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.
Handling Illegal Queries: If $\mathsf{sid} \neq \overline{\mathsf{sid}}$, then add $(x', \mathsf{out}, \mathsf{mid})$ to the list $\mathcal{Q}_{\overline{\mathsf{sid}}}$ that is initialized to be empty.

Transfer. Upon receiving $(\mathsf{transfer}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid})$ from S, find the unique stored tuple $(\mathrm{S}, \overline{\mathsf{sid}}, \mathsf{mid}, M)$. If no such tuple exists, do nothing. Otherwise,

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

Retrieve Queries: Upon receiving a request $(\mathsf{retreive}, \mathsf{sid})$ from a party R, return the list $\mathcal{Q}_{\mathsf{sid}}$ of illegitimate queries.

Figure 2: The global stateless token functionality.

We now provide the formal definition of UC-security in the Global Tamper-Proof model.

Definition 2.1 (GUC security in the global tamper-proof model) *Let $\mathcal{F}$ be an ideal functionality and let π be a multi-party protocol. Then protocol π GUC realizes $\mathcal{F}$ in $\mathcal{F}_{\text{gWRAP}}$-hybrid model, if for every uniform PPT hybrid-model 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,*

$$\left\{\mathbf{View}^{\mathcal{F}_{\text{gWRAP}}}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa)\right\}_{\kappa\in\mathbb{N}} \overset{c}{\approx} \left\{\mathbf{View}^{\mathcal{F}_{\text{gWRAP}}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa)\right\}_{\kappa\in\mathbb{N}}.$$

3 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 3.1 *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 $X \overset{c}{\approx} Y$, if for every* PPT *machine D, every $a \in \{0,1\}^*$, every positive polynomial $p(\cdot)$ and all sufficiently large κ's,*

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

Definition 3.2 *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 $SD(X_\kappa,Y_\kappa) \le \varepsilon(\kappa)$. 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.*

3.1 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 3.3 (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,*

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

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.

3.2 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 3.4 (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.*

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

where $\mathbf{View}^{R^*}_{\mathsf{Com}}(m, z)$ *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 3.5 (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* $(\tau, state) \leftarrow \mathcal{S}_1^{\mathrm{R}^*}(1^\kappa)$*,* $w \leftarrow \mathcal{S}_2(state, m)$.

3.3 Randomness Extractors

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

Definition 3.6 (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 3.7 (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 $\varepsilon = 2^{-\frac{2\kappa+1-1}{2}} = 2^{-\kappa}$.

3.4 Hardcore Predicates

Definition 3.8 (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 3.9 ([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

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

3.5 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.

4 Two-Round Oblivious Transfer in the Stand-Alone Model

4.1 Building Blocks: Commitment Schemes

Trapdoor commitment schemes. A core building block of our protocol is a trapdoor commitment scheme TCom (cf. Definition 3.5) introduced by Pass and Wee in [PW09]. In Figure 3 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 3, 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 4.2.1 for more details.

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 3: Trapdoor commitment scheme

4.2 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 and a one-way tokens transfer phase that involves sending a set of tokens from the sender to the receiver in *one direction*. We begin with a protocol that comprises of three rounds where the first round only transfers tokens from one party and then later modify it to obtain a two-round protocol where the tokens are reusable and need to be transferred once at the beginning of the protocol. 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 tokens transfer phase the sender sends two types of tokens. The PRF tokens $\{\mathsf{TK}_{\mathsf{S}}^{\mathsf{PRF},l}\}_{l\in[4\kappa^2]}$ 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 $\mathsf{TK}_{\mathsf{S}}^{\mathsf{Com}}$ 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 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 A.*

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].*

Next, we describe our OT protocol Π_{OT} in the $\mathcal{F}_{\mathrm{gWRAP}}$-hybrid 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 the description below).

Protocol 1 *Protocol Π_{OT} - OT with stateless tokens in the plain model.*

- **Input:** S *holds two strings* $s_0, s_1 \in \{0,1\}^{\kappa}$ *and* R *holds a bit* b. *The common input is* sid.
- **The Protocol:**

  $\mathrm{S} \to \mathrm{R}$: *The sender creates two sets of tokens as follows and sends them to the receiver.*

  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* $(\mathsf{Create}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}_l, M_1)$ *to* $\mathcal{F}_{\mathrm{gWRAP}}$, *that on input* $(\overline{\mathsf{sid}}, x)$ *outputs* $\mathsf{PRF}_{\gamma_l}(x)$, *where* M_1 *is the functionality; if* $\overline{\mathsf{sid}} \neq \mathsf{sid}$ *the token aborts.*
  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* $(\mathsf{Create}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}_{l+1}, M_2)$ *to* $\mathcal{F}_{\mathrm{gWRAP}}$ *where* M_2 *is the functionality that on input* $(\overline{\mathsf{sid}}, \mathsf{tcom}_{b_i}, i)$ *proceeds as follows:*
     - *For the case where* $\overline{\mathsf{sid}} \neq \mathsf{sid}$ *the token aborts;*
     - *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.*

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

  1. R *sends* $(\mathsf{Run}, \mathsf{sid}, \mathsf{mid}_{l+1}, (0^{\kappa}, 0))$ *and receives* com_e *and interprets it as* TCmsg_1.
  2. *For all* $i \in [\kappa]$ *and* $j \in [4\kappa]$, R *sends* $(\mathsf{Run}, \mathsf{sid}, \mathsf{mid}_{1_l}, u_i^j)$ *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,* $\forall j \in [\kappa]$ *the receiver picks* $\eta_{i,j} \leftarrow \{0,1\}^{\kappa}$ *as per Figure 3 and computes:*

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

  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* $(\mathsf{Create}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}_{l+1+i}, M_3)$ *to* $\mathcal{F}_{\mathrm{gWRAP}}$ *where* M_3 *implements the following functionality:*

   On input $(\overline{\mathsf{sid}}, b_i, \mathsf{TCdecom}_{b_i})$*:*

   – *For the case where* $\overline{\mathsf{sid}} \neq \mathsf{sid}$ *the token aborts;*
   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* $(\mathsf{Run}, \mathsf{sid}, \mathsf{mid}_{l+1}, (\mathsf{tcom}_{b_i}, i))$ *and receives* $\mathsf{com}_{s_0^i}, \mathsf{com}_{s_0^i}$.
2. *For all* $i \in [\kappa]$, R *sends* $(\mathsf{Run}, \mathsf{sid}, \mathsf{mid}_{l+1+i}, (b_i, \mathsf{TCdecom}_{b_i}))$ *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.*

Next, we prove the following theorem,

Theorem 4.1 *Assume the existence of one-way permutations, then Protocol 1 securely realizes* $\mathcal{F}_{\mathrm{OT}}$ *in the* $\mathcal{F}_{\mathrm{gWRAP}}$*-hybrid.*

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, $\mathcal{S}$ queries the tokens $\{\mathsf{TK}_i\}_{i\in[\kappa]}$ twice by communicating with $\mathcal{F}_{\mathrm{gWRAP}}$ and 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 $\{s_{b_i\oplus 1}^i\}_{i\in[\kappa]}$. Our proof of indistinguishability relies on the list-decoding ability of the Goldreich-Levin hardcore predicate (cf. Theorem 3.9), 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. In case the adversary $\mathcal{A}$ issues a transfer query $(\mathsf{transfer}, \cdot)$, $\mathcal{S}$ transfers the query to the $\mathcal{F}_{\mathrm{gWRAP}}$.

Note that in this protocol there is no need to allow transfer queries to the $\mathcal{F}_{\mathrm{gWRAP}}$ functionality.

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. Adversary $\mathcal{A}$ communicates with functionality $\mathcal{F}_{\text{gWRAP}}$ on behalf of the corrupted party by sending create messages $\{(\mathsf{Create}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}_l, M_1)\}_{l\in[4\kappa^2]}$ and $(\mathsf{Create}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}_{l+1}, M_2)$. Then $\mathcal{F}_{\text{gWRAP}}$ forwards these tokens to the honest party by sending receipt messages $\{(\mathsf{Receipt}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}_l, M_1)\}_{l\in[4\kappa^2]}$ and $(\mathsf{Receipt}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}_{l+1}, M_2)$.

3. Upon receiving acknowledgement messages $\{(\mathsf{Receipt}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}_l, \cdot)\}_{l\in[4\kappa^2+1]}$ that all $[4\kappa^2]+1$ tokens have been created by $\mathcal{A}$, $\mathcal{S}$ 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_i^j 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 $\mathcal{S}$ creates 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 $\mathcal{F}_{\text{OT}}$ and halts, outputting whatever $\mathcal{A}$ does.

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 4.3 *The following two ensembles are computationally indistinguishable,*

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

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 4.4 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 5.1 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 4.4 *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.

□

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 4.5 *For every $i \in [\kappa]$,*

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

Note that the proof regarding the expected running time of the simulator is identical to the proof of Lemma 4.4.

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 communicates with $\mathcal{F}_{\rm gWRAP}$ in order to run the tokens twice after the adversary's view is obtained and the rewinding phase is completed. Namely, it asks $\mathcal{F}_{\rm gWRAP}$ to run 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 4.6 *The following two ensembles are identical,*

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

■

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. $\mathcal{S}$ communicates with $\mathcal{F}_{\rm gWRAP}$ on behalf of the honest party by sending create messages $\{(\mathsf{Create},\mathsf{sid},\mathrm{S},\mathrm{R},\mathsf{mid}_l,M_1)\}_{l\in[4\kappa^2]}$ and $(\mathsf{Create},\mathsf{sid},\mathrm{S},\mathrm{R},\mathsf{mid}_{l+1},M_2)$, where the code M_1 implements truly random functions (that is, M_1 is encoded with a lookup table that includes some polynomial number of queries bounded by the running time of the adversary). Then $\mathcal{F}_{\rm gWRAP}$ forwards these tokens by sending receipt messages $\{(\mathsf{Receipt},\mathsf{sid},\mathrm{S},\mathrm{R},\mathsf{mid}_l,M_1)\}_{l\in[4\kappa^2]}$ and $(\mathsf{Receipt},\mathsf{sid},\mathrm{S},\mathrm{R},\mathsf{mid}_{l+1},M_2)$ to $\mathcal{A}$. For each query $u\in\{0,1\}^{5\kappa}$ made by $\mathcal{A}$ to token $\mathsf{TK}^{\mathsf{PRF},l}_{\mathrm{S}}$, functionality $\mathcal{F}_{\rm gWRAP}$ runs M_1 on that query and returns a random v from $\{0,1\}^\kappa$. Similarly, M_2 implements a random function that maps elements from $\{0,1\}^\kappa\to\{0,1\}^{p(\kappa)}$.

3. Next, $\mathcal{S}$ retrieves $\mathcal{A}$ queries for session sid from $\mathcal{F}_{\mathrm{gWRAP}}$ by sending a $(\mathsf{retreive}, \mathsf{sid})$ message receiving the list $\mathcal{Q}_{\mathsf{sid}}$. $\mathcal{S}$ splits the set of receiver's queries (tcom_b, i^*) to the token $\mathsf{TK}_{\mathrm{S}}^{\mathsf{Com}}$ (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 $\mathsf{TK}_{\mathrm{S}}^{\mathsf{Com}}$ 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 $\mathcal{F}_{\mathrm{OT}}$. 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 4.7 *The following two ensembles are computationally indistinguishable,*

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

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 $\mathbf{Hybrid}^i_{\mathcal{F}_{\mathrm{OT}},\mathcal{S}_i(z),I}(\kappa, (s_0, s_1), b)$ the random variable that corresponds to the adversary's view in hybrid execution $\mathrm{H_i}$ 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 $\mathrm{H}_{1,\mathrm{l}}$ 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 $\mathsf{TK}_{\mathsf{S}}^{\mathsf{PRF},l}$. 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 4.8 *For every $l \in [4\kappa^2]$,*

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

Hybrid H_2: Similarly, we consider a hybrid execution for which the code of token $\mathsf{TK}_{\mathsf{S}}^{\mathsf{Com}}$ 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 4.9

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

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 $\mathsf{TK}_{\mathsf{S}}^{\mathsf{PRF},l}$ have the same response. Specifically, in case of a collision simulator $\mathcal{S}_{3,l}$ aborts.

Claim 4.10 *For every $l \in [4\kappa^2]$,*

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

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 4.11

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

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 4.12 *For every $i \in [\kappa]$,*

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

and

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

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 3.9), 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,[7]

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

Therefore,

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

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 $\mathrm{H}_{5,\mathrm{i}}$ 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

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

$\frac{1}{2}+\varepsilon$. Using the list-decoding algorithm of Goldreich-Leving hardcore-predicate (cf. Theorem 3.9), 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. $\square$

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 4.13

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

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. $\square$

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

■

■

4.2.1 Relaxing to One-Way Functions

In our construction we rely on one-way permutations for a non-interactive perfectly binding commitment scheme. Recall that, the $\mathsf{TK}^{\mathsf{Com}}$ on input $(\cdot,0)$ is required to output a commitment to the challenge e and else commitments to the s^i_0, s^i_1's values. To relax this assumption to one-way functions, we instead need to rely on the two-message Naor's statistically binding commitment scheme [Nao91] where the receiver sends the first message. Instead of communicating this message to the sender, the receiver directly feeds it to the token as input. More precisely, let $\widehat{\mathsf{Com}}(m;r,R)$ denote the honest committer's strategy function that responds according to Naor's commitment with input message m and random tape r, where the receiver's first message is R. We make the following modification and incorporate the following functionality: On input $(\mathsf{tcom}_{b_i}, i, R_0, R_1)$ proceed 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 sent by the receiver to the sender, it produces R_0, R_1, the first messages corresponding to the commitments made so that the sender can reconstruct the values being committed to (using the same PRF function). We note that two issues arise when proving security using the modified token's functionality.

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.

4.3 Reusability of Tokens

Following the work of Choi et al. [CKS$^+$14], we investigate the possibility of exchanging tokens just once and (re-)using the tokens for an *unbounded* number of oblivious transfers. Namely, we extend our protocol from the previous section to achieve limited concurrency, namely sequential and parallel composition. Note that we still maintain that the tokens are created only by one party. We remark that this gets around the barrier of [CKS$^+$14] as we do not achieve full concurrent (i.e. UC)-security. More precisely, we show how the same set of tokens can be used to execute an arbitrary number of oblivious transfers. Towards achieving this, we show that it is possible to exchange all tokens at the beginning of the protocol. Analogous to [PVW08], we consider the multi-session extension $\widehat{\mathcal{F}}_{\rm OT}$ of the OT functionality which on a high-level enables arbitrary number of independent executions of $\mathcal{F}_{\rm OT}$ and coordinates interactions with every pair of parties via subsessions specified by the identifier ssid of a single session with identifier sid.

Recall that the sender sends three sets of tokens: PRF and commitment tokens in the first message and OT tokens in the second message whose codes depend on the first message of the receiver. We handle each of these sets in a different way.

Handling the OT tokens. Recall that these tokens are generated after receiving the receiver's message of the protocol. Then in order to generate them independently of this message, 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 4.2.1 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 a non-interactive perfectly binding commitment scheme 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 concurrently. We will identify every session with the identifier ssid. On a high-level, for every OT instance identified by ssid we generate a one-time signature/verification key pair by applying a PRF on $\tau = \mathsf{sid}\|\mathsf{ssid}\|\mathsf{com}_{b_j}\|j$ where com_{b_j} is the receiver's j^{th} commitment to its input's share for that instance. Then the receiver is allowed to query the OT token TK_j only if it possesses a valid signature corresponding to its commitment com_{b_j} for that instance. Now, since the signatures are unique, the signatures themselves do not carry any additional information. To conclude, we use the same protocol as described in the previous section with the following change:

- S provides a signature of com_{b_j} for every j, under key sk_τ for $\tau = \mathsf{sid}\|\mathsf{ssid}\|\mathsf{com}_{b_i}\|i$, along with its second message to the receiver.

Similarly, we handle the commitments com_{c_i} and tokens $\widehat{\mathsf{TK}}_i$.

We further note that since the tokens are sent prior to the protocol execution, we use an additional PRF key in order to sample the random strings x_0, x_1, for which the OT tokens use in order to decommit to their shares. Specifically, the OT tokens will generate this pair of strings by applying a PRF on an input $\mathsf{sid}\|\mathsf{ssid}$.

Handling the PRF tokens. 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 5.1, 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.[8] The rest of the proof against a corrupted sender follows similarly to the proof of Theorem 4.1.

Lemma 4.14 (Lemma 5.1 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.*

Proof: We continue with the formal proof of the extended Lemma 5.1. Specifically, instead of having the sender send $4\kappa^2$ tokens that independently implement the PRF, we will require the sender to send 2κ tokens implementing the same PRF functionality for each commitment, in order to implement a simple cut-and-choose strategy. In all, the sender sends $4\kappa^2 \times 2\kappa = 8\kappa^3$ tokens.

Namely, for each of the $4\kappa^2$ commitments, we modify the receiver's algorithm as follows: let the 2κ tokens (allegedly) implementing a particular PRF to be used for some 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.

[8] 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.

3. Commit by running an extractor on u as described Protocol 2.

For a given malicious sender S^*, let there be at most $T = \text{poly}(k)$ 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 5.1, 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 5.1 we have that except with probability $1/2^\kappa$, there is a set S_{V_i} of size at least $2^{4\kappa^2}$ 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 $|S_{V_j}| > 2^{4\kappa^2}$. 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. ■

Handling commitment tokens. As discussed in the beginning of the section, 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]. As mentioned above, we will identify every session by two quantifiers (sid, ssid) where sid is the sequential session identifier and ssid is the parallel session identifier.

Following these modifications we can allow for the tokens to be created in a setup phase only once and then used an arbitrary number of times. We exchange the following tokens in an initial setup phase. As in the previous protocol Π_{OT}, we additionally rely on a non-interactive perfectly binding commitment scheme Com and PRFs F, F'. In more details,

A Signature Token $\mathsf{TK}_{\mathrm{S}}^{\mathrm{SIG}}$**:** S chooses a PRF key γ for a PRF family $\mathsf{PRF}'_\gamma : \{0,1\}^* \to \{0,1\}^\kappa$. Let $\tau = \mathsf{sid}\|\mathsf{ssid}\|\mathsf{com}\|i$ then compute $V = \mathsf{PRF}'_\gamma(\tau)$ and output

$$\mathsf{vk}_\tau = \begin{pmatrix} \mathsf{Com}(x_1^0; r_1^0) & \cdots & \mathsf{Com}(x_{\kappa+1}^0; r_{\kappa+1}^0) \\ \mathsf{Com}(x_1^1, r_1^1) & \cdots & \mathsf{Com}(x_{\kappa+1}^1;, r_{\kappa+1}^1) \end{pmatrix}$$

where V is parsed as $(x_\ell^b)_{b\in\{0,1\},\ell\in[\kappa+1]} \| (r_\ell^b)_{b\in\{0,1\},\ell\in[\kappa+1]}$.

Then S creates token $\mathsf{TK}_{\mathrm{S}}^{\mathrm{SIG}}$ by sending $(\mathsf{Create}, \mathsf{sid}, \mathsf{ssid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_1, M_1)$, that on input $\mathsf{sid}\|\mathsf{ssid}\|\mathsf{com}\|i$ outputs vk, where M_1 is the above functionality.

Next, consider the following modified tokens.

1. $\{\mathsf{TK}_{1_j}^0, \mathsf{TK}_{1_j}^1, \ldots, \mathsf{TK}_{\kappa_j}^0, \mathsf{TK}_{\kappa_j}^1\}_{j\in[4\kappa^2]}$: S chooses $8\kappa^3$ random PRF keys $\{\gamma_l\}_{l\in[8\kappa^3]}$ 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 $\{\mathsf{Create}, \mathsf{sid}, \mathsf{ssid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_l, M_2\}_{l\in[8\kappa^3]}$ to R where M_1 is the machine that on input $(\mathsf{sid}, \mathsf{ssid}, x)$, outputs $\mathsf{PRF}_{\gamma_l}(\mathsf{sid}\|\mathsf{ssid}\|x)$.

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 $(\mathsf{Create}, \mathsf{sid}, \mathsf{ssid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{l+1}, M_3)$ to $\mathcal{F}_{\mathrm{gWRAP}}$ 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

   $$(\mathsf{com}_{s_0^i}, \mathsf{com}_{s_1^i}), \mathsf{vk}_\tau = \begin{pmatrix} \mathsf{Com}(x_1^0; r_1^0) & \cdots & \mathsf{Com}(x_{\kappa+1}^0; r_{\kappa+1}^0) \\ \mathsf{Com}(x_1^0, r_1^0) & \cdots & \mathsf{Com}(x_{\kappa+1}^0;, r_{\kappa+1}^1) \end{pmatrix}$$

   where V is parsed as $s_0^i\|s_1^i\|r_0\|r_1\|(x_\ell^b)_{b\in\{0,1\},\ell\in[\kappa+1]}\|(r_\ell^b)_{b\in\{0,1\},\ell\in[\kappa+1]}$ and $\mathsf{com}_{s_b^i} \leftarrow \mathsf{Com}(s_b^i; r_b)$ for $b = \{0,1\}$.

3. TK_i: For all $i \in [\kappa]$, S creates a token TK_i by sending $(\mathsf{Create}, \mathsf{sid}, \mathsf{ssid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{l+1+i}, M_4)$ to $\mathcal{F}_{\mathrm{gWRAP}}$ where M_4 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 $\mathsf{TK}_{\mathrm{S}}^{\mathsf{Com}}$ 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} retrieves all the queries made to both the tokens. 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 OT 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 4.7 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 4.10.

5 Two-Round Token-Based GUC Oblivious Transfer

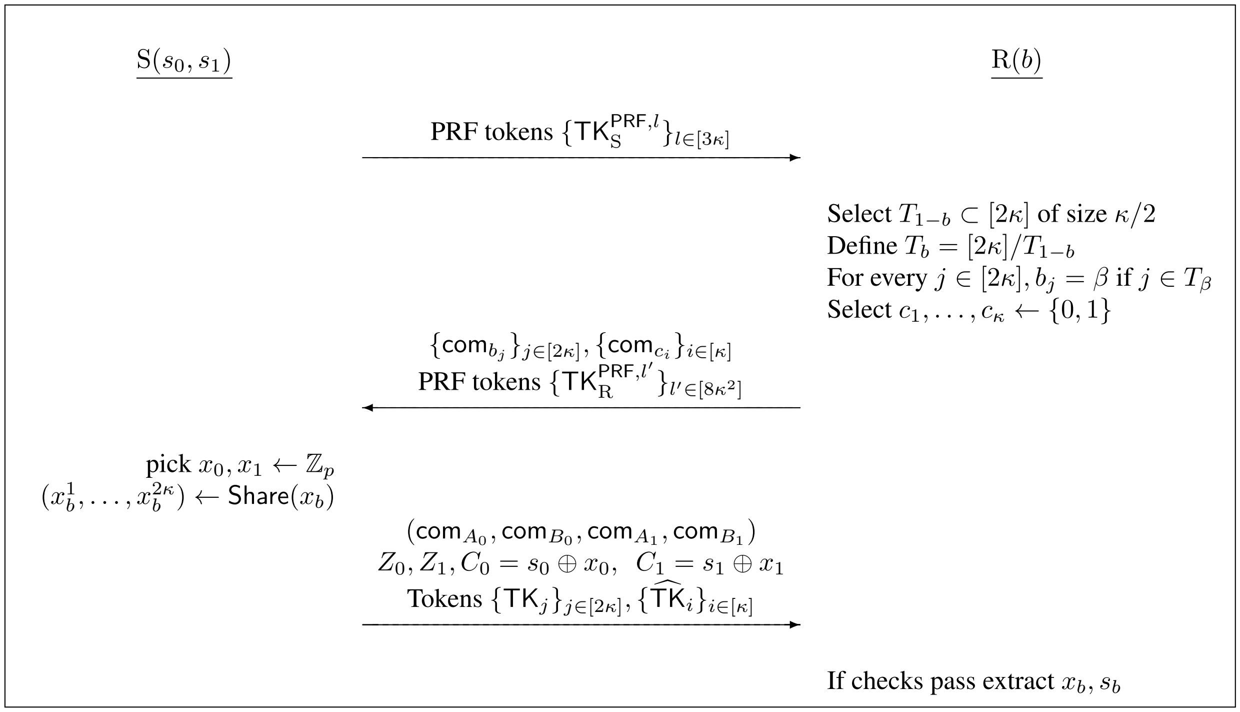
In this section we present our main protocol that implements GUC OT in two rounds. We first construct a three-round protocol and then show in Section 5.3, similarly to Section 4.3, how to obtain a two-round protocol by exchanging tokens just once in a setup phase. Recall that the counter example to the [GIS$^+$10] protocol shows that directly extracting the sender's inputs does not necessarily allow us to extract the sender's inputs correctly, as the tokens can behave maliciously. Inspired by the recently developed protocol from [ORS15] we consider a new approach here for which the sender's inputs are extracted directly by monitoring the queries it makes to the PRF tokens and using additional checks to ensure that the sender's inputs can be verified.

Protocol intuition. As a warmup consider the following sender's algorithm that first chooses two random strings 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.

Our oblivious transfer protocol. We obtain a two-round oblivious transfer 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, 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 retrieving its queries and we are able to show that there cannot be any input dependent behavior of the token if it passes both the shares consistency check and the shares validity check. See Figure 4 for the protocol overview. In Section 5.1 we discuss how to obtain a two-round two-party computation using our OT protocol.



$\underline{\mathrm{S}(s_0, s_1)}$
$\underline{\mathrm{R}(b)}$
PRF tokens $\{\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}\}_{l\in[3\kappa]}$
Select $T_{1-b} \subset [2\kappa]$ of size $\kappa/2$
Define $T_b = [2\kappa]/T_{1-b}$
For every $j \in [2\kappa], b_j = \beta$ if $j \in T_\beta$
Select $c_1, \ldots, c_\kappa \leftarrow \{0,1\}$
$\{\mathsf{com}_{b_j}\}_{j\in[2\kappa]}, \{\mathsf{com}_{c_i}\}_{i\in[\kappa]}$
PRF tokens $\{\mathsf{TK}_{\mathrm{R}}^{\mathsf{PRF},l'}\}_{l'\in[8\kappa^2]}$
pick $x_0, x_1 \leftarrow \mathbb{Z}_p$
$(x_b^1, \ldots, x_b^{2\kappa}) \leftarrow \mathsf{Share}(x_b)$
$(\mathsf{com}_{A_0}, \mathsf{com}_{B_0}, \mathsf{com}_{A_1}, \mathsf{com}_{B_1})$
$Z_0, Z_1, C_0 = s_0 \oplus x_0,\ \ C_1 = s_1 \oplus x_1$
Tokens $\{\mathsf{TK}_j\}_{j\in[2\kappa]}, \{\widehat{\mathsf{TK}}_i\}_{i\in[\kappa]}$
If checks pass extract x_b, s_b

Figure 4: A high-level diagram of $\Pi_{\mathrm{GUC}}^{\mathrm{OT}}$.

We now describe our protocol $\Pi_{\mathrm{OT}}^{\mathrm{GUC}}$ 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. See Protocol 2 for the complete description.

Protocol 2 *Protocol* $\Pi_{\mathrm{GUC}}^{\mathrm{OT}}$ *- GUC OT with stateless tokens.*

- **Inputs:** S *holds two strings* $s_0, s_1 \in \{0,1\}^\kappa$ *and* R *holds a bit* b. *The common input is* sid.
- **The protocol:**
  1. **S → R***: 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* $\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}$ *sending* $(\mathsf{Create}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}_l, M_1)$ *to* $\mathcal{F}_{\mathrm{gWRAP}}$ *where* M_1 *is the functionality of the token that on input* $(\overline{\mathsf{sid}}, x)$ *outputs* $\mathsf{PRF}_{\gamma_l}(x)$ *for all* $l \in [3\kappa]$*; For the case where* $\overline{\mathsf{sid}} \neq \mathsf{sid}$ *the token aborts;*

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

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

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

   (b) 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* $(\mathsf{Create}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{l'}, M_2)$ *to* $\mathcal{F}_{\mathrm{gWRAP}}$ *for all* $l' \in [8\kappa^2]$.

3. **S $\to$ R**: 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* $b \in \{0,1\}$. 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:*

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

   S *sends:*

   (a) *Matrices* $(\mathsf{com}_{A_0}, \mathsf{com}_{B_0}, \mathsf{com}_{A_1}, \mathsf{com}_{B_1})$ *to* R, *where,*

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

   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* $(\mathsf{Run}, \mathsf{sid}, \mathsf{mid}_{[i,j,\beta]}, u^{A_\beta[i,j]})$ *and* $(\mathsf{Run}, \mathsf{sid}, \mathsf{mid}_{2\kappa^2+[i,j,\beta]}, u^{B_\beta[i,j]})$, *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* $[2\kappa^2]$.

   (b) $C_0 = s_0 \oplus x_0$ *and* $C_1 = s_1 \oplus x_1$ *to* R.

   (c) *For all* $j \in [2\kappa]$, S *creates a token* TK_j *sending* $(\mathsf{Create}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}_{3\kappa+j}, M_3)$ *to* $\mathcal{F}_{\mathrm{gWRAP}}$ *where* M_3 *is the functionality that on input* $(\overline{\mathsf{sid}}, b_j, \mathsf{decom}_{b_j})$, *aborts if* $\overline{\mathsf{sid}} \neq \mathsf{sid}$ *or if* decom_{b_j} *is not verified correctly. Otherwise it outputs* $(A_{b_j}[\cdot,j], \mathsf{decom}_{A_{b_j}[\cdot,j]}, B_{b_j}[\cdot,j], \mathsf{decom}_{B_{b_j}[\cdot,j]})$.

   (d) *For all* $i \in [\kappa]$, S *creates a token* $\widehat{\mathsf{TK}}_i$ *sending* $(\mathsf{Create}, \mathsf{sid}, \mathrm{S}, \underline{\mathrm{R}}, \mathsf{mid}_{5\kappa+i}, M_4)$ *to* $\mathcal{F}_{\mathrm{gWRAP}}$ *where* M_4 *is the functionality that on input* $(\overline{\mathsf{sid}}, c_i, \mathsf{decom}_{c_i})$ *aborts if* $\overline{\mathsf{sid}} \neq \mathsf{sid}$ *or if* decom_{c_i} *is not verified correctly. Otherwise it outputs,*

   $$(A_0[i,\cdot], \mathsf{decom}_{A_0[i,\cdot]}, A_1[i,\cdot], \mathsf{decom}_{A_1[i,\cdot]}), \textit{ if } c = 0$$
   $$(B_0[i,\cdot], \mathsf{decom}_{B_0[i,\cdot]}, B_1[i,\cdot], \mathsf{decom}_{B_1[i,\cdot]}), \textit{ if } c = 1$$

4. **Output Phase:**
   For all $j \in [2\kappa]$, R *sends* $(\mathsf{Run}, \mathsf{sid}, \mathsf{mid}_{3\kappa+j}, (b_j, \mathsf{decom}_{b_j}))$ *and receives*

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

   For all $i \in [\kappa]$, R *sends* $(\mathsf{Run}, \mathsf{sid}, \mathsf{mid}_{5\kappa+i}, (c_i, \mathsf{decom}_{c_i}))$ *and receives*

   $$(A_0[\cdot,i], A_1[\cdot,i]) \textit{ or } (B_0[\cdot,i], B_1[\cdot,i]).$$

   (a) 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.*

(b) 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.*

(c) OUTPUT RECONSTRUCTION: *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 $s_b = C_b \oplus x_b$.*

Next, we prove the following theorem,

Theorem 5.1 *Assume the existence of one-way functions, then protocol $\Pi_{\mathrm{GUC}}^{\mathrm{OT}}$ GUC realizes $\mathcal{F}_{\mathrm{OT}}$ in the $\mathcal{F}_{\mathrm{gWRAP}}$-hybrid.*

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 retrieving 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 rely 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 retrieving 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.

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

Lemma 5.1 *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 3.7) 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 continue with the complete proof.

Proof: Let $\mathcal{A}$ be a malicious PPT real adversary attacking protocol $\Pi_{\mathrm{OT}}^{\mathrm{GUC}}$ in the $\mathcal{F}_{\mathrm{gWRAP}}$-hybrid model. We construct an ideal adversary $\mathcal{S}$ with access to $\mathcal{F}_{\mathrm{OT}}$ which simulates a real execution of $\Pi_{\mathrm{OT}}^{\mathrm{GUC}}$ 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 $\Pi_{\mathrm{OT}}^{\mathrm{GUC}}$ 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 $\mathcal{Z}$.

In case the adversary $\mathcal{A}$ issues a transfer query $(\mathsf{transfer}, \cdot)$, $\mathcal{S}$ relays the query to the $\mathcal{F}_{\mathrm{gWRAP}}$.

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

1. $\mathcal{A}$ communicates with the functionality $\mathcal{F}_{\mathrm{gWRAP}}$ on behalf of the corrupted parties by sending create messages $\{(\mathsf{Create}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}_l, M_1)\}_{l\in[3\kappa]}$. Then $\mathcal{F}_{\mathrm{gWRAP}}$ forwards these tokens to the honest parties by sending receipt messages $\{(\mathsf{Receipt}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}_l, M_1)\}_{l\in[3\kappa]}$.

2. Upon receiving acknowledgement messages $\{(\mathsf{Receipt}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}_l, M_1)\}_{l\in[3\kappa]}$ that all $[3\kappa]$ tokens have been created by $\mathcal{A}$, $\mathcal{S}$ communicates with the functionality $\mathcal{F}_{\mathrm{gWRAP}}$ on behalf of the honest parties by sending create messages $\{(\mathsf{Create}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{l'}, M_2)\}_{l'\in[8\kappa^2]}$, where the code M_2 implements

truly random functions (that is, M_2 is encoded with a lookup table that includes some polynomial number of queries bounded by the running time of the adversary). Then, the functionality $\mathcal{F}_{\text{gWRAP}}$ forwards receipt messages $\{(\mathsf{Receipt}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{l'}, M_2)\}_{l' \in [8\kappa^2]}$ to $\mathcal{A}$. For each query $u \in \{0,1\}^{5\kappa}$ made by $\mathcal{A}$ to the tokens $\mathsf{TK}_{\mathrm{R}}^{\mathsf{PRF},l'}$, functionality $\mathcal{F}_{\text{gWRAP}}$ runs M_2 on that query and returns a random v from $\{0,1\}^{\kappa}$.

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. 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 by $\mathcal{F}_{\text{gWRAP}}$ with respect to that token by sending a retrieve message $(\mathsf{retreive}, \mathsf{sid})$ to $\mathcal{F}_{\text{gWRAP}}$ which returns the list $\mathcal{Q}_{\mathsf{sid}}$ of illegitimate queries. 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 $\mathcal{F}_{\text{OT}}$ and halts, outputting whatever $\mathcal{A}$ does.

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

Lemma 5.2 $\left\{\mathbf{View}^{\mathcal{F}_{\text{gWRAP}}}_{\Pi^{\text{GUC}}_{\text{OT}},\mathcal{A},\mathcal{Z}}(\kappa)\right\}_{\kappa \in \mathbb{N}} \overset{\text{c}}{\approx} \left\{\mathbf{View}^{\mathcal{F}_{\text{OT}}}_{\pi_{\text{IDEAL}},\mathcal{S},\mathcal{Z}}(\kappa)\right\}_{\kappa \in \mathbb{N}}$.

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 $\mathcal{F}_{\text{OT}}$. Instead, we define a simulator $\mathcal{S}_0$ that receives the real input of the receiver and internally emulates the protocol $\Pi^{\text{GUC}}_{\text{OT}}$ 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 5.3 $\left\{\mathbf{View}^{\mathcal{F}_{\text{gWRAP}}}_{\Pi^{\text{GUC}}_{\text{OT}},\mathcal{A},\mathcal{Z}}(\kappa)\right\}_{\kappa \in \mathbb{N}} \approx \left\{\mathbf{View}_{\Pi_{\mathrm{H}_0},\mathcal{S}_0,\mathcal{Z}}(\kappa)\right\}_{\kappa \in \mathbb{N}}$.

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 $\mathsf{TK}_{\mathrm{R}}^{\mathsf{PRF},l'}$ 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 $\mathsf{TK}_{\mathrm{R}}^{\mathsf{PRF},l'}$. 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 5.4 *For every* $l' \in [8\kappa^2]$, $\{\mathbf{View}_{\Pi_{\mathrm{H}_{1,l'-1}},\mathcal{S}_{1,l'-1},\mathcal{Z}(\kappa)}\}_{\kappa\in\mathbb{N}} \overset{\mathrm{c}}{\approx} \{\mathbf{View}_{\Pi_{\mathrm{H}_{1,l'}},\mathcal{S}_{1,l'},\mathcal{Z}(\kappa)}\}_{\kappa\in\mathbb{N}}$.

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 $\mathsf{TK}_{\mathrm{R}}^{\mathsf{PRF},l'}$ results in the same response. Using a proof analogous to Lemma 4.10, we obtain the following claim.

Claim 5.5 *For every* $l' \in [8\kappa^2]$, $\{\mathbf{View}_{\Pi_{\mathrm{H}_{2,l'-1}},\mathcal{S}_{2,l'-1},\mathcal{Z}(\kappa)}\}_{\kappa\in\mathbb{N}} \overset{\mathrm{s}}{\approx} \{\mathbf{View}_{\Pi_{\mathrm{H}_{2,l'}},\mathcal{S}_{2,l'},\mathcal{Z}(\kappa)}\}_{\kappa\in\mathbb{N}}$.

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 $i \in [\kappa]$

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

then, 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 5.6 $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_{2,8\kappa^2}},\mathcal{S}_{2,8\kappa^2},\mathcal{Z}}(\kappa)\right\}_{\kappa\in\mathbb{N}} \overset{\mathrm{s}}{\approx} \left\{\mathbf{View}_{\Pi_{\mathrm{H}_3},\mathcal{S}_3,\mathcal{Z}}(\kappa)\right\}_{\kappa\in\mathbb{N}}$.

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 from Lemma 5.1 that the output of H_3 and H_4 are statistically-close. Therefore, we have the following claim,

Claim 5.7 $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_3},\mathcal{S}_3,\mathcal{Z}}(\kappa)\right\}_{\kappa\in\mathbb{N}} \overset{\mathrm{s}}{\approx} \left\{\mathbf{View}_{\Pi_{\mathrm{H}_4},\mathcal{S}_4,\mathcal{Z}}(\kappa)\right\}_{\kappa\in\mathbb{N}}$.

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 communicates with $\mathcal{F}_{\mathrm{gWRAP}}$ instead of creating/sending the tokens by itself and further 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 $\mathcal{F}_{\mathrm{OT}}$ functionality. It follows from our simulation that the view of the adversary in 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 H_4 and the ideal execution are identically distributed.

Claim 5.8 $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_4},\mathcal{S}_4,\mathcal{Z}}(\kappa)\right\}_{\kappa\in\mathbb{N}} \approx \left\{\mathbf{View}^{\mathcal{F}_{\mathrm{OT}}}_{\pi_{\mathrm{IDEAL}},\mathcal{S},\mathcal{Z}}(\kappa)\right\}_{\kappa\in\mathbb{N}}$.

■

Simulating the corrupted R**.** We begin by describing our simulation:

1. $\mathcal{S}$ communicates with $\mathcal{F}_{\mathrm{gWRAP}}$ on behalf of the honest parties by sending create messages $\{(\mathsf{Create}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}_l, M_1)\}_{l\in[3\kappa]}$, where the code M_1 implements truly random functions (that is, M_1 is encoded with a lookup table that includes some polynomial number of queries bounded by the running time of the adversary). Then $\mathcal{F}_{\mathrm{gWRAP}}$ forwards these tokens by sending receipt messages $\{(\mathsf{Receipt}, \mathsf{sid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}_l, M_1)\}_{l\in[3\kappa]}$ to $\mathcal{A}$. For each query $u \in \{0,1\}^{5\kappa}$ made by $\mathcal{A}$ to the tokens $\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}$, functionality $\mathcal{F}_{\mathrm{gWRAP}}$ runs M_1 on that query and returns a random v from $\{0,1\}^{\kappa}$.

2. $\mathcal{A}$ communicates with $\mathcal{F}_{\mathrm{gWRAP}}$ on behalf of the corrupted parties by sending create messages to the functionality $\{(\mathsf{Create}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{l'}, M_2)\}_{l'\in[8\kappa^2]}$. Then, the functionality $\mathcal{F}_{\mathrm{gWRAP}}$ forwards these tokens to the honest parties by sending receipt messages $\{(\mathsf{Receipt}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{l'}, M_2)\}_{l'\in[8\kappa^2]}$.

3. Upon receiving acknowledgement messages $\{(\mathsf{Receipt}, \mathsf{sid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{l'}, M_2)\}_{l' \in [8\kappa^2]}$ that all $[8\kappa^2]$ tokens have been created by $\mathcal{A}$, and 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 by sending a retrieve message $(\mathsf{retreive}, \mathsf{sid})$ to $\mathcal{F}_{\mathrm{gWRAP}}$ which returns the list $\mathcal{Q}_{\mathsf{sid}}$ of illegitimate queries. 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 $\mathcal{F}_{\mathrm{OT}}$ 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 5.9 $\{\mathbf{View}^{\mathcal{F}_{\mathrm{gWRAP}}}_{\Pi^{\mathrm{GUC}}_{\mathrm{OT}},\mathcal{A},\mathcal{Z}}(\kappa)\}_{\kappa \in \mathbb{N}} \overset{\mathrm{c}}{\approx} \{\mathbf{View}^{\mathcal{F}_{\mathrm{OT}}}_{\pi_{\mathrm{IDEAL}},\mathcal{S},\mathcal{Z}}(\kappa)\}_{\kappa \in \mathbb{N}}$.

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 sender and internally emulates the protocol $\Pi^{\mathrm{GUC}}_{\mathrm{OT}}$ 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 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 5.10 $\{\mathbf{View}^{\mathcal{F}_{\mathrm{gWRAP}}}_{\Pi^{\mathrm{GUC}}_{\mathrm{OT}},\mathcal{A},\mathcal{Z}}(\kappa)\}_{\kappa \in \mathbb{N}} \approx \{\mathbf{View}_{\Pi_{\mathrm{H}_0},\mathcal{S}_0,\mathcal{Z}}(\kappa)\}_{\kappa \in \mathbb{N}}$.

Hybrids $\mathrm{H}_{1,0} \ldots, \mathrm{H}_{1,3\kappa}$: We define a collection of hybrid executions such that for every $l \in [3\kappa]$ hybrid $\mathrm{H}_{1,l}$ is defined as follows. We modify the code of token $\mathsf{TK}^{\mathsf{PRF},l}_{\mathrm{S}}$ 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 $\mathsf{TK}^{\mathsf{PRF},l}_{\mathrm{S}}$ 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 5.11 *For every* $l \in [3\kappa]$, $\{\mathbf{View}_{\Pi_{\mathrm{H}_{1,l-1}},\mathcal{S}_{1,l-1},\mathcal{Z}}(\kappa)\}_{\kappa \in \mathbb{N}} \overset{\mathrm{c}}{\approx} \{\mathbf{View}_{\Pi_{\mathrm{H}_{1,l}},\mathcal{S}_{1,l},\mathcal{Z}}(\kappa)\}_{\kappa \in \mathbb{N}}$.

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 $\mathsf{TK}^{\mathsf{PRF},l}_{\mathrm{S}}$ results in the same response. Using a proof analogous to Lemma 4.10, we obtain the following claim.

Claim 5.12 *For every $l \in [3\kappa]$,* $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_{2,l-1}},\mathcal{S}_{2,l-1},\mathcal{Z}(\kappa)}\right\}_{\kappa\in\mathbb{N}} \overset{\mathrm{s}}{\approx} \left\{\mathbf{View}_{\Pi_{\mathrm{H}_{2,l}},\mathcal{S}_{2,l},\mathcal{Z}(\kappa)}\right\}_{\kappa\in\mathbb{N}}$.

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 5.13 $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_{2,3\kappa}},\mathcal{S}_{2,3\kappa},\mathcal{Z}(\kappa)}\right\}_{\kappa\in\mathbb{N}} \overset{\mathrm{s}}{\approx} \left\{\mathbf{View}_{\Pi_{\mathrm{H}_3},\mathcal{S}_3,\mathcal{Z}(\kappa)}\right\}_{\kappa\in\mathbb{N}}$.

Hybrid H_4: In this hybrid execution, the simulator instead of creating 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 4.11 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 5.14 $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_3},\mathcal{S}_3,\mathcal{Z}(\kappa)}\right\}_{\kappa\in\mathbb{N}} \overset{\mathrm{s}}{\approx} \left\{\mathbf{View}_{\Pi_{\mathrm{H}_4},\mathcal{S}_4,\mathcal{Z}(\kappa)}\right\}_{\kappa\in\mathbb{N}}$.

Hybrid H_5: In this hybrid execution, instead of creating 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 5.15 $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_4},\mathcal{S}_4,\mathcal{Z}(\kappa)}\right\}_{\kappa\in\mathbb{N}} \overset{\mathrm{s}}{\approx} \left\{\mathbf{View}_{\Pi_{\mathrm{H}_5},\mathcal{S}_5,\mathcal{Z}(\kappa)}\right\}_{\kappa\in\mathbb{N}}$.

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 5.16 $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_5},\mathcal{S}_5,\mathcal{Z}(\kappa)}\right\}_{\kappa\in\mathbb{N}} \overset{\mathrm{s}}{\approx} \left\{\mathbf{View}_{\Pi_{\mathrm{H}_6},\mathcal{S}_6,\mathcal{Z}(\kappa)}\right\}_{\kappa\in\mathbb{N}}$.

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 5.17 $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_6},\mathcal{S}_6,\mathcal{Z}}(\kappa)\right\}_{\kappa\in\mathbb{N}} \equiv \left\{\mathbf{View}_{\Pi_{\mathrm{H}_7},\mathcal{S}_7,\mathcal{Z}}(\kappa)\right\}_{\kappa\in\mathbb{N}}$.

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

Claim 5.18 $\left\{\mathbf{View}_{\Pi_{\mathrm{H}_7},\mathcal{S}_7,\mathcal{Z}}(\kappa)\right\}_{\kappa\in\mathbb{N}} \overset{\mathrm{c}}{\approx} \left\{\mathbf{View}_{\Pi_{\mathrm{H}_8},\mathcal{S}_8,\mathcal{Z}}(\kappa)\right\}_{\kappa\in\mathbb{N}}$.

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 two-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 two-round commitment scheme along with the first message of the protocol.

5.1 Two-Round 2PC Using Stateless Tokens

In [IKO+11], the authors provide a two-round UC secure protocol in the OT-hybrid between a sender and a receiver where the receiver obtains the output of the computation at the end of the second round. First, we observe that we can repeat our OT protocol in parallel. Then, obtaining UC secure two-party computation using tokens is carried out by running the two-round protocol of [IKO+11] in parallel with our OT protocol.

Namely, upon receiving the second message for the [IKO+11] and OT protocols, the receiver computes the OT outcome and uses these to compute the outcome of the [IKO+11] protocol.

In more details, in order to achieve simulation when the sender is corrupted, we rely on the receiver simulation for both our OT protocol and the [IKO+11] protocol. Next, we observe that, after the simulation submits the receiver's first message, it can extract the sender's input by extracting the sender's input to the OT tokens. To achieve simulation when the receiver is corrupted, the simulator first extracts the receiver's input by extracting the receiver's input to the OT tokens. Then the simulation queries the ideal functionality to obtain the output of the function evaluation on their private inputs. Using the output, the simulator next sets up the OT part of the sender's message using the OT simulation and submits the [IKO+11] sender's message using the [IKO+11] simulation. Thus, we obtain the following theorem:

Theorem 5.2 *Assuming one-way functions, there exists a two-round two-party protocol for any well-formed functionality that is GUC secure in the presence of static malicious adversaries.*

5.2 GUC-Secure MPC using Stateless Tokens from One-Way Functions

From the work of [IPS08], we know that assuming one-way functions, there exists a multiparty protocol in the OT-hybrid to securely realize any well-formed functionality with UC-security. Since, we realize the GUC-OT functionality combining with the works of [IPS08] we obtain the following corollary:

Theorem 5.3 *Assuming one-way functions, there exists a $O(d_f)$ multi-party protocol for any well-formed functionality f that is GUC secure in the presence of static malicious adversaries where d_f is the depth of the circuit implementing the function f.*

5.3 On Reusability

As in our protocol from Section 4, we discuss below how to handle exchange tokens just once and reusing them for an *unbounded* number of oblivious transfers. Recall that the sender sends two sets of tokens: PRF tokens in the first message and OT tokens in the second message whose codes depend on the first message of the receiver. We handle each of these sets in a different way, where reducing the number of PRF tokens is as discussed in Section 4.3. More concretely, we consider the following modified tokens.

PRF Tokens:

1. $\{\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l}\}_{l\in[6\kappa^2]}$: 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. Then for each $l' \in [3\kappa]$, S creates the sequence of tokens $\mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l'_1}, \ldots, \mathsf{TK}_{\mathrm{S}}^{\mathsf{PRF},l'_{2\kappa}}$ by sending the message $\{\mathsf{Create}, \mathsf{sid}, \mathsf{ssid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{1+(l'-1)2\kappa+j}, M_1\}_{j\in[2\kappa]}$, that on input x, outputs $\mathsf{PRF}_{\gamma_{l'}}(x)$, where M_1 is the functionality.

2. Similarly, R generates the tokens $\{\mathsf{TK}_{\mathrm{R}}^{\mathsf{PRF},\hat{l}}\}_{\hat{l}\in[16\kappa^3]}$ which are analogous to the sender's PRF tokens by sending $\{\mathsf{Create}, \mathsf{sid}, \mathsf{ssid}, \mathrm{S}, \mathrm{R}, \mathsf{mid}_{6\kappa^2+1+(\hat{l}-1)2\kappa+j}, M_2\}_{j\in[2\kappa]}$ for all $\hat{l}' \in [8\kappa^2]$.

OT Tokens:

1. $\{\mathsf{TK}_j\}_{j\in[2\kappa]}$: S chooses a random PRF key γ' for family F'. Let $\mathsf{PRF}'_{\gamma'} : \{0,1\}^{5\kappa} \to \{0,1\}^{\kappa}$ denote the pseudorandom function. Then, for each $j \in [2\kappa]$, S creates a token TK_j by sending $(\mathsf{Create}, \mathsf{sid}, \mathsf{ssid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{6\kappa^2+16\kappa^3+1+j}, M_3)$, where M_3 is the functionality that on input $(\sigma, \mathsf{sid}, \mathsf{ssid},$

$b_j, \mathsf{com}_{b_j}, \mathsf{decom}_{b_j})$, aborts if decom_{b_j} is not verified correctly or σ is not (the unique) valid signature of com_{b_j}, corresponding to the verification key vk_τ, where vk_τ is the key generated for $\tau = \mathsf{sid}\|\mathsf{ssid}\|\mathsf{com}_{b_i}\|j$.

If both the checks pass then the token computes $(x_0, x_1) = \mathsf{PRF}'_{\gamma'}(\mathsf{sid}\|\mathsf{ssid})$ and secret shares them using $\mathcal{SS}$ as in $\Pi^{\mathrm{OT}}_{\mathrm{GUC}}$. Finally, it outputs $(A_{b_j}[\cdot, j], \mathsf{decom}_{A_{b_j}[\cdot,j]}, B_{b_j}[\cdot, j], \mathsf{decom}_{B_{b_j}[\cdot,j]})$.

2. $\{\widehat{\mathsf{TK}}_i\}_{i\in[\kappa]}$: S chooses a PRF key γ' for family F' (same key as above). Let $\mathsf{PRF}'_{\gamma'} : \{0,1\}^* \to \{0,1\}^\kappa$ denote the pseudorandom function. Then, for each $i \in [\kappa]$, S creates a token $\widehat{\mathsf{TK}}_i$ by sending $(\mathsf{Create}, \mathsf{sid}, \mathsf{ssid}, \mathrm{R}, \mathrm{S}, \mathsf{mid}_{16\kappa^2+6\kappa^2+1+2\kappa+i}, M_4)$, where M_4 is the functionality that on input $(\sigma, \mathsf{sid}, \mathsf{ssid}, c_i, \mathsf{com}_{c_i}, \mathsf{decom}_{c_i})$ aborts if decom_{c_i} is not verified correctly or σ is not (the unique) valid signature of com_{c_i}, corresponding to the verification key vk_τ, where vk_τ is the key generated for $\tau = \mathsf{sid}\|\mathsf{ssid}\|\mathsf{com}_{c_i}\|i$.

   If both the checks pass then the token picks two random strings $(x_0, x_1) = \mathsf{PRF}'_{\gamma'}(\mathsf{sid}\|\mathsf{ssid})$ and secret shares them using $\mathcal{SS}$ as in $\Pi^{\mathrm{OT}}_{\mathrm{GUC}}$. Finally, it outputs

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

Condition on the event that none of the parties successfully forges a signature, then our proof for Theorem 5.1 follows similarly (with the modifications that involve extraction from 2κ tokens per commitment).

6 Three-Round Token-Based GUC Secure Multi-Party Computation

In this section, we show how to compile an arbitrary round semi-honest protocol Π to a three-round protocol using stateless tokens. As discussed in the introduction, the high-level of our approach is borrowing the compressing round idea from [GGHR14] which proceeds in three steps. In the first step, all parties commit to their inputs via an extractable commitment and then in the second step, each party provides a token to emulate their actions with respect to Π given the commitments. Finally, each party runs the protocol Π locally and obtains the result of the computation. For such an approach to work, it is crucial that an adversary, upon receiving the tokens, is not be able to "rewind" the computation and launch a resetting attack. This is ensured via zero-knowledge proofs that are provided in each round. In essence, the zero-knowledge proofs validates the actions of each party with respect to the commitments made in the first step. Such a mechanism is also referred to as a commit-and-prove strategy. In Section 6.1.1, we will present a construction of a commit-and-prove protocol in the $\mathcal{F}_{\mathrm{gWRAP}}$-hybrid and then design our MPC protocol using this protocol. We then take a modular approach by describing our MPC protocol in an idealized version of the commit-and-prove functionality analogous to [CLOS02] and then show how to realize this functionality. As we mentioned before we then rely on the approach of Garg et al. [GGHR14] to compress the rounds of our MPC protocol compiled with our commit and prove protocol in 3 rounds. Due to space constraints we present this in the full version [**?**].

6.1 One-Many Commit-and-Prove Functionality

The commit and prove functionality $\mathcal{F}_{\mathrm{CP}}$ introduced in [CLOS02] is a generalization of the commitment functionality and is core to constructing protocols in the GUC-setting. The functionality parameterized by an NP-relation $\mathcal{R}$ proceeds in two stages: The first stage is a commit phase where the receiver obtains a commitment to some value w. The second phase is a prove phase where the functionality upon receiving a statement x from the committer sends x to the receiver along with the value $\mathcal{R}(x, w)$. We will generalize the $\mathcal{F}_{\text{CP}}$-functionality in two ways. First, we will allow for asserting multiple statements on a single committed value w in the $\mathcal{F}_{\text{gWRAP}}$-hybrid. Second, we will allow a single party to assert the statement to many parties. In an MPC setting this will be useful as each party will assert the correctness of its message to all parties in each step. Our generalized functionality can be found in Figure 5 and is parameterized by an NP relation $\mathcal{R}$ and integer $m \in \mathbb{N}$ denoting the number of statements to be proved.

Functionality $\mathcal{F}_{\text{CP}}^{1:\text{M}}$

Functionality $\mathcal{F}_{\text{CP}}^{1:\text{M}}$ is parameterized by an NP-relation $\mathcal{R}$, an integer m and an implicit security parameter κ, and runs with set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$.

Commit Phase: Upon receiving a message $(\mathsf{commit}, \mathsf{sid}, \mathcal{P}, w)$ from P_i, where $w \in \{0,1\}^\kappa$, record the tuple $(\mathsf{sid}, P_i, \mathcal{P}, w, 0)$ and send $(\mathsf{receipt}, P_i, \mathcal{P}, \mathsf{sid})$ to all parties in $\mathcal{P}$.

Prove Phase: Upon receiving a message $(\mathsf{prove}, \mathsf{sid}, \mathcal{P}, x)$ from P_i, where $w \in \{0,1\}^{poly(\kappa)}$, find the record $(\mathsf{sid}, P_i, \mathcal{P}, w, ctr_{\mathsf{sid}})$. If no such record is found or $ctr_{\mathsf{sid}} \geq m$ then ignore. Otherwise, send $(\mathsf{proof}, \mathsf{sid}, \mathcal{P}, (x, \mathcal{R}(x, w)))$ to all parties in $\mathcal{P}$. Replace the tuple $(\mathsf{sid}, P_i, \mathcal{P}, w, ctr_{\mathsf{sid}})$ with $(\mathsf{sid}, P_i, \mathcal{P}, w, ctr_{\mathsf{sid}} + 1)$.

Figure 5: The one-many multi-theorem commit and prove functionality [CLOS02].

To realize this functionality, we will rely on the so-called input-delayed proofs [LS90, CPS$^+$16a, CPS$^+$16b, HV16]. In particular, we rely on the recent work of Hazay and Venkitasubramaniam [HV16], who showed how to obtain a 4-round commit-and-prove protocol where the underlying commitment scheme and one-way permutation are used in a black-box way, and requires the statement only in the last round. Below, we extend their construction and design a protocol Π_{CP} that securely realizes functionality $\mathcal{F}_{\text{CP}}^{1:\text{M}}$, and then prove the following theorem.

Theorem 6.1 *Assuming the existence of one-way functions, then protocol Π_{CP} securely realizes the $\mathcal{F}_{\text{CP}}^{1:\text{M}}$-functionality in the $\mathcal{F}_{\text{gWRAP}}$-hybrid.*

6.1.1 Realizing $\mathcal{F}_{\text{CP}}^{1:\text{M}}$ in the $\mathcal{F}_{\text{gWRAP}}$-Hybrid

In the following section we extend ideas from [HV16] in order to obtain a one-many commit-and-prove protocol with negligible soundness using a specialized randomized encodings (RE) [IK00, AIK04], where the statement is only known at the last round. Loosely speaking, RE allows to represent a "complex" function by a "simpler" randomized function. Given a string $w_0 \in \{0,1\}^n$, the [HV16] protocol considers a randomized encoding of the following function:

$$f_{w_0}(x, w_1) = (\mathcal{R}(x, w_0 \oplus w_1), x, w_1)$$

where $\mathcal{R}$ is the underlying NP relation and the function has the value w_0 hardwired in it. The RE we consider needs to be secure against *adaptively chosen inputs* and *robust*. Loosely speaking, an RE is secure against adaptive chosen inputs if both the encoding and the simulation can be decomposed into offline and online algorithms and security should hold even if the input is chosen adaptively after seeing the offline part of the encoding. Moreover, an offline/online RE is said to be robust if no adversary can produce an offline part following the honest encoding algorithm and a (maliciously generated) online part that evaluates to a value outside the range of the function. Then the ZK proof follows by having the prover generate the offline phase of the randomized encoding for this functionality together with commitments to the randomness r used for this generation and w_1. Next, upon receiving a challenge bit ch from the verifier, the prover completes the proof as follows. In case $ch = 0$, then the prover reveals r and w_1 for which the verifier checks the validity of the offline phase. Otherwise, the prover sends the online part of the encoding and a decommitment of w_1 for which the verifier runs the decoder and checks that the outcome is $(1, x, w_1)$.

A concrete example based on garbled circuits [Yao86] implies that the offline part of the randomized encoding is associated with the garbled circuit, where the randomness r can be associated with the input key labels for the garbling. Moreover, the online part can be associated with the corresponding input labels that enable to evaluate the garbled circuit on input x, w_1. Clearly, a dishonest prover cannot provide both a valid garbling and a set of input labels that evaluates the circuits to 1 in case x is a false statement. Finally, adaptive security is achieved by employing the construction from [HJO$^+$15] (see [HV16] for a discussion regarding the robustness of this scheme).

We discuss next how to extend Theorem 5.5 from [HV16] by adding the one-many multi-theorem features. In order to improve the soundness parameter of their ZK proof Hazay and Venkitasubramaniam repeated their basic proof sufficiently many times in parallel, using fresh witness shares each time embedding the [IKOS09] approach in order to add a mechanism that verifies the consistency of the shares. Consider a parameter N to be the number of repetitions and let m denote the number of proven theorems. Our protocol employs two types of commitments schemes: (1) Naor's commitment scheme [Nao91] denoted by Com. (2) Token based extractable commitment scheme in the $\mathcal{F}_{\text{gWRAP}}$-hybrid denoted by $\mathsf{Com}_{\text{gWRAP}}$ and defined as follows. First, the receiver R in the commitment scheme will prepare a token that computes a PRF under a randomly chosen key k and send it to the committer in an initial setup phase, incorporated with the session identifier sid. Such that on input (x, sid) the token outputs PRF evaluated on the input x. More, precisely, the receiver on input sid creates a token $\mathsf{TK}^{\mathsf{PRF}_k}$ with the following code:

- On input $(x, \widetilde{\mathsf{sid}})$: If $\widetilde{\mathsf{sid}} = \mathsf{sid}$ output $\mathsf{PRF}_k(x)$. Otherwise, output $\perp$.

Then, to commit to a bit b, the committer C first queries the token $\mathsf{TK}^{\mathsf{PRF}_k}$ on input (u, sid) where $u \in \{0,1\}^{5\kappa}$ is chosen at random and sid is the session identifier. Upon receiving the output v from the token, it sends $(\mathsf{Ext}(u) \oplus b, v)$ where Ext is a randomness extractor as used in Section 5. We remark here that if the tokens are exchanged initially in a token exchange phase, then the commitment scheme is non-interactive.

Protocol 3 *Protocol Π_{CP} - one-many commit-and-prove protocol.*

- **Input:** *The prover holds a witness w, where the prover is a designated party P_τ for some $\tau \in [n]$.*
- **The Protocol:**
  1. *Each party P_k for $k \neq \tau$ plays the role of the verifier and picks random m t-subsets I_j^k of $[N]$ for each $j \in [m]$ and $k \in [n-1]$ where m is the number of proven statements. It also picks t random challenge bits $\{ch_{i,j}^k\}_{i \in I_j^k}$ and commits to them using $\mathsf{Com}_{\text{gWRAP}}^k$. It further sends the first message of the Naor's commitment scheme.*
  2. *The prover then continues as follows:*
     (a) *It first generates $N \times m \times (n-1)$ independent XOR sharings of the witness w, say*

$$\{w_{i,j,k}^0, w_{i,j,k}^1\}_{(i \times j \times k) \in [N \times m \times (n-1)]}.$$

(b) *Next, for each $j \in [m]$ and $k \in [n-1]$, it generates the views of $2N$ parties $P^0_{i,j,k}$ and $P^1_{i,j,k}$ for all $i \in [N]$ executing a t-robust t-private MPC protocol, where $P^b_{i,j,k}$ has input $w^b_{i,j,k}$, that realizes the functionality that checks if $w^0_{i,j,k} \oplus w^1_{i,j,k}$ are all equal. Let $V^b_{i,j,k}$ be the view of party $P^b_{i,j,k}$.*

(c) *Next, for each $j \in [m]$ and $k \in [n-1]$, it computes N offline encodings of the following set of functions:*

$$f_{w^0_{i,j,k}, V^0_{i,j,k}}(x_j, w^1_{i,j,k}, V^1_{i,j,k}) = (b, x_j, w^1_{i,j,k}, V^1_{i,j,k})$$

where $b = 1$ if and only if $\mathcal{R}(x_j, w^0_{i,j,k} \oplus w^1_{i,j,k})$ holds and the views $V^0_{i,j,k}$ and $V^1_{i,j,k}$ are consistent with each other.

(d) *Finally, the prover broadcasts to all parties the set containing*

$$\big\{(f^{\mathsf{off}}_{w^0_{i,j,k}, V^0_{i,j,k}}(r_{i,j,k}), \mathsf{Com}(r_{i,j,k}), \mathsf{Com}(w^0_{i,j,k}), \mathsf{Com}(w^1_{i,j,k}), \mathsf{Com}(V^0_{i,j,k}), \mathsf{Com}(V^1_{i,j,k}))\big\}_{(i \times j \times k) \in [N \times m \times (n-1)]}.$$

Moreover, let $\mathsf{decom}_{r_{i,j,k}}, \mathsf{decom}_{w^0_{i,j,k}}, \mathsf{decom}_{w^1_{i,j,k}}, \mathsf{decom}_{V^0_{i,j,k}}, \mathsf{decom}_{V^1_{i,j,k}}$ be the respective decommitment information of the above commitments. Then for every $k \in [n-1]$, P_i commits to the above decommitment information with respect to party P_k and all $(i \times j) \in [N] \times [m]$, using $\mathsf{Com}_{\mathrm{gWRAP}}$.

3. *The verifier decommits to all its challenges.*
4. *For every index (i, j) in the t subset the prover replies as follows:*
   - *If $ch^i_{j,k} = 0$ then it decommits to $r_{i,j,k}$, $w^0_{i,j,k}$ and $V^0_{i,j,k}$. The verifier then checks if the offline part was constructed correctly.*
   - *If $ch^i_{j,k} = 1$ then it sends $f^{\mathsf{on}}_{w^0_{i,j,k}, V^0_{i,j,k}}(r_{i,j,k}, x_j, w^1_{i,j,k}, V^1_{i,j,k})$ and decommits $w^1_{i,j,k}$ and $V^1_{i,j,k}$. The verifier then runs the decoder and checks if it obtains $(1, x_j, w^1_{i,j,k}, V^1_{i,j,k})$.*

   Furthermore, from the decommitted views $V^{ch^i_{j,k}}_{i,j,k}$ for every index (i, j) that the prover sends, the verifier checks if the MPC-in-the-head protocol was executed correctly and that the views are consistent.

Theorem 6.2 *Assuming the existence of one-way functions, then protocol Π_{CP} GUC realizes $\mathcal{F}^{1:\mathrm{M}}_{\mathrm{CP}}$ in the $\mathcal{F}_{\mathrm{gWRAP}}$-hybrid.*

Proof: Let $\mathcal{A}$ be a malicious PPT real adversary attacking protocol Π_{CP} in the $\mathcal{F}_{\mathrm{gWRAP}}$-hybrid model. We construct an ideal adversary $\mathcal{S}$ with access to $\mathcal{F}^{1:\mathrm{M}}_{\mathrm{CP}}$ which simulates a real execution of Π_{CP} with $\mathcal{A}$ such that no environment $\mathcal{Z}$ can distinguish the ideal process with $\mathcal{S}$ and $\mathcal{F}_{\mathrm{gWRAP}}$-hybrid from a real execution of Π_{CP} with $\mathcal{A}$ in the $\mathcal{F}_{\mathrm{gWRAP}}$-hybrid. $\mathcal{S}$ 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 $\mathcal{Z}$. In case the adversary $\mathcal{A}$ issues a transfer query on any token $(\mathsf{transfer}, \cdot)$, $\mathcal{S}$ relays the query to the $\mathcal{F}_{\mathrm{gWRAP}}$.

Party P_τ is not corrupted. In this scenario the adversary only corrupts a subset of parties $\mathcal{I}$ playing the role of the verifiers in our protocol. The simulator proceeds as follows.

1. Upon receiving a commitment $\mathsf{Com}^k_{\mathrm{gWRAP}}$ from a corrupted party P_k, the simulator extracts the m committed t-subsets I^k_j and the challenge bits $\{ch^k_{i,j}\}_{i \in I^k_j}$ for all $j \in [m]$, by retrieving the queries made to the tokens.

2. For each $j \in [m]$ and $k \in [\mathcal{I}]$, the simulator generates the views of $2N$ parties $P^0_{i,j,k}$ and $P^1_{i,j,k}$ for all $i \in [N]$ emulating the simulator of the t-robust t-private MPC protocol underlying in the real proof, where the set of corrupted parties for the $(j,k)^{th}$ execution is fixed to be I^k_j extracted above. Let $V^b_{i,j,k}$ be the view of party $P^b_{i,j,k}$.

3. Next, for each $j \in [m]$ and $k \in [\mathcal{I}]$, the simulator computes N offline encodings as follows.

   - For every index i in the t subset I^k_j the simulator replies as follows:
     - If $ch^k_{i,j} = 0$, then the simulator broadcasts the following honestly generated message: $f^{\mathsf{off}}_{w^0_{i,j,k},V^0_{i,j,k}}(r_{i,j,k}), \mathsf{Com}(r_{i,j,k}), \mathsf{Com}(w^0_{i,j,k}), \mathsf{Com}(0),$ $\mathsf{Com}(V'^0_{i,j,k}), \mathsf{Com}(V'^1_{i,j,k})$. where $V'^0_{i,j,k} = 0$ and $V'^1_{i,j,k} = V^1_{i,j,k}$ if the matched challenge bit equals one, and vice versa.
     - Else, if $ch^k_{i,j} = 1$, then the simulator invokes the simulator for the randomized encoding and broadcasts the following message:

     $$\{\mathcal{S}^{\mathsf{off}}_{w^0_{i,j,k},V^0_{i,j,k}}(r_{i,j,k}), \mathsf{Com}(0), \mathsf{Com}(0), \mathsf{Com}(w^1_{i,j,k}), \mathsf{Com}(V'^0_{i,j,k}), \mathsf{Com}(V'^1_{i,j,k})\}_{(i \times j \times k) \in [N \times m \times (n-1)]}$$

     where $w^1_{i,j,k}$ is a random string and $V'^0_{i,j,k} = 0$ and $V'^1_{i,j,k} = V^1_{i,j,k}$ if the matched challenge bit equals one, and vice versa.
   - For every index i not in the t subset I^k_j the simulator broadcasts

   $$f^{\mathsf{off}}_{w^0_{i,j,k},V^0_{i,j,k}}(r_{i,j,k}), \mathsf{Com}(r_{i,j,k}), \mathsf{Com}(w^0_{i,j,k}), \mathsf{Com}(0), \mathsf{Com}(0), \mathsf{Com}(0).$$

   The simulator correctly commits to the decommitments information with respect to the honestly generated commitments (namely, as the honest prover would have done) using $\mathsf{Com}_{\mathrm{gWRAP}}$. Else, it commits to the zero string.

4. Upon receiving the decommitment information from the adversary, the simulator aborts if the adversary decommits correctly to a different set of messages than the one extracted above by the simulator.

5. Else, $\mathcal{S}$ completes the protocol by replying to the adversary as the honest prover would do.

Note that the adversary's view is modified with respect to the views it obtains with respect to the underlying MPC and both types of commitments. Indistinguishability follows by first replacing the simulated views of the MPC execution with a real execution. Namely the simulator for this hybrid game commits to the real views. Indistinguishability follows from the privacy of the protocol. Next, we modify the fake commitments into real commitments computed as in the real proof. The reduction for this proof follows easily as the simulator is not required to open these commitments.

Party P_τ is corrupted. In this scenario the adversary corrupts a subset of parties $\mathcal{I}$ playing the role of the verifiers in our protocol as well as the prover. The simulator for this case follows the honest verifier's strategy $\{P_k\}_{k \notin [\mathcal{I}]}$, with the exception that it extracts the prover's witness by extracting one of the witness' pairs. Recall that only the decommitment information is committed via the extractable commitment scheme $\mathsf{Com}_{\mathrm{gWRAP}}$. Since a commitment is made using tokens from every other party and there is at least one honest party, the simulator can extract the decommitment information and from that extract the real value. We point out that in general extracting out shares from only one-pair could cause the problem of "over-extraction" where the adversary does not necessarily commit to shares of the same string in each pair. In our protocol this is not an issue because in conjunction with committing to these shares, it also commits to the views of an MPC-in-the-head protocol which verifies that all shares are correct. Essentially, the soundness argument follows by showing that if an adversary deviates, then with high-probability the set $\mathcal{I}$ will include a party with an "inconsistent view". This involves a careful argument relying on the so-called t-robustness of the underlying MPC-in-the-head protocol. Such an argument is presented in [HV16] to get negligible soundness from constant soundness and this proof can be naturally extended to our setting (our protocol simply involves more repetitions but the MPC-in-the-head views still ensure correctness of all repetition simultaneously).

As for straight-line extraction, the argument follows as for the simpler protocol. Namely, when simulating the verifier's role the simulator extracts the committed values within the forth message of the prover. That is, following a similar procedure of extracting the committed message via obtaining the queries to the token, it is sufficient to obtain two shares of the witness as the robustness of the MPC protocol ensures that all the pairs correspond to the same witness. ■

6.2 Warmup: Simple MPC Protocol in the $\mathcal{F}_{\rm CP}^{\rm 1:M}$-Hybrid

We next describe our MPC protocol in the $\mathcal{F}_{\rm CP}^{\rm 1:M}$-hybrid. On a high-level, we follow GMW-style compilation [GMW87] of a semi-honest secure protocol Π to achieve malicious security using the $\mathcal{F}_{\rm CP}^{\rm 1:M}$-functionality. Without loss of generality, we assume that in each round of the semi-honest MPC protocol Π, each party broadcasts a single message that depends on its input and randomness and on the messages that it received from all parties in all previous rounds. We let $m_{i,j}$ denote the message sent by the i^{th} party in the j^{th} round in the protocol Π. We define the function π_i such that $m_{i,t} = \pi_i(x_i, r_i, (M_1, \ldots, M_{t-1}))$ where $m_{i,t}$ is the t^{th} message generated by party P_i in protocol Π with input x_i, randomness r_i and where M_r is the message sent by all parties in round i of Π.

Protocol description. Our protocol $\Pi_{\rm MPC}$ proceeds as follows:

Round 1. In the first round, the parties commit to their inputs and randomness. More precisely, on input x_i, party P_i samples random strings $r_{i,1}, r_{i,2}, \ldots, r_{i,n}$ and sends $(\mathsf{commit}, \mathsf{sid}, \mathcal{P}, \overline{w})$ to $\mathcal{F}_{\rm CP}^{\rm 1:M}$ and $\overline{w} = (x, R_i)$ where $R_i = (r_{i,1}, r_{i,2}, \ldots, r_{i,n})$.

Round 2. P_i broadcasts shares $\overline{R}_i = R_i - \{r_{i,i}\}$ and sends $(\mathsf{prove}, P_i, \mathcal{P}, \overline{R}_i)$. Let $M_0 = (\overline{R}_1, \ldots, \overline{R}_n)$.

Round $2+\delta$. Let $M_{\delta-1}$ be the messages broadcast by all parties in rounds $3, 4, \ldots, 2+(\delta-1)$ and let $m_{i,\delta} = \pi_i(x_i, r_i, (M_1, \ldots, M_{\delta-1}))$ where $r_i = \oplus_j r_{j,i}$. P_i broadcasts $m_{i,\delta}$ and sends to $\mathcal{F}_{\rm CP}^{\rm 1:M}$ the message $(\mathsf{prove}, P_i, \mathcal{P}, \overline{M}_{t-1} : m_{i,\delta})$ where $\overline{M}_{\delta-1} = (M_0, M_1, \ldots, M_{\delta-1})$.

The NP-relation $\mathcal{R}$ used to instantiate the $\mathcal{F}_{\rm CP}^{\rm 1:M}$ functionality will include:

1. (M_0, R_i) : if M_0 contains $\overline{R}_i$ as its i^{th} component where $\overline{R}_i = R_i - \{r_{i,i}\}$ and $R_i = \{r_{i,1}, \ldots, r_{i,n}\}$.
2. $((\overline{M}_{\delta-1}, m_{i,\delta}), (x_i, R_i))$: if $(M_0, R_i) \in \mathcal{R}$ and $m_{i,\delta} = \pi_i(x_i, r_i, (M_1, \ldots, M_{\delta-1}))$ where $r_i = \oplus_{j\in[n]} r_{j,i}$, $\overline{M}_{\delta-1} = (M_0, M_1 \ldots, M_{\delta-1})$ and $R_i = \{r_{i,1}, \ldots, r_{i,n}\}$.

Theorem 6.3 *Let f be any deterministic polynomial-time function with n inputs and a single output. Assume the existence of one-way functions and an n-party semi-honest MPC protocol Π. Then the protocol Π_{MPC} GUC realizes $\mathcal{F}_f$ in the $\mathcal{F}_{\text{CP}}^{1:\text{M}}$-hybrid.*

Proof: Let $\mathcal{A}$ be a malicious PPT real adversary attacking protocol Π_{MPC} in the $\mathcal{F}_{\text{CP}}^{1:\text{M}}$-hybrid model. We construct an ideal adversary $\mathcal{S}$ with access to $\mathcal{F}_f$ which simulates a real execution of Π_{MPC} with $\mathcal{A}$ such that no environment $\mathcal{Z}$ can distinguish the ideal process with $\mathcal{S}$ interacting with $\mathcal{F}_f$ from a real execution of Π_{MPC} with $\mathcal{A}$ in the $\mathcal{F}_{\text{CP}}^{1:\text{M}}$-hybrid. $\mathcal{S}$ 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 honest parties: Let $\mathcal{I}$ be the set of parties corrupted by the adversary $\mathcal{A}$. This means $\mathcal{S}$ needs to simulate all messages from parties in $\mathcal{P}/\mathcal{I}$. $\mathcal{S}$ emulates the $\mathcal{F}_{\text{CP}}^{1:\text{M}}$ functionality for $\mathcal{A}$ as follows. For every $P_j \in \mathcal{P}/\mathcal{I}$ it sends the commitment message $(\mathsf{receipt}, P_j, \mathcal{P}, \mathsf{sid})$ to all parties $P_i \in \mathcal{I}$. Next, for every message $(\mathsf{commit}, \mathsf{sid}, P_i, \mathcal{P}, \overline{w}_i)$ received from $\mathcal{A}$, it records $\overline{w}_i = (x_i, r_{i,1}, \ldots, r_{i,n})$. Upon receiving this message on behalf of every $P_i \in \mathcal{I}$, the simulator $\mathcal{S}$ sends x_i on behalf of every $P_i \in \mathcal{I}$ to $\mathcal{F}_f$ and obtains the result of the computation output. Then using the simulator of the semi-honest protocol Π, it generates random tapes r_i for every $P_i \in \mathcal{I}$ and messages $m_{j,\delta}$ for all honest parties $P_j \in \mathcal{P}/\mathcal{I}$ and all rounds δ. Next, it sends $\overline{R}_j$ on behalf of the honest parties $P_j \in \mathcal{P}/\mathcal{I}$ so that for every $P_i \in \mathcal{I}$, $r_i = \oplus r_{j,i}$. This is possible since there is at least one party P_j outside $\mathcal{I}$ and $\mathcal{S}$ can set $r_{j,i}$ so that it adds to r_i. Next, in round $2 + \delta$, it receives the messages from $P_i \in \mathcal{I}$ and supplies messages from the honest parties according to the simulation of P_i. Along with each message it receives the prove message that the parties in $\mathcal{I}$ send to $\mathcal{F}_{\text{CP}}^{1:\text{M}}$. $\mathcal{S}$ simply honestly emulates $\mathcal{F}_{\text{CP}}^{1:\text{M}}$ for these messages. For messages that the honest parties send to $\mathcal{F}_{\text{CP}}^{1:\text{M}}$, $\mathcal{S}$ simply sends the receipt message to all parties in $\mathcal{I}$.

Indistinguishability of the simulation follows from the following two facts:

- Given an input x_i and random tape r_i for every $P_i \in \mathcal{I}$ and the messages from the honest parties, there is a unique emulation of the semi-honest protocol Π where all the messages from parties P_i if honestly generated are deterministic.
- Since the simulation is emulating the $\mathcal{F}_{\text{CP}}^{1:\text{M}}$ functionality, the computation immediately aborts if a corrupted party P_i deviates from the deterministic strategy.

■

6.3 Three-Round MPC Protocol in the $\mathcal{F}_{\text{gWRAP}}$-Hybrid

In this section, we show how to modify the previous protocol Π_{MPC} to a three-round protocol. The high-level idea is similar to the work of [GGHR14] which shows how to compress the rounds of communication in an MPC protocol using obfuscation primitives. Instead of using obfuscation based primitives, we will directly make each party create tokens for their next-message function in the previous protocol.

As our starting point, we will look into certain properties of the protocol used to realize the $\mathcal{F}_{\text{CP}}^{1:\text{M}}$ to achieve our goal and explain this next. In more details, our methodology in this section is to first unravel the protocol Π_{MPC} from the previous section, namely, describe a protocol $\widetilde{\Pi}_{\text{MPC}}$ directly in the $\mathcal{F}_{\text{gWRAP}}$-hybrid as opposed to the $\mathcal{F}_{\text{CP}}^{1:\text{M}}$-hybrid. More formally, $\widetilde{\Pi}_{\text{MPC}}$ is the protocol Π_{MPC} where the calls to the $\mathcal{F}_{\text{CP}}^{1:\text{M}}$ protocol are replaced with the actual protocol Π_{CP} that realizes this functionality in the $\mathcal{F}_{\text{gWRAP}}$-hybrid. Then we show how to compress this protocol to a 3-round protocol $\widehat{\Pi}_{\text{MPC}}$ by adding extra tokens.

Step 1: Intermediate MPC protocol $\widetilde{\Pi}_{\text{MPC}}$ in the $\mathcal{F}_{\text{gWRAP}}$-hybrid. We begin with an overview of our protocol Π_{CP} that realizes the $\mathcal{F}_{\text{CP}}^{1:\text{M}}$ functionality. In our protocol Π_{CP} between a prover P_i and the rest of the parties as the verifier, the commit phase comprises of two messages and the prove phase for each statement comprises of two rounds. In more detail,

1. **Commit phase: first message.** In the commit phase, all parties first commit to a "challenge" for the zero-knowledge proof. The parties make the commitment using a token supplied by the prover, which, as we recall is simply a PRF token. The parties further send the first message for the Naor's commitment scheme [Nao91] for the prover.

2. **Commit phase: second message.** In the second message the prover P_i broadcasts a message, namely the offline part of randomized encodings (RE) of some function and a set of commitments. For each commitment, the prover first commits the actual string using the Naor's commitment. Then for each party P_j, the prover also commits to the decommitment information using the PRF token given by P_j (See Section 6.1).

3. **Prove phase: first message.** In the prove phase, for each statement in a first message, all receivers decommit to their challenges made using $\mathsf{Com}_{\text{gWRAP}}$. Let $\Pi_{\text{CP}}^{\text{CH}}(r)$ be the algorithm used by P_j to generate this message where ch is the message committed to in the commit phase and r the random tape used in the execution.

4. **Prove phase: second message.** If the challenges were decommitted to correctly, the prover generates a string and a set of decommitments. We denote by $\Pi_{\text{CP}}^{\text{RESP}}(x, w, r)$ the algorithm used by the prover to verify the previous message and generate this message, where x is the statement, w is the message committed in the commit phase and r the random tape.

We emphasize that the algorithms $\Pi_{\text{CP}}^{\text{CH}}$ and $\Pi_{\text{CP}}^{\text{RESP}}$ do not make use of any tokens and are deterministic polynomial-time computations on the inputs. We are now ready to describe our protocol $\widetilde{\Pi}_{\text{MPC}}$ where we replace all calls to $\mathcal{F}_{\text{CP}}^{1:\text{M}}$ with instructions from protocol Π_{CP} in the Π_{MPC}. First, we recall (and introduce) some notations first.

- $R_i = \{r_{i,1}, \ldots, r_{i,n}\}$ are the shares committed to by P_i, where $r_{i,j}$ will be one of the shares used in generating the random tape of party P_j.
- We denote by $\overline{R}_i = R_i - \{r_{i,i}\}$ and $\overline{M}_0 = (\overline{R}_1, \ldots, \overline{R}_n)$.
- $m_{i,\delta} = \pi_i(x_i, r_i, M_{\delta-1})$ is the next-message function according to the semi-honest protocol Π_{MPC}. To simplify our notation we will denote by $m_{i,0} = \overline{R}_i$.
- Let $\tau_{i,j,\delta}$ denote the random tape used by party P_i for the proof of the δ^{th} statement with P_j using Π_{CP} as the prover, and let $\tau_{i,\delta}$ denote the random tape for P_i to generate messages for Π_{CP} acting as the prover.

Round 1: Every party P_j generates the first message according to Π_{CP} acting as the receiver for a proof received from P_i (for every P_i) and broadcasts it to all parties.

Round 2: Every party P_i generates the second message according to $\Pi_{\rm CP}$ acting as the prover and broadcasts to all parties where party P_i commits to its input x_i and random strings $R_i = (r_{i,1}, \ldots, r_{i,n})$.

Then for $\delta = 0, \ldots, T$,

Round $3 + 2\delta$**:** Recall that in the $(2+\delta)^{th}$ round of $\Pi_{\rm MPC}$, party P_i sends $m_{i,\delta}$ and proves its correctness by sending the message to $\mathcal{F}_{\rm CP}^{1:{\rm M}}$. Since the prove phase of $\Pi_{\rm CP}$ comprises of two rounds, in round $3 + 2\delta$ all parties P_j acting as the receiver in $\Pi_{\rm CP}$ send their prove phase first message (which are decommitments to challenges). More precisely, party P_i computes $\Pi_{\rm CP}^{\rm CH}(\tau_{i,j,(\delta+1)})$ for every $j \in [n]$.

Round $3 + 2\delta + 1$**:** In round $3 + 2\delta + 1$ all parties P_i sends $m_{i,\delta}$ and the prover phase second message for the corresponding NP-statement, namely $\Pi_{\rm CP}^{\rm RESP}(\tau_{i,(\delta+1)})$

By the construction and the GUC-composition theorem, we know that $\widetilde{\Pi}_{\rm MPC}$ securely realizes $\mathcal{F}_f$ in the $\mathcal{F}_{\rm gWRAP}$-hybrid. Next, we briefly describe our simulation.

High-level description of the simulation. Given an adversary $\mathcal{A}$, the simulator $\widetilde{\mathcal{S}}$ will first extract the inputs and shares of random tapes supplied by the corrupted parties P_j and queries $\mathcal{F}_f$ with these inputs to obtain the output of computation. Next, it uses the simulation of $\Pi_{\rm MPC}$ as discussed in the previous section to generate the random tapes for the corrupted parties P_j and the messages from the honest parties. We remark here that once the random tapes for all corrupted parties P_j's are fixed, there is only a valid accepting transcript of the execution using $\widetilde{\Pi}_{\rm MPC}$. This follows from the soundness of protocol $\Pi_{\rm CP}$ where no corrupted P_j can deviate from the honest strategy.

Before, we proceed to our actual protocol, we make the following important observation regarding the protocol $\Pi_{\rm CP}$ used in $\widetilde{\Pi}_{\rm MPC}$.

Prove phase is token free. We remark that in protocol $\Pi_{\rm CP}$ the actions of each party (including the prover) in the prove phase can be computed without querying any token. Namely, in the first message of the prove phase the prover verifiers the decommitment of the challenges made using a token it supplied, and generates the online parts of an RE. The REs are generated using stand-alone algorithms and does not involve tokens. To verify the decommitments, since it knows the PRF keys used to make these commitments, the prover can verify the decommitments directly using the key. In the second message, each party P_j performs a "decoding" computation on the RE and verifies the decommitments. Decoding of the RE can be done again using stand-alone algorithms. To verify a decommitment, recall that P_j verifies a Naor commitment, which does not involve tokens and decom_j which involves a commitment made using a token supplied by P_j. Hence P_j can verify this without querying any token.

Step 2: our 3-round protocol $\widehat{\Pi}_{\rm MPC}$ in the $\mathcal{F}_{\rm gWRAP}$-hybrid. We describe our 3-round protocol and prove correctness. Our protocol $\widehat{\Pi}_{\rm MPC}$ is essentially $\widetilde{\Pi}_{\rm MPC}$ with the following modification. Each party P_i generates $2T + 2$ tokens (one for each round after round 2 in $\widetilde{\Pi}_{\rm MPC}$) for each party P_j. These tokens will execute the next-message function on behalf of P_i for P_j. In more detail, the r^{th} token will verify the partial transcript up until the $2 + (r-1)^{st}$ round according to $\widetilde{\Pi}_{\rm MPC}$, and if correct, generates P_i's message in the r^{th} round. We denote the r^{th} token from party P_i to party P_j that performs its actions in round $2 + r$ as $\mathsf{TK}_{i,j}^r$. More formally, the token $\mathsf{TK}_{i,j}^r$ is defined as follows:

Hardwired parameters: The messages exchanged in rounds 1 and 2, P_i's input x_i and its random tape τ_i for the protocol $\Pi_{\rm CP}$.

Code: On input a partial transcript $\overline{M}_{r-1}$, it verifies if the transcript is consistent with the protocol $\widetilde{\Pi}_{\mathrm{MPC}}$ and then outputs P_i's message in the r^{th} round. We recall here that given the partial transcript, all actions of party P_i after round 2 are polynomial-time computable from the hardwired parameters. In particular, these algorithms do not have to query any token supplied by other parties.

Theorem 6.4 *Let f be any deterministic polynomial-time function with n inputs and a single output. Assume the existence of one-way functions and an n-party semi-honest MPC protocol Π. Then protocol $\widehat{\Pi}_{\mathrm{MPC}}$ GUC realizes $\mathcal{F}_f$ in the $\mathcal{F}_{\mathrm{gWRAP}}$-hybrid model.*

Proof: Let $\mathcal{A}$ be a malicious PPT real adversary attacking protocol $\widehat{\Pi}_{\mathrm{MPC}}$ in the $\mathcal{F}_{\mathrm{gWRAP}}$-hybrid model. We construct an ideal adversary $\mathcal{S}$ with access to $\mathcal{F}_f$ which simulates a real execution of $\widehat{\Pi}_{\mathrm{MPC}}$ with $\mathcal{A}$ such that no environment $\mathcal{Z}$ can distinguish the ideal process with $\mathcal{S}$ interacting with $\mathcal{F}_f$ in the $\mathcal{F}_{\mathrm{gWRAP}}$-hybrid from a real execution of $\widehat{\Pi}_{\mathrm{MPC}}$ with $\mathcal{A}$ in the $\mathcal{F}_{\mathrm{gWRAP}}$-hybrid. $\mathcal{S}$ 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 rounds 1 and 2 of the protocol. The messages exchanged in rounds 1 and 2 in both protocols $\widetilde{\Pi}_{\mathrm{MPC}}$ and $\widehat{\Pi}_{\mathrm{MPC}}$ are identical. The simulator $\mathcal{S}$ follows the simulation $\widetilde{\mathcal{S}}$ of the protocol $\widetilde{\Pi}_{\mathrm{MPC}}$ for these rounds.

Simulating access to token $\mathsf{TK}^r_{i,j}$. Recall that in $\widehat{\Pi}_{\mathrm{MPC}}$, the parties exchange tokens $\mathsf{TK}^r_{i,j}$ after the first two rounds. Thus, $\mathcal{S}$ needs to generate these tokens and sends them to $\mathcal{F}_{\mathrm{gWRAP}}$. Towards this, we recall that the simulator $\widetilde{\mathcal{S}}$ for $\widetilde{\Pi}_{\mathrm{MPC}}$, generates the messages of the honest parties in rounds $2+r$ from $r = 1, \ldots, 2T+2$ by using the semi-honest simulation of the MPC protocol Π and the simulation of the commit-and-prove protocol Π_{CP}. More precisely, given the input and randomness (extracted out) of all corrupted parties P_j and the output of the computation, using the semi-honest simulation of Π, $\widetilde{\mathcal{S}}$ generates a transcript of messages that will be exchanged with every corrupted party P_j. Next, it forces the computation to proceed according to this transcript. This follows from the soundness of the prove phase of the protocol Π_{CP}. More formally, the simulation forces a corrupted party P_j's random tape to be a certain value, and then the protocol Π_{CP} enforces semi-honest behavior from all corrupted parties.

The simulator $\widehat{\mathcal{S}}$ for $\widehat{\Pi}_{\mathrm{MPC}}$ however does not have to generate messages on behalf of the honest parties, rather it simply needs to create the code that, on behalf of an honest party, will perform its action in a particular round of the MPC protocol. Since the transcript of the communication can be fixed before the adversary queries the tokens $\mathsf{TK}^r_{i,j}$, the simulator simply hardwires the message that P_i needs to send in the r^{th} round to P_j in the token and sends it to P_j. The code of the token would be to verify the partial transcript and to reveal P_i's message in the r^{th} round if the verification follows correctly.

Indistinguishability follows from the indistinguishability of the simulation by $\widetilde{\mathcal{S}}$ and the fact that the soundness of protocol Π_{CP} ensures that there is a unique valid transcript given the random tapes and inputs of the corrupted parties, and the fact that the adversary cannot deviate from the honest behavior. For any adversary that deviates, we can construct another adversary $\mathcal{B}$ and environment $\mathcal{Z}$ where $\mathcal{B}$ can prove a false statement using Π_{CP} and this will contradict the fact that Π_{CP} realizes the $\mathcal{F}^{1:\mathrm{M}}_{\mathrm{CP}}$-functionality. ■

7 Acknowledgements

We thank Vipul Goyal, Yuval Ishai, Jonathan Katz and Amit Sahai for several discussions regarding tamper-proof hardware tokens. The first author acknowledges support from the Israel Ministry of Science and Technology (grant No. 3-10883) and support by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office. The second author was supported by the Danish National Research Foundation; the National Science Foundation of China (grant no. 61061130540) for the Sino-Danish CTIC; the 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. The third author was supported by Google Faculty Research Grant and NSF Award CNS-1526377.

References

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *FOCS*, pages 166–175, 2004.

[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.

[Bea91] Donald Beaver. Foundations of secure interactive computing. In *CRYPTO*, pages 377–391, 1991.

[BOV15] Ioana Boureanu, Miyako Ohkubo, and Serge Vaudenay. The limits of composable crypto with transferable setup devices. In *CCS*, pages 381–392, 2015.

[BS05] Boaz Barak and Amit Sahai. How to play almost any mental game over the net - concurrent composition via super-polynomial simulation. In *FOCS*, pages 543–552, 2005.

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

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *TCC*, pages 61–85, 2007.

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

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

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In *CCS*, pages 597–608, 2014.

[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$^+$13] 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. *IACR Cryptology ePrint Archive*, 2013:840, 2013.

[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.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, 2002.

[CLP10] Ran Canetti, Huijia Lin, and Rafael Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In *FOCS*, pages 541–550, 2010.

[COSV16a] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Concurrent non-malleable commitments (and more) in 3 rounds. *To appear at CRYPTO*, 2016.

[COSV16b] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. On round-efficient non-malleable protocols. *IACR Cryptology ePrint Archive*, 2016:621, 2016.

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

[CPS$^+$16a] Michele Ciampi, Giuseppe Persiano, Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. Improved or-composition of sigma-protocols. In *TCC*, pages 112–141, 2016.

[CPS$^+$16b] Michele Ciampi, Giuseppe Persiano, Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. Online/offline OR composition of sigma protocols. In *EUROCRYPT*, pages 63–92, 2016.

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In *CRYPTO*, pages 265–281, 2003.

[DKM11] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. Unconditional and composable security using a single stateful tamper-proof hardware token. In *TCC*, pages 164–181, 2011.

[DKMN15a] Nico Döttling, Daniel Kraschewski, Jörn Müller-Quade, and Tobias Nilges. From stateful hardware to resettable hardware using symmetric assumptions. In *ProvSec*, pages 23–42, 2015.

[DKMN15b] Nico Döttling, Daniel Kraschewski, Jörn Müller-Quade, and Tobias Nilges. General statistically secure computation with bounded-resettable hardware tokens. In *TCC*, pages 319–344, 2015.

[DMMN13] Nico Döttling, Thilo Mie, Jörn Müller-Quade, and Tobias Nilges. Implementing resettable uc-functionalities with untrusted tamper-proof hardware-tokens. In *TCC*, pages 642–661, 2013.

[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.

[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.

[GLOV12] Vipul Goyal, Chen-Kuei Lee, Rafail Ostrovsky, and Ivan Visconti. Constructing non-malleable commitments: A black-box approach. In *FOCS*, pages 51–60, 2012.

[GMPP16] Sanjam Garg, Pratyay Mukherjee, Omkant Pandey, and Antigoni Polychroniadou. The exact round complexity of secure computation. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 448–476, 2016.

[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.

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

[GRRV14] Vipul Goyal, Silas Richelson, Alon Rosen, and Margarita Vald. An algebraic approach to non-malleability. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 41–50, 2014.

[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.

[HJO$^+$15] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. *IACR Cryptology ePrint Archive*, 2015:1250, 2015.

[HV15] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. On black-box complexity of universally composable security in the CRS model. In *ASIACRYPT*, pages 183–209, 2015.

[HV16] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. On the power of secure two-party computation. *To appear at CRYPTO*, 2016.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, pages 294–304, 2000.

[IKO$^+$11] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In *EUROCRYPT*, pages 406–425, 2011.

[IKOS09] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.*, 39(3):1121–1152, 2009.

[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.

[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.

[LPV12] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. A unified framework for UC from only OT. In *ASIACRYPT*, pages 699–717, 2012.

[LS90] Dror Lapidot and Adi Shamir. Publicly verifiable non-interactive zero-knowledge proofs. In *CRYPTO*, pages 353–365, 1990.

[MMN16] Jeremias Mechler, Jörn Müller-Quade, and Tobias Nilges. Universally composable (non-interactive) two-party computation from untrusted reusable hardware tokens. *IACR Cryptology ePrint Archive*, 2016:615, 2016.

[MR91] Silvio Micali and Phillip Rogaway. Secure computation (abstract). In *CRYPTO*, pages 392–404, 1991.

[MS08] Tal Moran and Gil Segev. David and goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. In *EUROCRYPT*, pages 527–544, 2008.

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

[Nil15] Tobias Nilges. *The Cryptographic Strength of Tamper-Proof Hardware*. PhD thesis, Karlsruhe Institute of Technology, 2015.

[ORS15] Rafail Ostrovsky, Silas Richelson, and Alessandra Scafuro. Round-optimal black-box two-party computation. In *CRYPTO*, pages 339–358, 2015.

[Pas03] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In *EUROCRYPT*, pages 160–176, 2003.

[PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: achieving universal composability without trusted setup. In *STOC*, pages 242–251, 2004.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, pages 554–571, 2008.

[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 *FOCS*, pages 162–167, 1986.

A Issue with *Over Extraction* in Oblivious Transfer Combiners [GIS$^+$10]

In the following we identify an issue that affects one of the feasibility results in [GIS$^+$10, Section 5]. More precisely, this result establishes that UC security for general functionalities is feasible in the tamper-proof hardware model in $O(\kappa)$-round assuming only OWFs (or $O(1)$-round based on CRHFs) based on stateless tokens. The issue arises as a result of *over extraction* where a fully-secure OT protocol is constructed from a weaker variant and the simulation extracts values for sender's inputs even on certain executions where the receiver aborts. The term over extraction has been studied before in the context of commitment schemes where a scheme with over extraction is constructed as an intermediate step towards achieving full security [PW09, GLOV12].

On a high-level, in the work of [GIS$^+$10], they first construct an OT protocol with milder security guarantees. More precisely, a QuasiOT protocol achieves UC-security against a malicious receiver and straight-line extraction against malicious sender. However, the scheme is not fully secure as a malicious sender can cause an input-dependent abort for an honest receiver. Towards amplifying the security, [GIS$^+$10] consider the following protocol:

1. The sender with input (s_0, s_1) and receiver with input b interact in n executions of QuasiOTs. The sender picks $z_1, \ldots, z_n$ and Δ at random and sets the inputs to the i^{th} QuasiOT instance as $(z_i, z_i+\Delta)$. The receiver on the other hand chooses bits $b_1, \ldots, b_n$ at random subject to the sum being its input b.
2. If the first step completes, the sender sends $(s'_0 = s_0+\sum_i z_i, s'_1 = s_1+\sum_i z_i+\Delta)$ to the receiver. The receiver computes its output as $s'_b + \sum_i w_i$ where w_i is the output of the receiver in the i^{th} QuasiOT.

This protocol remains secure against a malicious receiver. However, an issue arises with a malicious sender. To simulate a malicious sender in this protocol, [GIS$^+$10] rely on the straight-line extractor of the n QuasiOTs by sampling two sets of random $(b_1, \ldots, b_n)$, one set summing up to 0 and another set summing up to 1 and computing what the receiver outputs in the two cases. As we demonstrate below such a strategy leads to failure in the simulation. More precisely, consider the following malicious sender strategy.

- Pick $z_1, z_2, ..., z_{n-1}$ and Δ at random.
- The inputs of the first $n-1$ tokens are set to $z_1, z_1+\Delta, \ldots, z_{n-1}, z_{n-1}+\Delta$.
- Let $z_1 + \ldots + z_{n-1} = a$ and $z_1 + \ldots + z_{n-1} + \Delta = b$.
- The inputs to the n-th token are some fixed values c (when $b_n = 0$) and d (when $b_n = 1$), where $c + d \neq \Delta$.

Next, the sender modifies the code of the tokens used in the QuasiOT protocol so that the first $n-1$ QuasiOTs never abort. The n-th instantiations however is made to abort whenever the input b_n, the receiver's input is 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 what 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 n-th 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 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 in the the first $n-1$ QuasiOTs and c from the n-th QuasiOT. On the other hand, if $b_n = 1$ then the receiver aborts in the n-th QuasiOT and therefore aborts.

Case 2: The receiver's input is $b = 1$. 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 its inputs in the QuasiOTs by monitoring the queries to the corresponding PRF tokens. 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.

On a more general note, our attack presents the subtleties that need to be addressed with the "selective" abort strategy. Recent works by Ciampi et al. [COSV16a, COSV16b] have identified subtleties in recent construction of non-malleable commitments [GRRV14] where selective aborts were not completely addressed.