Practical UC security with a Global Random Oracle

Ran Canetti* Abhishek Jain† Alessandra Scafuro‡

Abstract

We show that there exist commitment, zero-knowledge and general function evaluation protocols with universally composable security, in a model where all parties and all protocols have access to a single, global, random oracle and no other trusted setup. This model provides significantly stronger composable security guarantees than the traditional random oracle model of Bellare and Rogaway [CCS'93] or even the common reference string model. Indeed, these latter models provide no security guarantees in the presence of arbitrary protocols that use the *same* random oracle (or reference string or hash function).

Furthermore, our protocols are highly efficient. Specifically, in the interactive setting, our commitment and general computation protocols are much more efficient than the best known ones due to Lindell [Crypto'11,'13] which are secure in the common reference string model. In the non-interactive setting, our protocols are slightly less efficient than the best known ones presented by Afshar et al. [Eurocrypt '14] but do away with the need to rely on a non-global (programmable) reference string.

1 Introduction

The random oracle model (ROM) [2] has been extremely successful as a tool for justifying the design of highly efficient cryptographic schemes that lack more direct proofs of security. Indeed, while security in the ROM does not in general imply security when the random oracle is replaced by a concrete, publically computable hash function [6, 11, 22], it is generally accepted that security analysis in the ROM does provide strong corroboration to the resilience of the protocol in question to practical attacks.

However, when attempting to use the ROM for analyzing security of general protocols, and in particular when attempting to assert simulation-based security definitions, the following question comes up: Does security analysis in the ROM provide any *composable* security guarantees? In particular, what does security analysis in the stand-alone ROM say about the security of the protocol within a larger system that involves also other protocols, where these protocols may have themselves been analyzed in the ROM?

To provide a positive answer to this question, we would like to come up with a model for analyzing security of protocols in a stand-alone fashion, while taking advantage of the ROM, and still be able to provide security guarantees in a composite system where multiple such protocols co-exist and interact.

*Boston University and Tel-Aviv University, USA and Israel

†Johns Hopkins University, USA. Part of this work was done while the author was a postdoc at MIT and Boston University.

‡UCLA, USA

A natural approach to devising such a model is to start from an existing framework with composability guarantees, and try to add the ROM to that framework. Specifically, start from the universally composable (UC) security framework [3] that provides strong composability, and formulate the ROM as a "trusted functionality" that is available to the parties (i.e., this "random oracle functionality" simply returns an independent random value $RO(x)$ to each query x, while maintaining consistency among different queries with the same x).

However, it turns out that if one wants to use the full power of the ROM, and in particular to allow the simulator, in the security analysis, to have free access to the adversary's RO queries and furthermore to set the responses of the random oracle to values of its choice, the resulting modeling loses all composability guarantees. More precisely, secure composition holds only if each instance of each protocol uses a completely different and independent random oracle than all other instances. This of course does not correspond to the practice of replacing the random oracle in *all* executions with a *single* hash function.

Furthermore, this is not just a modeling issue: Using the same instance of the RO across multiple protocols inevitably gives rise to some unavoidable attacks. For instance, consider a UC non-interactive zero knowledge (NIZK) protocol in the RO model. If RO is available outside the individual instance of the protocol then the transcript of the protocol (i.e., the proof) becomes transferable - it is verifiable not only by the intended verifier, but rather by anyone who has access to RO. This stands in contrast to the ideal zero knowledge functionality, which allows only the intended verifier to verify the verity of the statement. (It should be remarked that the same issue happens even in interactive Zero Knowledge protocols in the ROM, but is perhaps most evident for non-interactive ones.)

Indeed, this discrepancy between the abstract model and its intended use was already noticed in the context of the common reference string setup [4]. To handle this discrepancy, [4] suggests to explicitly consider only trusted setup constructs that are *global,* namely only a single instance of this setup exists in the system. In particular, this construct exists even in the "ideal model", where the protocol is replaced by a "trusted party", or an ideal functionality representing the task at hand. They then proceed to propose such a global trusted setup construct. However their construct is not just a public hash function (or a random oracle). Rather, it consists of a global public key for which each party has its own identity-based secret key. Furthermore, they argue that no "public setup construct", namely no construct that provides only public information that is available to all, can suffice for realizing tasks such as commitment or zero knowledge in the UC framework. Given that the random oracle does provide only public information, the avenue of coming up with a useful, global ROM that provides composable security guarantees seemed to have reached a dead end.

1.1 Our Contributions

1.1.1 The global random oracle model

We formulate a natural, global variant of the "random oracle functionality". As per the formalism of [4], this functionality, which we denote by gRO – standing for *global* Random Oracle – is accessible to all parties (both honest and corrupted), both in the ideal model and in the model for protocol execution. This functionality answers consistently to all queries made by all parties. Furthermore, only a single instance of this functionality exists. As shown in [4], the universal composition theorem holds in this model - even though multiple protocols and instances thereof use the same instance of gRO.

In addition, we incorporate in gRO a mechanism that captures the fact, sketched above, that the global random oracle, being a single global construct that provides the same information to all, allows third parties to double up as adversarial protocol participants and mount a transferability attack.

In a nutshell, this analytical mechanism provide each protocol session with a unique domain of queries: queries that pertain to a session that the querying party does not belong to are considered "illegitimate" and are disclosed to the adversary. This mechanism allows capturing security properties such as "transferable non-interactive zero knowledge", namely protocols that are zero-knowledge except for the fact that proofs may be transferable. As we explain in more details later, transferability attacks are the only ones allowed in this model. Intuitively the reason is that, whatever a malicious third party can do by accessing the RO in some concurrent adversarial protocol execution, can be done by the adversary itself in her protocol execution.

We then observe that simple variants of known protocols, such as the two-message zero-knowledge protocol of Pass [24], is in fact UC zero knowledge in our global random oracle model (gRO model).

The gRO model, the CRS/RO model and the JUC model. The advantage of the gRO model is that it guarantees secure composition even with arbitrary protocols that use the *same instance* of the RO (or the same hash function). In practice it means that the RO can be *reused* by any protocol in the system, without jeopardizing security. Neither the standard (programmable) RO model nor the standard (programmable) CRS model give any security guarantees when the same setup is reused. One might object that there exist protocols that are UC-secure in the Joint State model (JUC) [8] where the same CRS is reused. However, the form of reusability guaranteed by the JUC model is very limited as that protocols must be *pre-designed* to work well together with same CRS instance. Instead, in the gRO model protocols do not need to synchronize their access to the RO.

Discussion on the random oracle model. One might wonder about the utility in rigorously arguing secure composition of protocols in the gRO model, given that this model anyway does not provide rigorous security guarantees once the random oracle is replaced by a publicly computable hash function. We provide several answers to this valid question. First, we note that attacks that take advantage of insecure composition might come up even when no other attacks are found against a protocol instantiated with some hash function. (In fact, the transferability attack is a quintessential example for such a situation.)

Second, we observe that protocols in the gRO model give us a level of security that was not known to exist in any other general computation (or even zero knowledge or commitment) protocol that was proven secure in the UC framework: Indeed, protocols in that framework cannot exist without some "trusted setup" construct. All known such constructs require trust in some third party or an external entity that is outside the control of the players. Furthermore, these protocols invariably provide the trusted external entity with the ability, if played maliciously, to completely compromise the security of the players.

The gRO model is different, in that it "only" reduces the security guarantees to the level of the stand-alone security guarantees provided by the random oracle methodology to begin with. That is, the level of reassurance provided by analysis in the gRO model with respect to universally composable security is no lower than the level of assurance provided by analysis in the ROM for traditional, stand-alone security.

Certainly, in some respects, a protocol that was analyzed in the gRO model and instantiated with a concrete hash function may well provide better security than a protocol analyzed in the

(non-global) CRS model and instantiated with a globally available reference string.

Still, it should be stressed that (similar to standard ROM) as soon as the gRO is replaced by a concrete hash function, the security guarantees provided by this model are inevitably only heuristic.

1.1.2 Highly-Efficient Protocols

We design efficient protocols for a variety of tasks in the gRO model.

Starting from the work of Lindell and Pinkas [18], who constructed efficient two-party computation protocols based on Yao's garbled circuit [30] via a novel cut-and-choose technique, a prolific sequence of works [20, 19, 14, 28, 21, 17, 13, 29, 1] have shown increasingly more efficient protocols for secure computation, which are either only stand-alone secure, or UC-secure in the CRS model [19, 16, 17, 1]. We show how to construct very efficient protocols in the gRO model, which in most cases improve on the highly efficient known existing solutions in the CRS model. In particular, we first show a highly efficient UC-commitment scheme, which compares very favorably to the most efficient known UC-commitment scheme of Lindell in [16] (in the CRS model). When plugged in [17], this construction directly yields very efficient UC-secure protocol for two-party computation. Finally we show that non-interactive secure two-party computation (NISC) is also achievable in the gRO model, building on the work of Afshar et al. [1]. More specifically we provide the following protocols.

UC COMMITMENTS IN THE gRO MODEL. We show a general construction that combines the use of any straight-line extractable commitment in the gRO model and any trapdoor commitment, to construct a UC-secure commitment scheme in the gRO model. By instantiating the extractable commitment with the protocol provided by Pass in [24] and the trapdoor commitment with Pedersen's scheme [25], we obtain an extremely efficient UC-secure commitment scheme that is significantly more efficient than the best known UC-secure commitment scheme in the CRS model of [16]. Concretely our protocol requires only 5 exponentiations per party and 5 rounds of communication in total (including the commitment and decommitment phases).

UC TWO-PARTY COMPUTATION IN THE gRO MODEL. We observe that the highly efficient UC-secure two-party computation protocol (2PC for short) of Lindell [17], that works in the CRS model, requires the use of UC-secure zero knowledge proofs, which in turns is based on UC-Commitments only. By instantiating Lindell's construction with our UC-commitments, we obtain a more efficient UC-2PC protocol in the gRO model.

UC NISC IN THE gRO MODEL. NISC – non-interactive secure computation– is a two-message protocol run between parties P_1 and P_2, where P_1 speaks first. Very recently, Afshar et al. in [1] presented the most efficient NISC protocol, which is UC-secure in the CRS model. We show how to instantiate this construction *without using the CRS*, in the gRO model, while preserving the non-interactive nature of the protocol. Our construction is slightly less efficient than the protocol of [1] but do away with the need to rely on a non-global (programmable) reference string.

1.2 Our Techniques

Here we provide an overview of the main technical ideas underlying our constructions.

1.2.1 Efficient UC Commitment in gRO Model

Recall that a UC secure commitment scheme requires two main properties: (a) *Equivocation:* When the receiver is corrupted, the simulator should be able to commit (on behalf of the honest sender) in such a way that it is able to decommit to any desired value. (b) *Extractability:* When the sender is corrupted, the simulator should be able to extract the committed value during the commitment phase.

Note that the observability property of the gRO naturally yields the desired extraction property discussed above. Indeed, this was already used to build extractable commitments in prior works (see e.g., [24]). How to achieve the equivocation property, however, is not immediately clear. Indeed, as discussed earlier, in the gRO model the simulator *cannot* program the outputs of the random oracle. Further, since we do not allow for trusted setups (such as a CRS), the simulator does not have immediate access to a "trapdoor" that allows for equivocation.

Towards that end, our starting point is the observation (already implicit in prior works) that the task of equivocation can, in fact, be reduced to the task of *trapdoor extraction*. More concretely, consider a trapdoor commitment scheme in the CRS model where the knowledge of the CRS trapdoor allows for equivocation (but does not compromise the hiding property of the scheme). For example, Pedersen's commitment scheme [25] satisfies these properties. Then, consider the following protocol template: first, the receiver chooses the CRS of the trapdoor commitment scheme on its own and sends it to the sender along with an extractable commitment to the associated trapdoor. For concreteness, let us think of the extractable commitment as simply the answer of the gRO when queried with the trapdoor string. Next, the committer commits to its input string by simply using the trapdoor commitment scheme. Since we want to preserve the extractability property from the committer side, we further require the committer to query the gRO on the opening of the above commitment and then commit to the answer of the gRO via another instance of trapdoor commitment. (Similar ideas were used in [23].)

Now, consider the following simple strategy for equivocation: the simulator first extracts the value committed by the receiver in the first message (by simply observing its query to the gRO) and then uses it as a trapdoor to later equivocate in both of the trapdoor commitments. While such an approach would indeed work against a semi-honest receiver, unfortunately, it does not work against a *malicious* receiver. The problem is that the above protocol does not preclude a cheating receiver from committing to some bogus value (instead of the correct trapdoor). Note that here we cannot simply require the receiver to provide a proof of consistency since proving that a given string is the output of the random oracle is not an NP statement.

Going further, one can observe that an extractable commitment in the gRO model is, in fact, only effective if it is later decommitted. This is because otherwise the adversary can choose to simply not query the gRO at all! Thus, in order to verify that the receiver actually commits to the valid trapdoor, we ask it to open its commitment in the decommitment phase. Now, the simulator can indeed extract the trapdoor from the first message of the receiver and be convinced of its validity since otherwise the receiver would fail to decommit properly later on.

While the above modification yields us the desired equivocation property, unfortunately, the resultant protocol is no longer sound against adversarial committers. This is because after viewing the trapdoor revealed by the receiver, a cheating committer can now also equivocate in the same manner as the simulator. Indeed, it may seem that now the simulator and an adversarial committer have the exact same power (i.e., both have access to the trapdoor). In order to solve this problem, we leverage the asymmetry between the simulator and the cheating committer. In particular,

note that the simulator knows how to equivocate even at the start of the decommitment phase (conditioned on the event that it previously extracted a valid trapdoor from the receiver), while the cheating sender can only equivocate after the receiver reveals the trapdoor. Thus, we now require the committer to commit to its openings of the commitments (from the commitment phase) *before* the receiver reveals the trapdoor. This immediately prevents the committer from being able to equivocate, but still preserves the equivocation property of the simulator. Due to technical reasons, we require the above commitment to also be extractable. Very briefly, this is necessary to formally reduce the binding property of the UC commitment scheme to the binding property of the trapdoor commitment scheme.

Discussion on Efficiency. We compare the efficiency of our scheme with Lindell's commitment scheme [16] (which, to the best of our knowledge, is the most efficient UC secure commitment scheme in the CRS model, in the literature). *Round Complexity.* The commitment phase in our scheme requires 2 rounds while the commitment phase in [16] is non-interactive. On the positive side, our decommitment phase requires 3 rounds, while [16] requires 5 rounds. *Computational Complexity.* Prior works have demonstrated that the main bottleneck in the computational efficiency is the number of exponentiations. When instantiated with Pedersen's commitment scheme, our protocol requires only 5 exponentiations per party: the commitment phase requires 1 exponentiation from the receiver to compute the parameters for Pedersen commitment, and 4 exponentiations from the committer to compute two Pedersen's commitments; in the decommitment phase the same exponentiations are required in the reverse order for the verification of the parameters and the decommitments. In contrast, in [16], requires 13 exponentiations per party. Our protocol additionally requires 6 random oracle evaluations.

1.2.2 Efficient NISC in gRO Model.

Our starting point is the NISC protocol of [1], which is UC-secure in the CRS model. Our goal is to emulate their approach in the gRO model. Towards that end, we observe that this task can be reduced to implementing a UC secure oblivious transfer (OT) protocol in the gRO model. In particular, since our focus is on efficiency, recall that the NISC protocol of [1] relies on the highly efficient UC OT protocol of Peikert et. al [26]. (For convenience, let us refer to this protocol as PVW OT.) Therefore, our goal then is to realize a version of PVW OT in the gRO model.

Realizing this simple idea, however, turns out to be highly problematic. Note that since a CRS is not available in our setting, the natural approach is to have the OT receiver choose the OT parameters (that comprise the CRS in [26]) and provide a zero-knowledge proof of knowledge (ZKPoK) of consistency. We stress that both the ZK and PoK properties of the proof are crucial here to ensure that the resulting OT protocol is fully simulatable. Specifically, the ZK property is necessary to allow the simulator to cheat in the computation of the parameters and extract both the inputs of a malicious OT sender. The PoK property, on the other hand, allows the simulator to extract the input of the receiver. Note, however, that a ZK proof in the gRO model requires at least two rounds [24]. As such, the resulting OT protocol in the gRO model becomes 3 rounds which violates the non-interactivity requirement for NISC.

Towards that end, upon closer inspection of the NISC protocol of [1], we make the following observation: Let P_1 and P_2 denote the two parties in the NISC protocol where P_1 is the evaluator of the garbled circuit, and therefore the receiver of the OT, and P_2 is the generator of the garbled circuit and therefore the sender of the OT. Then, [1] uses the simulatability property of OT against

malicious OT senders to extract the input of the sender P_2.

Our first idea is to extract the critical information from P_2 by exploiting the observability property of gRO. Specifically, we modify the NISC protocol of [1] by requiring that the randomness used to compute the commitments and the garbled circuits is generated by querying gRO. This enables the simulator –that observes the queries– to extract all the keys of the garbled circuits in "straight-line" *without* simulating the OT protocol for adversarial OT sender. Therefore, the problem of implementing the NISC protocol of [1] in the gRO model now reduces to constructing a 2 round *one-sided simulatable* OT, namely, an OT which is UC simulatable against malicious receivers but only guarantees *indistinguishability security* against malicious senders.

Our next contribution is to provide such a construction. The high-level strategy is to replace the (2-round) ZKPoK in the above construction of PVW OT with a *non-interactive* witness hiding (or witness indistinguishable) PoK in the gRO model. Implementing this idea, however, turns out to be quite non-trivial. Recall that the security of PVW OT against a malicious sender relies on the hardness of the DDH problem: an adversary distinguishing the input bit of the honest receiver can be used to construct an adversary that distinguishes a DDH tuple from a non-DDH tuple. This reduction goes smoothly when the receiver in PVW OT gives a ZKPoK proof of the correctness of the OT parameters since the DDH distinguisher can use its challenge tuple as the OT parameters and give a simulated ZK proof of correctness without knowledge of the corresponding witness.

However, when we replace the ZK proof with (say) a witness hiding (WH) proof, then the above reduction does not work because the DDH distinguisher does not know the witness for the proof. Towards that end, we pursue the idea of using a witness-indistinguishable (WI) proof instead of a WH proof. We have the following two-fold requirement: first, the statement for the WI proof should enable a secondary witness that can be used by the DDH distinguisher in the above reduction to construct a valid proof (without knowledge of the witness corresponding to the challenge tuple). Second, the PoK property of the proof should enable extraction of a cheating OT receiver's input *even if she uses the secondary witness*.

As we discuss later in Sec.5, realizing the above two properties *simultaneously* turns out to be a difficult task. Our final idea towards this end is to essentially run the OT protocol twice in parallel. Specifically, we require the OT receiver to choose two independent OT parameters and give a *single* WIPoK proof that proves the correctness of one of them. The sender then secret shares each of its OT input into two parts and then computes two different OT messages (using different OT parameters), one for each set of input shares.

Now, in order to argue security against cheating senders, we can construct a DDH distinguisher who uses the challenge tuple as one of the two OT parameters and generates the second one on its own. This allows the distinguisher to successfully give a WI proof of correctness. On the other hand, when the OT receiver is corrupted, the soundness of WIPoK ensures that at least one of the OT parameters is honestly generated. Therefore, we can ensure that a cheating receiver cannot learn both the inputs of the honest sender. In particular, the simulator uses the PoK property to extract the input of the receiver (which may be $\perp$ if the one set of OT parameters chosen by the receiver is malformed, since in this case, the receiver will not learn either of the inputs of the honest sender). We refer the reader to Sec. 5 for more details.

Discussion on Efficiency. Our one-sided simulatable OT in the gRO model is more expensive in terms of exponentiations compared with the PVW OT in the CRS model. This is due to the WIPoK that the OT receiver has to perform at the beginning of the protocol to prove the consistency of the OT parameters. The WIPoK protocol that we use requires t parallel repetitions of a Σ-protocol

(which are necessary to achieve straight-line extractability [10] in the gRO model), where t is the statistical security parameter. The underlying Σ-protocol is based on the Σ-protocol provided in [9] to prove OR-statements, and requires 8 exponentiations. Therefore, in total the proof requires $8t$ exponentiations. We stress that this proof is executed only once at the beginning, and the same parameters can be reused for all the subsequent transfers.

Furthermore, as our OT-protocol consists of a double repetition of PVW OT, each transfer is twice more expensive than a transfer with PVW OT. Additionally, our protocol requires $4.5t$ random oracle evaluations (the explanation of such values are deferred to Sec. 5). However, we observe that the NISC protocol of [1] requires $\mathcal{O}(tn + t^2)$ exponentiations, where t is the statistical parameter for the cut-and-choose protocol and n is the size of the input of one of the parties. Therefore, when plugged into NISC, our one-sided simulatable OT construction does not add a significant overhead.

2 The Global Random Oracle Model

In this section we describe our model and explain our definitional choices. Toward this end we briefly discuss the UC definition and its extensions JUC (Joint State UC) and GUC (Generalized UC).

2.1 Basic UC

Informal. The Universal Composability (UC) framework was introduced in [3] by Canetti to simplify the security analysis of protocols running in the complex network environment, like the Internet, where arbitrary protocols are potentially run concurrently. Namely, assume that one has to design a protocol and to prove that the protocol is secure even if run concurrently with many instances of arbitrary protocols. In the proof of security one should somehow take into account all such possible executions and prove that no matter what all these protocols are doing, the protocol remains secure. Let us call this protocol –the one for which we want to prove security– *challenge* protocol.

The UC framework allows one to analyze the security of the challenge protocol in isolation (without having to consider the concurrent executions) and then use the composition theorem to conclude that the protocol will be secure also when composed with arbitrary protocols.

The crucial aspect of the UC framework is indeed its *modularity*: when programs call subroutines, these subroutines are treated as separate entities that can be analyzed *separately* for their security properties by way of realizing a functionality $\mathcal{G}$. It is then argued, via the universal composition theorem, that any protocol that uses subroutine calls to $\mathcal{G}$ keeps all its security properties when $\mathcal{G}$ is replaced by a protocol that realizes it.

This powerful composition theorem holds only when subroutines *do not share* any part of their internal states with each other or with the calling protocol. In particular, a setup functionality that is modeled as a subroutine of the analyzed protocol cannot be invoked by more than one protocol session. In practice this means that even setup functionalities like PKI or CRS cannot be shared by more than one protocol.

Technical. The UC framework is based on the ideal/real world paradigm. In the ideal world, one specifies an ideal functionality $\mathcal{F}$ as an interactive Turing machine that privately communicates with the parties and the adversary S and computes a task in a trusted manner. The specification of the functionality also models the adversary's ability to influence the computation and the information that the protocol leaks.

In the real world, one specifies a protocol Π that is executed by the parties. Parties communicate over the channel in presence of an adversary $\mathcal{A}$ which controls the schedule of the communication over the channel, and can corrupt parties. Both in the ideal world and in the real world, parties are identified by a unique id, called PID together with a session id, SID. When a party is corrupted the adversary receives its secret input and its internal state. In this work, we consider only *static* adversaries, which means that $\mathcal{A}$ can corrupt a party only before the protocol execution starts.

The presence of arbitrary protocol running in the network is modeled via the concept of the environment $\mathcal{Z}$. The environment $\mathcal{Z}$ determines the inputs to the parties running the challenge protocol and see the outputs generated by these parties. The environment communicates with the adversary $\mathcal{A}/S$ and corrupts parties through the adversary. Typically, wlog one assume that the adversary $\mathcal{A}$ is dummy in the sense that he just acts as a proxy between the environment and the honest parties participating in the challenge protocol.

A protocol Π securely realizes a functionality $\mathcal{F}$ in the UC framework if for any real world adversary $\mathcal{A}$, there exists an ideal adversary S, such that for any PPT environment $\mathcal{Z}$ the view of an interaction with the protocol and $\mathcal{A}$ is indistinguishable from the view of an interaction with the ideal functionality $\mathcal{F}$ and S. One also considers a $\mathcal{G}$-*hybrid model*, where the real-world parties are additionally given access to an ideal setup functionality $\mathcal{G}$. During the execution of the protocol, the parties can send inputs to, and receive outputs from, the functionality $\mathcal{G}$.

In the basic UC framework the environment $\mathcal{Z}$ cannot directly access to the ideal setup functionality, but it can do so through the adversary. Namely, any query that $\mathcal{Z}$ wishes to make to the ideal functionality $\mathcal{G}$ is observed by the adversary/simulator, who queries the ideal functionality for $\mathcal{Z}$ and forwards the answer. This implicitly means that the setup $\mathcal{G}$ is treated as a "private subroutine" of the protocol, and is thus *local* to the challenge protocol instance. For example, the CRS functionality in the basic UC model is captured as a trusted setup that gives the reference string *only* to the adversary and the parties running the protocol. Technically, this assumption allows the simulator to *program* the CRS because the environment has no access to the "real" CRS.

2.2 JUC: UC with Joint State

The basic UC-framework demands that each execution of a protocol ρ is independent and uses its own local setup functionality. Therefore, if one wants to analyze a protocol π which executes, say t copies of ρ, one would need t independent instantiations of the setup functionality, e.g., t independent CRS. However, in practice one wants to use the same setup, e.g., the same CRS, for all executions of ρ. In order to reuse the same CRS in the UC-framework, one has to analyze the entire system of t executions of ρ as a single unit. Indeed, early works in UC implemented directly the multi-version of a functionality, e.g., functionality for multiple commitments $\mathcal{F}_{\mathsf{mcom}}$ instead of just functionality for a commitment $\mathcal{F}_{\mathsf{com}}$, and prove security directly of $\mathcal{F}_{\mathsf{mcom}}$ where all commitments share the same setup.

In [8] Canetti and Rabin introduce Universal Composition with Joint State (JUC), and a new composition theorem that allows to prove composition of protocols that share some state. Continuing the example of $\mathcal{F}_{\mathsf{mcom}}$ and the CRS setup, [8] introduces a mechanisms that allows to do the following. Instead of designing a protocol that directly implements the multi-instance version $\mathcal{F}_{\mathsf{mcom}}$, it is sufficient to provide a protocol that implements $\mathcal{F}_{\mathsf{com}}$ in the standard CRS model and then "compile" $\mathcal{F}_{\mathsf{com}}$ into $\mathcal{F}_{\mathsf{mcom}}$ using the same CRS.

A bit more precisely, functionality $\mathcal{F}_{\mathsf{mcom}}$ is a multi-session functionality that has a global sid and for each sub-session has a a sub-session id ssid. The idea of the compiler of [8] is to derive fresh

CRS_{ssid} for each ssid starting from a single CRS, and then run each execution of $\mathcal{F}_{\mathsf{com}}$ using setup CRS_{ssid}.

In the JUC model the CRS is locally available to only a specific set of protocols. It is not a public setup and cannot be used globally by any protocol. Thus, any protocol which is not pre-specified in this set must use a freshly sampled CRS. Technically this means that again the environment $\mathcal{Z}$ has not direct access to the CRS, but it needs to access it through the adversary/simulator, and therefore the CRS is programmable.

The JUC can be seen more as a proof technique and does not provide a stronger level of security than basic UC.

2.3 Generalized UC model

In both basic UC and JUC model, the environment is *constrained*: it does not have access to the setup functionality. The consequence in practice is that a protocol proved secure in these models is secure only if the setup is not public. In the CRS example, the CRS must be communicated privately to the parties participating the protocol.

However this assumption might be too strong or impractical in real life applications where it is instead more plausible that there is a single CRS published and used by many protocols, or, in the case of PKI (Public Key Infrastructure), one party uses the same public key in all protocol executions.

Ideally, one would like to have the nice modularity of the UC composition theorem that works also in presence of a *global setup* that is available to all parties/protocol executions. In [4], Canetti et al introduce the "Generalized UC model": they provide the formalism to describe global functionalities and show that the composition theorem still holds in this setting. The model of [4] is very general as it considers general shared functionalities (not necessarily setup functionalities). In the following we restrict our attention to setup functionalities only.

A setup functionality is *global* if it can be accessed by any protocol running in the system besides the challenge protocol. To model this, [4] first grants the environment the power of initiating several sessions besides the challenge session, and second, allows the environment to access to the setup functionality *directly*, without going through the simulator/adversary. This indeed captures the fact that any protocol in the network can use the same setup[1]. The first consequence of this modeling is that a global setup functionality must be *non-programmable*. Too see why, think of the CRS setup: If the CRS is publicly available then the simulator cannot program it as the environment $\mathcal{Z}$ can create new parties for arbitrary protocols – *of which the simulator is not aware of* – and check the CRS.

It is easy to see that GUC security is impossible to achieve in the CRS model as the simulator has no advantage over the adversary. In fact, as noticed in [4] it seems that any global setup that provides only public information is of little help for the simulator as it does not provide any trapdoor. Thus, achieving security in GUC model seems to require a special publicly available resource that also "hides" a trapdoor.

Motivated by this intuition [4] introduces the Augmented CRS (ACRS for short), where there is a public, global CRS consisting in signature verification keys, one for each party participating in

[1] [4] actually considers also a simplified definition called *externalized* UC model, where the environment has direct access to the setup functionality but does not initiate any new session except the challenge session. The access to the setup functionality allows $\mathcal{Z}$ to internally mimic the behavior of multiple sessions or more sessions of the challenge protocol.

the protocols. Additionally, associated to each verification key, there is a secret signing key that is not revealed to the parties. This is the trapdoor. An honest party never asks for the signing key. The catch is that a corrupted party can ask for the signing key and this is the trapdoor that allows the simulator to cheat in the simulation. At the same time, knowledge of her own signing key does not help the real adversary to break the privacy of other parties.

Note that security in the ACRS model is preserved only for "pid-wise" adversary: Namely it assumes that a party pid is corrupted in all the sessions that he plays. Also, note that even if the environment does not instruct the adversary to actually get the private signing key, still the simulator will ask the key to the ACRS functionality, i.e., a corrupt guy always gets the secret key.

2.4 Our Global Random Oracle Model

In this work we aim to use the random oracle as global setup functionality and achieve the stronger GUC security using this setup.

Let us first consider a simplistic candidate formulation for the global random oracle functionality: When queried by anyone for a value x, the random oracle functionality simply checks if x was queried before by anyone. If not, then it returns a freshly chosen random string of some pre-specified length. If yes then the previously chosen value is returned again — even if the earlier query was made by another party. No other information is disclosed to anyone. Let us call this functionality $\mathcal{G}_{\mathsf{sRO}}$ (where the s stands for "strict"). While $\mathcal{G}_{\mathsf{sRO}}$ is natural, it seems to be of little help for proving security of protocols. For one, $\mathcal{G}_{\mathsf{sRO}}$ does not allow the simulator to "emulate" the random oracle functionality to the environment, or in other words to "program" the answers of the random oracle. Indeed, recall that the environment can create additional parties that query $\mathcal{G}_{\mathsf{sRO}}$ and report the answer directly back. More importantly, $\mathcal{G}_{\mathsf{sRO}}$ is of little help to the simulator for another reason: The environment can obtain random-oracle values via the auxiliary parties, *without having the adversary/simulator be aware of the queried values or answers.* This means that $\mathcal{G}_{\mathsf{sRO}}$ is essentially useless to the simulator. Indeed, the impossibility results for UC computation in the plain model (e.g. [5, 7]) are easily extendible to the $\mathcal{G}_{\mathsf{sRO}}$ model.[2] And in fact, as mentioned earlier, [4] already observed any global setup functionality that provides only public information cannot be useful for achieving UC-security as the simulator has no advantage over the real adversary.

We can move forward using the observation that the Random Oracle does keep secret information: the queries made by the parties. Such queries are the *trapdoor* that can allow the simulator to extract crucial information from the adversary. We want to provide a reasonable formulation of the global random oracle functionality that allows extractability of the queries.

First note that if we attempt to modify the definition of $\mathcal{G}_{\mathsf{sRO}}$ so that it will disclose to the simulator *all* the queries made by other parties then we will again lose the usefulness of the functionality altogether, since the adversary too would be able to see the queries made by uncorrupted parties. Instead, we want to disclose only the queries made by the adversary/environment, while the queries made by any honest party should be invisible to external entity.

The queries made by the dummy adversary are directly observed by the simulator. The problem is: how to extract the queries made by the environment?

We propose the following mechanism. Queries are expected to have an explicit session identifier field, namely, a query x is parsed as the pair $x = (s, x')$ where s is the SID. The random oracle

[2] Still, it should be kept in mind that having access to a random oracle such as $\mathcal{G}_{\mathsf{sRO}}$ is not something that can be emulated in the standard model using an efficiently computable hash function family. In particular, the impossibility results of [6] still hold even with respect to this model.

Functionality $\mathcal{G}_{\mathsf{gRO}}$

Parameters: output length $\ell(n)$ and a list $\bar{\mathcal{F}}$ of ideal functionality programs.

1. Upon receiving a query x, from some party $P = (\mathsf{pid}, \mathsf{sid})$ or from the adversary S do:
   - If there is a pair (x, v) for some $v \in \{0,1\}^{\ell(n)}$ in the (initially empty) list $\mathcal{Q}$ of past queries, return v to P. Else, choose uniformly $v \in \{0,1\}^{\ell(n)}$ and store the pair (x, v) in $\mathcal{Q}$. Return v to P.
   - Parse x as (s, x'). If $\mathsf{sid} \neq s$ then add (s, x', v) to the (initially empty) list of `illegitimate` queries for SID s, that we denote by $\mathcal{Q}_{|s}$.
2. Upon receiving a request from an instance of an ideal functionality in the list $\bar{\mathcal{F}}$, with SID s, return to this instance the list $\mathcal{Q}_{|s}$ of illegitimate queries for SID s.

Figure 1: $\mathcal{G}_{\mathsf{gRO}}$

continues to answer the queries as before, namely it answers with a random value. The only difference is that some of the queries can be marked as "illegitimate" and potentially disclosed. Let us explain when a query is illegitimate. Recall that in the UC framework, a party P is identified by the unique pair (PID, SID) where PID is the program identifier and SID is the session identifier. Illegitimate queries are identified as follows. If the content of the SID field of the query differs from the content of the SID field of the querying party P, then the queries is considered "illegitimate". The functionality will record such queries and potentially disclosed to *to the instance of $\mathcal{F}$ whose SID is the one in the query.*

Therefore, our global random oracle functionality answers the queries just like the strict $\mathcal{G}_{\mathsf{sRO}}$, but additionally it agrees to disclose, to some pre-specified set of ideal functionalities, the illegitimate queries. We stress that illegitimate queries are answered as usual; they are just recorded separately and potentially disclosed. We stress that the random oracle in not programmable. The resulting random oracle functionality, denoted $\mathcal{G}_{\mathsf{gRO}}$, is described in Fig. 1.

The rationale behind this way of defining (il-)legitimate queries is the following. On the one hand, it allows designing protocols where the legitimate participants make RO queries that are never disclosed (simply prefix each query by the SID of the present session). Furthermore, it forces an ideal functionality to explicitly represent the information that is leaked by ideal functionalities regarding the oracle queries.

To further exemplify the properties of $\mathcal{G}_{\mathsf{gRO}}$, let us consider the case of zero-knowledge protocols in the presence of $\mathcal{G}_{\mathsf{gRO}}$. Recall that the traditional ideal Zero Knowledge functionality, $\mathcal{F}_{\mathsf{ZK}}$, allows a prover to convince a verifier (whose identity is determined by the prover) of the correctness of a statement without revealing any additional information, and without allowing the verifier to "transfer" the proof to another party. In contrast, as discussed in the Introduction, any proof in the global ROM is inherently transferable: To transfer a proof to party C, the verifier V simply lets C act as the verifier, and in particular have C make all the oracle queries herself. Consequently, any formal modeling of the global ROM should mirror this property of the global ROM. Indeed, it can be seen that $\mathcal{F}_{\mathsf{ZK}}$ is not realizable if the parties only have access to $\mathcal{G}_{\mathsf{sRO}}$. (Intuitively, disclosing the queries of third parties to the adversary/simulator has the effect that these third parties can no longer use $\mathcal{G}_{\mathsf{gRO}}$ to verify claims made by parties that participate in the session under consideration. This, for

instance, means that $\mathcal{G}_{\mathsf{gRO}}$ can no longer be used, in the model, to "transfer" proofs made within the session to third parties.) Further discussion on the definition of transferable Zero-Knowledge in the gRO model can be found in Appendix A.

We now provide the formal definition of UC-security in the Global Random oracle model.

Definition 1. [UC-SECURITY IN THE GLOBAL RANDOM ORACLE MODEL.] *Let $\mathcal{F}_t$ be an ideal m-party functionality and π be a protocol. We say that Π UC-**realizes $\mathcal{F}_t$ in the $\mathcal{G}_{\mathsf{gRO}}$-hybrid model** if for any hybrid-model PPT adversary $\mathcal{A}$, there exists an ideal process expected PPT adversary $\mathcal{S}$ such that for every PPT environment $\mathcal{Z}$, it holds that:*

$$\{\mathsf{IDEAL}^{\mathcal{G}_{\mathsf{gRO}}}_{\mathcal{F}_t,\mathcal{S},\mathcal{Z}}(\bar{x},n,z)\}_{\bar{x}\in\{0,1\}^{*m},n,z} \approx \{\mathsf{REAL}^{\mathcal{G}_{\mathsf{gRO}}}_{\pi,\mathcal{A},\mathcal{Z}}(\bar{x},n,z)\}_{\bar{x}\in\{0,1\}^{*m},n,z}$$

where REAL denotes the outputs of the honest parties and the adversary A after a real execution of protocol π, where $\bar{x}$ is the vector of inputs to the parties $P_1,\ldots,P_m$, $z\in\{0,1\}^*$ is the auxiliary input for $\mathcal{A}$ and n is the security parameter. IDEAL is the analogous distribution in an ideal execution with a trusted party that computes $\mathcal{F}_t$ for the parties and hands the output to the designated players. We provide the formal description of the ideal functionalities in the $\mathcal{G}_{\mathsf{gRO}}$ model that we implement in this paper: the commitment functionality Fig. 2, the OT functionality Fig. 3 (OT) and the NISC functionality Fig. 4.

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

$\mathcal{F}_{t\mathsf{com}}$ running in presence of an adversary S proceeds as follows:

- **Commitment Phase:** Upon receiving a message $(\mathsf{commit},\mathsf{sid},P_i,P_j,m)$ from P_i where $m\in\{0,1\}^n$: Record the tuple (sid,P_i,P_j,m) and send the message $(\mathsf{receipt},\mathsf{sid},P_i,P_j)$ to P_j and S. Ignore any subsequent commit messages.
- **Decommit Phase:** Upon receiving $(\mathsf{decommit},\mathsf{sid},P_i,P_j)$ from P_i, if the tuple (sid,P_i,P_j,m) is recorded then send $(\mathsf{decommit},\mathsf{sid},P_i,P_j,m)$ to P_j and to S and halt. Otherwise ignore the message.
- When asked by the adversary, obtain from $\mathcal{G}_{\mathsf{gRO}}$ the list $\mathcal{Q}_{\mathsf{sid}}$ of illegitimate queries that pertain to SID sid, and send $\mathcal{Q}_{\mathsf{sid}}$ to S.

Figure 2: The Commitment Functionality $\mathcal{F}_{t\mathsf{com}}$.

2.5 Discussion

Comparison with the JUC model. Our mechanism of illegitimate queries can be seen as a multiplexing mechanisms where from one global random oracle we derive many *independent* random oracles based on the sid. This technique is reminiscent of the multiplexing technique of [8] for the Joint State UC model, where they derive many independent CRS from a shared CRS.

We stress important differences between our model and the model of [8]. First, the shared CRS in [8] is still *local* to a pre-specified set of parties. This means that the environment does not have access to the CRS and that the simulator can program it. Thus, there is no global setup.

In our case the RO is global and can be used by any protocol in the world.

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

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

- Upon receiving a message $(\mathsf{send}, \mathsf{sid}, s_0, s_1, S, R)$ from S where each $s_0, s_1 \in \{0,1\}^n$, record the tuple (sid, s_0, s_1) and send $(\mathsf{send}, \mathsf{sid})$ to R and $\mathcal{S}$. Ignore any subsequent send messages.
- Upon receiving a message $(\mathsf{receive}, \mathsf{sid}, b)$ from R, where $b \in \{0,1\}$ send (sid, s_b) to R and $(\mathsf{received}, \mathsf{sid})$ to S and $\mathcal{S}$ and halt. (If no $(\mathsf{send}, \cdot)$ message was previously sent, do nothing).
- When asked by the adversary, obtain from $\mathcal{G}_{\mathsf{gRO}}$ the list $\mathcal{Q}_{\mathsf{sid}}$ of illegitimate queries that pertain to SID sid, and send $\mathcal{Q}_{\mathsf{sid}}$ to $\mathcal{S}$.

Figure 3: The Oblivious Transfer Functionality $\mathcal{F}_{\mathsf{tot}}$.

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

$\mathcal{F}_{\mathsf{nisc}}$ running in presence of an adversary S proceeds as follows:

- Initialize a list L of tuples.
- Upon receiving a message $\mathsf{input}(x)$ from P_1, store x.
  - Upon receiving a message $\mathsf{input}(\mathsf{sid}, y)$ from a player P_i, insert the tuple (P_i, sid, y) to L. If P_1 is corrupted send $(P_i, f(x,y))$ to P_1. Else send $\mathsf{message} - \mathsf{received}(P_i)$ to P_1.
  - Upon receiving a message $\mathsf{get\text{-}outputs}$ from P_1, send $\{(P_i, \mathsf{sid}, f(x,y)\}_{(P_i,\mathsf{sid},y)\in L}$.
  - When asked by the adversary, on input sid obtain from $\mathcal{G}_{\mathsf{gRO}}$ the list $\mathcal{Q}_{\mathsf{sid}}$ of illegitimate queries that pertain to SID sid and send $\mathcal{Q}_{\mathsf{sid}}$ to S.

Figure 4: The Multi-server NISC functionality $\mathcal{F}_{\mathsf{nisc}}$ (adapted from [1]).

Comparison with the GUC model. Our model is weaker than the GUC model of [4] as it does not achieve *on-line* deniability (we discuss more on this in the next paragraph).

However, our leaking of queries has the same flavor of ACRS. Let us explain. Consider the transferability attack described earlier where there are parties A,B,C and B is simply forwarding messaging between A and C. In this real attack the adversary B is not doing anything but relaying messaging. In the ACRS model of [4] this attack is simulatable for the following reason. B is a corrupted party and as such the simulator can ask for B's secret key, even if in real life B did not actually tried to obtain its secret key neither to use it. Their model somehow "leaks" the secret key of the corrupt party, but this leakage does not give any advantage to the adversary in breaking the security of honest parties.

Our mechanism with illegitimate queries is of the the same flavor. Our functionality leaks illegitimate queries, but knowledge of this queries does not give any advantage to the adversary in breaking the security of the honest parties. As in the ACRS model where an honest party would never ask for its secret key, in our model an honest party will not make illegal queries, and its security is therefore preserved.

About pid-wise corruption. In the constructions provided in [4] security holds only in presence of pid-wise corruption. Namely, a corrupted party is corrupt in all sessions (i.e., if a player is corrupt in one session, she cannot be honest in another one). This is due to the fact that, once a party is

corrupt, her secret key is leakead. Once the secret key is revealed, security for this party cannot be argued anymore. We note that in our model we do not need such restriction.

Off-line Deniability As pointed out in [4] it is hard to precisely define what deniability is as such definition might differ depending on the application. The definition that [4] gives is that a protocol is deniable if any party can simulate the protocol with just the knowledge of the output. Namely, any party can be a simulator.

As we discussed before, our model does not guarantee *on-line* deniability, as if the corrupted party is forwarding the messages to a third party, then the protocol is not deniable. However, we note that our protocols do provide off-line deniability, as any party, given the output of the function can simulate an accepting transcript.

Off-line deniability is instead impossible to achieve in the programmable CRS/RO model.

3 Building Blocks

In this section we provide the definitions of the building blocks that we use in our protocols.

3.1 Trapdoor Commitment

Definition 2 (Trapdoor Commitment Scheme)**.** *A tuple of PPT algorithms* $(\mathsf{TCGen}, \mathsf{TVer}, \mathsf{TCom}, \mathsf{TRec}, \mathsf{TEquiv})$ *is a* trapdoor *commitment scheme if,* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{TCGen}(1^n)$ *is the generation algorithm that outputs the* public key pk *and the* trapdoor key sk*,* $\mathsf{TVer}(\mathsf{pk}, \mathsf{sk})$ *outputs* 1 *iff* sk *is a valid trapdoor for* pk*,* TCom *is the algorithm that takes in input* pk *and a message* m *and outputs the commitment* c *and the decommitemt* d*;* TRec *is the verification algorithm that on input* (c, d, m) *outputs* 1 *iff the* d *is a valid decommitment of* c *for message* m *and* TEquiv *is a polynomial-time algorithm that satisfies the following:*

- trapdoor property: *for all PPT* $\mathcal{A}$*, for any* $m \in \{0,1\}^n$ *the following two probability distributions are computationally indistinguishable:*

- (pk, c, d, m) *where* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{A}(1^n)$ *s.t.* $\mathsf{TVer}(\mathsf{pk}, \mathsf{sk}) = 1$*, and* $(c, d) \leftarrow \mathsf{TCom}(\mathsf{pk}, m)$
- (pk, c', d', m) *where* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{A}(1^n)$ *s.t.* $\mathsf{TVer}(\mathsf{pk}, \mathsf{sk}) = 1$*,* $(c', z) \leftarrow \mathsf{TCom}(\mathsf{pk}, \cdot)$*, and* $d' \leftarrow \mathsf{TEquiv}(\mathsf{sk}, z, m)$

The above definition considers trapdoor commitments with the following properties: (a) the trapdoor is used only to compute the decommitment, (b) knowledge of the trapdoor allows to equivocate any previously computed commitment (as long as the state z is known). Such a commitment scheme can be based on Pedersen's perfectly hiding commitment scheme [25] that we describe in Fig. 5 for completeness.

3.2 Non-interactive WIPoK in the gRO model

Based on non-interactive straight-line commitment and Σ-protocols, Pass [24] shows how to construct an efficient straight-line non-interactive witness-indistinguishable proof of knowledge, for short $\mathsf{NIWIpok}$, in the non-programmable random oracle model. The idea behind the construction is the following. Let (α, β, γ) be the three messages of a Σ-protocol. Recall that the special soundness of Σ-protocols guarantees that if for the same first message α, one gets two accepting answers (β_0, γ_0) and (β_1, γ_1), then one can extract the witness. To achieve proof of knowledge, the idea is

- TCGen_P: The generation algorithm is a randomized algorithm that outputs $\mathsf{pk} = (\mathbb{G}, p, q, g, h)$ and $\mathsf{sk} = \mathsf{trap}$ where $p = 2q + 1$, p and q are primes, $\mathbb{G}$ is a subgroup of order q of Z_p^*, g and $h = g^{\mathsf{trap}}$ are generators of $\mathbb{G}$.
- TVer_P: The trapdoor verification algorithm outputs 1 if $h = g^{\mathsf{sk}}$; otherwise it outputs 0.
- TCom_P: The commitment algorithm is implemented as: $\mathsf{TCom}_P(\mathsf{pk}, m) = (g^m h^r \pmod p), d = (m, r)$ where $r \in \mathbb{Z}_p^\star$ is randomly chosen.
- TRec_P: The receiver algorithm takes as input (c, m, r) and outputs 1 iff $c = g^m h^r$.
- TEquiv_P: The equivocation algorithm takes as input $(c, m, r, m', \mathsf{sk})$, where c denotes a commitment, (m, r) denotes the decommitment for c, m' is the new message (for which equivocation is required) and sk is the trapdoor. The algorithm computes the new decommitment (for m') as $r' = m' - m + r$ and outputs it.

Figure 5: Pedersen's commitment scheme.

to force the prover to commits to both pairs (β_0, γ_0) and (β_1, γ_1) using the straight-line extractable commitment, but open to only one of the answers.

The answers to open is decided by querying the RO with the commitments so computed. The extractor obtains the witness by running the extractor associated to the extractable commitments. This protocol achieves only soundness 1/2. For soundness 2^L, one needs to repeat this protocol L times. The protocol is formally described in Protocol 1.In the protocol the RO is invoked with the prefix corresponding to the ID of the prover. Due to the unpredictability of the RO, adding this prefix ensures non-malleability.

PROTOCOL 2. *Non-interactive proof of knowledge* $\mathsf{NIWIpok}$ *[24].* Let (α, β, γ) denote the three messages of a Σ protocol for a language L. Let $\mathsf{gRO} : \{0,1\}^{2n} \leftarrow \{0,1\}^{\ell(n)}$ where $\ell(n) = \omega(\log n)$. Let ExtCom the following commitment algorithm. On input id, x, pick a random string r and outputs $\mathsf{gRO}(\mathsf{id}, x, r)$. The decommitment consists in the pair x, r.

Public input: x.
Private input to P: a witness w for theorem "$x \in L$".

- P computes α, the first message of the Σ-protocol. P picks two challenges β_0, β_1 (with $\beta_0 \neq \beta_1$), and computes the corresponding answers γ_0, γ_1 using knowledge of the witness w.
- P commits to β_0, β_1, γ_0, γ_1: it computes $c\beta_b = \mathsf{ExtCom}(\text{“}P0'' \circ \beta_b)$ and $c\gamma_b = \mathsf{ExtCom}(\text{“}P0'' \circ \gamma_b)$, for $b = 0, 1$. Let $\mathsf{D}\beta_b, \mathsf{D}\gamma_b$ the corresponding decommitments.
- P computes the challenge e as follows. Let $\bar{\mathsf{C}} = (\mathsf{C}\beta_0, \mathsf{C}\beta_1, \mathsf{C}\gamma_0, \mathsf{C}\gamma_1)$ the concatenations of the commitments so obtained. P compute $e = \mathsf{gRO}(\alpha, \bar{\mathsf{C}})$.
- The non-interactive proof consists of the message: $\alpha, \bar{\mathsf{C}}, \mathsf{D}\beta_e, \mathsf{D}\gamma_e$.
- V checks that: 1)$\mathsf{D}\beta_e, \mathsf{D}\gamma_e$ are valid decommitments to β_e, γ_e, 2) $\alpha, \beta_e, \gamma_e$ is an accepting transcript for the theorem $x \in L$.

3.3 Sigma Protocols

In this section we present the Σ-protocol that will be used in Protocol 2.

Σ-protocol to prove Knowledge of a Discrete Logarithm (DLog). Let G be a group of prime order q generated by g and let $w \in Z_q$. Let n be the security parameter, and $2^n < q$. Let $\mathcal{R}_{DL}$ be the relation for the Discrete Log (DLog) problem as follows: $\mathcal{R}_{DL} = \{((G, q, g, h), w) | h = g^w\}$. The Σ-protocol for $\mathcal{R}_{DL}$ is depicted in Figure 6.

Common input: $x = (G, q, g, h)$.
P's secret input: w s.t. $(x, w) \in \mathcal{R}_{DL}$.
Σ-protocol:

- P: choose $r \xleftarrow{\$} \mathbb{Z}_q$ and send $a = g^r$ to V.
- V: choose $e \xleftarrow{\$} \mathbb{Z}_q$ and send it to P.
- P: compute $z \leftarrow r + ew \mod q$ and send it to V.
- V: accept if and only if $g^z = ah^e$.

Figure 6: Σ-protocol for Relation $\mathcal{R}_{DL}$.

Proof of Knowledge of a Compound Statement. Let (x_0, x_1) be a pair of statements. Let P be a prover who wants to prove that he knows a witness w such that either $(x_0, w) \in \mathcal{R}_0$ or $(x_1, w) \in \mathcal{R}_1$ without revealing which is the case. Let π_0 be the Σ-protocol for the relation $\mathcal{R}_0$ and π_1 be the one for $\mathcal{R}_1$. Figure 7 shows a Σ-protocol for $\mathcal{R}_0 \vee \mathcal{R}_1$. This protocol exploits the honest verifier zero knowledge (HVZK) property of Σ protocol. This property allows to compute an accepting transcript (a, e, z) given in input the theorem x and the challenge e and without knowing the witness.

Common input: (x_0, x_1).
P's secret input: w s.t. $(x_b, w) \in \mathcal{R}_b$.
Σ-protocol:

- P: compute a_b according to π_0 using (x_b, w) as input: then choose e_{1-b} and run the HVZK simulator S for π_{1-b} on input (x_{1-b}, e_{1-b}) to obtain $(a_{1-b}, e_{1-b}, z_{1-b})$; send (a_0, a_1) to V.
- V: choose $s \in \{0, 1\}^t$ and send it to P.
- P : set $e_b \leftarrow s \oplus e_{1-b}$, compute z_b according to π_b and send (e_0, z_0, e_1, z_1) to V.
- V: check that $e_0 \oplus e_1 = s$ and that both transcript (x_0, a_0, e_0, z_0) and (x_1, a_1, e_1, z_1) are accepting according to π_0 and π_1 respectively.

Figure 7: Witness-Indistinguishable PoK of a Compound Statement.

4 UC Commitments in the gRO Model

In this section, we present a UC secure commitment scheme in the global RO model. Our commitment scheme can be based on any stand-alone secure trapdoor commitment scheme (see Def.2).

However, in order to obtain the concrete efficiency parameters as discussed earlier in Sec.1.2, we instantiate the trapdoor commitment scheme with Pedersen's perfectly hiding commitment scheme [25] described in Fig. 5. The intuition behind the security proof was given in Sec.1.2.

We now describe a UC secure commitment scheme, denoted $\langle C, R\rangle$, in the global random oracle model. Here, C denotes the algorithm of the committer and R denotes the algorithm of the receiver. See Fig. 2 for a formal description of the ideal commitment functionality. Let $(\mathsf{TCGen}, \mathsf{TVer}, \mathsf{TCom}, \mathsf{TRec}, \mathsf{TEquiv})$ be a trapdoor commitment scheme. Let n be the security parameter and $m \in \{0,1\}^{|n|}$ denote the input string of the committer. Let sid denote the session identifier. The commitment scheme $\langle C, R\rangle$ consists of two phases, namely, the commitment phase and the decommitment phase, described as follows:

PROTOCOL 1. UC COMMITMENT IN THE gRO MODEL.
Inputs. C has in input $m \in \{0,1\}^n$. R has no input. Let sid denote the session identifier.

Commitment Phase: This phase consists of two rounds.

- $R \to C$: R first computes $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{TCGen}(1^n)$. Next, it samples a random string r (of appropriate length) and queries the gRO on the string $(\mathsf{sid}, \text{'R'} \| \mathsf{sk} \| r)$. Let a_R be the resulting answer. R sends (pk, a_R) to C.

- $C \to R$: C first computes a trapdoor commitment to its input string m, namely, $(c_{\mathsf{msg}}, d_{\mathsf{msg}}) \leftarrow \mathsf{TCom}(\mathsf{pk}, m)$. Next, it samples a random string s and queries the gRO on the string $(\mathsf{sid}, \text{'C'} \| m \| d_{\mathsf{msg}} \| s)$. Let a_C be the resulting answer. Finally, it computes a trapdoor commitment to a_C as $(c_{\mathsf{ro}}, d_{\mathsf{ro}}) \leftarrow \mathsf{TCom}(\mathsf{pk}, a_C)$.

  C sends $(c_{\mathsf{msg}}, c_{\mathsf{ro}})$ to R.

Decommitment Phase: This phase consists of 3 rounds.

- $C \to R$: C commits to the decommitments $d_{\mathsf{msg}}, d_{\mathsf{ro}}$: it first samples a random string s' and queries the gRO on the string $(\mathsf{sid}, \text{'C'} \| d_{\mathsf{msg}} \| d_{\mathsf{ro}} \| s')$. It then sends the resulting answer a'_C to R.

- $R \to C$: R sends (sk, r) to C.

- $C \to R$: C aborts the protocol if either of the following verifications fails: (a) $a_R = \mathsf{gRO}(\mathsf{sid}, \text{'R'} \| \mathsf{sk} \| r)$, (b) $\mathsf{TVer}(\mathsf{pk}, \mathsf{sk}) = 1$. Otherwise, if both the checks succeed, then C reveals $(m, d_{\mathsf{msg}}, d_{\mathsf{ro}}, a_C, s, s')$ to R.

  The receiver R accepts m as the decommitted value iff *all* of the following verifications succeed: (a) $a'_C = \mathsf{gRO}(\mathsf{sid}, \text{'C'} \| d_{\mathsf{msg}} \| d_{\mathsf{ro}} \| s')$, (b) $\mathsf{TRec}(c_{\mathsf{ro}}, a_C, d_{\mathsf{ro}}) = 1$, (c) $a_C = \mathsf{gRO}(\mathsf{sid}, \text{'C'} \| m \| d_{\mathsf{msg}} \| s)$, (d) $\mathsf{TRec}(c_{\mathsf{msg}}, m, d_{\mathsf{msg}}) = 1$.

Efficiency The commitment protocol has the following complexity. *Round Complexity.* The protocol requires 2 rounds for the commitment phase and 3 rounds for the decommitment phase. *Exponentiations.* The total number of exponentiations is 10. *Hash evaluations.* The total number of hash evaluations is 6.

4.1 Proof of Security

We present the security proof for $\langle C, R\rangle$ in two parts. The first part concerns with the case where the environment corrupts the receiver, while the second part concerns with the corrupted committer. In each case, we construct a simulator $\mathcal{S}_A$ for the corrupted party A and argue indistinguishability of the real and ideal world experiments.

Security Against Malicious Receiver We describe the strategy of the simulator $\mathcal{S}_R$ for the commitment and decommitment phases:

Commitment phase: Upon receiving the message $(\mathsf{receipt}, \mathsf{sid}, \text{'C'}, \text{'R'})$ from the trusted party, $\mathcal{S}_R$ first obtains the list $\mathcal{Q}_{|\mathsf{sid}}$ from the ideal functionalities $\mathcal{F}_{t\mathsf{com}}$ and $\mathcal{G}_{\mathsf{gRO}}$, then computes the second round of the commitment phase as follows. It computes two trapdoor commitments to the all zeros string, namely, $(c_{\mathsf{msg}}, d_{\mathsf{msg}}) \leftarrow \mathsf{TCom}(\mathsf{pk}, 0^{|n|})$ and $(c_{\mathsf{ro}}, d_{\mathsf{ro}}) \leftarrow \mathsf{TCom}(\mathsf{pk}, 0^{|n|})$, and then sends $(c_{\mathsf{msg}}, c_{\mathsf{ro}})$ to R^*.

Decommitment phase: Upon receiving the decommitment message $(\mathsf{decommit}, \mathsf{sid}, \text{'C'}, \text{'R'}, m)$ from the trusted party, $\mathcal{S}_R$ proceeds as follows:

- Trapdoor extraction: If there exists a query in $\mathcal{Q}_{\mathsf{sid}}$ of the form $\mathsf{sid}\|\text{'R'}\|\mathsf{sk}'\|r'$ such that $\mathsf{TVer}(\mathsf{pk}, \mathsf{sk}') = 1$, then $\mathcal{S}_R$ records sk' as the trapdoor. If no such query exists, then it sets $\mathsf{sk}' = \perp$.
- Equivocation: If $\mathcal{S}_R$ successfully extracted a valid trapdoor in the previous step, then it proceeds to perform the following "equivocation" steps: (a) It starts by computing $\widetilde{d_{\mathsf{msg}}} \leftarrow \mathsf{TEquiv}(\mathsf{sk}, c_{\mathsf{msg}}, m)$ as a valid opening for c_{msg} to string m. (b) Next, it samples a random string s and queries the gRO on the string $\mathsf{sid}\|\text{'C'}\|m\|\widetilde{d_{\mathsf{msg}}}\|s$. Let a_C be the answer received from gRO. (c) It then computes $\widetilde{d_{\mathsf{ro}}} \leftarrow \mathsf{TEquiv}(\mathsf{sk}, c_{\mathsf{ro}}, a_C)$ as a valid opening for c_{ro} to string a_C.

  Now, $\mathcal{S}_R$ samples a random string s' and performs the following steps: If $\mathsf{sk}' = \perp$, then $\mathcal{S}_R$ simply computes $a'_C \leftarrow \mathsf{gRO}(\mathsf{sid}\|\text{'C'}\|0^{|n|}\|s')$; otherwise, it computes $a'_C \leftarrow \mathsf{gRO}(\mathsf{sid}\|\text{'C'}\|\widetilde{d_{\mathsf{msg}}}\|\widetilde{d_{\mathsf{ro}}}\|s')$. It then sends a'_C to R^* in the first round of this phase. Next, upon receiving the message (sk, r) from R^*, $\mathcal{S}_R$ verifies its correctness by following the honest committer strategy as described above. If the verification fails, then it aborts the protocol. Otherwise, it reveals $(m, \widetilde{d_{\mathsf{msg}}}, \widetilde{d_{\mathsf{ro}}}, s, s')$ to R.

This completes the description of $\mathcal{S}_R$. We now briefly argue the correctness of simulation.

First note that it follows from the hiding property of the trapdoor commitment scheme that the (trapdoor) commitments sent by $\mathcal{S}_R$ in the commitment phase are indistinguishable from those send by an honest committer in the real world. Further, since the outputs of the random oracle are random, the random oracle points sent by $\mathcal{S}_R$ in the commitment and decommitment phase are indistinguishable from those sent by the honest committer. Finally, we argue that conditioned on the event that the adversarial receiver reveals a valid trapdoor value in the decommitment phase, the decommitments sent by $\mathcal{S}_R$ are accepting. (This is sufficient since if the adversary fails to reveal a valid trapdoor, then both the honest committer and $\mathcal{S}_R$ abort the protocol.) This follows from the observability property of the gRO model. Specifically, since $\mathcal{S}_R$ extracts the trapdoor from the receiver during the commitment phase, it follows from the correctness of the equivocation algorithm (of the trapdoor commitment scheme) that $\mathcal{S}_R$ can compute accepting decommitments

for the output received from the trusted party, and then behave honestly in the decommitment phase.

Security Against Malicious Committer We describe the strategy of the simulator $\mathcal{S}_C$ for the commitment and decommitment phases:

Commitment phase: $\mathcal{S}_C$ follows the honest receiver strategy to generate the first receiver message.

Upon receiving the second message $(c_{\mathsf{msg}}, c_{\mathsf{ro}})$ of this phase from C^*, $\mathcal{S}_C$ performs the following steps:

- Extraction: Let $\mathcal{Q}_{\mathsf{sid}}$ be the list of queries made to gRO by any party. If there exists a query q of the form $q = \mathsf{sid}\|\text{`C'}\|m\|d_{\mathsf{msg}}\|s$ such that $\mathsf{TRec}(c_{\mathsf{msg}}, m, d_{\mathsf{msg}}) = 1$, then it sets $m' = m$; otherwise it sets m' to a dummy value (say) $0^{|n|}$.

$\mathcal{S}_C$ sends the message $(\mathsf{commit}, \mathsf{sid}, \text{`C'}, \text{`R'}, m')$ to the trusted party.

Decommitment phase: $\mathcal{S}_C$ follows the honest receiver strategy in this phase. If the receiver algorithm R accepts a value m^* from C^*, then $\mathcal{S}_C$ performs the following steps: If $m^* = m$, it sends the message $(\mathsf{decommit}, \mathsf{sid}, \text{`C'}, \text{`R'})$ to the trusted party. Otherwise, if $m^* \neq m$, then $\mathcal{S}_C$ outputs the special $\perp$ symbol and stops.

This completes the description of $\mathcal{S}_C$. We now briefly argue the correctness of simulation. Note that since $\mathcal{S}_C$ follows the honest receiver strategy in sending its messages, we only need to argue the following: Conditioned on the event that the adversarial sender successfully decommits to a string m in the decommitment phase, the simulator $\mathcal{S}_C$ successfully extracts the *same* string m during the commitment phase. This follows from the binding property of the trapdoor commitment scheme. Specifically, we build an adversary $\mathcal{A}$ for the trapdoor commitment scheme that obtains the public key pk from its challenger. $\mathcal{A}$ runs the adversarial committer C^* for the protocol $\langle C, R\rangle$ and plays the role of the honest receiver using the public key pk. We only consider a partial protocol execution that stops at the end of the first message in the decommitment phase. By the observability of the gRO model, it follows that $\mathcal{A}$ can extract the committed value from C^* if this partial transcript is a prefix of a full protocol transcript that is sampled from the space of accepting transcripts.

5 One-sided UC-Simulatable OT

The NISC protocol of [1] builds upon the efficient PVW OT protocol of [26], which is UC-secure in the CRS model (more specifically, they rely on a modification of it due to [28], which we will explain later). UC-realizing NISC in the gRO model amounts to provide a 2-round OT protocol which is UC secure in the gRO model.

As we discussed in Sec. 1.2, one promising approach to implement efficient UC-secure OT, is to take the PVW OT that works without the CRS and adapt it to the gRO model. This version of PVW OT was shown by Lindell and Pinkas in [19], and it requires the receiver to choose the parameters for the OT and to provide a zero-knowledge PoK of their correctness.

This naive approach is correct, but it yields a 3-round OT protocol. The reason is that any ZKPoK in the gRO model requires at least two rounds (as observed in [24]).

Our first observation is that the *zero knowledge* property is required only to extract from the sender, thus if we relax this requirement and demands only that a malicious sender cannot distinguish whether the receiver is playing with input 0 or input 1, then a witness indistinguishable proof

– which can be made non-interactive in the gRO model – suffices.

Before proceeding with our discussion, let us recall the PVW OT protocol in the plain model (without CRS). Let $g_0, q, \mathbb{G}$ be public parameters, where g_0 is the generator of the group $\mathbb{G}$ of prime order q. PVW OT consists of two steps. **Step 1.** The receiver R picks α_0, y at random in $\mathbb{Z}_q$ and set $\alpha_1 = \alpha_0 + 1$, and $g_1 = g_0^y$. It computes $h_0 = g_0^{\alpha_0}$ and $h_1 = g_1^{\alpha_1}$. It sends parameters (g_0, h_0, g_1, h_1) to the sender S and additionally proves that $(g_0, h_0, g_1, \frac{h_1}{g_1})$ is a DDH tuple using a ZK PoK. (Such parameters can be reused among several transfers with the same sender). Concretely, such ZKPoK is instantiated with the zero-knowledge version of a Σ-protocol for Discrete Log (due to [27] and shown in Figure 6), for the theorem $\left(h_0 = g_0^{\alpha_0} \wedge \frac{h_1}{g_1} = g_1^{\alpha_0}\right)$. For the actual transfer, R sends the temporary key $g = g_b^r, h = h_b^r$, where $r \leftarrow_R \mathbb{Z}_q$ and b is R's input. **Step 2.** If the ZK proof is accepting, S uses parameters (g_0, h_0, g_1, h_1) and h, g to encrypt its two strings s_0, s_1.

The security of the receiver relies on the DDH assumptions. To see why, note that if $b = 0$, then (g_0, h_0, g, h) is a DDH tuple, if $b = 1$, then (g_1, h_1, g, h) is a DDH tuple. Thus, a malicious sender distinguishing bit 0 from 1 can be transformed in DDH distinguisher.

Our first attempt is to replace the ZK protocol with a (non-interactive) witness-hiding PoK protocol (WHPoK). The crucial problem of this approach is that the reduction to the DDH problem does not go through. Indeed, in order to complete the OT protocol, and thus to be able to exploit the distinguishing power of a malicious sender S^*, the DDH distinguisher needs to provide a valid WHPoK, for which he does not know the witness. Although intuitively it seems that the witness hiding property should help, it is not clear how to exploit S^* to extract the witness. (Note that technically one can also use a WI proof for the following theorem: $(h_0 = g_0^{\alpha_0} \wedge \frac{h_1}{g_1} = g_1^{\alpha_0})$ OR $(g_1 = g_0^y \wedge \frac{h_1}{g_0} = h_0^y)$. The problem here is that extracting the witness y does not help the simulator in extracting the bit of R^*).

Our approach is to run two parallel executions of Step 1 (with independent parameters), and prove using a *single* WIPoK that one of the parameters is correctly generated. In Step 2, the sender will secret share its inputs in two shares, and compute two parallel executions of Step 2, one for each share. Due to the the soundness of the proof, in one of the executions the parameters are correctly computed, and thus a malicious receiver can never get two shares for both inputs.

This solution works against the malicious sender. Indeed, now we have the freedom to choose between two *independent* DDH tuples, and place the challenge of the DDH experiment in one of the two, while computing the WI proof with the witness of the other tuple. Against the malicious receiver, the simulator can extract the bit played in one of the sessions only. Nevertheless, this knowledge is sufficient, as the receiver will be able to get two shares only for the bit committed in the "correct" session.

We now present our protocol in details.

Notation. Our OT protocol essentially consists of two parallel sessions of the PVW OT, that we denote by session 0 and session 1 respectively. To identify the parameters used in each session we use two indexes in the subscript, where the second index identifies the session. For example, in our OT the receiver send two set of PVW OT's parameters (g_0, h_0, g_1, h_1), one for each session. Hence, we denote the parameters for session 0 by $(g_{0,0}, h_{0,0}, g_{1,0}, h_{1,0})$, and parameters for session 1 by $(g_{0,1}, h_{0,1}, g_{1,1}, h_{1,1})$. In general notation $h_{b,s}$ must be interpreted as follows: $h_{b,s}$ is the parameter h_b (as in PVW OT) played in session s.

Similarly, the sender will break up its input (s_0, s_1) in the shares $(s_{0,0}, s_{1,0})$ to play in session 0 and $(s_{0,1}, s_{1,1})$ to play in session 1, where notation $(s_{0,s}, s_{1,s})$ means, the share of s_0 in session s and the the share of s_1 in session s.

Finally, for easiness of explanation we omit some validity checks that S has to perform on the parameters sent by R.

PROTOCOL 2. *One-sided UC-simulatable OT.*

Common parameter. $(\mathbb{G}, q, g_{0,0}, g_{0,1})$.
Inputs. Inputs to S: a pair (s_0, s_1). Input to R : a bit b.

Round 1. R performs the following steps.

1. **Generate OT Parameters.** (Session 0) Pick $y_0, \alpha_{0,0}$. Set $\alpha_{1,0} = \alpha_{0,0} + 1$. (Session 1) Pick $y_1, \alpha_{0,1}$. Set $\alpha_{1,1} = \alpha_{0,1} + 1$. Compute $g_{1,e} = (g_{0,e})^{y_e}$, $h_{0,e} = (g_0)^{\alpha_{0,e}}$, $h_{1,e} = (g_1)^{\alpha_{1,e}}$, for $e = 0, 1$.

2. **Generate proof of consistency.** Run $\mathsf{NIWIpok}$ shown in Prot. 3.2 for the theorem $\{h_{0,0} = (g_{0,0})^{\alpha_{0,0}} \wedge \frac{h_{1,0}}{g_{1,0}} = (g_{1,0})^{\alpha_{0,0}}\}$ OR $\{h_{0,1} = (g_{0,1})^{\alpha_{0,1}} \wedge \frac{h_{1,1}}{g_{1,1}} = (g_{1,1})^{\alpha_{0,1}}\}$ using witness $\alpha_{0,e}$ for a randomly chosen bit e. We denote this proof by $\mathsf{proof}_{\mathsf{cons}}$. For this computation R needs to query the random oracle gRO. The details of $\mathsf{NIWIpok}$ are provided in Protocol 1.

3. **Generate temporary public keys.** Pick $r_0, r_1 \in \mathbb{Z}_q$ at random. Compute temporary key for session 0. Set $pk_0 = (g^0, h^0)$ where $g^0 = (g_{b,0})^{r_0}, h^0 = (h_{b,0})^{r_0}$. Set $pk_1 = (g^1, h^1)$ where $g^1 = (g_{b,1})^{r_1}, h^1 = (h_{b,1})^{r_1}$.

4. **Send parameters.** Send $\mathsf{par0} = (g_0, h_{0,0}, g_{1,0}, h_{1,0})$; $\mathsf{par1} = (g_0, h_{0,1}, g_{1,1}, h_{1,1})$; $\mathsf{proof}_{\mathsf{cons}}$, pk_0, pk_1 to S.

Round 2. If the proof is accepting, S performs the following steps.

1. **Compute shares.** Pick $s_{0,0}, s_{1,0}$ at random, and compute $s_{1,0} = s_{0,0} + s_0$; $s_{1,1} = s_{1,0} + s_1$.

2. **OT transfer.**

   - Session 0. Play as sender of PVW OT with input $(s_{0,0}, s_{1,0})$. Namely, compute $(u_{d,0}, v_{d,0})$ $= \mathsf{RAND}(g_{d,0}, h_{d,0}, pk_0)$ for $d = 0, 1$ and $w_{0,0} = v_{0,0} \cdot s_{0,0}$ $w_{1,0} = v_{1,0} \cdot s_{1,0}$.
   - Session 1. Play as sender of PVW OT with input $(s_{0,1}, s_{1,1})$. Namely, compute $(u_{d,0}, v_{d,0})$ $= \mathsf{RAND}(g_{d,0}, h_{d,0}, pk_0)$, for $d = 0, 1$ and $w_{0,1} = v_{0,1} \cdot s_{0,1}$ $w_{1,1} = v_{1,1} \cdot s_{1,1}$.
     Where $\mathsf{RAND}(g, h, g', h')$ is the following functionality: pick $s, t \in \mathbb{Z}_q$, output $u = g^s \cdot h^t$ and $v = g'^s \cdot h'^s$.

3. Send $(u_{0,0}, w_{0,0})$, $(u_{1,0}, w_{1,0})$ for session 0. Send $(u_{0,1}, w_{0,1})$, $(u_{1,1}, w_{1,1})$ for session 1.

Decryption. R obtains $s_{b,0} = \frac{w_{b,0}}{(u_{b,0})^{r_0}}$ and $s_{b,1} = \frac{w_{b,1}}{(u_{b,1})^{r_1}}$ and outputs $s_b = s_{b,0} + s_{b,1}$.

Efficiency The above protocol has the following complexity. *Round Complexity.* The protocol requires one message per party only. *Exponentiations.* In the initialization phase the receiver computes 6 exponentiation for the parameter generation and $2t$ exponentiations for the non-interactive WI proof $\mathsf{proof}_{\mathsf{cons}}$. This is done only once and the same parameter can be re-used for more than one transfer. For each transfer the receiver computes 6 exponentiations. The sender computes $2t$ exponentiations to check the WI proof, and 8 exponentiations for each transfer. The total number of exponentiation is 4t+20 exponentiations, where t is the statistical parameter. *Hash evaluations.* The sender compute $2t + 1$ hash evaluations for the WI proof $\mathsf{proof}_{\mathsf{cons}}$, while the receiver evaluates the hash $t + 1$ times.

Batch Committing OT The notion of Committing-OT has been introduced by Kiraz and Schoenmakers in [15], and is a modification of standard OT functionality where, at the end of the protocol, the OT receiver additionally receives commitments to the inputs of the sender, and the OT sender outputs the opening of such commitments.

More specifically, the sender runs OT with input (s_0, r_0), (s_1, r_1) and the receiver runs with input b. At the end of the protocol the receiver additionally obtains commitments $\mathsf{Com}(s_0; r_0)$, $\mathsf{Com}(s_1, r_1)$ and the sender outputs r_0, r_1.

The work [1] on which we build upon, requires a committing OT protocol which is a property already satisfied by PVW OT. To see why, note that the message sent by the OT sender in the second round can be seen as a commitment of the sender's input. (E.g., the message to retrieve string s_b corresponds to the pair (u_b, w_b) where $u_b = g_b^s \cdot h_b^t$ and $w = g'^s \cdot h'^t \cdot s_b$ and g', h' is the temporary key sent by the receiver). Our one-sided simulatable OT is also a committing OT. The reason is that it can be seen as a modification of the PVW OT where sender and receiver basically repeats the PVW OT twice in parallel.

Furthermore, as in [28] our protocol can be modified to allow batch OT, where the sender plays with m strings for 0 and m strings for 1: $[K_{0,1}, \ldots, K_{0,m}]$ and $[K_{1,1}, \ldots, K_{1,m}]$ and the receiver plays with one bit b only, and select one of the m-tuples. In order to send m strings, the sender simply runs procedure $\mathsf{RAND}()$ $2m$ times reusing the same temporary keys sent by the receiver for that transfer.

5.1 Proof of security

We show that Protocol 2 is one-sided UC-simulatable. Namely, we show that it is UC-simulatable when the receiver is malicious, while it is only secure in the indistinguishability sense when the sender is malicious. Before proceeding with the formal proof, we write the formal definition of one-sided simulation adapted from [12] to the Oblivious Transfer functionality in the gRO model.

Definition 3. *Let $\mathcal{F}_{\mathsf{tot}}$ be the Oblivious Transfer functionality as shown in Fig. 3. We say that a protocol π securely computes $\mathcal{F}_{\mathsf{tot}}$ with* `one-sided simulation` *if the following holds:*

1. *For every non-uniform PPT adversary R^* controlling the receiver in the real model, there exists a non-uniform ppt adversary $\mathcal{S}$ for the ideal model, such that for any environment $\mathcal{Z}$,*

$$\{\mathsf{IDEAL}^{\mathcal{G}_{\mathsf{gRO}}}_{\mathcal{F}_{\mathsf{t}},\mathcal{S},\mathcal{Z}}(s_0, s_1, b, z)\}_{z\in\{0,1\}^*} \approx \{\mathsf{REAL}^{\mathcal{G}_{\mathsf{gRO}}}_{\pi,\mathsf{R}^*,\mathcal{Z}}(s_0, s_1, b, z)\}_{z\in\{0,1\}^*}$$

2. *For every non-uniform PPT adversary S^* controlling the sender it holds that:*

$$\{\mathsf{view}^R_{\pi,S^*(z)}(s_0, s_1, 0)\}_{z\in\{0,1\}^*} \approx \{\mathsf{view}^R_{\pi,S^*(z)}(s_0, s_1, 1)\}_{z\in\{0,1\}^*}\}$$

   where $\mathsf{view}^R_{\pi,S^*(z)}$ denotes the view of adversary S^* after a real execution of protocol π with the honest receiver R.

We are now ready to prove the following theorem.

Theorem 1. *If $\mathsf{NIWIpok}$ is a witness-indistinguishable proof of knowledge in the gRO model and under the assumption that DDH is hard in $\mathbb{G}$, Protocol 2 securely computes $\mathcal{F}_{\mathsf{tot}}$ with* `one-sided simulation` *in the gRO model.*

Proof. **UC-security against a Malicious Receiver.** We show a simulator $\mathcal{S}_R$ that extracts the input of the malicious receiver R^* using the observability of the RO. Let sid the session id of the challenge protocol that $\mathcal{S}$ must simulate. Let $\mathcal{Q}_{|\mathsf{sid}}$ be the list of queries made by R^* to the random oracle *and* of `illegitimate` queries performed by the environment $\mathcal{Z}$ in concurrent sessions of arbitrary protocols accessing gRO.

$\mathcal{S}_R$ extracts the witness $\alpha_{0,e}$ by running the straight-line extractor associated to the $\mathsf{NIWIpok}$ protocol on input $\mathcal{Q}_{|\mathsf{sid}}$ and the proof $\mathsf{proof}_{\mathsf{cons}}$. If the extraction fails, $\mathcal{S}_R$ aborts. Due to the proof of knowledge property of $\mathsf{NIWIpok}$, if $\mathsf{proof}_{\mathsf{cons}}$ is accepting, then $\mathcal{S}_R$ will not abort with all but negligible probability.

$\mathcal{S}_R$ detects in which session R^* sent the correct parameters by checking if $h_{0,0} = (g_{0,0})^{\alpha_{0,e}} \wedge h_{1,0} = (g_{1,0})^{\alpha_{0,e}+1}$ or $h_{0,1} = (g_{0,1})^{\alpha_{0,e}} \wedge h_{1,1} = (g_{1,1})^{\alpha_{0,e}+1}$. Let e be such session. To extract the input played by R^*, $\mathcal{S}_R$ looks at the temporary key pk_e. If $h^e = (g_e)^{\alpha_{0,e}}$ it sets the secret bit $b = 0$, else if $h^e = (g_e)^{\alpha_{0,e}+1}$ it sets $b = 1$. $\mathcal{S}_R$ plays b in the ideal OT functionality and obtains s_b. Finally it computes Round 2 of the OT protocol as the honest sender but on input the pair $(s_b, 0)$. Indistinguishability follows from the indistinguishability of the PVW OT in session e and from the fact that the inputs of the sender are secret shared between the two sessions. UC-security follows from the fact that the simulator is straight-line and that the proofs of knowledge sent by R^* are non-malleable in the gRO model.

Indistinguishability against a Malicious Sender. We want to show that for any PPT malicious sender S^*, the view of S^* when R is playing with bit 0 is indistinguishable from the view of S^* when R is playing with bit 1. Informally, this holds due to the hardness of the DDH problem and the WI property of the $\mathsf{NIWIpok}$ shown in Prot. 3.2.

The proof goes by hybrids arguments. Recall that the OT protocol consists of two parallel sessions of PVW OT (that we denote as session 0 and session 1) and one WI proof. A receiver playing with bit b, runs both the sessions with input b.

We start with an execution where the receiver plays with input 0 in both sessions. Through a sequence of hybrids (i.e., using the WI property of $\mathsf{proof}_{\mathsf{cons}}$ and to the DDH assumption) we move to an execution where the receiver plays bits 0 and 1 and we argue that this hybrid is indistinguishable from the case in which R plays with input $0, 0$. Following the same arguments we end with the case in which R played with input $1, 1$, therefore proving the claim. More precisely, the hybrids are the following.

Hybrid H_0. In this hybrid R plays with input 0, namely it runs the two parallel OT sessions with secret bit 0. This hybrid corresponds to a real execution where the receiver plays with bit 0:

$$\{\mathsf{view}^R_{\pi,S^*(z)}(s_0, s_1, 0)\}_{z\in\{0,1\}^*}$$

Hybrid H_1. In this hybrid R picks a bit e and computes parameters $\mathsf{par}_{\bar{e}}$ as a DDH tuple. Namely $h_{1,\bar{e}} = g_{1,e}^{\alpha_{0,\bar{e}}}$. (Therefore, the tuple $\{g_{0,\bar{e}}, h_{0,\bar{e}}, g_{1,\bar{e}}, \frac{h_{1,\bar{e}}}{g_{1,\bar{e}}}$ is *not* a DDH tuple.) R compute the WI proof with witness $\alpha_{0,e}$. The difference between H_0 and H_1 is in the fact that in H_0 $\mathsf{par}_{\bar{e}}$ is a not a DDH tuple while in H_1 it is. Assume that there is a distinguisher D for the views output by S^* in hybrids H_0 and H_1. We can use S^* and D to help a DDH distinguisher as follows. The DDH distinguisher receives the challenge tuple (g, g_1, g_2, g_3) from the DDH experiment. It sets $\mathsf{par}_{\bar{e}} = (g, g_1, g_2, g_3)$, and computes any other message as in experiment H_0. Then it runs the distinguisher D on the view of S^* and outputs whatever D outputs. If (g, g_1, g_2, g_3) was a DDH tuple, then the messages computed by the DDH distinguisher are distributed identically to H_1,

otherwise they are distributed identically to H_0. From the hardness of the DDH problem, H_0 and H_1 are computationally indistinguishable.

Hybrid H_2. In this hybrid we change the bit played in session $\bar{e}$, from 0 to 1. Because $\mathsf{par}_{\bar{e}}$ is a DDH tuple, then the temporary key $pk_{\bar{e}}$ computed in this hybrid is distributed identically to the $pk_{\bar{e}}$ computed in H_1. Therefore, the two hybrids are identical.

Hybrid H_3. In this hybrid we change again the parameters $\mathsf{par}_{\bar{e}}$ so that they are not a DDH tuple. Again, the only difference between H_2 and H_3 is the fact that in H_2, parameters $\mathsf{par}_{\bar{e}}$ is a DDH tuple and in hybrid H_3 is not. H_2 and H_3 are indistinguishable due to the hardness of the DDH problem.

Hybrid H_4 In this hybrid R changes the witness used to compute $\mathsf{proof}_{\mathsf{cons}}$. Namely, the proof is computed using as witness $\alpha_{0,\bar{e}}$. Assume that there is a distinguisher D for the view of S^* in these two hybrids, then we can use S^* and D to distinguish the witness used in $\mathsf{proof}_{\mathsf{cons}}$ and break its witness indistinguishability property.

Hybrid H_5. In this hybrid R set parameter par_e as a DDH tuple. It follows from the same arguments for the indistinguishability of H_0, H_1 that H_5 and H_4 are indistinguishable.

Hybrid H_6. In this hybrid R uses the bit 1 also in session e. Therefore, in this hybrid R is playing with bit 1. It follows from the same arguments for the indistinguishability of H_1, H_2 that H_5 and H_6 are indistinguishable. The view of the S^* in this hybrids corresponds to:

$$\{\mathsf{view}^R_{\pi,S^*(z)}(s_0, s_1, 1)\}_{z\in\{0,1\}^*}$$

and the claim is proved.

□

6 UC-secure NISC

Given any two-round UC-OT, a non-interactive UC-2PC (UC-NISC, for short) for any function $f(x, y)$ can be easily constructed in the CRS model as follows. P_1 sends the first message of the OT based on its input x, P_2 prepares a garbled circuit for f and sends the garbled keys for P_1 using the second message of OT. Additionally P_2 sends the garbled keys for its own input y and a NIZK proof proving that the garbled circuit is correct, the garbled keys sent in the OT are consistent with the garbled circuit and with P_2's input. This solution is very inefficient as the NIZK proof requires the use of the circuit of the underlying primitives and thus expensive NP reductions.

In [1] Afshar et al. show how to implement UC-secure NISC protocol very efficiently in the CRS model. They start with the multi-round highly-efficient 2PC protocol presented by Lindell [17] and use several tricks to squash it down to two rounds. The round-complexity of Lindell's construction stems from the cut-and-choose phase. The main contribution of [1] is to show how to perform all the checks required by the cut-an-choose technique *non-interactively*. We built upon their protocol to achieve the same result in the gRO model. In the following, we first outline the NISC protocol of [1] and then we discuss how we modify their construction to achieve our result.

NISC in the CRS model [1] To implement the cut-and-choose in only two rounds [1] uses several techniques. We provide an overview of the technique used in [1] in Appendix B. Here we

discuss only two salient points of their protocol that allow us to achieve the same result in the gRO model.

Recall that in a typical cut-and-choose protocol, P_2 sends t garbled circuits $gc_1, \ldots, gc_t$ to P_1, who tests the correctness of them by asking P_2 to "open" half of the circuits (i.e., to reveal the randomness used to generate them). If all the checks go through, P_1 is convinced that most of the remaining circuits are correct. The first idea to achieve non-interactiveness, is to let P_1 select the circuits that she wants to check, via the selection bits of several OTs. Namely, additionally to the OT for the input, called `input-OT` in [1], P_1 runs one OT for each garbled circuit. Such OTs are called `circuit-OT`: P_1 participates to the i-th `circuit-OT` with input $b_i = 0$ if she wants to check (i.e., obtain the randomness for) circuit gc_i, otherwise if she wants to evaluate gc_i, she sends $b_i = 1$.

The second idea is to let P_2 compute the garbled circuit, the messages for `input-OT` protocols, and other relevant information related to input/output consistency proofs, using randomness generated by a PRF. Namely, all such messages are computed by invoking a PRF with key $seed_i$. In the `circuit-OT` protocol P_2 will place the string $seed_i$ for the case $b_i = 0$ and thus gc_i is a circuit that will be checked, and will instead send the keys corresponding to her input, in case $b_i = 1$ and thus gc_i is an evaluation circuit.

Security of NISC [1] in the CRS model. The UC-security of NISC [1] relies crucially on the fully simulatability of the underlying PVW OT protocol. Informally, the intuition behind the proof is the following. *Simulating Malicious* P_1^*. The simulator extracts the secret input x^* of P_1^* from the `input-OT`, by running the UC-simulator granted by PVW OT in the CRS model. It sends x^* to the ideal functionality and receives the output $z = f(x^*, y)$. Next, from the `circuit-OTs` it extracts the indexes of the circuits that will be checked, and hence it computes the evaluation and checking garbled circuits accordingly.

Simulating Malicious P_2^*. The simulator generates the first message of P_1 for `circuit-OTs` and `input-OTs`, by running honest P_1 with a random input. Due to the security of PVW OT in the CRS model, these messages are distributed identically for the real input and the random one. When the simulator receives P_2's message it performs the same correctness checks as P_1. If P_2 cheats in one of the checked garbled circuits, the simulator will abort, and this happens with the same probability that P_1 would abort in the real execution. If the emulated P_1 does not abort, the cut-and-choose guarantees that whp, at least one evaluated garbled circuit is correct. Therefore, the simulator derives the input of P_2 by extracting the seed $seed_i$ of the correct circuit i from the i-th `circuit-OT`.

NISC in the gRO model: Our techniques In the original NISC protocol the extraction of the input of the sender is done by extracting the keys and the seeds from the executions of `input-OT` and `circuit-OT`. Our idea is to extract the input of P_2 by observing the queries that P_2 makes to the RO. Toward this end, we require that each seed $seed_i$ used to generate the randomness for the computation of the circuits and other critical information, is not picked by P_2 but it is computed as the output of the RO. Namely, P_2 queries the RO on some random input q_i and set $seed_i = \mathsf{gRO}(\mathsf{sid}, P_2 \| q_i)$. Then, in the cut-and-choose phase, P_2 plays the OT protocol using q_i instead of $seed_i$. In order to pass the consistency checks, P_2 must query the RO to compute *most* of the PRF seeds. In this way the simulator, which obtains all the queries made to $\mathcal{G}_{\mathsf{gRO}}$ for a specific session sid, gets most of the seeds and is able to recompute the keys and garbled circuits without having to extract them from the `input-OTs` and `circuit-OTs`.

This idea allows us to replace the PVW OT protocol used in [1] which is fully simulatable, with

our one-sided UC-simulatable OT protocol, and still be able to simulate a malicious P_2^*.

Security of our NISC in the gRO model. We now outline the ideas behind the security proof of our NISC protocol. *Simulating Malicious* P_1^*. Because our OT protocol is UC-simulatable against a malicious receiver the proof in this case follows the same proof provided in [1]. *Simulating Malicious* P_2^*. Our simulator works similarly to the simulator of [1] except that the information necessary to derive the input of P_2^* is obtained by looking at the queries to the RO. Due to the cut-and-choose, P_2 is forced to query the RO to compute most of the PRF seeds used in the protocol, otherwise it will be caught whp by P_1. Therefore, if P_2^* passes all the checks, whp the simulator will extract the relevant information.

The crucial difference in our case is in the indistinguishability of the first message of `input-OTs` computed by the simulator. In [1], the simulator, following the UC-simulator of PVW OT, sets the OT parameters contained in the CRS to be a DDH tuple. In this way, the first message of `input-OTs` computed using a random input is distributed identically to the one played by an honest P_1 computed with the actual input.

In our case, as we are using our one-sided simulatable OT in the gRO model, the OT parameters are honestly generated to be a non-DDH tuple. Thus, we cannot claim that the view generated by the simulator is identically distributed to the view generated by an honest P_1. Nevertheless, due to the indistinguishability property of our OT against malicious senders, the messages computed by the simulator are computationally indistinguishable from the messages computed by P_1, and thus we can use a malicious P_2 to break the indistinguishability of our OT.

6.1 The Protocol

In this section we present our modification of the NISC protocol of [1] which is UC-secure in the gRO model. We stress that the protocol is essentially the same as [1] with the following two modifications: (a) the underlying OT is one-sided UC-simulatable; (b) the seed of the PRF is computed using the RO. In the description of the protocol we will highlight the points where the two protocols differ.

Notation and sub-protocols. We denote by $\mathsf{oneside\text{-}COT}_1(b)$ and $\mathsf{oneside\text{-}COT}_2((k_1^0,k_1^1),\ldots,(k_t^0,k_t^1))$ Round 1 and Round 2 of our one-sided UC-simulatable (batch) committing OT shown in Section 5, respectively. Notation oneside stresses that we use a one-sided UC-simulatable OT instead of a fully UC simulatable OT. Let $\mathbb{G}$ be a group of prime order q with generator g. Define $\mathsf{EGCom}(h,b,r) = (g^r, h^r g^b)$ as the "El Gamal" commitment for a bit b. This commitment has two important properties: (1) there exists a very efficient way to prove the equality of two commitments Namely, let $c_0, c_1 = \mathsf{EGCom}(h,m,r) = g^r, h^r g^m$ and $c_0', c_1' = \mathsf{EGCom}(h,m,r') = g^{r'}, h^{r'} g^m$, one can verify that $(c_0')^{r-r'} = c_0$ and $(c_1')^{r-r'} = c_1$; (2) there exists a trapdoor that allows the extraction of the committed value. I.e., if $h = g^w$, and $c = (c', c'')$ with $c' = g^r$ and $c'' = h^r g^b$, one can extract b by checking if $\frac{c''}{c'^w}$ is equal to 1 or g. Both properties are used crucially to guarantee correctness against a malicious sender. Let ReHash be a collision-resistant hash function that is a suitable randomness extractor.

Protocol 3. NISC in the gRO model(built upon [1]).

Inputs. P_1 has input $x \in \{0,1\}^{n_1}$ and P_2 has input $y \in \{0,1\}^{n_2}$. Let $f : \{0,1\}^{n_1} \times : \{0,1\}^{n_2} \to \{0,1\}^m$ and C be a circuit computing f. The inputs wires of P_1 and P_2 are denote by $\mathsf{IN}(1)$ and $\mathsf{IN}(2)$ respectively. The set of output wires is denoted by OUT.

Enc denotes an encryption scheme, Commit denotes a commitment scheme and prf a pseudo-random function. Let t denote the number of circuits and sid the session identifier.

P_1**'s message**:

- (Choose circuits to check) Pick a random t-bit string $c_1, \ldots, c_t$. Let T be the set of i such that $c_i = 1$.
- (`Circuit-OT` queries) For each index $i \in [t]$, publish $\mathsf{oneside\text{-}COT}_1(c_i)$.
- (`Input-OT` queries) For each input $j \in \mathsf{IN}(1)$ publish $\mathsf{oneside\text{-}COT}_1(x_j)$. **Difference with [1].** We use one-sided UC-simulatable OT, instead of UC-secure OT.

P_2**'s message**:

Commit to inputs, outputs and trapdoor.

- (Trapdoor) Pick $w \in_R \mathbb{Z}_q$ and send $h = g^w$.
- (Input commitment) Send $\mathsf{EGCom}(h; y_j, r_j)$ for $j \in \mathsf{IN}(2)$, where y_j is its input for input-wire j and r_j is randomly chosen.
- (Output commitment) Send $h_{j,0} = g^{w_{j,0}}$ and $h_{j,1} = g^{w_{j,1}}$ for each output wire $j \in \mathsf{OUT}$, where $w_{j,0} \in_R \mathbb{Z}_q$ and $w_{j,1} = w - w_{j,0}$.

Generate garbled circuits. For each circuit $i \in [t]$.

- Randomly choose q_i and set $seed_i = \mathsf{gRO}(\mathsf{sid}, P_2 \| q_i)$. **Difference with [1].** The seed $seed_i$ is computed by invoking the random oracle gRO.
- Compute $u_{i,j,b} = \mathsf{EGCom}(h; b, r_{i,j,b})$ for all wire $j \in \mathsf{IN}(2)$, $b \in \{0,1\}$ and $r_{i,j,b} = \mathsf{prf}_{seed_i}(\text{“}EGCom\text{”} \circ j \circ b)$.
- (Compute Garbled Circuit) Compute gc_i:
  - (Labels for input wires) For $j \in \mathsf{IN}(2)$ and $b \in \{0,1\}$, set $\mathsf{label}(gc_i, j, b) = \mathsf{ReHash}(P_2 \circ \mathsf{sid}, u_{i,j,b})$.
  - (Labels for other wires) Any other label for wire j and bit b is constructed as usual using randomness generated by $seed_i$, i.e., with randomness $\mathsf{prf}_{seed_i}(\text{“}label\text{”} \circ j \circ b)$.
- (Commitments to the input labels) Send commitments $\{\mathsf{Commit}(u_{i,j,\delta_{i,j}}), \mathsf{Commit}(u_{i,j,1-\delta_{i,j}})\}_{j \in \mathsf{IN}(2)}$ where bit $\delta_{i,g}$ is randomly chosen. Such commitments are computed using randomness derived from $seed_i$. We denote by $d_{i,j,\delta}$ the decommitment of the *commitment* of $u_{i,j,\delta}$.

Prepare cheating recovery box. For $j \in \mathsf{OUT}$ send:

- *Output recovery commitments*: $h_{j,0} \cdot g^{K_{i,j,0}}$ and $h_{j,1} \cdot g^{K_{i,j,1}}$ where $K_{i,j,0}, K_{i,j,1}$ are randomly chosen.
- Two encryptions to the labels of the output: $\mathsf{Enc}(\mathsf{label}(gc_i, j, 0), K_{i,j,0})$, $\mathsf{Enc}(\mathsf{label}(gc_i, j, 1), K_{i,j,1})$.

Prepare proofs of input/output consistency.

- Let inputs_i be the set $\{u_{i,j,y_j}, d_{i,j,y_j}\}_{j \in \mathsf{IN}(2)}$. Let $\mathsf{inputsEq}_i$ be the set $\{r_j - r_{i,j,y_j}\}_{j \in \mathsf{IN}(2)}$.
- Let $\mathsf{outputsDecom}_i$ be the set $\{w_{j,0} + K_{i,j,0},\ w_{j,1} + K_{i,j,1}\}_{j \in \mathsf{OUT}(2)}$.

- Pick a random k_i and send encryption $\mathsf{Enc}(k_i, \mathsf{inputs}_i \circ \mathsf{inputsEq}_i \circ \mathsf{outputsDecom}_i)$.

Prepare the OT answers.

- (`Input-OT`) P_2 prepares the messages of oneside-COT_2 for the `input-OT` executions. These messages are computed on input the labels: $\{\mathsf{label}(gc_i, j, 0)\}$ and $\{\mathsf{label}(gc_i, j, 1)\}$ with $i \in [t]$ and $j \in \mathsf{IN}(1)$.

  The randomness $r_{i,j}$ used to compute the OT message for wire j and circuit i is set as $r_{i,j} = \mathsf{prf}_{seed_i}(\text{“}OT\text{”} \circ 1 \circ \text{“}r\text{”} \circ i \circ j)$.

- (`Circuit-OT`) P_2 prepares the messages oneside-COT_2 for the `circuit-OT` executions. These messages are computed on input the pairs (q_i, k_i), for $i \in [t]$. **Difference with [1].** P_2 transfers the query q_i instead of the seed $seed_i$.

P_1**'s Computation**:

- **Check circuit consistency.** For each $i \in T$, compute $seed_i = \mathsf{gRO}(\mathsf{sid}, P_2 \| q_i)$ and check that $seed_i$ correctly generated circuit gc_i and the answers of the i-th execution of `input-OT`. If any check fails, abort.

- **Check consistency of the input/output of P_2.** For all circuits $i \in [t]/T$.

  - Verify that $h_{j,0} \cdot h_{j,1} = h$ for $j \in \mathsf{OUT}$.
  - Check that $\mathsf{outputsDecom}_i$ are correct discrete-logs of the values in set $\{h_{j,b} g^{K_{i,j,b}}\}_{j \in \mathsf{OUT}, b \in \{0,1\}}$.
  - Check that $\mathsf{inputs}, \mathsf{inputsEq}$ are consistent with the input-commitments: check $u_{i,j,y_j}(g^{r_j - r_{i,j,y_j}}, h^{r_j - r_{i,j,y_j}}) = \mathsf{EGCom}(h; y_j, r_j)$. Otherwise abort.
  - Evaluate circuit g_i. Say P_1 learns the labels $\{l_{i,j}\}_{j \in \mathsf{OUT}}$. P_1 uses these labels to decrypt the corresponding encryptions $\mathsf{Enc}(\mathsf{label}(gc_i, j, b), K_{i,j,b})$ from the cheating recovery box. Then it checks if the result is the correct "decommitment" of the output recovery commitment $h_{j,b} g^{K_{i,j,b}}$ where the b are the actual output bits P_1 received from gc_i. If all these steps pass for all output wires, label circuit gc_i as *semi-trusted*.

- **Compute the output.** If the output of *all* semi-trusted circuits is the same, then output such value. Otherwise,

  - Let $gc_i, gc_{i'}$ be two semi-trusted circuits that have different outputs in the j-th output wire, and let $l_{i,j}$ and $l_{i',j}$ be their output labels. P_1 learns $w_{j,0}$ from one of the labels and $w_{j,1}$ from the other (since it learns $K_{i,j,b}, K_{i',j,1-b}$ from the cheating recovery boxes, and $w_{j,b} + K_{i,j,b}$, $w_{j,1-b} + K_{i',j,1-b}$ from $\mathsf{outputsDecom}_i, \mathsf{outputsDecom}_{i'}$).
  - P_1 computes $w = w_{j,0} + w_{j,1}$ and decrypts the input-commitments provided by P_2. Say y is the input so obtained. P_1 outputs $f(x, y)$.

This concludes the description of the protocol.

Efficiency Our protocol inherits the same complexity of the NISC protocol of [1] with the following additional overhead. Concerning the exponentiations, the additional overhead corresponds to the overhead of the one-sided OT protocol that we discussed in Section 5. Concerning hash evaluations, our protocol requires t hash evaluations for P_2 necessary to obtain each seed $seed_i$ for the PRF evaluations, and $t/2$ hash evaluations for the receiver when checking the garbled circuits.

6.2 Proof of security of NISC protocol

In this section we prove that Protocol 3 is UC-secure in the gRO model. We prove the following theorem:

Theorem 2 (Protocol 3 is UC-secure in the gRO model.)**.** *Assuming that the DDH problem is hard in* $\mathbb{G}$*,* prf *is a pseudo-random function,* ReHash *is an extractor and a collision-resistance hash function, then Protocol 3 is GUC-secure in the* gRO *model.*

Proof. P_1 **is corrupted.** We describe the strategy of the simulator $\mathcal{S}_1$ for the case in which P_1^* is malicious. We stress that this proof is very similar to the proof provided in [1], which we rewrite for completeness.

High-level description. Informally, $\mathcal{S}_1$ first extracts the inputs (x, T) of P_1^* invoking the UC-simulator of the one-sided simulatable OT protocol, where T is the set of circuits that P_1^* wants to check. $\mathcal{S}_1$ sends message $\mathsf{input}(x)$ to $\mathcal{F}_{\mathsf{nisc}}$ and obtains $(P_2, f(x, y))$ from $\mathcal{F}_{\mathsf{nisc}}$. Let $z = f(x, y)$. It then proceeds as follows. For all $i \in T$, $\mathcal{S}_1$ honestly computes the garbled circuits gc_i with the difference that in the `circuit OT` it will play replacing k_i with a random string. Additionally, it replaces all the encryptions under key k_i with the encryption of 0. These differences however are not detected by a malicious P_1^* due to the semantic security of the encryption scheme and the UC-security of the OT. For all $i \in T$, $\mathcal{S}_1$ replaces circuits gc_i with fake circuits which outputs always z, using fresh randomness (instead of PRF). It computes the labels $u_{i,j,b}$ with commitment of 0, and plays the `input-OT` using random values in place of the keys corresponding to the bits of $\bar{x}$ and in the `circuit-OT` using a random value instead of $seed_i$.

The indistinguishability of the messages computed in this case follows from the pseudo-randomness of the PRF, the security of EGCom, the security of garbled circuits and the OT.

Formal proof. The formal strategy of the $\mathcal{S}_1$ is the following.

Extraction of P_1^*'s input.

The one-side UC-simulatability of the OT protocol guarantees that when the receiver of the OT is corrupted, the UC simulator obtains the input used by the receiver to query the $\mathcal{F}_{\mathsf{tot}}$ functionality. Under this assumption, we have that P_1^*, which is the receiver of the OT in the NISC protocol, sent $(\mathsf{receive}, \mathsf{sid\text{-}ot}, c_i)$ for $i \in [t]$ and $(\mathsf{receive}, \mathsf{sid\text{-}ot}, x_j)$ for $j \in \mathsf{IN}(1)$, where sid-ot is the session id of the OT protocol, to the $\mathcal{F}_{\mathsf{tot}}$ functionality, and such messages are intercepted by $\mathcal{S}_1$.

Therefore, $\mathcal{S}_1$ obtains $x = x_1, \ldots, x_{n_1}$ and the set T, it sends $\mathsf{input}(x)$ to $\mathcal{F}_{\mathsf{nisc}}$ and obtains $z = f(x, y)$. Then it proceeds as follows.

Simulation of P_2's second message. Let T be the set of circuits that P_1 chose to open.

- (Evaluation circuits) For all $i \in [t] \backslash T$, $\mathcal{S}_1$ prepares the second message as an honest P_2 playing with input 0^{n_2} with the following differences.
  - Garbled circuits. $\mathcal{S}_1$ replaces each gc_i with a garbled circuit that always output z, regardless of the input.
  - Garbled keys. $\mathcal{S}_1$ sends garbled keys corresponding to input 0^{n_2} (instead of input y). Namely, send inputs_i as the set $\{u_{i,j,0}, d_{i,j,0}\}_{j \in \mathsf{IN}(2)}$.
  - $\{\mathsf{Commit}(u_{i,j,\delta_{i,j}}, \mathsf{Commit}(u_{i,j,1-\delta_{i,j}}\}_{j \in \mathsf{IN}(2)}$: replace the commitment of $u_{i,j,1}$ with commitments to 0.
  - `input OT`. In the input OT, $\mathcal{S}_1$ replaces the garbled key that P_1 did not choose with random values.

- (Checked circuits) For $i \in T$, $\mathcal{S}_1$ prepares the second message as P_2 with the following differences.
  - `Circuit OT`. $\mathcal{S}_1$ replaces the keys k_i in the OT, with random values.
  - Encrypted values. $\mathcal{S}_1$ replaces the values encrypted under key k_i with encryption of 0 (under k_i).

Indistinguishability. Indistinguishability of the simulation is proven through a sequence of hybrids.

Hybrid H_0. This is the real world executions. $\mathcal{S}_1$ on input y runs as an honest P_2.

Hybrid H_1. (Changing `circuit OT` answers). In this hybrid $\mathcal{S}_1$ replaces the OT answers as follows. For $i \in T$ (the circuits that will be checked), it replaces the value k_i played in the i-th `circuit OT` using a random value. For $i \notin T$ (the circuits that will be evaluated), replaces the keys not asked by P_1^* with random values. Due to security of the OT H_1 and H_0 are identical.

Hybrid H_2. (Replacing PRF with fresh random values). In this hybrid $\mathcal{S}_1$ generated the garbled circuits that must be evaluated (i.e., gc_i with $i \notin T$) and the OT, commitments etc., using fresh randomness instead of the PRF evaluated on $seed_i$. uses knowledge of the set T extracted from the OT. Due to the pseudo-randomness of the PRF H_2 and H_1 are computationally indistinguishable.

Hybrid H_3. (Changing `input OT` answers). In this hybrid $\mathcal{S}_1$ replaces the keys not asked by P_1^*, with random values. Due to security of the OT H_3 and H_2 are identical.

Hybrid H_4. (Playing with input 0^{n_2} in the checked circuits). In this hybrid $\mathcal{S}_1$ computes the messages for the circuits that will be checked, running on input 0^{n_2} instead of y. Namely, it computes the encryption $\mathsf{Enc}(k_i, \mathsf{inputs}_i \circ \mathsf{inputsEq}_i \circ \mathsf{outputsDecom}_i)$ so that it encrypts the data corresponding to the input 0^{n_2}. Due to the semantic security of the encryption scheme, hybrids H_3 and H_4 are indistinguishable. At this point $\mathcal{S}$ is not using the input y in the sessions $i \in T$.

Hybrid H_5. (Changing commitments $u_{i,j,1}$ in the evaluated circuits). In this hybrid, for all the evaluated circuits i, $\mathcal{S}_1$ computes commitments $u_{i,j,1}$ as commitment to 0. Due to the hiding of EGCom, hybrids H_4 and H_5 are computationally indistinguishable.

Hybrid $H_{6,j}$ (with $j = 1$ to $t - |T|$). (Replacing evaluated circuits with fake circuits). In Hybrid $H_{6,j}$ $\mathcal{S}_1$ replaces the first j evaluated circuits with fake circuits that always output z. Due to the security of the garbling scheme hybrids H_5 and $H_{6,1}$ and $H_{6,j}$ and $H_{6,j+1}$ are indistinguishable.

Hybrid H_7. (Changing the output labels in the cheating recovery box). In this hybrid $\mathcal{S}_1$ replaces the entries corresponding to $\bar{z}$ in the cheating recovery box. Namely, it replaces encryptions $\mathsf{Enc}(\mathsf{label}(gc_i, j, \bar{z}_i), K_{i,j,\bar{z}_i})$ with encryption of 0. Due to the semantic security of the underlying encryption scheme, hybrids H_7 and $H_{6,t-|T|}$ are indistinguishable.

Hybrid $H_{8,i}$. (Playing with input 0^{n_2} in the evaluated circuits). In this hybrid $\mathcal{S}_1$ plays with input $y_i = 0$ in all the evaluated circuits. This means that the input commitments are computed as $\mathsf{EGCom}(h; 0, r_i)$, where $h = g^w$. Hybrids $H_{8,i}$ and $H_{8,i+1}$ can be proved indistinguishable under the DDH assumptions.

Hybrid H_{8,n_2}. This is the procedure of $\mathcal{S}_1$ playing in the ideal world.

P_2 is corrupted. We describe the strategy of the simulator $\mathcal{S}_2$ for the case in which P_2^* is malicious. The strategy of $\mathcal{S}_2$ is simple.

- Compute the first message honestly as P_1 but running with random inputs.
- When receiving the message from P_2^* proceed as follows: Emulate an honest P_1 until the end of the protocol. If P_1 aborts, then abort. If P_1 did not abort, then obtain the list of queries $\mathcal{Q}_{|\mathsf{sid}}$ from $\mathcal{F}_{\mathsf{nisc}}$, and learn $(q_i, seed_i)$ for $i \in [t] \backslash T$.
- If there exists at least an i such that the randomness computed using $seed_i$ is consistent with the circuit gc_i and all the messages computed for session i, then use $seed_i$ to extract P_2's input.
- Send $\mathsf{input}(\mathsf{sid}, y)$ to the ideal functionality $\mathcal{F}_{\mathsf{nisc}}$.

Indistinguishability. Indistinguishability of the simulation is proven through a sequence of hybrids.

Define a circuit gc_i to be *good* if it passes all the checks: Query q_i yields to $seed_i$ that properly generates the randomness for (a) the OT, (b) the labels $u_{i,j,b}$,(c) the commitment to the labels $\{\mathsf{Commit}(u_{i,j,\delta_{i,j}}, \mathsf{Commit}(u_{i,j,1-\delta_{i,j}}\}_{j \in \mathsf{IN}(2)}$, (d) the circuit gc_i.

If a set of circuits is good, then it must hold that the input labels of such circuits are all consistent with the same input[3].

Hybrid H_0. This is the real execution with $\mathcal{S}_2$ running as a honest P_1 with input x.

Hybrid H_1. In this hybrid $\mathcal{S}_2$ extracts the query q_i for a *good* circuit gc_i that was chosen for evaluation (due to the cut-and-choose many such i exist). Knowledge of q_i allows $\mathcal{S}_2$ to recompute the garbled circuit gc_i and knowledge k_i (obtained from OT) allows to compute the input y ($\mathcal{S}_2$ obtain y also from the knowledge of $K_{i,j,0}, K_{i,j,1}$ for each $j \in IN(2)$, that allows to reconstruct the trapdoor w and therefore extract from the El-Gamal input commitment). $\mathcal{S}_2$ then computes $z = f(x, y)$ itself without evaluating the circuits. Note that if there exists an i' such that evaluation $gc_{i'}$ is semi-trusted and that outputs z', then an honest P_1 would have extracted w from the two different outputs[4], and obtains y from the input commitments (which are statistically binding, therefore there exists one unique y that can be decommitted).

The observability of the gRO allows $\mathcal{S}_2$ to extract queries for session sid and therefore derive the input y.

[3] If not then the proof of equality would not go through. To see why, recall that P_2^* sends the El-Gamal commitments of its input, the labels $u_{i,j,b}$ and a proof of equality $\mathsf{inputsEq}$ between the labels and the commitment of the input. Because the El-Gamal commitments are perfectly binding, finding two accepting proofs $\mathsf{inputsEq}$ is impossible.

[4] Indeed, a semi-trusted has passed the tests for the decommitment of the output recovery box

Due to cut-and-choose, the probability that $\mathcal{S}_2$ fails in extracting the input from a *good* circuit that was chosen for evaluation is 2^{-t}, and therefore hybrid H_0 and H_1 are statistically indistinguishable.

Hybrid $H_{2,i}$ with $i = 1, \ldots, i = n_1$. (Changing the `input-OT` with 0^{n_1}.) This hybrid is the same as H_1 except that now the simulator plays in the i-th `input OT` with $x_i = 0$. Assume that there is a distinguisher between $H_{2,i}$ and $H_{2,i+1}$ then one can construct a malicious sender S^*_{OT} of the one-sided OT, which is able to distinguish the input of an honest external receiver R with the same probability.

The reduction work as follows. S^*_{OT} receives the message $\mathsf{oneside\text{-}COT}_1(b)$ from the external receiver R.

More specifically, this message consists in the parameters g_0, h_0, g_1, h_1, a proof $\mathsf{proof}_{\mathsf{cons}}$ and the temporary public key $\bar{p}k_b$ that allows to retrieve the k-tuple corresponding to a bit b.

S^*_{OT} will use the same parameters and the same proof $\mathsf{proof}_{\mathsf{cons}}$. Then for each index $j < i$ it computes parameters $\bar{p}k_j$ so to retrieve bit $x_j = 0$. For each index $j > i$, S^*_{OT} computes parameters $\bar{p}k_j$ so to retrieve bit x_j. Instead for index i it will use the parameters received from the external receiver R.

Now, when receiving the message from P^*_2, the sender S^*_{OT} computes the output of the function using the same procedure of the simulator $\mathcal{S}_2$ and using the observability of gRO and outputs the same view.

Finally, S^*_{OT} presents the view to the distinguisher, and outputs whatever the distinguisher outputs. Due to the indistinguishability of the one-sided OT protocol, hybrid $H_{2,i}$ and $H_{2,i+1}$ are computationally indistinguishable.

Hybrid H_3. (Changing the input in `circuit-OT`.) In this sequence of hybrids we want to show that regardless of the input played by the simulator in the execution of `circuit-OT`, due to the security of the OT, the probability that P_1 aborts with a different probability than $\mathcal{S}_2$ is negligible.

Towards this end, we consider two sequence of hybrids: in hybrids $\{H^0_{3,i}\}_{i\in[t]}$, the simulator plays the `circuit-OT` with the i-th input set to 0. Then in hybrids $\{H^{c_i}_{3,i}\}_{i\in[t]}$ the simulator plays using as input a random bits.

Hybrid $H^0_{3,i}$, for $i = 1, \ldots, t$. In this hybrid we wish to change the bits used in the `circuit-OT`, with all zeros. The difference between $H^0_{3,i}$ and $H^0_{3,i+1}$ is that in $H_{3,i}$ $\mathcal{S}_2$ plays $\mathsf{oneside\text{-}COT}_1(c_i)$ with a random bit c_i, while in hybrid $H^0_{3,i+1}$, $\mathcal{S}_2$ plays with $\mathsf{oneside\text{-}COT}_1(0)$. A distinguisher between hybrids $H^0_{3,i}$ and $H^0_{3,i+1}$ can be reduced to a malicious sender S^*_{OT} for `circuit-OT`, as before.

Note that $H^0_{3,1} = H_{2,n_1}$.

Hybrid $H^r_{4,i}$, for $i = 1, \ldots, t$. In this hybrid we wish to change the bits used in the `circuit-OT` to be all random. The difference between $H^r_{4,i}$ and $H^r_{4,i+1}$ is that in $H^r_{4,i}$ $\mathcal{S}_2$ plays $\mathsf{oneside\text{-}COT}_1(0)$, while in hybrid $H^r_{4,i+1}$, $\mathcal{S}_2$ plays with $\mathsf{oneside\text{-}COT}_1(c_i)$ with a random bit c_i. A distinguisher between hybrids $H^r_{4,i}$ and $H^r_{34,i+1}$ can be reduced to a malicious sender S^*_{OT} for `circuit-OT`, as before.

Note that hybrid $H^r_{4,t}$ corresponds to the description of the simulator $\mathcal{S}_2$, and this completes the proof.

□

Acknowledgment

We thanks Vassilis Zikas for pointing out ways to improve the presentation of our model.

The first author is supported by the Check Point Institute for Information Security, the NSF SaTC MACS project, and NSF Algorithmic Foundations grant no. 1218461 The third author is supported in part by NSF grants 09165174, 1065276, 1118126 and 1136174, US-Israel BSF grant 2008411, OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award, and Lockheed-Martin Corporation Research Award. This material is based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014 -11 -1-0392. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

[1] A. Afshar, P. Mohassel, B. Pinkas, and B. Riva. Non-interactive secure computation based on cut-and-choose. In *EUROCRYPT*, pages 387–404, 2014.

[2] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

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

[4] R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. In *TCC*, pages 61–85, 2007.

[5] R. Canetti and M. Fischlin. Universally composable commitments. In *CRYPTO*, pages 19–40, 2001.

[6] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited (preliminary version). In *STOC*, 1998.

[7] R. Canetti, E. Kushilevitz, and Y. Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In *EUROCRYPT*, pages 68–86, 2003.

[8] R. Canetti and T. Rabin. Universal composition with joint state. In *CRYPTO*, pages 265–281, 2003.

[9] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *CRYPTO*, pages 174–187, 1994.

[10] M. Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *CRYPTO*, pages 152–168, 2005.

[11] S. Goldwasser and Y. T. Kalai. On the (in)security of the fiat-shamir paradigm. In *FOCS*, pages 102–113, 2003.

[12] C. Hazay and Y. Lindell. *Efficient Secure Two-Party Protocols - Techniques and Constructions*. Information Security and Cryptography. Springer, 2010.

[13] Y. Huang, J. Katz, and D. Evans. Efficient secure two-party computation using symmetric cut-and-choose. In *CRYPTO (2)*, pages 18–35, 2013.

[14] Y. Ishai, E. Kushilevitz, R. Ostrovsky, M. Prabhakaran, and A. Sahai. Efficient non-interactive secure computation. In *EUROCRYPT*, pages 406–425, 2011.

[15] M. Kiraz and B. Schoenmakers. A protocol issue for the malicious case of yao's garbled circuit construction. In *Symposium on Information Theory in the Benelux*, pages 283–290, 2006.

[16] Y. Lindell. Highly-efficient universally-composable commitments based on the ddh assumption. In *EUROCRYPT*, pages 446–466, 2011.

[17] Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *CRYPTO (2)*, pages 1–17, 2013.

[18] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, pages 52–78, 2007.

[19] Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In *TCC*, pages 329–346, 2011.

[20] P. Mohassel and M. K. Franklin. Efficiency tradeoffs for malicious two-party computation. In *Public Key Cryptography*, pages 458–473, 2006.

[21] P. Mohassel and B. Riva. Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In *CRYPTO (2)*, pages 36–53, 2013.

[22] J. B. Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *CRYPTO*, 2002.

[23] R. Ostrovsky, A. Scafuro, I. Visconti, and A. Wadia. Universally composable secure computation with (malicious) physically uncloneable functions. In *EUROCRYPT*, pages 702–718, 2013.

[24] R. Pass. On deniability in the common reference string and random oracle model. In *CRYPTO*, pages 316–337, 2003.

[25] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.

[26] C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, pages 554–571, 2008.

[27] C.-P. Schnorr. Efficient Signature Generation for Smart Cards. *Journal of Cryptology*, 4(3):239–252, 1991.

[28] A. Shelat and C.-H. Shen. Two-output secure computation with malicious adversaries. In *EUROCRYPT*, pages 386–405, 2011.

[29] A. Shelat and C.-H. Shen. Fast two-party secure computation with minimal assumptions. In *ACM Conference on Computer and Communications Security*, pages 523–534, 2013.

[30] A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

A Zero Knowledge Functionality in the gRO model

The ideal functionality $\mathcal{G}_{\mathsf{gRO}}$ allows us to capture a more nuanced property: "Zero Knowledge up to transferability". This is done via a relaxed variant of the ideal zero knowledge functionality, $\mathcal{F}_{\mathsf{tZK}}$ where tzk stands for transferable zero knowledge, which leaks to the simulator the values of the illegitimate queries made with the present SID. Intuitively, the leakage of the adversarial queries made by third parties does not compromise zero knowledge beyond allowing the ability to transfer proofs. Indeed, since any query made by a third party could have been made by the adversary itself, any adversary that uses $\mathcal{G}_{\mathsf{gRO}}$'s answer to queries made by third parties can be simulated by an adversary that makes the same queries by itself. We describe the $\mathcal{F}_{\mathsf{tZK}}$ functionality in Fig. 8.

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

$\mathcal{F}_{\mathsf{tZK}}$ is parameterized by a relation $\mathcal{R} \subset \{0,1\}^* \times \{0,1\}^*$. We require that there exists a polynomial p such that $(x,w) \in \mathcal{R}$ implies that $|w| \leq p(|x|)$ and such that $(x,w) \in \mathcal{R}$ can be checked in time $p(|x|)$. $\mathcal{F}_{\mathsf{tZK}}$ runs with a prover P, a verifier V and an adversary S and proceeds as follows.

- Upon receiving the first input $(\texttt{ZK-prover}, \mathsf{sid}, P, V, x, w)$ from P, do: If $(x,w) \in R$, then send $(\texttt{ZK-proof}, \mathsf{sid}, P, V, x)$ to V and S.
- When asked by the adversary, obtain from $\mathcal{G}_{\mathsf{gRO}}$ the list $\mathcal{Q}_{\mathsf{sid}}$ of illegitimate queries that pertain to SID sid, and send $\mathcal{Q}_{\mathsf{sid}}$ to S.

Figure 8: The Zero-Knowledge functionality $\mathcal{F}_{\mathsf{tZK}}$

B NISC protocol [1]

For completeness we review the ideas behind the NISC protocol of [1].

The starting point of [1] is the highly efficient 2PC protocol of Lindell [17] which is based on cut-and-choose. The cut-and-choose is used to force the sender P_2 to send correct and consistent garbled circuits.

In the cut-and-choose, P_2 sends $3t$ garbled circuits. P_1 chooses a subset of such circuits and requires to see the randomness used to compute them. In this way it can check that the circuits are correctly computed. If they all pass the consistency check, then P_1 is convinced that *most* of the

remaining (not checked) circuits are correct as well. P_1 then evaluates the remaining circuits and takes the output on which the majority of the circuits agree upon. To implement this approach one has to solve the following issues:

1. How to enforce that the garbled keys transferred to P_1 via OT are consistent with the garbled circuits?

   This issue is solved by using committing OT in place of standard OT. In the multi server setting, committing OT works as long as the output of the computation is not published. Once the output is published then P_1 needs to refresh the first message (specifically the first message for the `circuit OT`).

2. How to enforce that P_2 uses the same inputs in all the remaining garbled circuits? Indeed, P_1 obtains a set of garbled keys (called labels) for each of the remaining garbled circuits. However, it has no guarantee that each set correspond to the same input.

   This issues is solved in [1] by requiring P_2 to send a commitment of its input, bit by bit, and then attaching to each garble keys (input label), a proof that the garbled keys corresponds to the bit committed. We explain this step in more details below.

3. How to reduce the total number of circuits from $3t$ to t? When the remaining circuits *all* output the same value, then P_1 is guaranteed that the output is correct with probability $1 - 2^{-t}$. However, when the circuits output different values it is necessary to provide a mechanism that allows P_1 to obtain the correct answer (with only t circuits the majority rule does not give correctness with probability $1 - 2^{-t}$). The mechanism provided in [17] requires several rounds of communication and therefore cannot be applied in the NISC setting where P_1 sends and receives only one message.

   The idea in [1] is to ask P_2 to commit to its input and to a "trapdoor" that, if revealed, allows to compute that decommitment. The trapdoor can be reconstructed by P_1 anytime it obtains two different results as the output of the garbled circuits. We explain this step in more details below.

[1] overcomes the above issues by crucially exploiting the specific properties of PVW OT and El-Gamal commitments. In this sense [1] is heavily non-black-box. We now describe in details the techniques used in [1].

Consistency of the labels exchanged via OT. The goal is to force P_2 to send consistent labels in the OTs for P_1's input wire. This is achieved adopting a committing OT (PVW OT satisfies this property) and by requiring that the randomness used in the OTs for circuit i, is derived from the PRF run on input $seed_i$. In this way when getting the seed $seed_i$, P_1 is able to check both the correctness of the circuit and the correctness of the OT.

Consistency of P_2's input in all the remaining circuits. For the circuits that will not be checked, P_1 obtains a set of garbled keys representing the input of P_2. However, how to check that such garbled keys all represent the *same* input? This is done as follows.

- The input labels of the garbled circuit are *commitment* of bits. More precisely, let i be the index of a circuit, let j be the input wire. First compute El-Gamal commitments for bit 0 and bit 1: $u_{i,j,b} \leftarrow \mathsf{EGCom}(h; b, r_{i,j,b})$, for $b \in \{0,1\}$ and where $\mathsf{EGCom}(h; b, r_{i,j,b}) = g^{r_{i,j,b}}, h^{r_{i,j,b}} g^b$ and the randomness $r_{i,j,b}$ is derived from PRF and $seed_i$. Then, the label is computed as: $\mathsf{label}(gc_i, j, b) = \mathsf{ReHash}(\mathsf{id}_2, u_{i,j,b})$, where id_2 is the identity of P_2.

- Each bit of the input of P_2 is committed using El-Gamal commitment. Namely, for each bit j, P_2 sends commitment $\mathsf{EGCom}(h; y_j, r_j)$.
- For each label P_2 sends a proof of equality with the input committed. Proving equality of two El-Gamal commitment, without revealing anything about the values committed, is very simple. It is sufficient to reveal the difference between the randomness used to compute the two commitments. Namely, let $c_0, c_1 = \mathsf{EGCom}(h, m, r) = g^r, h^r g^m$ and $c'_0, c'_1 = \mathsf{EGCom}(h, m, r') = g^{r'}, h^{r'} g^m$, one can verify that $(c'_0)^{r-r'} = c_0$ and $(c'_1)^{r-r'} = c_1$.

  Therefore, P_2 computes and send $\mathsf{inputsEq}_i = \{r_j - r_{i,j,y_j}\}$ for each circuit gc_i and input wire j.

Reducing the number of garbled circuits. When the remaining circuits do not output the same value, we need a mechanism of cheating recovering that enables P_1 to compute the correct output. This is implemented using a special extractability property of El-Gamal commitment Indeed, El-Gamal commitment is extractable given a trapdoor: let $h = g^w$ and let $c_0 = g^r, c_1 = h^r g^b$ an El-Gamal commitment. Knowledge of the trapdoor w allows the extraction of the bit b by checking if $c_1/(c_0)^w = g$ or $c_1/(c_0)^w = 1$. Using this property, the cheating recovery mechanism is implemented as follows.

- P_2 commits to shares of the trapdoor. Namely, let $h = g^w$ be the parameter used for all the El-Gamal commitments send by the P_2 (in particular recall that the input of P_2 is committed using h). P_2 computes w_0, w_1 s.t. $w_0 + w_1 = 1$ and sends $h_0 = g^{w_0}$ and $h_1 = g^{w_1}$ to P_1 who checks that $h = h_0 \cdot h_1$.
- The output labels are connected to the trapdoor. Let $l_{i,0}$ and $l_{i,1}$ be the output labels of the i-th garbled circuit gc_i (for simplicity assume that the output of the circuit is one bit only).

  P_2 prepares:

  - Encryption of each trapdoor under each labels ($\mathsf{outputsDecom}_i$): For each circuit i, P_2 sends $K_{i,0} + w_0$ and $K_{i,1} + w_1$, for randomly chosen $K_{i,0}, K_{i,1}$. (These values are revealed only for the circuits that will be evaluated.) The keys $K_{i,0}, K_{i,1}$ are then encrypted under the labels $l_{i,0}$ and $l_{i,1}$: P_2 sends $\mathsf{Enc}(l_{i,0}, K_{i,0})$ and $\mathsf{Enc}(l_{i,0}, K_{i,0})$.
  - Commitment of the labels (`output recovery commitments`): $C_0 = h_0 g^{K_{i,0}}$ and $C_1 = h_1 g^{K_{i,1}}$. This is a Pedersen commitment and is therefore not binding for P_2 which knows the trapdoors w_0, w_1. However, it binds P_2 to $w_b + l_{i,b}$ for $b \in \{0, 1\}$. P_1 checks the consistency of $K_{i,0} + w_0$ and $K_{i,1} + w_1$ by checking that $C_b = g^{K_{i,b} + w_b}$. If this last check goes through, then P_1 is guaranteed that the labels of the circuit gc_i are indeed connected to the trapdoors w_0, w_1, and thus the circuit is marked as *semi-trusted*. P_1 continues her executions only if all circuits are semi-trusted.
- **Two different outputs enable P_1 to reconstruct the trapdoor.** Let gc_i and gc_k two garbled circuits that output two different values. Because they are both *semi-trusted*, this means that gc_i outputs label $l_{i,0}$ that enables P_1 to decrypt $K_{i,0}$ and therefore w_0 and gc_k outputs $l_{k,1}$ enabling P_1 to decrypt $K_{k,1}$ and therefore w_1. P_1 then obtains the trapdoor $w = w_0 + w_1$ necessary to decrypt the commitments of the input of P_2 and computes the value $f(x, y)$.