Endemic Oblivious Transfer via Random Oracles, Revisited[*]

Zhelei Zhou[†] Bingsheng Zhang[‡] Hong-Sheng Zhou[§] Kui Ren[¶]

March 11, 2023

Abstract

The notion of Endemic Oblivious Transfer (EOT) was introduced by Masny and Rindal (CCS'19). EOT offers a weaker security guarantee than the conventional random OT; namely, the malicious parties can fix their outputs arbitrarily. The authors presented a 1-round UC-secure EOT protocol under a tailor-made and non-standard assumption, Choose-and-Open DDH, in the RO model.

In this work, we systematically study EOT in the UC/GUC framework. We present a new 1-round UC-secure EOT construction in the RO model under the DDH assumption. Under the GUC framework, we propose the first 1-round EOT construction under the CDH assumption in the Global Restricted Observable RO (GroRO) model proposed by Canetti *et al.* (CCS'14). We also provide an impossibility result, showing there exist *no* 1-round GUC-secure EOT protocols in the Global Restricted Programmable RO (GrpRO) model proposed by Camenisch *et al.* (Eurocrypt'18). Subsequently, we provide the first round-optimal (2-round) EOT protocol with adaptive security under the DDH assumption in the GrpRO model. Finally, we investigate the relations between EOT and other cryptographic primitives.

As side products, we present the first 2-round GUC-secure commitment in the GroRO model as well as a separation between the GroRO and the GrpRO models, which may be of independent interest.

[*]Corresponding authors: Bingsheng Zhang *bingsheng@zju.edu.cn*, and Hong-Sheng Zhou *hszhou@vcu.edu*.

[†]Zhejiang University, and ZJU-Hangzhou Global Scientific and Technological Innovation Center.

[‡]Zhejiang University, and ZJU-Hangzhou Global Scientific and Technological Innovation Center. Work supported by the National Key R&D Program of China (No. 2021YFB3101601), the National Natural Science Foundation of China (Grant No. 62072401), and "Open Project Program of Key Laboratory of Blockchain and Cyberspace Governance of Zhejiang Province". This project is also supported by Input Output (iohk.io).

[§]Virginia Commonwealth University. Work supported in part by NSF grant CNS-1801470, and a Google Faculty Research Award.

[¶]Zhejiang University, and ZJU-Hangzhou Global Scientific and Technological Innovation Center.

Contents

1 Introduction

The security of a cryptographic protocol is typically analyzed under the simulation paradigm [GMW87], where the "formal specification" of the security requirements is modeled as an ideal process, and a real-world protocol is said to securely realize the specification if it "emulates" the ideal process. In the past decades, many variants were proposed: Initially, protocol security was considered in the *standalone* setting, in the sense that the challenged protocol is executed in isolation. Later, *Universal Composibility* (UC) [Can01] was introduced to analyze protocol security in arbitrary execution environments; in particular, multiple protocol sessions may be executed concurrently in an adversarially coordinated way. Note that protocols in the UC framework must be *subroutine respecting*, in the sense that all the underlying subroutines are only created for the challenged protocol instance and cannot be directly accessed by any other protocols or even the other instances of the same protocol. To address this drawback, Canetti *et al.* [CDPW07] proposed the Generalized Universal Composibility (GUC) framework.

Endemic Oblivious Transfer. The notion of *Endemic Oblivious Transfer* (EOT) was introduced by Masny and Rindal [MR19a] as a weaker version of Random OT (ROT). In an EOT protocol, the sender has no input, and the receiver inputs a choice bit $b \in \{0,1\}$; at the end of EOT, the sender outputs two random elements (m_0, m_1), and the receiver outputs m_b. Although EOT looks similar to the conventional ROT, EOT offers a weaker security guarantee — the malicious sender can fix its output (m_0, m_1) arbitrarily, and the malicious receiver can fix its output m_b arbitrarily. The first 1-round[1] (a.k.a. non-interactive) EOT/ROT protocol was proposed by Bellare and Micali [BM90]. It achieves standalone security against semi-honest adversaries under the DDH assumption in the Common Reference String (CRS) model. As shown in [GS17], this scheme can also be transformed to achieve malicious security using the Groth-Sahai proof [GS08]. Later, Garg *et al.* proposed several 1-round UC-secure EOT protocols under the well-understood assumptions, (e.g., Decisional Diffie-Hellman (DDH), Quadratic Residuosity (QR) and Learning With Errors (LWE)), in the CRS model [GIS18a]. Recently, Masny and Rindal [MR19a] demonstrated a generic construction for 1-round EOT by using any non-interactive key exchange scheme in the Random Oracle (RO) model; however, their generic construction only achieves standalone security. Masny and Rindal [MR19a, MR19b] then provided a 1-round *UC-secure* EOT protocol but under a tailor-made computational assumption called "*Choose-and-Open DDH* (CODDH)", in the RO model. We remark that, different from the DDH, the CODDH is a new assumption, and its hardness is yet to be further studied.

(Global) Random Oracle Models. Random oracle (RO) model [BR93] is a popular idealized setup model that has been widely used to justify the security of efficient cryptographic protocols. In spite of its known inability to provide provable guarantees when RO is instantiated with a real-world hash function [CGH98], RO is still a promising setup since it is generally accepted that security analysis in the RO model does provide strong evidences to the resilience of the protocol in question in the presence of practical attacks [CJS14]. In fact, RO model draws increasing attention along with recent advancement of the blockchain technology.

Local RO Model vs. Global RO Models. The "local" RO model is often used in the UC framework where the simulator is allowed to simulate it in the ideal world, and it grants the simulator two advantages: (i) observability: the simulator can see what values the parties query the RO on; (ii) programability: the simulator can program RO query responses as long as they "look" indistinguishable from the real ones. In the GUC framework [CDPW07], a "global" RO is external to the simulator; to facilitate simulation, some "extra power" needs to be granted to the simulator. In the literature, two main strengthened variants of the global RO model were proposed: global RO with restricted observability (GroRO) model proposed by Canetti *et al.* [CJS14] and global RO with restricted programmability (GrpRO) model proposed by Camenisch *et al.* [CDG+18]. Here, the restricted observability and programmability stand for the "extra power" that the simulator has but the adversary does not have.

1.1 Problem Statement

Constructing EOT in (Global) RO Models. As mentioned above, it is known that one can build a 1-round UC-secure EOT protocol under the well-known assumptions in the local CRS model [GIS18a]; however, in the local RO model, the recent construction by Masny and Rindal [MR19a, MR19b] was based on a *non-standard* assumption

[1]In this work, we consider the simultaneous communication model with a rushing adversary, where both parties can send messages to each other within the same round. The rushing adversary can delay sending messages on behalf of corrupted parties in a given round until the messages sent by all the uncorrupted parties in that round have been received. Note that this is different from the simultaneous messaging requirement in [Kat07], which deals with a non-rushing adversary.

i.e., the CODDH assumption. A natural question to ask is: can we construct a 1-round UC-secure EOT protocol under well-understood assumptions (e.g., DDH assumption) in the local RO model?

Compared to local setups (e.g., local CRS and local RO), global setups are more practical in real life applications. However, very little research work has been done for constructing EOT protocols under a global setup. Our main goal here is to construct a 1-round EOT protocol using global setups. We emphasize that local setups are helpful for us to construct a provably secure 1-round EOT protocol. For example, in the local CRS model, both parties can utilize the shared string, i.e., the CRS, to generate the correlated information for the remaining protocol execution. In other words, *the CRS can be viewed as an extra round of communication messages* during the protocol execution. Intuitively, the security analysis can go through: the simulator is allowed to generate the CRS along with the trapdoor; then the trapdoor information will help the simulator to complete the simulation. In the local RO model, the situation is similar: in the protocol execution, the protocol players may query the RO at certain predefined points to obtain corresponding responses; in a very fuzzy way, it also can be viewed as an extra round of communication messages. In the security analysis, the simulator is allowed to program the RO on those predefined points; this gives the simulator advantages over the adversary which will help the simulator to complete the simulation.

The situation is very different when we use a **global** setup for constructing 1-round EOT protocols. First, we remark that, as already proven in [CDPW07], it is impossible to construct a non-trivial two-party computation protocols (including EOT) using a global CRS. To bypass this impossibility, Canetti *et al.* proposed the Augmented CRS (ACRS) model [CDPW07]; however, known technique of building non-trivial two-party computation protocols in the ACRS model requires coin-flipping [CDPW07, DSW08], which increases round complexity. The good news is that it might be possible to construct a 1-round EOT protocol using a global RO model; note that, different global RO models (e.g., the GroRO [CJS14] and the GrpRO [CDG+18]) have been introduced for constructing non-trivial two-party computation protocols. We must remark that, technical difficulty remains. Typically, a global RO is instantiated with a predefined hash function. It seems that the aforementioned design and analysis ideas using local ROs still work: both parties may still be able to utilize the shared hash function on some predefined points to generate the corresponding responses for the remaining protocol execution; unfortunately, it is not true. Below, we provide our elaboration: (1) in the GroRO model, the simulator is not allowed to program the global RO and thus cannot obtain the "trapdoor" of the corresponding responses; as a result, it is unclear how we will be able to complete the security analysis; (2) in the GrpRO model, the simulator is only allowed to program the unqueried points, and the simulator may not be able to program the global RO on those predefined points since the environment may have already queried them before the protocol execution. Given the technical difficulty, we ask the following major research question:

> *In the GUC framework, does there exist a 1-round EOT protocol under well-understood assumptions in the GroRO/GrpRO model?*

For completeness, we also construct new 1-round UC-secure EOT protocols in the local RO model.

Understanding the Complexity of EOT. n addition to the concrete protocol constructions, we are also interested in understanding the complexity, including the power and the limits, of the cryptographic task of EOT. More precisely, what are the relations between EOT and other well-known secure computation tasks? For example, is EOT fundamentally different from ROT or (1-out-of-2) OT? In [MR19a], Masny and Rindal have already initialized the investigation of this interesting problem: They proposed a new OT notion called Uniform OT (UOT) which also looks similar to the conventional ROT, except that it offers a strong security guarantee that no adversary can bias the distribution of the ROT outputs. They showed that it is possible to build UOT based on an EOT and a coin-tossing protocol; however, it is unclear if the coin-tossing protocol can be built from an EOT protocol. We thus ask the following question:

> *What is the relation between the EOT and other cryptographic primitives (such as coin-tossing and UOT etc.)?*

Understanding the Complexity of Global RO Models. Finally, let us go back to the global setups we used in this work. Recall that, the GroRO and the GrpRO models provide different aspects of "extra power" to the simulator. Are these two different global RO models, essentially equivalent? Or one is strictly stronger than the other? It raises our last question:

> *What is the relation between the GroRO model and the GrpRO model?*

Our goal is to provide a comprehensive and thorough investigation of constructing EOT via ROs. From a practical point of view, if the above questions could be answered, we would see highly efficient constructions for EOT. From a theoretical point of view, if (some of) the above questions could be answered, we would have a better understanding of the relation between EOT and many secure computation tasks; we could also have a better understanding of the power and limits of different global RO models.

1.2 Our Results

In this work, we investigate the above problems. Our results can be summarized as follows.

1.2.1 Constructing EOT in the (Global) Random Oracles

Table 1 depicts a selection of our new constructions.

Protocol	#Round	Security	Computational Assumption	Setup Assumption
Garg *et al.* [GIS18a, GIS18b][a]	1	UC+Static	DDH	CRS
Masny and Rindal [MR19a, MR19b]	1	UC+Static	CODDH[b]	RO
Canetti *et al.* [CSW20]	1	UC+Adaptive	DDH	GrpRO+CRS
$\Pi_{\text{EOT-RO}}$(Sec. 3)	1	UC+Static	DDH	RO
$\Pi_{\text{EOT-GroRO}}$(Sec. 5.1.1)[c]	1	GUC+Static	CDH	GroRO
$\Pi_{\text{EOT-GrpRO}}$(Sec. 5.2.2)[d]	2	GUC+Adaptive	DDH	GrpRO

[a] Garg et al's constructions can be instantiated from different assumptions (e.g., DDH, LWE and QR); but in this table, we focus on constructions using (cyclic) group based assumptions.
[b] Here, CODDH refers to the "Choose-and-Open DDH" assumption which is not known to be reducible to the DDH assumption.
[c] Although protocol $\Pi_{\text{EOT-GroRO}}$ uses a weaker computational assumption and a less idealized setup than protocol $\Pi_{\text{EOT-RO}}$ does, the former is less efficient than the latter.
[d] This construction is round-optimal due to Theorem 5, below.

Table 1: Comparison with state-of-the-art round-optimal EOT protocols under computational assumptions that related to the cyclic groups.

Next, we provide the technical overview for our EOT protocol constructions in the (global) RO models. We first show how to construct a 1-round UC-secure EOT protocol under DDH assumption in the RO model against static adversaries. After that, we turn to the global RO models and show how to construct a 1-round GUC-secure EOT protocol under CDH assumption in the GroRO model. Note that, the situation in the GrpRO model is complicated: We find that there exists no 1-round GUC-secure EOT protocols in the GrpRO model even with static security, and we give a round-optimal (2-round) EOT protocol under DDH assumption against adaptive adversaries.

New technique: 1-round UC-secure EOT protocol in the RO model. We present a new technique that enables the first UC-secure 1-round EOT protocol in the RO model under the DDH assumption (cf. Section 3). The basic scheme achieves static security. Intuitively, our technique is as follows. We start with the two-round standalone ROT/EOT protocol in the RO model proposed in [CO15]. In the 1st round, the sender sends $h := g^s$ to the receiver; in the 2nd round, the receiver uses sender's message to compute $B := g^r h^b$ and sends B back, where $b \in \{0, 1\}$ is the choice bit; finally, the sender outputs $m_0 := \mathtt{Hash}(B^s)$ and $m_1 := \mathtt{Hash}((\frac{B}{h})^s)$, where $\mathtt{Hash}$ is a predefined hash function and it is modeled as a RO; the receiver outputs $m_b := \mathtt{Hash}(h^r)$. Although this protocol is simple and efficient, it cannot achieve UC security [LM18, GIR20].

Our technique is presented as follows. The dependence of the sender's message in [CO15] can be eliminated such that the receiver's message can be produced simultaneously in the same round. The idea is to let the receiver produce the commitment key h instead of waiting it from the sender. How to generate a random group element and be oblivious to its discrete logarithm? This can be achieved by setting $h := \mathtt{Hash}(\mathsf{seed})$, where seed is some randomly sampled string. Similar technique can be found in [CSW20]. Now the 1-round (non-interactive) version of [CO15] roughly works as follows. The sender sends $z := g^s$ to the receiver; meanwhile, the receiver picks $h := \mathtt{Hash}(\mathsf{seed})$ and computes $B := g^r h^b$, and then it sends (seed, B) to the sender; finally, the sender computes $h := \mathtt{Hash}(\mathsf{seed})$ and outputs $m_0 := \mathtt{Hash}(B^s)$ and $m_1 := \mathtt{Hash}((\frac{B}{h})^s)$; the receiver outputs $m_b := \mathtt{Hash}(z^r)$.

Further, to make the protocol UC-secure, certain *extractability* is needed: (i) when the sender is malicious, the simulator should be able to extract the sender's private randomness s, so the simulator can compute both m_0

and m_1; (ii) when the receiver is malicious, the simulator should be able to extract the receiver's choice bit. In order to extract the sender's s, we let the sender additionally generate a RO-based straight-line extractable NIZK argument [Pas03, Fis05, Ks22]. In order to extract the receiver's choice bit b, we let the receiver computes the ElGamal encryption of b instead of the Pedersen commitment. We then let the receiver additionally generate a NIZK argument to ensure the correctness of the ElGamal encryption. Note that, we do not need the straight-line extractability here, since the simulator can program the RO to obtain $\log_g h$ and thus be able to decrypt the ElGamal ciphertext to extract b.

1-round GUC-secure EOT protocol in the GroRO model. Turning to the GUC setting, we propose the first 1-round EOT construction under the CDH assumption in the GroRO model (cf. Section 5.1.1). Compared to our UC-secure construction, this one requires *weaker* a computational assumption.

Recall that, in our UC-secure EOT protocol, we let the sender send $z := g^s$ together with a straight-line extractable NIZK argument. The straight-line extractable NIZK argument gives the simulator the ability to extract s. However, Pass showed that it is impossible to construct NIZK arguments in observable RO model [Pas03], let alone NIZK arguments with straight-line extractability. The good news is that straight-line extractable NIWH argument is sufficient for our purpose, and it exists in the GroRO model [Pas03]. Therefore, we let the sender generate a straight-line extractable NIWH argument of s such that $z = g^s$ instead. Next, to extract the receiver's choice bit, our UC-secure construction utilizes the programmability of RO; however, $\mathcal{G}_{\mathsf{roRO}}$ does not offer programability, so a different approach shall be taken. In particular, we let the receiver compute a Pedersen commitment to the choice bit $B := g^r h^b$, and generate a straight-line extractable NIWH argument of (r, b) such that $B = g^r h^b$. Analogously to the sender side, the straight-line extractable NIWH argument gives the simulator extractability.

1.2.2 Understanding the Power/Limits of Different Global Random Oracles

A separation between the GroRO model and the GrpRO model. To show this separation, we first give a new impossibility result, showing that there exists *no* 1-round GUC-secure EOT protocol in the GrpRO model even with static security (cf. Section 5.2.1). By combining this negative result in the GrpRO model and the aforementioned positive result in the GroRO model, we demonstrate a separation between the GroRO model and the GrpRO model. More precisely, let $\mathcal{G}_{\mathsf{roRO}}, \mathcal{G}_{\mathsf{rpRO}}$ be the functionalities of the GroRO and the GrpRO model, we present the relation of these global RO models in Figure 1.

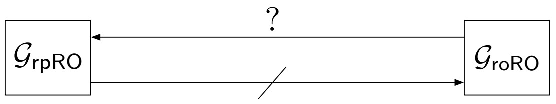

$$\begin{tikzcd} \boxed{\mathcal{G}_{\mathsf{rpRO}}} \arrow[r, shift right, "/" marking] & \boxed{\mathcal{G}_{\mathsf{roRO}}} \arrow[l, shift right, "?"'] \end{tikzcd}$$

Figure 1: The relation between the $\mathcal{G}_{\mathsf{roRO}}$ model and the $\mathcal{G}_{\mathsf{rpRO}}$ model. Here, "A $\nrightarrow$ B" denotes that A does not imply B. In addition, "A $\overset{?}{\to}$ B" denotes that whether A implies B remains unknown.

New impossibility and feasibility results in the GrpRO model. Here we will present more details about the aforementioned impossibility result in the GrpRO model. Furthermore, to complete the picture, we also provide a round-optimal GUC-secure EOT protocol with adaptive security in the GrpRO model.

New impossibility results in the GrpRO model. The impossibility is proven by contradiction (cf. Section 5.2.1). Suppose that there exists such a 1-round GUC-secure EOT protocol in the GrpRO model. Let us first consider the case where the receiver is corrupted, and the simulator needs to extract the choice bit of the receiver from its message. Recall that, the GrpRO only grants the simulator the restricted programmability: the simulator can program the unqueried points without being detected. More importantly, unlike local RO, the simulator cannot program a global RO on the fly, as it cannot see which point is queried at this moment. Thus, the simulator needs to find a way to enforce the corrupt receiver to query the simulator's programmed points. However, in a one simultaneous round protocol, the messages between parties have no dependency. Hence the simulator cannot enforce the corrupt receiver to produce its message on the programmed points, and has no advantages. If the simulator still succeeds to extract the corrupted receiver's choice bit, then we have the following attack. The adversary corrupts the sender, and instructs the sender to run the simulator algorithm above to extract the choice bit from the message sent by the receiver/simulator. However, the simulator has no idea about the real choice bit, thus with 1/2 probability the simulation would fail.

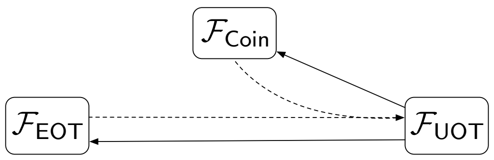
New feasibility: Round-optimal GUC-secure EOT protocol in the GrpRO model. To complete the picture, we also give a round-optimal (2-round) EOT protocol with adaptive security under the DDH assumption in the GrpRO model (cf. Section 5.2.2). Here, we do not consider simultaneous messaging in the same round. Our intuition comes from the UC-secure EOT protocol in the CRS+GrpRO model proposed by Canetti *et al.* [CSW20]. In their protocol, the CRS consists of two group elements $g, h \in \mathbb{G}$, and the simulator knows $\log_g h$. The sender computes $z := g^r h^s$, while the receiver generates $(G, H) := \texttt{Hash}(\text{seed})$ and computes two Pedersen commitments to the choice bit using two sets of the parameter, i.e., (g, G) and (h, H), and the same randomness.

To eliminate the CRS, we let the sender generate the first set of the parameter $(g, h) := \texttt{Hash}(\text{seed}_1)$ where seed_1 is an uniformly sampled string. At the same time, the sender computes $z := g^r h^s$ using random $r, s \leftarrow \mathbb{Z}_q$ and sends seed, z to the receiver in the first round. In the second round, the receiver first checks if seed_1 is a programmed point. If not, the receiver generates the second set of the parameter $(G, H) := \texttt{Hash}(\text{seed}_2)$ where seed_2 is an uniformly sampled string. Then the receiver can compute two Pedersen commitments to the choice bit, i.e., $(B_1, B_2) := (g^x G^b, h^x H^b)$ using random $x \leftarrow \mathbb{Z}_q$. Finally, we let the receiver send $(\text{seed}_2, B_1, B_2)$ to the sender. How to make the protocol simulatable in the GrpRO model? We show the simulation strategy as follows: when the receiver is malicious (and the sender is honest), the simulator can extract the receiver's choice bit b by programming the GrpRO (the simulator always succeeds to program the GrpRO since seed_1 is sampled by the honest sender itself) and knowing α such that $h = g^\alpha$; when the sender is malicious (and the receiver is honest), the simulator can compute both m_0 and m_1 by programming the GrpRO (the simulator always succeeds to program the GrpRO since seed_2 is sampled by the honest receiver itself) such that (g, h, G, H) is a DDH tuple.

1.2.3 Understanding the Relation between EOT and Other Cryptographic Primitives

EOT implies UOT and commitment. In [MR19a], the authors showed that UOT implies EOT. But the work on the opposite direction is incomplete. Let $\mathcal{F}_{\mathsf{EOT}}$, $\mathcal{F}_{\mathsf{UOT}}$ and $\mathcal{F}_{\mathsf{Coin}}$ be the ideal functionalities of EOT, UOT and coin-tossing protocol, respectively. They showed that a UOT protocol can be constructed in the $\{\mathcal{F}_{\mathsf{EOT}}, \mathcal{F}_{\mathsf{Coin}}\}$-hybrid world with unconditional security, and they constructed $\mathcal{F}_{\mathsf{Coin}}$ via only $\mathcal{F}_{\mathsf{UOT}}$. However, it remains unclear whether $\mathcal{F}_{\mathsf{Coin}}$ can be constructed via only $\mathcal{F}_{\mathsf{EOT}}$; therefore, it is still an open question on whether EOT implies UOT? We present the relations that they claimed in Figure 2(a).

It is known that bit commitment can be constructed via 1-out-of-2 OT with unconditional security [Kil88, BFSK11]. What about EOT? Nevertheless, surprisingly, we show that bit commitment can be constructed via a weaker primitive, i.e., EOT, with unconditional security (cf. Section 4.1). Our key observation is that the receiver's message can be viewed as the commitment to the choice bit b, and the locally computed message m_b together with b can be viewed as the opening. Typically, a commitment protocol requires both hiding and binding properties. The hiding property holds since the malicious receiver in the EOT cannot learn m_{1-b}, even if it can influence the distribution of m_b. The binding property holds since the malicious sender in the EOT cannot know which message is received by the receiver, even if it can influence the distributions of both m_0 and m_1.


$$\begin{tikzcd} & \mathcal{F}_{\mathsf{Coin}} \arrow[dr, dashed, bend right] & \\ \mathcal{F}_{\mathsf{EOT}} \arrow[rr, dashed] & & \mathcal{F}_{\mathsf{UOT}} \arrow[ul] \arrow[ll] \end{tikzcd}$$
(a) The relations claimed in [MR19a]

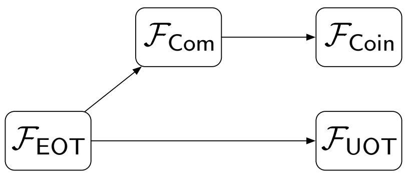

$$\begin{tikzcd} & \mathcal{F}_{\mathsf{Com}} \arrow[r] & \mathcal{F}_{\mathsf{Coin}} \\ \mathcal{F}_{\mathsf{EOT}} \arrow[ur] \arrow[rr] & & \mathcal{F}_{\mathsf{UOT}} \end{tikzcd}$$
(b) The relations in this work.

Figure 2: The relations between EOT and other primitives. "A $\rightarrow$ B" denotes that A implies B. "A $\dashrightarrow$ B" denotes that A can be transformed into B.

Since it is well-known how to construct $\mathcal{F}_{\mathsf{Coin}}$ via only $\mathcal{F}_{\mathsf{Com}}$, where $\mathcal{F}_{\mathsf{Com}}$ is the commitment functionality, we show that EOT implies UOT and completes the relation between EOT and UOT (cf. Section 4.2). We present the relations that explored in this work in Figure 2(b).

Furthermore, as a side product, we present the first 2-round GUC-secure commitment in the GroRO model (cf. Section 5.1.2), which may be of independent interest. The previous state-of-the-art protocols need 3 rounds [MRS17, ZZZR22]. Note that this result does not contradict Zhou *et al.*'s impossibility result [ZZZR22], as their work did not consider simultaneous communication model.

1.3 Related Work

We first review the EOT (and 1-out-of-2 OT) protocols in the CRS model. For simplicity, we often use OT to refer to the 1-out-of-2 OT in this work. Bellare and Micali gave an efficient and 1-round (standalone) EOT protocol under DDH assumption in the CRS model against semi-honest adversaries [BM90]. Later, Garg and Srinivasan compiled Bellare and Micali's protocol to against malicious adversaries [GS17] by additionally utilizing Groth-Sahai proofs [GS08]. As for the UC security, Canetti *et al.* proposed the first UC-secure OT protocol [CLOS02], but their protocol is quite inefficient. Next, Peikert *et al.* provided a dual-mode framework for constructing UC-secure OT protocols along with efficient instantiations under DDH, QR and LWE assumptions in the CRS model [PVW08]. Basing on Peikert *et al*'s dual mode framework, Garg *et al.* proposed the 1-round UC-secure EOT protocols under DDH, QR and LWE assumptions in the CRS model [GIS18a].

We mainly focus on the EOT (and OT) protocols in the different variants of RO models, i.e. the local RO model, the GroRO model and the GrpRO model.

In terms of the local RO model, Chou and Orlandi proposed a 3-round OT protocol called "the simplest OT protocol" [CO15]. This protocol and the protocol proposed in [HL17] have been found to suffer from a number of issues [BPRS17, LM18, GIR20] and are not UC-secure. In the following, Masny and Rindal showed how to construct EOT protocols from the key exchange schemes in the local RO model [MR19a]. In particular, they provided a 1-round UC-secure construction under a non-standard assumption, i.e., Choose-and-Open DDH (CODDH) assumption [MR19a, MR19b].

Regarding the GroRO model, Canetti *et al.* proposed a 2-round OT protocol under DDH assumption [CJS14], but their protocol is only one-sided GUC-simulatable. Later, fully GUC-secure OT protocols in the GroRO model are proposed [DKLs18, DD20]. Their protocols only need CDH assumption but require no less than 5 rounds of communication. To achieve round-optimal, Canetti *et al.* proposed a 2-round GUC-secure OT protocol in the GroRO model [CSW20], but their protocol requires a stronger assumption, i.e., DDH assumption.

As for the GrpRO model, Canetti *et al.* proposed an adaptive-secure 1-round EOT protocol in the GrpRO+CRS hybrid model [CSW20], but their protocol is only UC-secure since their simulator must know the trapdoor of the CRS. The lower bounds on round complexity in the GrpRO model have been explored by Zhou *et al.* [ZZZR22], however, they mainly focused on commitments and ZK protocols and they did not consider the simultaneous communication model.

1.4 Organization

In Section 2, we present the preliminaries that will be used in this work. We propose the first 1-round UC-secure EOT protocol in the RO model against static adversaries in Section 3. We then investigate the relations between the EOT and other primitives in Section 4, showing EOT implies commitment and UOT in Section 4.1 and Section 4.2 respectively.

We also explore the feasibility and the impossibility for EOT in the global random oracle models. As for the GroRO model, we propose the first 1-round GUC-secure EOT protocol in Section 5.1.1. Basing on that, we propose the first 2-round GUC-secure commitment protocol in the GroRO model in Section 5.1.2. Regarding the GrpRO model, we show that it is impossible to construct a 1-round GUC-secure EOT protocol even with static security in Section 5.2.1 and subsequently propose the 2-round (round-optimal) GUC-secure EOT protocol in Section 5.2.2.

We put the formal description of Pedersen commitment and ElGamal encryption in Appendix A and put the the building blocks of (straight-line extractable) NIZK/NIWH arguments in Appendix B. Finally, we put the additional security proofs in Appendix C.

2 Preliminaries

2.1 Notations

We denote by $\lambda \in \mathbb{N}$ the security parameter. We say that a function $\mathsf{negl} : \mathbb{N} \to \mathbb{N}$ is negligible if for every positive polynomial $\mathrm{poly}(\cdot)$ and all sufficiently large λ, it holds that $\mathsf{negl}(\lambda) < \frac{1}{\mathrm{poly}(\lambda)}$. We use the abbreviation PPT to denote probabilistic polynomial-time. For an NP relation $\mathcal{R}$, we denote by $\mathcal{L}$ its associate language, i.e. $\mathcal{L} = \{x \mid \exists w \text{ s.t. } (x, w) \in \mathcal{R}\}$. We say that two distribution ensembles $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ are identical (resp. computationally indistinguishable), which we denote by $\mathcal{X} \equiv \mathcal{Y}$ (resp., $\mathcal{X} \overset{c}{\approx} \mathcal{Y}$), if for any unbounded (resp., PPT) distinguisher $\mathcal{D}$ there exists a negligible function negl such that $|\Pr[\mathcal{D}(\mathcal{X}_\lambda) = 1] - \Pr[\mathcal{D}(\mathcal{Y}_\lambda) = 1]| = 0$ (resp., $\mathsf{negl}(\lambda)$).

We denote by $y := \mathsf{Alg}(x; r)$ the event where the algorithm Alg on input x and randomness r, outputs y. We denote by $y \leftarrow \mathsf{Alg}(x)$ the event where Alg selects a randomness r and sets $y := \mathsf{Alg}(x; r)$. We denote by $y \leftarrow S$ the process for sampling y uniformly at random from the set S. Let q be a λ-bit prime, and $p = 2q + 1$ also be a prime. Let $\mathbb{G}$ be a subgroup of order q of $\mathbb{Z}_p^*$ with the generator g.

2.2 Universal Composability

We formalize and analyze the security of our protocols in the Canetti's Universal Composability (UC) framework [Can01] and Canetti *et al*'s Generalized UC (GUC) framework [CDPW07].

UC Framework. The UC framework was proposed by Canetti [Can01], and it lays down a solid foundation for designing the protocols and analyzing the security against attacks in an arbitrary and complex network execution environment.

In the UC framework, a protocol Π is defined to be a computer program (or several programs) which is intended to be executed by multiple interconnected parties. Furthermore, we call a protocol, the one for which we want to prove security, the *challenged* protocol. We only consider the *terminating* protocols in this work which terminates in polynomial time. Each party is identified by a unique identity pair $(\mathsf{pid}, \mathsf{sid})$, where pid is the Party ID (PID) and sid is the Session ID (SID). Parties running with the same code and the same SID are said to be a part of the same protocol session, and the PIDs are used to distinguish among the various parties in a particular protocol session. Typically, all SIDs are unique, i.e. no two protocol sessions share the same SID. The adversarial behaviors in the protocol are captured by a adversary $\mathcal{A}$ who can control the network and corrupt the parties. When a party is corrupted, the adversary $\mathcal{A}$ receives its secret input and its internal state.

The UC framework is based on the "simulation paradigm" (a.k.a., the ideal/real world paradigm). In the ideal world, there is an ideal functionality $\mathcal{F}$ communicating a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{S}$ (a.k.a., the simulator) and computing a task in a trusted manner. The corrupted parties are controlled by the simulator $\mathcal{S}$. In the real world, a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ communicate with each other and the adversary $\mathcal{A}$ to execute the protocol Π. The corrupted parties are controlled by the adversary $\mathcal{A}$.

The presence of arbitrary protocols running in the network is modeled via the concept of the environment $\mathcal{Z}$, which determines the inputs to parties and see the outputs generated by those parties. The environment $\mathcal{Z}$ can also communicate with the adversary $\mathcal{A}$/simulator $\mathcal{S}$ and corrupts parties through it. The crucial part of the ideal/real world paradigm is that for every PPT adversary $\mathcal{A}$ attacking an execution of Π, there is a PPT simulator $\mathcal{S}$ attacking the ideal process that interacts with $\mathcal{F}$ (by corrupting the same set of parties), such that the executions of Π with $\mathcal{A}$ and that of $\mathcal{F}$ with $\mathcal{S}$ makes no difference to the environment $\mathcal{Z}$. We denote by $\mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ (resp. $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$) the output of the environment $\mathcal{Z}$ in the ideal world (resp. real world) execution. Formally, we define a protocol to be UC-secure through the following definition.

Definition 1. *We say a protocol Π UC-realizes the functionality $\mathcal{F}$, if for any PPT environment $\mathcal{Z}$ and any PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ s.t.* $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

Furthermore, we describe the *modularity* which is a crucial aspect of the UC framework: when a protocol calls subroutines, these subroutines can be treated as separate entities and can be analyzed separately for their security by way of realizing an ideal functionality. We here introduce the notion of "hybrid world", and a protocol Π is said to be realized "in the $\mathcal{G}$ hybrid world" if Π invokes the ideal functionality $\mathcal{G}$ as a subroutine.

Definition 2. *We say protocol Π UC-realizes the functionality $\mathcal{F}$ in the $\mathcal{G}$ hybrid world, if for any PPT environment $\mathcal{Z}$ and any PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ s.t.* $\mathsf{EXEC}^{\mathcal{G}}_{\Pi,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

We note that, in the UC framework, the environment $\mathcal{Z}$ cannot have the direct access to $\mathcal{G}$, but it can do so through the adversary. More precisely, in the real world (resp. ideal world), the adversary $\mathcal{A}$ (resp. the simulator $\mathcal{S}$) can access the ideal functionality $\mathcal{G}$ directly. It queries $\mathcal{G}$ for $\mathcal{Z}$ and forwards the answers. This implicitly means that the ideal functionality $\mathcal{G}$ is local to the challenge protocol instance. Therefore, the simulator $\mathcal{S}$ is allowed to simulate $\mathcal{G}$ in the ideal world as long as it "looks" indistinguishable from $\mathcal{G}$ hybrid world.

GUC Framework. In the UC framework, the environment $\mathcal{Z}$ is constrained: it cannot have the direct access to the setup. In other words, the setups in the UC framework are not global. This is an impractical assumption in real life applications, where it is more plausible that there is a global setup published and used by many protocol instances.

Motivated by solving problems caused by modeling setup as a local subroutine, the Generalized UC (GUC) framework was introduced by Canetti *et al.* [CDPW07], which can be used for properly analyzing concurrent execution of protocols in the presence of global setups. Compared to the UC framework, the environment $\mathcal{Z}$ in the GUC framework is unconstrained: it is allowed to access the setup directly without going through the simulator/adversary. Furthermore, the environment $\mathcal{Z}$ can invoke arbitrary protocols alongside the challenge protocol. Formally, Canetti *et al.* introduced the notion of *shared functionality*: it is completely analogous to an ideal functionality, except that it may interact with more than one protocol sessions. The global setup can be modeled as a shared functionality. This indeed captures the fact that any protocol in the network can use the same global setup.

To distinguish from the basic UC execution, we denote the output of the unconstrained PPT environment $\mathcal{Z}$ in the real world (resp. ideal world) execution as $\mathsf{GEXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$ (resp. $\mathsf{GEXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$).

Definition 3. *We say a protocol* Π *GUC-realizes functionality* $\mathcal{F}$*, if for any unconstrained PPT environment* $\mathcal{Z}$ *and any PPT adversary* $\mathcal{A}$ *there exists a PPT simulator* $\mathcal{S}$ *s.t.* $\mathsf{GEXEC}_{\Pi,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{GEXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

Since the unconstrained environment $\mathcal{Z}$ we described above is granted a high-level of flexibility (i.e., $\mathcal{Z}$ is allowed to invoke arbitrary protocols in parallel with the challenge protocol and cause arbitrary interactions with shared functionalities), it becomes extremely difficult to prove the GUC security. Therefore, a simplified framework called Externalized UC (EUC) framework was introduced in [CDPW07]. In the EUC framework, the environment $\mathcal{Z}$ has direct access to the shared functionality $\mathcal{G}$ but does not initiate any new protocol sessions except the challenge protocol session. We call such an environment is $\mathcal{G}$-externalized constrained. We say a protocol Π is $\mathcal{G}$-subroutine respecting if it only shares state information via a single shared functionality $\mathcal{G}$.

Definition 4. *Let the protocol* Π *be* $\mathcal{G}$*-subroutine respecting. We say a protocol* Π *EUC-realizes the functionality* $\mathcal{F}$ *with respect to shared functionality* $\mathcal{G}$*, if for any PPT* $\mathcal{G}$*-externalized constrained environment* $\mathcal{Z}$ *and any PPT adversary* $\mathcal{A}$ *there exists a PPT simulator* $\mathcal{S}$ *s.t.* $\mathsf{EXEC}^{\mathcal{G}}_{\Pi,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

Furthermore, Canetti *et al.* showed that for any $\mathcal{G}$-subroutine respecting protocol Π, proving Π EUC-realizes $\mathcal{F}$ with respect to $\mathcal{G}$ is equivalent to proving Π GUC-realizes $\mathcal{F}$ [CDPW07]. For that reason, when we want to prove the GUC security of a protocol, we always turn to EUC security for the sake of simplicity.

Adversarial Model. In this work, we consider both static corruption (where the adversary corrupts the parties at the beginning of the protocol) and adaptive corruption (where the adversary corrupts the parties at any time). We also consider *rushing* adversaries, who may delay sending messages on behalf of corrupted parties in a given round until the messages sent by all the uncorrupted parties in that round have been received [Kat07].

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

The functionality interacts with a set of the parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{S}$.

Send. Upon receiving $(\textsc{Send}, \mathsf{sid}, P_j, m)$ from P_i, do:

- Send $(\textsc{Send}, \mathsf{sid}, P_i, P_j, m)$ to the adversary $\mathcal{S}$.
- Initialize a boolean corruption status $s := 0$.
- Upon receiving $(\textsc{Ok}, \mathsf{sid})$ from $\mathcal{S}$: if not yet generated output, output $(\textsc{Send}, \mathsf{sid}, P_i, P_j, m)$ to P_j.
- Upon receiving $(\textsc{Corrupt}, \mathsf{sid}, m', P_k)$ from $\mathcal{S}$, do:
  - Set $s := 1$.
  - If not yet generated output, output $(\textsc{Send}, \mathsf{sid}, P_i, P_k, m')$ to P_k.
- Upon receiving $(\textsc{ReportCorrupted}, \mathsf{sid})$ from P_i, do:
  - If $s = 0$, output $(\textsc{No}, \mathsf{sid})$ to P_i; else, output $(\textsc{Yes}, \mathsf{sid})$ to P_i.
- Ignore any subsequent messages.

Figure 3: The Ideal Functionality $\mathcal{F}_{\mathsf{Auth}}$ for Authenticated Channel

Secure Communication Model. Many UC-secure protocols assume the parties are interconnected with secure channels or authenticated channels [CLOS02, CDPW07, CDG+18]. The secure channel and authenticated channel can be modeled as ideal functionalities $\mathcal{F}_{\mathsf{SC}}$ and $\mathcal{F}_{\mathsf{Auth}}$ respectively [CK02, Can01]. In this work, most of our protocols are designed in the simultaneous communication channel with rushing adversaries, which is different from that [Kat07] deals with non-rushing adversaries. For this reason, we often assume the synchronous channel which can be modeled as $\mathcal{F}_{\mathsf{Syn}}$ [Can01]. Note that, intuitively, $\mathcal{F}_{\mathsf{Syn}}$ can be viewed an authenticated communication network with storage, which proceeds in a round-based fashion [Can01, KMTZ13]. For readability, we will mention which secure communication channel is used in the context and omit it in the protocol description. For completeness, we will give the formal description of the communication channel that will be used in this work (i.e., the authenticated channel and the synchronous channel) in the following. We present the ideal functionality for authenticated channel $\mathcal{F}_{\mathsf{Auth}}$ in Figure 3 and ideal functionality for synchronous channel $\mathcal{F}_{\mathsf{Syn}}$ in Figure 4.

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

The functionality interacts with a set of the parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{S}$. It maintains an initial round counter $r := 1$, an initial boolean s_P^ρ and an initially empty list N_P^ρ for all $\rho \geq 0$ and all $P \in \mathcal{P}$.

Send. Upon receiving $(\textsc{Send}, \mathsf{sid}, M)$ from P_i, where M is a list of pairs in the form of (P_j, m), do:

- Set $s_{P_i}^r := 1$.
- For each pair $(P_j, m) \in M$, add (P_i, m) to $N_{P_j}^r$.
- If all the uncorrupted parties in $\mathcal{P}$ have already provided their messages for round r (i.e., if $s_P^r = 1$ for all uncorrupted $P \in \mathcal{P}$), then increment the round counter $r := r + 1$.
- Send $(\textsc{Send}, \mathsf{sid}, P_i, M, r)$ to the adversary $\mathcal{S}$.

Receive. Upon receiving $(\textsc{Receive}, \mathsf{sid}, r')$ from P_i, do:

- If $r' < r$, output $(\textsc{Received}, \mathsf{sid}, N_{P_i}^{r'})$ to P_i; else, output $(\textsc{Incomplete}, r)$ to P_i.

Backdoor. Upon receiving a backdoor message $(\textsc{Corrupt}, \mathsf{sid}, P_i)$ from P_i, mark P_i as corrupted.

Figure 4: The Ideal Functionality $\mathcal{F}_{\mathsf{Syn}}$ for Synchronous Channel

2.3 Sigma Protocols

A Sigma-protocol is a 3-move public coin protocol [Dam02], where both the prover P and the verifier V hold a statement x, and the prover P additionally holds a private witness w such that $(x, w) \in \mathcal{R}$. In the first move, the prover P generates the message a by invoking $(a, \mathsf{st}) \leftarrow \mathsf{P1}(x, w)$ on a statement-witness pair (x, w). In the second move, the verifier V samples an uniformly random string $e \leftarrow \mathsf{V1}(1^\lambda)$ as the challenge. In the last move, the prover P computes the response $z \leftarrow \mathsf{P2}(x, w, e, \mathsf{st})$ using the statement-witness pair (x, w), the received challenge e and the previously computed state st. Finally, the verifier outputs a bit $b := \mathsf{Verify}(x, a, e, z)$ indicating acceptance $(b = 1)$ or rejection $(b = 0)$.

The Sigma-protocol should satisfy the completeness, special soundness and Special Honest Verifier Zero-Knowledge (SHVZK). The completeness requires that the honest prover will always make the verifier accept. The special soundness requires that there exists a PPT extractor algorithm Ext that given two distinct accepting transcripts (i.e., (a, e, z) and (a, e', z') where $e \neq e'$), it can output a valid witness w. The SHVZK property requires that there exists a PPT simulator algorithm Sim that given the challenge e ahead, it can output (a, z) that are computational indistinguishable from the real ones. Formally, we have the following definition.

Definition 5. *Fix an NP relation $\mathcal{R}$ and its associate language $\mathcal{L}$. We say a protocol $\Pi = (\Pi.\mathsf{P1}, \Pi.\mathsf{V1}, \Pi.\mathsf{P2}, \Pi.\mathsf{Verify}, \Pi.\mathsf{Ext}, \Pi.\mathsf{Sim})$ is a Sigma-protocol for $\mathcal{R}$ if the following conditions hold:*

1. ***(Completeness)*** *For any $(x, w) \in \mathcal{R}$, we say it is complete if*

$$\Pr\left[\begin{array}{l}(a, \mathsf{st}) \leftarrow \mathsf{P1}(x, w); e \leftarrow \mathsf{V1}(1^\lambda);\\ z \leftarrow \mathsf{P2}(x, w, e, \mathsf{st})\end{array} : \mathsf{Verify}(x, a, e, z) = 1\right] = 1$$

2. ***(Special Soundness)*** *For any $x \in \mathcal{L}$, we say it is special sound if for any PPT $\mathcal{A}$,*

$$\Pr\left[\begin{array}{l}(a, e, z, e', z') \leftarrow \mathcal{A}(x);\\ w \leftarrow \mathsf{Ext}(x, a, e, z, e', z')\end{array} : \begin{array}{l}\mathsf{Verify}(x, a, e, z) = \mathsf{Verify}(x, a, e', z') = 1\\ \wedge\, e \neq e' \ \wedge\ (x, w) \in \mathcal{R}\end{array}\right] = 1$$

3. ***(Special Honest Verifier Zero-Knowledge)*** *Given the challenge e ahead, for any $(x, w) \in \mathcal{R}$, we say it is SHVZK if*

$$\{(a, z) \mid (a, \mathsf{st}) \leftarrow \mathsf{P1}(x, w); z \leftarrow \mathsf{P2}(x, w, e, \mathsf{st})\} \overset{c}{\approx} \{(a, z) \mid (a, z) \leftarrow \mathsf{Sim}(x, e)\}$$

2.4 Non-Interactive Arguments in the Random Oracle Model

In this section, we introduce different variants of non-interactive argument systems in the RO model. In a non-interactive argument $\Pi = (\Pi.\mathsf{Prove}^{\mathcal{O}}, \Pi.\mathsf{Verify}^{\mathcal{O}})$ in the RO model, both prover and verifier are allowed to query the RO $\mathcal{O}$ at any time, during the protocol execution. A non-interactive argument system allow the prover to generate the proof π using the statement-witness pair (x, w). Upon receiving the proof π, the verifier can decide whether to accept or not. Formally, a non-interactive argument systems has the following algorithms:

- $\pi \leftarrow \mathsf{Prove}^{\mathcal{O}}(x, w)$ takes input as a statement-witness pair (x, w), and it outputs a proof π.
- $b := \mathsf{Verify}^{\mathcal{O}}(x, \pi)$ takes input as a statement x and a proof π, and it outputs a bit b indicating acceptance $(b = 1)$ or not $(b = 0)$.

The non-interactive argument systems should satisfies completeness and computational soundness. The former property requires that honest provers can always make the verifiers accept. The latter property requires that any computationally bounded provers cannot convince the verifiers that a false statement is true.

Definition 6. *Fix an NP relation $\mathcal{R}$ and its associate language $\mathcal{L}$. Consider a random oracle $\mathcal{O}$. We say $\Pi = (\Pi.\mathsf{Prove}^{\mathcal{O}}$, $\Pi.\mathsf{Verify}^{\mathcal{O}})$ is a non-interactive argument system for $\mathcal{R}$ in the RO model if the following conditions hold:*

1. ***(Completeness)*** *For any $(x, w) \in \mathcal{R}$, we say it is complete if*

$$\Pr\left[\pi \leftarrow \mathsf{Prove}^{\mathcal{O}}(x, w) : \mathsf{Verify}^{\mathcal{O}}(x, \pi) = 1\right] = 1$$

2. ***(Computational Soundness)*** *For any $x \notin \mathcal{L}$, we say it is computational sound if for any PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that*

$$\Pr\left[\pi^* \leftarrow \mathcal{A}^{\mathcal{O}}(x) : \mathsf{Verify}^{\mathcal{O}}(x, \pi^*) = 1\right] \leq \mathsf{negl}(\lambda)$$

2.4.1 NIWH Arguments in the Random Oracle Model

An Non-Interactive Witness Hiding (NIWH) argument $\Pi = (\Pi.\mathsf{Prove}^{\mathcal{O}}, \Pi.\mathsf{Verify}^{\mathcal{O}})$ in the RO model is a non-interactive argument in the RO model that additionally satisfies the witness hiding property. The witness hiding property means that any computationally bounded adversary cannot extract the witness w from any accepting proof π. Before giving the formal definition of witness hiding, we have to define the hard instance ensembles, as in [Pas03].

Definition 7 (Hard Instance Ensembles)**.** *Let $\mathcal{R}$ be an NP relation and $\mathcal{L}$ be its associate language, and $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ be a probability ensemble such that $\mathcal{X}_\lambda$ ranges over $\mathcal{L} \cap \{0,1\}^\lambda$. We say $\mathcal{X}$* a hard instance ensemble for $\mathcal{R}$ *if for any PPT adversary $\mathcal{A}$ and any $x \in \mathcal{X}$, there exists a negligible function* negl *such that* $\Pr[(x, \mathcal{A}(x)) \in \mathcal{R}] \leq \mathsf{negl}(\lambda)$.

Now we can formally define the NIWH argument in the RO model.

Definition 8. *Fix an NP relation $\mathcal{R}$ and its associate language $\mathcal{L}$. Consider a random oracle $\mathcal{O}$. We say $\Pi = (\Pi.\mathsf{Prove}^{\mathcal{O}}$, $\Pi.\mathsf{Verify}^{\mathcal{O}})$ is an NIWH argument system for $\mathcal{R}$ in the RO model if it satisfies completeness, computational soundness and the following property:*

1. ***(Witness Hiding)*** *For any $(x, w) \in \mathcal{R}$ and $x \in \mathcal{X}$ where $\mathcal{X}$ a hard instance ensemble for $\mathcal{R}$, we say it is* witness hiding *if for any PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that*

$$\Pr\left[\pi \leftarrow \mathsf{Prove}^{\mathcal{O}}(x, w); w' \leftarrow \mathcal{A}(x, \pi) : (x, w') \in \mathcal{R}\right] \leq \mathsf{negl}(\lambda)$$

2.4.2 NIZK Arguments in the Random Oracle Model

Now let us consider another strengthened version of non-interactive argument in the RO model: Non-Interactive Zero-Knowledge (NIZK) arguments $\Pi = (\Pi.\mathsf{Prove}^{\mathcal{O}}, \Pi.\mathsf{Verify}^{\mathcal{O}}, \Pi.\mathsf{Sim})$ in the RO model. It is known that the NIZK argument systems imply NIWH argument systems in the RO model [Pas03]. An NIZK argument is a non-interactive argument that additionally satisfies the zero-knowledge property. Here we adopt the definition from [FKMV12], the zero-knowledge property requires that there exists a PPT stateful simulator algorithm $\Pi.\mathsf{Sim}$

that can operate in two modes: $(h_i, \mathsf{st}) \leftarrow \mathsf{Sim}(1, \mathsf{st}, q_i)$ that takes care of answering the queries to RO, it takes the mode number 1, the state st and the query q_i as inputs, and it outputs the simulated RO's answer h_i and the updated state st; $(\pi, \mathsf{st}) \leftarrow \mathsf{Sim}(2, \mathsf{st}, x)$ that simulates the actual proof, it takes the mode number 2, the state st and the statement x as inputs, and it outputs the proof π and the updated state st. Note that, calls to $\mathsf{Sim}(1, \cdot, \cdot)$ and $\mathsf{Sim}(2, \cdot, \cdot)$ share the common state st which is updated after each operation. Formally, we have the following definition.

Definition 9. *Fix an NP relation $\mathcal{R}$ and its associate language $\mathcal{L}$. Consider a RO $\mathcal{O}$. We say $\Pi = (\Pi.\mathsf{Prove}^{\mathcal{O}}, \Pi.\mathsf{Verify}^{\mathcal{O}}, \Pi.\mathsf{Sim})$ is an NIZK argument system for $\mathcal{R}$ in the RO model if it satisfies completeness, computational soundness and the following property:*

1. ***(Zero-Knowledge)*** *Denote by $(\tilde{\mathcal{O}}_1, \tilde{\mathcal{O}}_2)$ the oracles such that $\tilde{\mathcal{O}}_1(q_i)$ returns the first output of $(h_i, \mathsf{st}) \leftarrow \mathsf{Sim}(1, \mathsf{st}, q_i)$ and $\tilde{\mathcal{O}}_2(x, w)$ returns the first output of $(\pi, \mathsf{st}) \leftarrow \mathsf{Sim}(2, \mathsf{st}, x)$ if $(x, w) \in \mathcal{R}$. For any $(x, w) \in \mathcal{R}$, we say it is* zero-knowledge *if for all PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that*

$$\left|\Pr[\mathcal{A}^{\mathcal{O}(\cdot), \mathsf{Prove}^{\mathcal{O}}(\cdot,\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{\tilde{\mathcal{O}}_1(\cdot), \tilde{\mathcal{O}}_2(\cdot,\cdot)}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda) \ ,$$

where $\mathcal{A}$ is given oracle access to either $\mathcal{O}, \mathsf{Prove}^{\mathcal{O}}$ or $\tilde{\mathcal{O}}_1, \tilde{\mathcal{O}}_2$, and both $\mathsf{Prove}^{\mathcal{O}}(x, w)$ and $\tilde{\mathcal{O}}_2(x, w)$ output $\perp$ if $(x, w) \notin \mathcal{R}$.

2.4.3 Straight-Line Extractability in the Random Oracle Model

Now let us consider a stronger security property, i.e. *(straight-line)* extractability, and our extractability definition is adopted from that by Pass [Pas03]. The extractability means that there exists a PPT extractor algorithm Ext which can extract the witness from a maliciously generated and accepting proof. To enable straight-line extractability, typically, the extractor Ext can be developed by simulating the RO for the prover and the verifier, and thus Ext has full control of the RO. Note that, by the simulating the RO, the extractor Ext is granted two advantages: (i) *observability:* the simulator can see the query-answer list of the RO; (ii) *programmability:* the simulator can answer the queries to the RO.

In this work, we consider the straight-line extractability in a much more restricted RO, that is, the extractor Ext is granted only the observability of RO. For that reason, we denote by $\mathcal{Q}^*$ the RO queries and answers posed by the malicious prover, and we will feed the extractor Ext with $\mathcal{Q}^*$. Now we can formally define the straight-line extractability.

Definition 10 (Straight-line extractability)**.** *Fix an NP relation $\mathcal{R}$ and its associate language $\mathcal{L}$. Consider a RO $\mathcal{O}$, and a non-interactive argument system for $\mathcal{R}$ in the RO model $\Pi = (\Pi.\mathsf{Prove}^{\mathcal{O}}, \Pi.\mathsf{Verify}^{\mathcal{O}}, \Pi.\mathsf{Ext})$. Denote by $\mathcal{Q}^*$ the RO queries and answers posed by the adversary $\mathcal{A}$. For any $x \in \mathcal{L}$, we say the non-interactive argument Π is* straight-line extractable *if for any PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that*

$$\Pr\left[\begin{array}{l}\pi^* \leftarrow \mathcal{A}^{\mathcal{O}}(x); b := \mathsf{Verify}^{\mathcal{O}}(x, \pi^*); \\ w^* \leftarrow \mathsf{Ext}(x, \pi^*, \mathcal{Q}^*)\end{array} : b = 1 \ \wedge \ (x, w^*) \in \mathcal{R}\right] \geq 1 - \mathsf{negl}(\lambda)$$

Note that, the NIWH argument and NIZK argument are developed on top of the non-interactive argument. Hence the Definition 10 can be easily extended to the straight-line extractable NIZK (NIWH) argument in the RO model.

Definition 11. *Fix an NP relation $\mathcal{R}$ and its associate language $\mathcal{L}$. Consider a RO $\mathcal{O}$. We say $\Pi = (\Pi.\mathsf{Prove}^{\mathcal{O}}, \Pi.\mathsf{Verify}^{\mathcal{O}}, \Pi.\mathsf{Ext})$ is a* straight-line extractable NIWH argument system *for $\mathcal{R}$ in the RO model if it satisfies completeness, computational soundness, witness hiding and straight-line extractability.*

Definition 12. *Fix an NP relation $\mathcal{R}$ and its associate language $\mathcal{L}$. Consider a RO $\mathcal{O}$. We say $\Pi = (\Pi.\mathsf{Prove}^{\mathcal{O}}, \Pi.\mathsf{Verify}^{\mathcal{O}}, \Pi.\mathsf{Sim}, \Pi.\mathsf{Ext})$ is a* straight-line extractable NIZK argument system *for $\mathcal{R}$ in the RO model if it satisfies completeness, computational soundness, computational zero-knowledge and straight-line extractability.*

2.5 Ideal Functionalities

In this section, we provide ideal functionalities that will be used in UC/GUC security analysis.

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

The functionality interacts with two parties P_1, P_2 and an adversary $\mathcal{S}$.

Toss. Upon receiving $(\textsc{Toss}, \mathsf{sid}, P_1, P_2)$ from P_1 and P_2, do:

- Sample $r \leftarrow \{0,1\}^\lambda$, send $(\textsc{Proceed?}, \mathsf{sid}, P_1, P_2)$ to $\mathcal{S}$.
- Upon receiving $(\textsc{Proceed}, \mathsf{sid}, P_1)$ from $\mathcal{S}$, send $(\textsc{Tossed}, \mathsf{sid}, P_1, P_2, r)$ to P_1; Upon receiving $(\textsc{No}, \mathsf{sid}, P_1)$ from $\mathcal{S}$, send $(\textsc{Abort}, \mathsf{sid}, P_1)$ to P_1. Upon receiving $(\textsc{Proceed}, \mathsf{sid}, P_2)$ from $\mathcal{S}$, send $(\textsc{Tossed}, \mathsf{sid}, P_1, P_2, r)$ to P_2; Upon receiving $(\textsc{No}, \mathsf{sid}, P_2)$ from $\mathcal{S}$, send $(\textsc{Abort}, \mathsf{sid}, P_2)$ to P_2.
- Ignore any subsequent $\textsc{Toss}$ command.

Figure 5: The Ideal Functionality $\mathcal{F}_{\mathsf{Coin}}$ for Coin-Tossing

2.5.1 Coin-Tossing

Coin-tossing is one of the most minimal building blocks for cryptographic protocols. Coin-tossing is a two-party protocol which allows each party to receive an uniformly random string at the end of the protocol execution. Formally, we put the coin-tossing functionality $\mathcal{F}_{\mathsf{Coin}}$ in Figure 5.

2.5.2 OT, UOT and EOT

We start with the Oblivious Transfer (OT). We often use OT to refer to the 1-out-of-2 OT. In an OT protocol, there is a sender S holding two private input $m_0, m_1 \in \{0,1\}^\lambda$ and a receiver R holding a choice bit $b \in \{0,1\}$. At the end of the honest execution of the OT protocol, the receiver R will compute m_b. At the same time, the sender should learn nothing about b while the receiver should learn nothing about m_{1-b}. We present the OT functionality $\mathcal{F}_{\mathsf{OT}}$ in Figure 6.

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

The functionality interacts with two parties S, R and an adversary $\mathcal{S}$.

Transfer. Upon receiving $(\textsc{Send}, \mathsf{sid}, S, R, m_0, m_1)$ from the sender S, do:

- Record $(\mathsf{sid}, S, R, m_0, m_1)$, and send $(\textsc{Send}, \mathsf{sid}, S, R)$ to R and the adversary $\mathcal{S}$.
- Ignore any subsequent $\textsc{Send}$ commands.

Choose. Upon receiving $(\textsc{Receive}, \mathsf{sid}, S, R, b)$ from R where $b \in \{0,1\}$, do:

- Record (sid, S, R, b), and send $(\textsc{Receive}, \mathsf{sid}, S, R)$ to the sender S and the adversary $\mathcal{S}$.
- Ignore any subsequent $\textsc{Receive}$ commands.

Process. When both $(\mathsf{sid}, S, R, m_0, m_1)$ and (sid, S, R, b) are recorded, do:

- Send $(\textsc{Proceed?}, \mathsf{sid}, S, R)$ to the adversary $\mathcal{S}$.
- Upon receiving $(\textsc{Proceed}, \mathsf{sid}, S)$ from the adversary $\mathcal{S}$, output $(\textsc{Received}, \mathsf{sid}, S, R)$ to the sender S; Upon receiving $(\textsc{No}, \mathsf{sid}, S)$ from the adversary $\mathcal{S}$, output $(\textsc{Abort}, \mathsf{sid}, S)$ to the sender S. Upon receiving $(\textsc{Proceed}, \mathsf{sid}, R)$ from the adversary $\mathcal{S}$, output $(\textsc{Received}, \mathsf{sid}, S, R, m_b)$ to R; Upon receiving $(\textsc{No}, \mathsf{sid}, R)$ from the adversary $\mathcal{S}$, output $(\textsc{Abort}, \mathsf{sid}, R)$ to R.

Figure 6: The Ideal Functionality $\mathcal{F}_{\mathsf{OT}}$ for Oblivious Transfer

In [MR19a], Masny and Rindal proposed two notions that called Uniform OT (UOT) and Endemic OT (EOT). Both of them are similar to OT, except that the senders have no inputs. The main difference between the UOT and the EOT is that they provide different levels of security guarantees. We describe the UOT first. The UOT functionality samples two uniformly random strings m_0, m_1, and outputs m_0, m_1 to the (potentially malicious) sender and m_b to the (potentially malicious) receiver. The UOT gives a strong security guarantee that any malicious party cannot influence the distribution of the OT messages. Formally, we put the UOT functionality $\mathcal{F}_{\mathsf{UOT}}$ in Figure 7.

Now let us turn to EOT. Compared to UOT, the EOT functionality gives a weak security guarantee: no matter whether the sender or the receiver is malicious, the malicious party can always determine the distribution of the OT messages. Roughly speaking, if both sender and receiver are honest, the EOT functionality acts as the UOT functionality. If the sender is malicious and the receiver is honest, the EOT functionality lets the adversary determine the message strings m_0, m_1, and it returns the adversarial chosen m_b to the honest receiver after receiving b. If the receiver is malicious and the sender is honest, the EOT functionality lets the adversary determine

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

The functionality interacts with two parties S, R and an adversary $\mathcal{S}$.

Transfer. Upon receiving $(\textsc{Send}, \mathsf{sid}, S, R)$ from S, do:

- Sample $m_0, m_1 \leftarrow \{0,1\}^\lambda$, record $(\mathsf{sid}, S, R, m_0, m_1)$, and send $(\textsc{Send}, \mathsf{sid}, S, R)$ to R and the adversary $\mathcal{S}$.
- Ignore any subsequent SEND commands.

Choose. Upon receiving $(\textsc{Receive}, \mathsf{sid}, S, R, b)$ from R where $b \in \{0,1\}$, do:

- Record (sid, S, R, b), and send $(\textsc{Receive}, \mathsf{sid}, S, R)$ to the sender S and the adversary $\mathcal{S}$.
- Ignore any subsequent RECEIVE commands.

Process. When both $(\mathsf{sid}, S, R, m_0, m_1)$ and (sid, S, R, b) are recorded, do:

- Send $(\textsc{Proceed?}, \mathsf{sid}, S, R)$ to the adversary $\mathcal{S}$.
- Upon receiving $(\textsc{Proceed}, \mathsf{sid}, S)$ from the adversary $\mathcal{S}$, output $(\textsc{Received}, \mathsf{sid}, S, R, m_0, m_1)$ to the sender S; Upon receiving $(\textsc{No}, \mathsf{sid}, S)$ from the adversary $\mathcal{S}$, output $(\textsc{Abort}, \mathsf{sid}, S)$ to the sender S. Upon receiving $(\textsc{Proceed}, \mathsf{sid}, R)$ from the adversary $\mathcal{S}$, output $(\textsc{Received}, \mathsf{sid}, S, R, m_b)$ to R; Upon receiving $(\textsc{No}, \mathsf{sid}, R)$ from the adversary $\mathcal{S}$, output $(\textsc{Abort}, \mathsf{sid}, R)$ to R.

Figure 7: The Ideal Functionality $\mathcal{F}_{\mathsf{UOT}}$ for Uniform Oblivious Transfer

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

The functionality interacts with two parties S, R and an adversary $\mathcal{S}$.

Transfer. Upon receiving $(\textsc{Send}, \mathsf{sid}, S, R)$ from S, do:

- Sample $m_0, m_1 \leftarrow \{0,1\}^\lambda$, record $(\mathsf{sid}, S, R, m_0, m_1)$, and send $(\textsc{Send}, \mathsf{sid}, S, R)$ to R and the adversary $\mathcal{S}$.
- Ignore any subsequent SEND commands.

Choose. Upon receiving $(\textsc{Receive}, \mathsf{sid}, S, R, b)$ from R where $b \in \{0,1\}$, do:

- Record (sid, S, R, b), and send $(\textsc{Receive}, \mathsf{sid}, S, R)$ to the sender S and the adversary $\mathcal{S}$.
- Ignore any subsequent RECEIVE commands.

Process. When both $(\mathsf{sid}, S, R, m_0, m_1)$ and (sid, S, R, b) are recorded, do:

- If both the sender S and the receiver R are honest, output $(\textsc{Received}, \mathsf{sid}, S, R, m_0, m_1)$ to the sender S, $(\textsc{Received}, \mathsf{sid}, S, R, m_b)$ to R and $(\textsc{Received}, \mathsf{sid}, S, R)$ to the adversary $\mathcal{S}$.
- Else if the sender S is corrupted and the receiver R is honest, send $(\textsc{Proceed?}, \mathsf{sid}, R)$ to the adversary $\mathcal{S}$. Upon receiving $(\textsc{Proceed}, \mathsf{sid}, R, m_0^*, m_1^*)$ from the adversary $\mathcal{S}$, set $m_0 := m_0^*$, $m_1 := m_1^*$, and output $(\textsc{Received}, \mathsf{sid}, S, R, m_0, m_1)$ to the sender S, $(\textsc{Received}, \mathsf{sid}, S, R, m_b)$ to R; Upon receiving $(\textsc{No}, \mathsf{sid}, R)$ from the adversary $\mathcal{S}$, output $(\textsc{Abort}, \mathsf{sid}, R)$ to R.
- Else if the sender S is honest and the receiver R is corrupted, send $(\textsc{Proceed?}, \mathsf{sid}, S)$ to the adversary $\mathcal{S}$. Upon receiving $(\textsc{Proceed}, \mathsf{sid}, S, m_b^*)$ from the adversary $\mathcal{S}$, set $m_b := m_b^*$, and output $(\textsc{Received}, \mathsf{sid}, S, R, m_0, m_1)$ to the sender S, $(\textsc{Received}, \mathsf{sid}, S, R, m_b)$ to R; Upon receiving $(\textsc{No}, \mathsf{sid}, S)$ from the adversary $\mathcal{S}$, output $(\textsc{Abort}, \mathsf{sid}, S)$ to the sender S.
- Else if both the sender S and the receiver R are corrupted, halt.

Figure 8: The Ideal Functionality $\mathcal{F}_{\mathsf{EOT}}$ for Endemic Oblivious Transfer

the message string m_b, and it returns the adversarial chosen m_b and an uniformly sampled m_{1-b} to the honest sender. If both sender and receiver are malicious, the EOT functionality simply aborts. Formally, we put the EOT functionality $\mathcal{F}_{\mathsf{EOT}}$ in Figure 8.

2.5.3 Commitment

A commitment protocol allows a committer to compute a commitment message c to a message m towards a receiver in the committing phase. Later in the opening phase, the committer can open c to m by sending the opening message to the receiver. The commitment protocol should satisfies two properties: (i) hiding property, i.e., the receiver should learn nothing about the committed message from c; (ii) binding property, i.e., the committer cannot open c to another message $m' \neq m$. We put the commitment functionality $\mathcal{F}_{\mathsf{Com}}$ in Figure 9, and both hiding and binding properties are captured by $\mathcal{F}_{\mathsf{Com}}$.

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

The functionality interacts with two parties C, R and an adversary $\mathcal{S}$.

Commit. Upon receiving $(\text{COMMIT}, \mathsf{sid}, C, R, m)$ from C, do:

- Record the tuple (sid, C, R, m), and send $(\text{PROCEED?}, \mathsf{sid}, C, R)$ to $\mathcal{S}$.
- Upon receiving $(\text{PROCEED}, \mathsf{sid}, C, R)$ from $\mathcal{S}$, send $(\text{RECEIPT}, \mathsf{sid}, C, R)$ to R and C; Upon receiving $(\text{NO}, \mathsf{sid}, C, R)$ from $\mathcal{S}$, send $(\text{ABORT}, \mathsf{sid}, C, R)$ to C and R and halt.
- Ignore any subsequent COMMIT command.

Open. Upon receiving $(\text{DECOMMIT}, \mathsf{sid}, C, R)$ from C, do:

- If there is a tuple (sid, C, R, m) recorded, send $(\text{PROCEED?}, \mathsf{sid}, C, R)$ to $\mathcal{S}$. Upon receiving $(\text{PROCEED}, \mathsf{sid}, C, R)$ from $\mathcal{S}$, send $(\text{DECOMMIT}, \mathsf{sid}, C, R, m)$ to R and $\mathcal{S}$; Upon receiving $(\text{NO}, \mathsf{sid}, C, R)$ from $\mathcal{S}$, send $(\text{ABORT}, \mathsf{sid}, C, R)$ to C and R.
- Otherwise, ignore the message.

Figure 9: The Ideal Functionality $\mathcal{F}_{\mathsf{Com}}$ for Commitment

2.5.4 Random Oracles

We introduce the local RO model, and two well-known global RO models: Global Restricted Observable Random Oracle (GroRO) model proposed by Canetti *et al.* [CJS14] and Global Restricted Programmable Random Oracle (GrpRO) model proposed by Camenisch *et al.* [CDG+18].

The Local RO Model. Typically, the local RO is modeled as an ideal functionality $\mathcal{F}_{\mathsf{RO}}$. As depicted in Figure 10, upon receiving $(\text{QUERY}, \mathsf{sid}, x)$ from any party, $\mathcal{F}_{\mathsf{RO}}$ first checks whether the query (sid, x) has been queried before. If not, $\mathcal{F}_{\mathsf{RO}}$ selects a random value of pre-specified length $v \leftarrow \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}$, answers with the value v and records the tuple (sid, x, v); otherwise, the previously chosen value v is returned again, even if the earlier query was made by another party.

Ideal Functionality $\mathcal{F}_{\mathsf{RO}}$

The functionality interacts with a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{S}$. It is parameterized by the output length $\ell_{\mathsf{out}}(\lambda)$. It maintains an initially empty list List.

Query. Upon receiving $(\text{QUERY}, \mathsf{sid}, x)$ from a party $P_i \in \mathcal{P}$, or the adversary $\mathcal{S}$:

- Check if $\exists\, v \in \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}$ such that $(\mathsf{sid}, x, v) \in \mathsf{List}$. If not, select $v \leftarrow \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}$ and record the tuple (sid, x, v) in List.
- Return $(\text{QUERYCONFIRM}, \mathsf{sid}, v)$ to the requestor.

Figure 10: The Local Random Oracle Model $\mathcal{F}_{\mathsf{RO}}$

The GroRO Model. Compared to $\mathcal{F}_{\mathsf{RO}}$, the GroRO is modeled as a shared functionality $\mathcal{G}_{\mathsf{roRO}}$ which may interact with more than one protocol sessions. The $\mathcal{G}_{\mathsf{roRO}}$ answers to the queries in the same way as $\mathcal{F}_{\mathsf{RO}}$. The simulator is only granted the restricted observability: some of the queries can be marked as "illegitimate" and potentially disclosed to the simulator. As depicted in Figure 11, the $\mathcal{G}_{\mathsf{roRO}}$ interacts with a list of ideal functionalities $\bar{\mathcal{F}} = \{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$, where $\mathcal{F}_1, \ldots, \mathcal{F}_n$ are the ideal functionalities for protocols. For any query (sid', x) from any party $P = (\mathsf{pid}, \mathsf{sid})$ where sid' is the content of the SID field, if $\mathsf{sid}' \neq \mathsf{sid}$, then this query is considered "illegitimate". After that, $\mathcal{G}_{\mathsf{roRO}}$ adds the tuple (sid', x, v) to the list of illegitimate queries for SID sid', which we denote as $\mathcal{Q}_{\mathsf{sid}'}$. The illegitimate queries $\mathcal{Q}_{\mathsf{sid}'}$ may be disclosed to an instance of ideal functionality $\mathcal{F} \in \bar{\mathcal{F}}$ whose SID is the one of the illegitimate queries, and the ideal functionality instance $\mathcal{F}$ may leak the illegitimate queries $\mathcal{Q}_{\mathsf{sid}'}$ to the simulator.

Share Functionality $\mathcal{G}_{\mathsf{roRO}}$

The functionality interacts with a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{S}$. It is parameterized by the output length $\ell_{\mathsf{out}}(\lambda)$ and a list of ideal functionalities $\bar{\mathcal{F}} := \{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$. It maintains an initially empty list List.

Query. Upon receiving $(\textsc{Query}, \mathsf{sid}', x)$ from a party $P_i \in \mathcal{P}$ where $P_i = (\mathsf{pid}, \mathsf{sid})$, or the adversary $\mathcal{S}$:

- Check if $\exists\, v \in \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}$ such that $(\mathsf{sid}, x, v) \in \mathsf{List}$. If not, select $v \leftarrow \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}$ and record the tuple (sid', x, v) in List.
- If $\mathsf{sid} \neq \mathsf{sid}'$, add the tuple (sid', x, v) to the (initially empty) list of illegitimate queries for SID sid', which we denote by $\mathcal{Q}_{\mathsf{sid}'}$.
- Return $(\textsc{QueryConfirm}, \mathsf{sid}', v)$ to the requestor.

Observe. Upon receiving a request from an instance of an ideal functionality $\mathcal{F}_i \in \bar{\mathcal{F}}$ with SID sid', return the list of illegitimate queries $\mathcal{Q}_{\mathsf{sid}'}$ for SID sid' to this instance $\mathcal{F}_i$.

Figure 11: The Global Restricted Observable Random Oracle Model $\mathcal{G}_{\mathsf{roRO}}$

The GrpRO Model. The GrpRO is also modeled as a share functionality $\mathcal{G}_{\mathsf{rpRO}}$, and it answers to the queries in the same way as $\mathcal{F}_{\mathsf{RO}}$. The simulator is only granted the restricted programmability: both the adversary and the simulator are allowed to program the unqueried points of the random oracle, but only the simulator can program it without being detected. More precisely, as depicted in Figure 12, upon receiving $(\textsc{Program}, \mathsf{sid}, x, v)$ from the simulator/adversary, $\mathcal{G}_{\mathsf{rpRO}}$ first checks whether (sid, x) has been queried before. If not, $\mathcal{G}_{\mathsf{rpRO}}$ stores (sid, x, v) in the query-answer lists. Any honest party can check whether a point has been programmed or not by sending the $(\textsc{IsPrograMed}, \mathsf{sid}, x)$ to $\mathcal{G}_{\mathsf{rpRO}}$. Thus, in the real world, the programmed points can always be detected. However, in the ideal world, the simulator $\mathcal{S}$ can escape the detection since it can return $(\textsc{IsPrograMed}, \mathsf{sid}, 0)$ when the adversary invokes $(\textsc{IsPrograMed}, \mathsf{sid}, x)$ to verify whether a point x has been programmed or not.

Share Functionality $\mathcal{G}_{\mathsf{rpRO}}$

The functionality interacts with a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{S}$. It is parameterized by the output length $\ell_{\mathsf{out}}(\lambda)$. It maintains two initially empty lists $\mathsf{List}, \mathsf{Prog}$.

Query. Upon receiving $(\textsc{Query}, \mathsf{sid}, x)$ from a party $P_i \in \mathcal{P}$, or the adversary $\mathcal{S}$:

- Check if $\exists\, v \in \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}$ such that $(\mathsf{sid}, x, v) \in \mathsf{List}$. If not, select $v \leftarrow \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}$ and record the tuple (sid, x, v) in List.
- Return $(\textsc{QueryConfirm}, \mathsf{sid}, v)$ to the requestor.

Program. Upon receiving $(\textsc{Program}, \mathsf{sid}, x, v)$ with $v \in \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}$ from the adversary $\mathcal{S}$:

- Check if $\exists\, v' \in \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}$ s.t. $(\mathsf{sid}, x, v') \in \mathsf{List}$ and $v \neq v'$. If so, ignore this input.
- Set $\mathsf{List} := \mathsf{List} \cup \{(\mathsf{sid}, x, v)\}$ and $\mathsf{Prog} := \mathsf{Prog} \cup \{(\mathsf{sid}, x)\}$.
- Return $(\textsc{ProgramConfirm}, \mathsf{sid})$ to $\mathcal{S}$.

IsPrograMed. Upon receiving $(\textsc{IsPrograMed}, \mathsf{sid}', x)$ from $P_i \in \mathcal{P}$ where $P_i = (\mathsf{pid}, \mathsf{sid})$, or the adversary $\mathcal{S}$:

- If the input was given by $P_i = (\mathsf{pid}, \mathsf{sid})$ and $\mathsf{sid} \neq \mathsf{sid}'$, ignore this input.
- If $(\mathsf{sid}', x) \in \mathsf{Prog}$, set $b := 1$; otherwise, set $b := 0$.
- Return $(\textsc{IsPrograMed}, \mathsf{sid}', b)$ to the requester.

Figure 12: The Global Restricted Programmable Random Oracle Model $\mathcal{G}_{\mathsf{roRO}}$

2.6 Computational Assumptions

Recall that, throughout the work, we let q be a λ-bit prime, and $p = 2q + 1$ also be a prime. We also let $\mathbb{G}$ be a subgroup of order q of $\mathbb{Z}_p^*$ with the generator g. We then present the formal descriptions about the computation assumptions that will be used in this work, i.e., Computational Diffie-Hellman (CDH) assumption and Decisional Diffie-Hellman (DDH) assumption [DH76], as follows.

Definition 13 (Computational Diffie-Hellman Assumption)**.** *We say that the Computational Diffie-Hellman (CDH) assumption holds in a group $\mathbb{G}$ if there is a negligible function* negl *such that for any PPT adversary $\mathcal{A}$, we have*

$$\Pr[\mathcal{A}(\mathbb{G}, p, q, g, g^r, g^s) = g^{rs}] \leq \mathsf{negl}(\lambda)$$

where $r, s \leftarrow \mathbb{Z}_q$.

Definition 14 (Decisional Diffie-Hellman Assumption)**.** *We say that the Decisional Diffie-Hellman (DDH) assumption holds in a group $\mathbb{G}$ if there is a negligible function* negl *such that for any PPT adversary $\mathcal{A}$, we have*

$$|\Pr[\mathcal{A}(\mathbb{G}, p, q, g^r, g^s, g^{rs}) = 1] - \Pr[\mathcal{A}(\mathbb{G}, p, q, g, g^r, g^s, Z) = 1]| \leq \mathsf{negl}(\lambda)$$

where $Z \leftarrow \mathbb{G}$ and $r, s \leftarrow \mathbb{Z}_q$.

3 UC-Secure Endemic OT via Random Oracles

In this section, we provide a new 1-round UC-secure EOT protocol construction under standard assumptions in the RO model. Note that, in our protocol construction, we make use of the well-known ElGamal encryption [ElG84], and we refer interesting readers to see formal description about it in Appendix A.

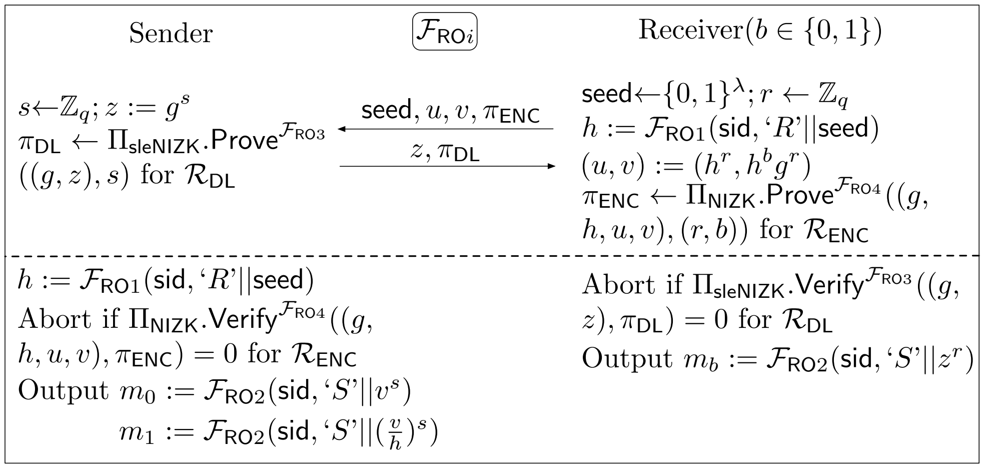
We start with the two-round standalone EOT protocol in the RO model proposed in [CO15]: in the first round, the sender sends $h := g^s$ using $s \leftarrow \mathbb{Z}_q$; in the second round, the receiver uses sender's message to compute $B := g^r h^b$ based on its choice bit b and its secret randomness $r \leftarrow \mathbb{Z}_q$; finally, the sender computes and outputs $m_0 := \mathcal{F}_{\mathsf{RO}}(B^s)$ and $m_1 := \mathcal{F}_{\mathsf{RO}}((\frac{B}{h})^s)$ while the receiver outputs $m_b := \mathcal{F}_{\mathsf{RO}}(h^r)$. Here we use to notation $y := \mathcal{F}_{\mathsf{RO}}(x)$ to describe the process for querying x to the random oracle $\mathcal{F}_{\mathsf{RO}}$ and obtaining the output y, which aligns with the notation in [CSW20]. Since B can be viewed as the Pedersen commitment of the choice bit b, b is perfectly hidden in B and thus a malicious sender cannot learn b from the receiver's message. On the other hand, since the sender's randomness s is kept secret from the receiver, a malicious receiver cannot compute both messages as it requires computing h^s (as it involves querying $B^s, (\frac{B}{h})^s$ to the RO). Such a computation is hard by CDH assumption. This completes the standalone security of this protocol. Although this protocol is very simple and efficient, it cannot achieve UC security [LM18, GIR20].

Our goals are: (i) reduce the round complexity of this protocol to one simultaneous round; (ii) add new mechanisms to make this protocol UC-secure. In order to reduce the round complexity, we let the receiver generate h by invoking the RO on a randomly sampled string seed. In this way, the receiver can compute its message without the sender's message, thus only one simultaneous round is needed. This technique can be found in [CSW20].

We then discuss how to provide UC security. The UC-secure EOT protocol requires *extractability*: (i) when the sender is malicious, the simulator should be able to extract the sender's secret randomness, so the simulator can compute both m_0 and m_1; (ii) when the receiver is malicious, the simulator should be able to extract the receiver's choice bit b. In order to extract the sender's secret randomness s, we let the sender additionally generate a straight-line extractable NIZK argument [Pas03, Fis05, Ks22] of s such that $z = g^s$. The straight-line extractability relies on the observability of the RO model. In this way, the simulator can extract the malicious sender's secret randomness. In order to extract the receiver's choice bit b, we let the receiver generate an ElGamal encryption of b instead of a Pedersen commitment of b, i.e., the receiver computes $(u, v) := (h^r, h^b g^r)$ using $r \leftarrow \mathbb{Z}_q$. We also let the receiver generate a NIZK argument of (b, r) such that $(u, v) = (h^r, h^b g^r)$ to ensure that (u, v) is an ElGamal encryption of a bit b. In this way, the simulator knows $\log_g h$ by making use of the programmability of the RO model, and thus be able to extract b from (u, v).

Let g be the generator of $\mathbb{G}$. Let $\mathcal{F}_{\mathsf{RO1}} : \{0,1\}^* \to \mathbb{G}$ and $\mathcal{F}_{\mathsf{RO2}} : \{0,1\}^* \to \{0,1\}^\lambda$ be random oracles. Let $\mathcal{R}_{\mathsf{ENC}} := \{((g, h, u, v), (r, b)) \mid (b = 0 \wedge (u, v) = (h^r, g^r)) \vee (b = 1 \wedge (u, v) = (h^r, g^r h))\}$ and $\mathcal{R}_{\mathsf{DL}} := \{((g, z), s) \mid z = g^s\}$. We denote by Π_{sleNIZK} the straight-line extractable NIZK argument in the $\mathcal{F}_{\mathsf{RO3}}$-hybrid world. We denote by Π_{NIZK} the NIZK argument in the $\mathcal{F}_{\mathsf{RO4}}$-hybrid world. We note that, the domain and range of $\mathcal{F}_{\mathsf{RO3}}$ and $\mathcal{F}_{\mathsf{RO4}}$ depend on the concrete instantiations of the protocols, for that reason, we do not write them explicitly. Here we assume the synchronous channel $\mathcal{F}_{\mathsf{Syn}}$ is available to the protocol players.

Protocol Description. Our protocol $\Pi_{\mathsf{EOT\text{-}RO}}$ works as follows. We first present the case where both sender and receiver are honest in Figure 13. More precisely, the sender computes $z := g^s$ where g is a common group generator and $s \leftarrow \mathbb{Z}_q$ is randomly sampled, then the sender generates a proof $\pi_{\mathsf{DL}} \leftarrow \Pi_{\mathsf{sleNIZK}}.\mathsf{Prove}^{\mathcal{F}_{\mathsf{RO3}}}((g, z), s)$ for relation $\mathcal{R}_{\mathsf{DL}}$. After that, the sender sends z, π_{DL} to the receiver via $\mathcal{F}_{\mathsf{Syn}}$. At the same time, the receiver samples $\mathsf{seed} \leftarrow \{0,1\}^\lambda$ and $r \leftarrow \mathbb{Z}_q$, then computes $h := \mathcal{F}_{\mathsf{RO1}}(\mathsf{sid}, `R'||\mathsf{seed})$ where 'R' is the public identifier of the receiver, and computes $u := h^r, v := h^b g^r$, finally generates a proof $\pi_{\mathsf{ENC}} \leftarrow \Pi_{\mathsf{NIZK}}.\mathsf{Prove}^{\mathcal{F}_{\mathsf{RO4}}}((g, h, u, v), (r, b))$ for $\mathcal{R}_{\mathsf{ENC}}$. The receiver then sends $(\mathsf{seed}, u, v, \pi_{\mathsf{ENC}})$ to the sender via $\mathcal{F}_{\mathsf{Syn}}$. After querying $\mathcal{F}_{\mathsf{Syn}}$ and obtaining $(\mathsf{seed}, u, v, \pi_{\mathsf{ENC}})$, the sender computes $h := \mathcal{F}_{\mathsf{RO1}}(\mathsf{sid}, `R'||\mathsf{seed})$, and checks if $\Pi_{\mathsf{NIZK}}.\mathsf{Verify}^{\mathcal{F}_{\mathsf{RO4}}}((g, h, u, v), \pi_{\mathsf{ENC}}) = 1$ for $\mathcal{R}_{\mathsf{ENC}}$. If so, the sender computes and outputs $m_0 := \mathcal{F}_{\mathsf{RO2}}(\mathsf{sid}, `S'||v^s)$ and $m_1 := \mathcal{F}_{\mathsf{RO2}}(\mathsf{sid}, `S'||(\frac{v}{h})^s)$ where 'S' is the public identifier of the sender; otherwise, the sender simply aborts. After querying $\mathcal{F}_{\mathsf{Syn}}$ and obtaining (z, π_{DL}), the receiver first checks if $\Pi_{\mathsf{sleNIZK}}.\mathsf{Verify}^{\mathcal{F}_{\mathsf{RO3}}}((g, z), \pi_{\mathsf{DL}}) = 1$ for $\mathcal{R}_{\mathsf{DL}}$. If so, the receiver computes $m_b := \mathcal{F}_{\mathsf{RO2}}(\mathsf{sid}, `S'||z^r)$ and outputs (b, m_b); otherwise, the receiver simply aborts. Note that, if the honest sender (resp. receiver) cannot obtain the desired message form $\mathcal{F}_{\mathsf{Syn}}$, the honest party simply aborts. Then we talk about the malicious cases. When the sender (resp. receiver) is statically corrupted and the receiver (resp. sender) is honest, after sending its message to $\mathcal{F}_{\mathsf{Syn}}$ and waiting for a long time, the honest receiver (resp. sender) will query $\mathcal{F}_{\mathsf{Syn}}$ to obtain the other party's message. If $\mathcal{F}_{\mathsf{Syn}}$ replies the desired message which is in the correct form, the honest party will compute and output the local message according to Figure 13; otherwise, the honest party simply aborts. The security of the protocol has been stated in Theorem 1.



Sender	$\boxed{\mathcal{F}_{\mathsf{RO}i}}$	Receiver$(b \in \{0,1\})$
$s \leftarrow \mathbb{Z}_q; z := g^s$ $\pi_{\mathsf{DL}} \leftarrow \Pi_{\mathsf{sleNIZK}}.\mathsf{Prove}^{\mathcal{F}_{\mathsf{RO3}}}$ $((g, z), s)$ for $\mathcal{R}_{\mathsf{DL}}$	$\xleftarrow{\mathsf{seed}, u, v, \pi_{\mathsf{ENC}}}$ $\xrightarrow{z, \pi_{\mathsf{DL}}}$	$\mathsf{seed} \leftarrow \{0,1\}^\lambda; r \leftarrow \mathbb{Z}_q$ $h := \mathcal{F}_{\mathsf{RO1}}(\mathsf{sid}, `R'||\mathsf{seed})$ $(u, v) := (h^r, h^b g^r)$ $\pi_{\mathsf{ENC}} \leftarrow \Pi_{\mathsf{NIZK}}.\mathsf{Prove}^{\mathcal{F}_{\mathsf{RO4}}}((g,$ $h, u, v), (r, b))$ for $\mathcal{R}_{\mathsf{ENC}}$
$h := \mathcal{F}_{\mathsf{RO1}}(\mathsf{sid}, `R'||\mathsf{seed})$ Abort if $\Pi_{\mathsf{NIZK}}.\mathsf{Verify}^{\mathcal{F}_{\mathsf{RO4}}}((g,$ $h, u, v), \pi_{\mathsf{ENC}}) = 0$ for $\mathcal{R}_{\mathsf{ENC}}$ Output $m_0 := \mathcal{F}_{\mathsf{RO2}}(\mathsf{sid}, `S'||v^s)$ $m_1 := \mathcal{F}_{\mathsf{RO2}}(\mathsf{sid}, `S'||(\frac{v}{h})^s)$		Abort if $\Pi_{\mathsf{sleNIZK}}.\mathsf{Verify}^{\mathcal{F}_{\mathsf{RO3}}}((g,$ $z), \pi_{\mathsf{DL}}) = 0$ for $\mathcal{R}_{\mathsf{DL}}$ Output $m_b := \mathcal{F}_{\mathsf{RO2}}(\mathsf{sid}, `S'||z^r)$

Figure 13: 1-round EOT protocol $\Pi_{\mathsf{EOT\text{-}RO}}$ in the $\{\mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{Syn}}\}$-hybrid world, where $\mathcal{F}_{\mathsf{RO}} = \{\mathcal{F}_{\mathsf{RO}i}\}_{i\in[4]}$. Let g be the generator of $\mathbb{G}$. Let $\mathcal{F}_{\mathsf{RO1}} : \{0,1\}^* \to \mathbb{G}$ and $\mathcal{F}_{\mathsf{RO2}} : \{0,1\}^* \to \{0,1\}^\lambda$ be random oracles. Let $\mathcal{R}_{\mathsf{ENC}} := \{((g, h, u, v), (r, b)) \mid (b = 0 \wedge (u, v) = (h^r, g^r)) \vee (b = 1 \wedge (u, v) = (h^r, g^r h))\}$ and $\mathcal{R}_{\mathsf{DL}} := \{((g, z), s) \mid z = g^s\}$ be NP relations.

Theorem 1. *Assume the DDH assumption holds in group $\mathbb{G}$. Let $\mathcal{F}_{\mathsf{RO1}} : \{0,1\}^* \to \mathbb{G}$ and $\mathcal{F}_{\mathsf{RO2}} : \{0,1\}^* \to \{0,1\}^\lambda$ be the random oracles. Let Π_{NIZK} be an NIZK argument in the $\mathcal{F}_{\mathsf{RO3}}$-hybrid world. Let Π_{sleNIZK} be a straight-line extractable NIZK argument in the $\mathcal{F}_{\mathsf{RO4}}$-hybrid world. The protocol $\Pi_{\mathsf{EOT\text{-}RO}}$ depicted in Figure 13 UC-realizes the functionality $\mathcal{F}_{\mathsf{EOT}}$ depicted in Figure 8 in the $\{\mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{Syn}}\}$-hybrid world against static malicious corruption, where $\mathcal{F}_{\mathsf{RO}} = \{\mathcal{F}_{\mathsf{RO}i}\}_{i\in[4]}$.*

Proof. We leave the formal proof in Appendix C.1. □

Instantiation. We instantiate Π_{sleNIZK} for relation $\mathcal{R}_{\mathsf{DL}}$ with the Schnorr's protocol [Sch90] and the *randomized Fischlin transform* [Ks22] which improves the efficiency and applicability of Fischlin transform [Fis05]. We present the formal description of the Schnorr's protocol and the randomized Fischlin transform in Appendix B.1 and Appendix B.2 respectively.

We instantiate Π_{NIZK} for relation $\mathcal{R}_{\mathsf{ENC}}$ with the following techniques: we first employ the OR-composition [CDS94] to the Chaum-Pedersen protocols [CP93] to prove either (g, h, v, u) is a DDH tuple (which means $b = 0$) or $(g, h, \frac{v}{h}, u)$ is a DDH tuple (which means $b = 1$), we then apply the the Fiat-Shamir transform [FS87] to remove the interaction. We present the formal description of the Chaum-Pedersen protocols and the OR-composition of the Sigma-protocols in Appendix B.1.

Efficiency. Here we compare the efficiency in the amortized setting where the sender and the receiver can reuse some elements for multiple instances of the EOT protocol (in this protocol, the sender can reuse s, π_{DL} while the receiver can reuse the string seed). The amortized setting is also used in [CSW20] for efficiency comparison. By taking the parameter (that achieves 128-bit security) from [Ks22], our protocol $\Pi_{\mathsf{EOT\text{-}RO}}$ requires 18 exponentiations w.r.t. computation and 10 group/field elements w.r.t. communication; while the state-of-the-art 1-round UC-secure RO-based protocol in [MR19a] requires 4 exponentiations w.r.t. computation and 2 group elements w.r.t. communication. Note that, our protocol is based on a standard assumption; whereas the protocol in [MR19a] is based on a non-standard assumption.

4 The Relations between Endemic OT and Other Primitives

In this section, we first show how to construct a bit commitment protocol via EOT with unconditional security. Subsequently, we complete the picture of OT relations in [MR19a], showing that UOT can be constructed via EOT with unconditional security.

4.1 From Endemic OT to Commitment

It is known that bit commitment can be constructed via 1-out-of-2 OT with unconditional security [Kil88, BFSK11]. What about EOT? Inspired by Brzuska et al.'s construction [BFSK11], here we show that bit commitment can also be constructed via a weaker primitive, i.e., EOT, with unconditional security.

We observe that the receiver's message can be viewed as the commitment to the receiver's choice bit b, and the locally computed message m_b together with b can be viewed as the opening. Typically, a commitment protocol requires two properties: hiding and binding. The hiding property comes from the fact: even if the malicious EOT receiver can influence the distribution of m_b, it cannot learn the other message m_{1-b}. The binding property comes from the fact: even if the malicious EOT sender can influence the distributions of both m_0 and m_1, it cannot tell which one is received by the receiver. Furthermore, if we use a UC-secure EOT protocol as the building block, the resulting commitment protocol is also UC-secure. Note that, we only assume authenticated channel $\mathcal{F}_{\mathsf{Auth}}$ is available to the protocol players.

Protocol Description. Our protocol Π_{Com} works as follows. We first present the case where both committer and receiver are honest in Figure 14. More precisely, in the committing phase, the committer acts as the EOT receiver and uses RECEIVE command to send b to $\mathcal{F}_{\mathsf{EOT}}$ and the receiver acts as the EOT sender and sends SEND command to $\mathcal{F}_{\mathsf{EOT}}$, then the committer obtains m_b from $\mathcal{F}_{\mathsf{EOT}}$ and the receiver obtains m_0, m_1 from $\mathcal{F}_{\mathsf{EOT}}$. Note that, if the honest committer (resp. receiver) obtains ABORT from $\mathcal{F}_{\mathsf{EOT}}$, the honest party simply aborts. In the opening phase, the committer simply sets $m := m_b$ and sends b, m to the receiver via $\mathcal{F}_{\mathsf{Auth}}$. If the receiver obtains the desired message from $\mathcal{F}_{\mathsf{Auth}}$, the receiver will accept b if and only if $m = m_b$; otherwise, the receiver simply aborts. We then discuss the case where one party is statically corrupted and the other one is honest. We will describe the case where the committer is statically corrupted and the receiver is honest, and omit the case where the receiver is statically corrupted and the committer is honest since it is similar. When the committer is statically corrupted and the receiver is honest, in the committing phase, the honest receiver sends its message to $\mathcal{F}_{\mathsf{EOT}}$ and waits for $\mathcal{F}_{\mathsf{EOT}}$ to response. If $\mathcal{F}_{\mathsf{EOT}}$ replies m_0, m_1, the honest receiver continues the protocol; otherwise (i.e., $\mathcal{F}_{\mathsf{EOT}}$ replies ABORT), the honest receiver simply aborts. In the opening phase, if honest receiver receives the corrupted committer's message from $\mathcal{F}_{\mathsf{Auth}}$ which is in the correct form, the honest receiver will complete the protocol according to Figure 14; otherwise, the honest receiver simply aborts. The security of the protocol has been stated in Theorem 2.

Theorem 2. *The protocol Π_{Com} depicted in Figure 14 UC-realizes the functionality $\mathcal{F}_{\mathsf{Com}}$ depicted in Figure 9 with unconditional security in the $\{\mathcal{F}_{\mathsf{EOT}}, \mathcal{F}_{\mathsf{Auth}}\}$-hybrid world against static malicious corruption.*

Proof. We leave the formal proof in Appendix C.2. □

4.2 From Endemic OT to Uniform OT

In [MR19a], the Masny and Rindal showed how to construct UOT with unconditional security in the $\{\mathcal{F}_{\mathsf{EOT}}, \mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{Auth}}\}$-hybrid world, and we recall their protocol construction in Figure 15. However, they only showed how to construct the coin-tossing protocol via UOT. Therefore, whether EOT implies UOT remains an open question.

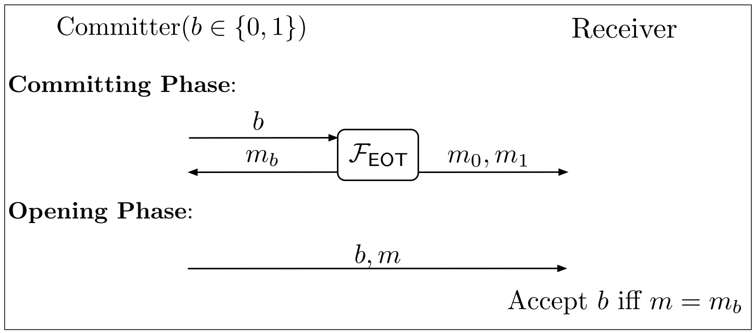


Figure 14: Bit Commitment Protocol Π_{Com} in the $\{\mathcal{F}_{\mathsf{EOT}}, \mathcal{F}_{\mathsf{Auth}}\}$-Hybrid World

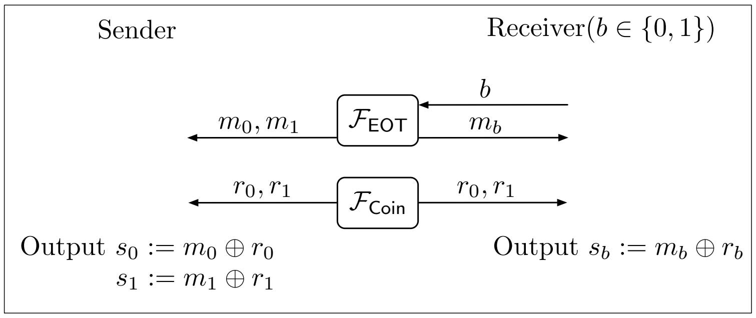

Sender
Receiver($b \in \{0,1\}$)
b
m_0, m_1
$\mathcal{F}_{\mathsf{EOT}}$
m_b
r_0, r_1
$\mathcal{F}_{\mathsf{Coin}}$
r_0, r_1
Output $s_0 := m_0 \oplus r_0$
$s_1 := m_1 \oplus r_1$
Output $s_b := m_b \oplus r_b$

Figure 15: UOT Protocol Π_{UOT} in the $\{\mathcal{F}_{\mathsf{EOT}}, \mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{Auth}}\}$-Hybrid World from [MR19a]

Lemma 1 ([MR19a])**.** *The protocol Π_{UOT} depicted in Figure 15 UC-realizes $\mathcal{F}_{\mathsf{UOT}}$ depicted in Figure 7 with unconditional security in the $\{\mathcal{F}_{\mathsf{EOT}}, \mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{Auth}}\}$-hybrid world against static malicious corruption.*

In this section, we provide a positive answer to this unsolved question. Our solution is as follows: we have showed that EOT implies commitment in Section 4.1, and the coin-tossing protocol can be easily constructed via only commitment; putting things together, we show that EOT implies UOT.

Protocol Description. The protocol Π_{Coin} works as follows. We first present the case where both player 1 and player 2 are honest in Figure 16. More precisely, the player 1 samples a uniformly random string m_1 and sends m_1 to $\mathcal{F}_{\mathsf{Com}}$ in the first round. If the player 2 receives RECEIPT from $\mathcal{F}_{\mathsf{Com}}$, the player 2 samples a uniformly random string m_2 to the player 1 via $\mathcal{F}_{\mathsf{Auth}}$; otherwise, the player 2 simply aborts. If the player obtains m_1 from $\mathcal{F}_{\mathsf{Auth}}$, the player 1 sends DECOMMIT command to $\mathcal{F}_{\mathsf{Com}}$ and outputs $m := m_1 \oplus m_2$; otherwise, the player 1 simply aborts. If the player 2 obtains m_2 from $\mathcal{F}_{\mathsf{Auth}}$, the player 2 outputs $m := m_1 \oplus m_2$. We then discuss the case where one party is statically corrupted and the other one is honest. We will describe the case where the player 2 is statically corrupted and the player 1 is honest, and omit the other case since it is similar. When the player 2 is statically corrupted and the player 1 is honest, the honest player 1 will execute the protocol according to Figure 16, except that whenever $\mathcal{F}_{\mathsf{Com}}$ sends ABORT to the player 1 or the player 1 does not receive the player 2's message from $\mathcal{F}_{\mathsf{Auth}}$ in the second round, the player 1 simply aborts. The security of the protocol has been stated in Theorem 3.

Theorem 3. *The protocol Π_{Coin} depicted in Figure 16 UC-realizes the functionality $\mathcal{F}_{\mathsf{Coin}}$ depicted in Figure 5 with unconditional security in the $\{\mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{Auth}}\}$-hybrid world against static malicious corruption.*

Proof. We leave the formal proof in Appendix C.3. □

Formally, we prove that EOT implies UOT through Corollary 1. The security proof of Corollary 1 directly comes from Lemma 1, Theorem 2 and Theorem 3, and thus we omit the trivial proof here.

Corollary 1. *The protocol Π_{UOT} depicted in Figure 15 UC-realizes $\mathcal{F}_{\mathsf{UOT}}$ depicted in Figure 7 with unconditional security in the $\{\mathcal{F}_{\mathsf{EOT}}, \mathcal{F}_{\mathsf{Auth}}\}$-hybrid world against static malicious corruption.*

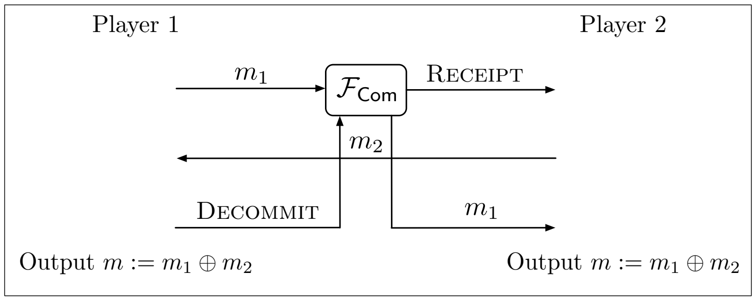

Player 1
Player 2
m_1
$\mathcal{F}_{\mathsf{Com}}$
RECEIPT
m_2
DECOMMIT
m_1
Output $m := m_1 \oplus m_2$
Output $m := m_1 \oplus m_2$

Figure 16: Coin-Tossing Protocol Π_{Coin} in the $\{\mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{Auth}}\}$-Hybrid World

5 GUC-Secure Endemic OT via Global Random Oracles

In this section, we turn to global RO models to seek a stronger variant of UC security, i.e., GUC security. As for the GroRO model, we construct the *first* 1-round GUC-secure EOT protocol under CDH assumption against static adversaries. Using our 1-round GUC-secure EOT protocol as the main building block, we propose the *first* 2-round GUC-secure commitment protocol in the GroRO model.

Regarding the GrpRO model, we prove that there exists *no* 1-round GUC-secure EOT protocol in the GrpRO model even with static security. By combining this negative result in the GrpRO model and the positive result in the GroRO model, we reveal a separation between these two models. Furthermore, we construct the *first* 2-round (round-optimal) GUC-secure EOT protocol under DDH assumption in the GrpRO model against adaptive adversaries.

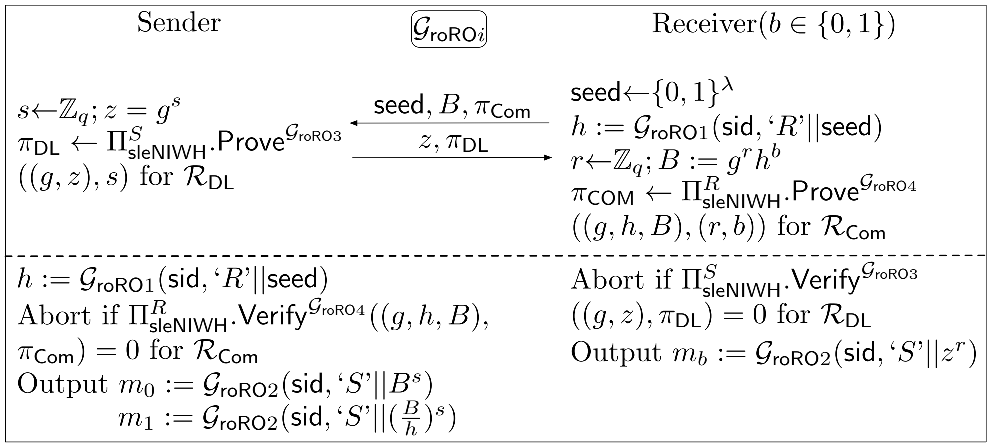
5.1 Feasibility Results in the GroRO Model

5.1.1 Our EOT Protocol

We start with our UC-secure EOT protocol $\Pi_{\mathsf{EOT\text{-}RO}}$ depicted in Figure 13. Recall that, we let the sender send $z := g^s$ using $s \leftarrow \mathbb{Z}_q$, together with a straight-line extractable NIZK argument of s such that $z = g^s$ in $\Pi_{\mathsf{EOT\text{-}RO}}$. The straight-line extractable NIZK argument gives the simulator chance of extracting the sender's secret randomness. However, Pass showed that it is impossible to construct NIZK arguments in observable RO model [Pas03], let alone NIZK arguments with straight-line extractability. The good news is that we find that straight-line extractable NIWH argument is sufficient for our purpose, and it is possible in the GroRO model [Pas03]. Therefore, we let the sender generate a straight-line extractable NIWH argument of s such that $z = g^s$. Now let us consider the receiver. In order to extract the receiver's choice bit, we make full use of the programmability of random oracles in $\Pi_{\mathsf{EOT\text{-}RO}}$. Since $\mathcal{G}_{\mathsf{roRO}}$ does not permit anyone to program the random oracle, we need to take a different strategy: we let the receiver generate h by invoking the $\mathcal{G}_{\mathsf{roRO}}$ on a randomly sampled string seed, compute a Pedersen commitment to the choice bit $B := g^r h^b$ using $r \leftarrow \mathbb{Z}_q$, and generate a straight-line extractable NIWH argument of (r, b) such that $B = g^r h^b$. Analogously to the sender side, the simulator can extract the malicious receiver's choice bit b.

Let g be the generator of $\mathbb{G}$. Let $\mathcal{G}_{\mathsf{roRO1}} : \{0,1\}^* \to \mathbb{G}$ and $\mathcal{G}_{\mathsf{roRO2}} : \{0,1\}^* \to \{0,1\}^\lambda$. Let $\mathcal{R}_{\mathsf{Com}} := \{((g, h, B), (r, b)) \mid B = g^r h^b\}$ and $\mathcal{R}_{\mathsf{DL}} := \{((g, z), s) \mid z = g^s\}$. We denote by Π^S_{sleNIWH} the straight-line extractable NIWH argument in the $\mathcal{G}_{\mathsf{roRO3}}$-hybrid world which is used for generating the proof by sender. We denote by Π^R_{sleNIWH} the straight-line extractable NIWH argument in the $\mathcal{G}_{\mathsf{roRO4}}$-hybrid world which is used for generating the proof by receiver. We assume synchronous channel $\mathcal{F}_{\mathsf{Syn}}$ is available to the protocol players..

Protocol Description. Our protocol $\Pi_{\mathsf{EOT\text{-}GroRO}}$ works as follows. We first present the case where both sender and receiver are honest in Figure 17. More precisely, the sender computes $z := g^s$ where g is a common group generator and $s \leftarrow \mathbb{Z}_q$, then the sender generates a proof $\pi_{\mathsf{DL}} \leftarrow \Pi_{\mathsf{sleNIWH}}.\mathsf{Prove}^{\mathcal{G}_{\mathsf{roRO3}}}((g, z), s)$ for relation $\mathcal{R}_{\mathsf{DL}}$. After that, the sender sends (z, π_{DL}) to the receiver via $\mathcal{F}_{\mathsf{Syn}}$. At the same time, the receiver selects seed$\leftarrow \{0,1\}^\lambda$, and computes $h := \mathcal{G}_{\mathsf{roRO1}}(\mathsf{sid}, `R'||\mathsf{seed})$. Then the receiver computes $B := g^r h^b$ where g is a common group generator and $r \leftarrow \mathbb{Z}_q$, and generates a proof $\pi_{\mathsf{Com}} \leftarrow \Pi_{\mathsf{sleNIWH}}.\mathsf{Prove}^{\mathcal{G}_{\mathsf{roRO4}}}((g, h, B), (r, b))$ for relation $\mathcal{R}_{\mathsf{Com}}$. After that, the receiver sends $(\mathsf{seed}, B, \pi_{\mathsf{Com}})$ to the sender via $\mathcal{F}_{\mathsf{Syn}}$. After querying $\mathcal{F}_{\mathsf{Syn}}$ and obtaining $(\mathsf{seed}, B, \pi_{\mathsf{Com}})$, the sender computes $h := \mathcal{G}_{\mathsf{roRO1}}(\mathsf{sid}, `R'||\mathsf{seed})$ where 'R' is the public identifier of the receiver. Then the sender checks if $\Pi_{\mathsf{sleNIWH}}.\mathsf{Verify}^{\mathcal{G}_{\mathsf{roRO4}}}((g, h, B), \pi_{\mathsf{Com}}) = 1$ for $\mathcal{R}_{\mathsf{Com}}$. If so, the sender outputs $m_0 := \mathcal{G}_{\mathsf{roRO2}}(\mathsf{sid}, `S'||B^s)$ and



Sender
$\boxed{\mathcal{G}_{\mathsf{roRO}i}}$
Receiver$(b \in \{0,1\})$
$s \leftarrow \mathbb{Z}_q; z = g^s$
$\pi_{\mathsf{DL}} \leftarrow \Pi^S_{\mathsf{sleNIWH}}.\mathsf{Prove}^{\mathcal{G}_{\mathsf{roRO3}}}$
$((g,z),s)$ for $\mathcal{R}_{\mathsf{DL}}$
$\mathsf{seed}, B, \pi_{\mathsf{Com}}$
z, π_{DL}
$\mathsf{seed} \leftarrow \{0,1\}^\lambda$
$h := \mathcal{G}_{\mathsf{roRO1}}(\mathsf{sid}, `R' || \mathsf{seed})$
$r \leftarrow \mathbb{Z}_q; B := g^r h^b$
$\pi_{\mathsf{COM}} \leftarrow \Pi^R_{\mathsf{sleNIWH}}.\mathsf{Prove}^{\mathcal{G}_{\mathsf{roRO4}}}$
$((g,h,B),(r,b))$ for $\mathcal{R}_{\mathsf{Com}}$
$h := \mathcal{G}_{\mathsf{roRO1}}(\mathsf{sid}, `R' || \mathsf{seed})$
Abort if $\Pi^R_{\mathsf{sleNIWH}}.\mathsf{Verify}^{\mathcal{G}_{\mathsf{roRO4}}}((g,h,B),$
$\pi_{\mathsf{Com}}) = 0$ for $\mathcal{R}_{\mathsf{Com}}$
Output $m_0 := \mathcal{G}_{\mathsf{roRO2}}(\mathsf{sid}, `S' || B^s)$
$m_1 := \mathcal{G}_{\mathsf{roRO2}}(\mathsf{sid}, `S' || (\frac{B}{h})^s)$
Abort if $\Pi^S_{\mathsf{sleNIWH}}.\mathsf{Verify}^{\mathcal{G}_{\mathsf{roRO3}}}$
$((g,z),\pi_{\mathsf{DL}}) = 0$ for $\mathcal{R}_{\mathsf{DL}}$
Output $m_b := \mathcal{G}_{\mathsf{roRO2}}(\mathsf{sid}, `S' || z^r)$

Figure 17: 1-round EOT protocol $\Pi_{\mathsf{EOT\text{-}GroRO}}$ in the $\{\mathcal{G}_{\mathsf{roRO}}, \mathcal{F}_{\mathsf{Syn}}\}$-hybrid world, where $\mathcal{G}_{\mathsf{roRO}} = \{\mathcal{G}_{\mathsf{roRO}i}\}_{i\in[4]}$. Let g be the generator of $\mathbb{G}$. Let $\mathcal{G}_{\mathsf{roRO1}} : \{0,1\}^* \to \mathbb{G}$ and $\mathcal{G}_{\mathsf{roRO2}} : \{0,1\}^* \to \{0,1\}^\lambda$ be random oracles. Let $\mathcal{R}_{\mathsf{Com}} := \{((g,h,B),(r,b)) \mid B = g^r h^b\}$ and $\mathcal{R}_{\mathsf{DL}} := \{((g,z),s) \mid z = g^s\}$ be NP relations.

$m_1 := \mathcal{G}_{\mathsf{roRO2}}(\mathsf{sid}, `S' || (\frac{B}{h})^s)$ where 'S' is the public identifier of the sender. After querying $\mathcal{F}_{\mathsf{Syn}}$ and obtaining (z, π_{DL}), the receiver checks if $\Pi_{\mathsf{sleNIWH}}.\mathsf{Verify}^{\mathcal{G}_{\mathsf{roRO3}}}((g,z),\pi_{\mathsf{DL}}) = 1$ for $\mathcal{R}_{\mathsf{DL}}$. If so, the receiver simply computes $m_b := \mathcal{G}_{\mathsf{roRO2}}(\mathsf{sid}, `S' || z^r)$ and outputs (b, m_b). Note that, if the honest sender (resp. receiver) cannot obtain the desired message form $\mathcal{F}_{\mathsf{Syn}}$, the honest party simply aborts. Then we talk about the malicious cases. When sender (resp. receiver) is statically corrupted and receiver (resp. sender) is honest, after sending its message to $\mathcal{F}_{\mathsf{Syn}}$ and waiting for a long time, the honest receiver (resp. sender) will query $\mathcal{F}_{\mathsf{Syn}}$ to obtain the other party's message. If $\mathcal{F}_{\mathsf{Syn}}$ replies the desired message which is in the correct form, the honest party will compute and output the local message according to Figure 17; otherwise, the honest party simply aborts. The security of the protocol has been stated in Theorem 4.

Before giving the theorem, we have to give the transferable EOT functionality $\mathcal{F}_{\mathsf{tEOT}}$ in Figure 18. The main difference with the traditional EOT functionality is that in $\mathcal{F}_{\mathsf{tEOT}}$, the simulator can request the list of illegitimate queries, which fits the $\mathcal{G}_{\mathsf{roRO}}$ model.

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

The functionality interacts with two parties S, R and an adversary $\mathcal{S}$.

Transfer/Choose/Process. Same as $\mathcal{F}_{\mathsf{EOT}}$ depicted in Figure 8.

Observe. When asked by the adversary $\mathcal{S}$, obtain from $\mathcal{G}_{\mathsf{roRO}}$ the list of illegitimate queries $\mathcal{Q}_{\mathsf{sid}}$ that pertain to SID sid, and send $\mathcal{Q}_{\mathsf{sid}}$ to the adversary $\mathcal{S}$.

Figure 18: The Transferable Ideal Functionality $\mathcal{F}_{\mathsf{tEOT}}$ for Endemic Oblivious Transfer in $\mathcal{G}_{\mathsf{roRO}}$ Model

Theorem 4. *Assume the CDH assumption holds in group $\mathbb{G}$. Let $\mathcal{G}_{\mathsf{roRO1}} : \{0,1\}^\lambda \to \mathbb{G}$ and $\mathcal{G}_{\mathsf{roRO2}} : \mathbb{G} \to \{0,1\}^\lambda$ be the random oracles. Let Π^S_{sleNIWH} be a straight-line extractable NIWH argument in the $\mathcal{G}_{\mathsf{roRO3}}$-hybrid world. Let Π^R_{sleNIWH} be a straight-line extractable NIWH argument in the $\mathcal{G}_{\mathsf{roRO4}}$-hybrid world. The protocol $\Pi_{\mathsf{EOT\text{-}GroRO}}$ depicted in Figure 17 GUC-realizes the functionality $\mathcal{F}_{\mathsf{tEOT}}$ depicted in Figure 18 in the $\{\mathcal{G}_{\mathsf{roRO}}, \mathcal{F}_{\mathsf{Syn}}\}$-hybrid world against static malicious corruption, where $\mathcal{G}_{\mathsf{roRO}} = \{\mathcal{G}_{\mathsf{roRO}i}\}_{i\in[4]}$.*

Proof. We leave the formal proof in Appendix C.4. □

Instantiation. We instantiate Π^S_{sleNIZK} for relation $\mathcal{R}_{\mathsf{DL}}$ with the Schnorr's protocol and the randomized Fischlin transform as in Section 3. Note that, although we use the same instantiation as in Section 3, we only obtain a straight-line extractable NIWH argument, since here we use a observable RO model [Pas03].

We instantiate Π^R_{sleNIWH} for relation $\mathcal{R}_{\mathsf{Com}}$ with the Okamoto's protocol [Oka93] and the randomized Fischlin transform. We present the formal description of the Okamoto's protocol and the randomized Fischlin transform in Appendix B.1 and Appendix B.2 respectively.

Efficiency. We consider the efficiency of our GUC-secure protocol $\Pi_{\mathsf{EOT\text{-}GroRO}}$ in the amortized setting here, just like we did in Section 3. By taking the parameter (that achieves 128-bit security) in [Ks22], our GUC-secure protocol $\Pi_{\mathsf{EOT\text{-}GroRO}}$ requires 53 exponentiations w.r.t. computation and 41 group/field elements w.r.t. communication; while the state-of-the-art 2-round GroRO-based OT protocol in [CSW20] requires 5 exponentiations w.r.t. computation and 2 group elements + 2λ bits string w.r.t. communication. The efficiency of our GUC-secure protocol $\Pi_{\mathsf{EOT\text{-}GroRO}}$ is slightly worse than the protocol in [CSW20], but our protocol only requires CDH assumption, whereas the protocol proposed in [CSW20] requires the DDH assumption, which is stronger.

5.1.2 Our Commitment Protocol

At the very beginning, we introduce the transferable commitment functionality $\mathcal{F}_{\mathsf{tCom}}$ from [CJS14] in Figure 19. The main difference with the traditional commitment functionality is that in $\mathcal{F}_{\mathsf{tCom}}$, the simulator can request the list of illegitimate queries, which fits the $\mathcal{G}_{\mathsf{roRO}}$ model.

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

The functionality interacts with two parties S, R and an adversary $\mathcal{S}$.

Commit/Open. Same as $\mathcal{F}_{\mathsf{Com}}$ depicted in Figure 9.

Observe. When asked by the adversary $\mathcal{S}$, obtain from $\mathcal{G}_{\mathsf{roRO}}$ the list of illegitimate queries $\mathcal{Q}_{\mathsf{sid}}$ that pertain to SID sid, and send $\mathcal{Q}_{\mathsf{sid}}$ to the adversary $\mathcal{S}$.

Figure 19: The Transferable Ideal Functionality $\mathcal{F}_{\mathsf{tCom}}$ for Commitment in $\mathcal{G}_{\mathsf{roRO}}$ Model

Recall that, we construct a commitment protocol Π_{Com} depicted in Figure 14 in the $\{\mathcal{F}_{\mathsf{EOT}}, \mathcal{F}_{\mathsf{Auth}}\}$-hybrid world with unconditional security (cf. Section 4.1). It is easy to see that if we replace the ideal functionality $\mathcal{F}_{\mathsf{EOT}}$ with transferable ideal functionality $\mathcal{F}_{\mathsf{tEOT}}$ and call the resulting protocol Π_{tCom}, then the protocol Π_{tCom} will GUC-realize $\mathcal{F}_{\mathsf{tCom}}$ in the $\{\mathcal{F}_{\mathsf{tEOT}}, \mathcal{F}_{\mathsf{Auth}}\}$-hybrid world with unconditional security. Formally, we have the following corollary, and its security proof is analogously to the proof of Theorem 2.

Corollary 2. *The protocol Π_{tCom} GUC-realizes the functionality $\mathcal{F}_{\mathsf{tCom}}$ depicted in Figure 19 with unconditional security in the $\{\mathcal{F}_{\mathsf{tEOT}}, \mathcal{F}_{\mathsf{Auth}}\}$-hybrid world against static malicious corruption.*

Instantiation. We instantiate $\mathcal{F}_{\mathsf{tEOT}}$ with our 1-round GUC-secure EOT protocol depicted in Figure 17. Then we immediate obtain a 2-round GUC-secure commitment protocol Π_{tCom} in the GroRO model; note that, the first round messages are communicated over the synchronous channel $\mathcal{F}_{\mathsf{Syn}}$ and the second round message is communicated over the authenticated channel $\mathcal{F}_{\mathsf{Auth}}$. The security is guaranteed by Theorem 4 and Corollary 2.

Comparison. Our commitment protocol is the *first* 2-round GUC-secure commitment in the GroRO model, while the previous state-of-the-art protocols achieves 3 rounds [MRS17, ZZZR22]. Note that, Zhou *et al.* proved that it is impossible to construct 2-round GUC-secure commitment protocol in the GroRO model even with static security [ZZZR22]; but they did not consider the simultaneous communication model. Our 2-round commitment protocol contains a simultaneous round, so we do not contradict their impossibility result. We also note that, our protocol and protocols in [MRS17, ZZZR22] are all 3-move static-secure protocols, but ours is the only one whose first two moves can be executed in one simultaneous round; hence, ours is the only one that can achieve 2-round. The details of the comparison are presented in Table 2.

Protocol	#Round	Computational Assumption
[MRS17]	3	DL
[ZZZR22]	3	OWF
Π_{tCom}	2	CDH

Table 2: Comparison with state-of-the-art GUC-secure commitment against static adversaries in the GroRO model.

5.2 Impossibility and Feasibility Results in the GrpRO Model

5.2.1 Our Impossibility Result

Here we show that there exists *no* 1-round GUC-secure EOT protocol against static adversaries in the GrpRO model.

We prove this impossibility by contradiction. Suppose that there exists such a 1-round GUC-secure EOT protocol. Let us first consider the case where the receiver is corrupted, and the simulator needs to extract the choice bit of the receiver from its message. Recall that, the $\mathcal{G}_{\mathsf{rpRO}}$ only grants the simulator the restricted programmability: although the simulator can program the unqueried points without being detected, the simulator is external to the $\mathcal{G}_{\mathsf{rpRO}}$ and it can not know in real time what queries other parties are sending to $\mathcal{G}_{\mathsf{rpRO}}$. Thus, the simulator needs to program the points in advance and find a way to enforce the corrupt receiver to generate its message on the simulator's programmed points. In that way, the simulator can have the chance of extracting the choice bit of the receiver. However, in a one simultaneous round protocol, the messages between parties have no dependency. Hence the simulator cannot enforce the corrupt receiver to produce its message on the programmed points, and has no advantages over the real world adversary. If the simulator still succeeds to extract the corrupted receiver's choice bit, then distinctions will be revealed when the adversary performs the following attacks. The adversary corrupts the sender, and instructs the sender to run the simulator algorithm above to extract the choice bit from the message sent by the receiver/simulator. However, the receiver/simulator has no idea about the real choice bit, thus with high probability the simulation would fail. Formally, we prove this impossibility through Theorem 5.

Theorem 5. *There exists no terminating 1-round protocol* Π *that GUC-realizes* $\mathcal{F}_{\mathsf{EOT}}$ *depicted in Figure 8 with static security in the* $\{\mathcal{G}_{\mathsf{rpRO}}, \mathcal{F}_{\mathsf{Syn}}\}$*-hybrid world.*

Proof. We proceed by contradiction. Suppose there exists such a protocol Π that GUC-realizes $\mathcal{F}_{\mathsf{EOT}}$ in the $\{\mathcal{G}_{\mathsf{rpRO}}, \mathcal{F}_{\mathsf{Syn}}\}$-hybrid world. Then there must exist a PPT simulator $\mathcal{S}$ such that $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{rpRO}}}_{\mathcal{F}_{\mathsf{EOT}},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}^{\mathcal{G}_{\mathsf{rpRO}},\mathcal{F}_{\mathsf{Syn}}}_{\Pi,\mathcal{A},\mathcal{Z}}$ for any PPT adversary $\mathcal{A}$ and any PPT $\mathcal{G}_{\mathsf{rpRO}}$-externally constrained environment $\mathcal{Z}$.

In particular, let us first consider the session with SID sid_1, and let $\mathcal{A}$ be a dummy adversary that simply forwards protocol flows between the corrupt party and the environment $\mathcal{Z}$. Our $\mathcal{Z}$ proceeds by corrupting the receiver R^* at first. Then $\mathcal{Z}$ waits for S to send its message π_S. Note that, once the message π_S is given to $\mathcal{F}_{\mathsf{Syn}}$, $\mathcal{A}$ is allowed to see it. After obtaining π_S from $\mathcal{A}$, $\mathcal{Z}$ chooses a random bit $b \leftarrow \{0,1\}$ and instructs R^* to perform the receiver algorithm on input b, and send its message π_R to S. Finally, $\mathcal{Z}$ performs the local computation of R^* to obtain m_b, and waits for S to output m'_0, m'_1 at the end of the protocol. If $m_b = m'_b$, $\mathcal{Z}$ outputs 1; otherwise, $\mathcal{Z}$ outputs 0.

In order to make the GUC experiment above remain indistinguishable, the simulator $\mathcal{S}$ needs to perform the following strategy. First, $\mathcal{S}$ extracts the choice bit b of the receiver from message π_R. Then $\mathcal{S}$ performs the local computation of S to obtain m_0, m_1. Finally, $\mathcal{S}$ sends $(\textsc{Receive}, \mathsf{sid}, b, m_b)$ to $\mathcal{F}_{\mathsf{EOT}}$ on behalf of the dummy corrupted receiver. It is easy to see that the crucial part lies in the extraction of the choice bit b of the receiver. Recall that, the main advantage of $\mathcal{S}$ is that it can program the $\mathcal{G}_{\mathsf{rpRO}}$ on unqueried points without being detected, but the simulator is external to the $\mathcal{G}_{\mathsf{rpRO}}$ and it can not know in real time what queries other parties are sending to $\mathcal{G}_{\mathsf{rpRO}}$. For notation convenience, we denote by $\mathsf{Prog}_{\mathsf{sid}_1}$ the queries programmed by $\mathcal{S}$. If the adversary happens to use the points that belongs to $\mathsf{Prog}_{\mathsf{sid}_1}$, then $\mathcal{S}$ has the chance of extracting the private information of the malicious party. We also note that, the simulator $\mathcal{S}$ also can query $\mathcal{G}_{\mathsf{rpRO}}$ just like other parties. In order to describe the process of querying to $\mathcal{G}_{\mathsf{rpRO}}$, we denote by $\mathcal{G}^*_{\mathsf{rpRO}}$ the simplified version of the $\mathcal{G}_{\mathsf{rpRO}}$, i.e., the $\mathcal{G}_{\mathsf{rpRO}}$ without the $\textsc{Program}$ interface. We write $\mathcal{S}^{\mathcal{G}^*_{\mathsf{rpRO}}}$ to denote the event that $\mathcal{S}$ is given query access to $\mathcal{G}_{\mathsf{rpRO}}$ and can continuously query to $\mathcal{G}_{\mathsf{rpRO}}$. With notations above, we denote by $b \leftarrow \mathcal{S}^{\mathcal{G}^*_{\mathsf{rpRO}}}(\mathsf{sid}_1, \pi_R, \mathsf{Prog}_{\mathsf{sid}_1})$ the event where $\mathcal{S}$ extracts the choice bit b from π_R using $\mathsf{Prog}_{\mathsf{sid}_1}$. Recall that, (i) the simulator $\mathcal{S}$ should be able to handle any PPT adversary $\mathcal{A}$ and any PPT environment $\mathcal{Z}$; (ii) there should be no dependency between π_S and π_R in one simultaneous round protocol, i.e., the computation of π_R is totally independent of π_S. Hence we can consider the following case: the environment $\mathcal{Z}$ queries $\mathcal{G}_{\mathsf{rpRO}}$ everything that will be needed to compute the receiver's message π_R in advance (we denote these queries as $\mathcal{Q}_{\mathsf{sid}_1,\mathcal{Z}}$), then $\mathcal{Z}$ starts the protocol Π and instructs R^* to run the receiver algorithm on those previously queried points. In this case, we have $\Pr[\mathsf{Prog}_{\mathsf{sid}_1} \cap \mathcal{Q}_{\mathsf{sid}_1,\mathcal{Z}} = \emptyset] = 0$ where $\mathcal{Q}_{\mathsf{sid}_1,\mathcal{Z}}$ is the queries used for generating the receiver's message π_R. However, the simulator $\mathcal{S}$ should still be able to extract the choice bit; otherwise, $\mathcal{Z}$ will distinguish the ideal world from the real world. In other words, the algorithm $b \leftarrow \mathcal{S}^{\mathcal{G}^*_{\mathsf{rpRO}}}(\mathsf{sid}_1, \pi_1, \emptyset)$ should work even if we replace the programmed queries $\mathsf{Prog}_{\mathsf{sid}_1}$ with an empty set $\emptyset$. We note that $\mathcal{S}^{\mathcal{G}^*_{\mathsf{rpRO}}}(\mathsf{sid}_1, \pi_R, \emptyset)$ works as long as the appropriate inputs are provided, even if we switch to the session with a different SID.

Next, we show that the existence of the simulator $\mathcal{S}$ above contradicts the security of Π against static corruptions, by creating a particular $\mathcal{Z}'$ which distinguishes $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{rpRO}}}_{\mathcal{F}_{\mathsf{EOT}},\mathcal{S}',\mathcal{Z}'}$ from $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{rpRO}}}_{\Pi,\mathcal{A}',\mathcal{Z}'}$ after a static corruption operation for any PPT simulator $\mathcal{S}'$. Let us consider the session with SID sid_2. We let $\mathcal{Z}'$ corrupt the sender S^* at first. Then $\mathcal{Z}'$ feeds the honest receiver with a random bit b, and waits for the arrival of message π_R. Note that, b is the hidden input of the honest receiver, and the receiver sends b only to $\mathcal{F}_{\mathsf{EOT}}$ which hides it from $\mathcal{S}$ information theoretically. Therefore, the entire computation of π_R is totally independent of b. Now $\mathcal{Z}'$ instructs S^* to invoke $b' \leftarrow \mathcal{S}^{\mathcal{G}^*_{\mathsf{rpRO}}}(\mathsf{sid}_2, \pi_R, \emptyset)$. In the real world, we always have $b' = b$. In the ideal world, we have $b' = b$ with probability at most $\frac{1}{2}$ since π_R is totally independent of b. Therefore, $\mathcal{Z}'$ can distinguish between the real world and the ideal world with a non-negligible probability, contradicting our assumption that Π is GUC-secure. □

By combining this negative result in the GrpRO model and the positive result in the GroRO model depicted in Section 5.1.1, we demonstrate a separation between the GroRO and the GrpRO model.

5.2.2 Our EOT Protocol

Theorem 5 rules out the possibility of 1-round GUC-secure EOT protocols in the $\{\mathcal{G}_{\mathsf{rpRO}}, \mathcal{F}_{\mathsf{Syn}}\}$-hybrid world. It makes us wonder if we do not let the sender and the receiver send their messages simultaneously but in a specific order, can we obtain a 2-move (also 2-round) GUC-secure protocol?

We start with the UC-secure EOT protocol in the CRS+GrpRO hybrid model proposed by Canetti *et al.* [CSW20]. Their CRS consists of two group elements $g, h \in \mathbb{G}$, and the simulator knows $\log_g h$. They let the receiver generate parameter G, H by invoking the RO on a randomly sampled string seed, and compute two instances of Pedersen commitment to the choice bit $(B_1, B_2) := (g^x G^b, h^x H^b)$ using two sets of different parameters $(g, G), (h, H)$ and the same randomness $x \leftarrow \mathbb{Z}_q$. As for the sender, they let the sender compute $z := g^r h^s$ using randomness $r, s \leftarrow \mathbb{Z}_q$. Finally, the sender outputs $m_0 := \mathcal{G}_{\mathsf{rpRO}}(B_1^r B_2^s)$ and $m_1 := \mathcal{G}_{\mathsf{rpRO}}((\frac{B_1}{G})^r (\frac{B_2}{H})^s)$ while the receiver outputs $m_b := \mathcal{G}_{\mathsf{rpRO}}(z^x)$.

Our goals are: (i) remove the CRS setup of this protocol; (ii) make this protocol GUC-secure in the GrpRO model. To achieve the former goal, we let the sender generate g, h by invoking random oracle on a randomly sampled string seed_1. Then the sender computes $z := g^r h^s$ using $r, s \leftarrow \mathbb{Z}_q$, and sends seed_1, z to the receiver. On the other hand, we let the receiver generate G, H by invoking $\mathcal{G}_{\mathsf{rpRO}}$ on another randomly sampled string seed_2, computes two instances of Pedersen commitment to the choice bit $(B_1, B_2) := (g^x G^b, h^x H^b)$ using the same randomness $x \leftarrow \mathbb{Z}_q$. The local computation is the same as Canetti *et al*'s protocol. In order to show that our modified protocol achieves the latter goal, we show the simulation strategy as follows: when the receiver is malicious, the simulator can extract the receiver's choice bit b by programming the $\mathcal{G}_{\mathsf{rpRO}}$ and knowing α such that $h = g^\alpha$. Then the simulator can extract b by the following strategy: if $B_2 = B_1^\alpha$, it sets $b := 0$; else if $\frac{B_2}{H} = (\frac{B_1}{G})^\alpha$, it sets $b := 1$; else, it sets $b := \perp$. Note that, when B_1, B_2 are not correctly constructed (i.e., the simulator sets $b := \perp$), the malicious receiver cannot compute either m_0 or m_1. When the sender is malicious, the simulator can compute both m_0 and m_1 by programming the $\mathcal{G}_{\mathsf{rpRO}}$ such that (g, h, G, H) is a DDH tuple, i.e., $G = g^t, H = h^t$. In this way, the simulator can compute $m_0 := \mathcal{G}_{\mathsf{rpRO}}(z^x)$ and $m_1 := \mathcal{G}_{\mathsf{rpRO}}(z^{x-t})$.

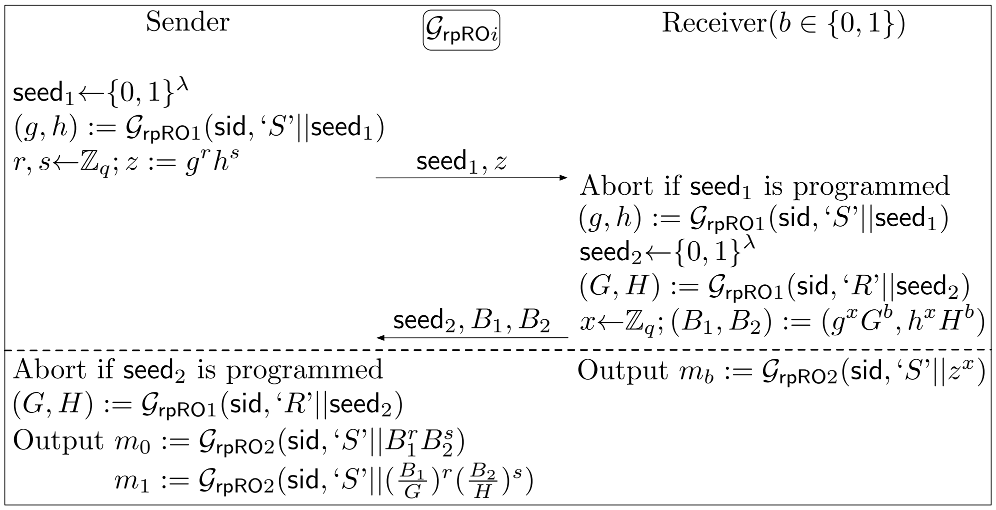

Sender
$\mathcal{G}_{\mathsf{rpRO}i}$
Receiver$(b \in \{0,1\})$
$\mathsf{seed}_1 \leftarrow \{0,1\}^\lambda$
$(g, h) := \mathcal{G}_{\mathsf{rpRO1}}(\mathsf{sid}, `S'||\mathsf{seed}_1)$
$r, s \leftarrow \mathbb{Z}_q; z := g^r h^s$
seed_1, z
Abort if seed_1 is programmed
$(g, h) := \mathcal{G}_{\mathsf{rpRO1}}(\mathsf{sid}, `S'||\mathsf{seed}_1)$
$\mathsf{seed}_2 \leftarrow \{0,1\}^\lambda$
$(G, H) := \mathcal{G}_{\mathsf{rpRO1}}(\mathsf{sid}, `R'||\mathsf{seed}_2)$
$\mathsf{seed}_2, B_1, B_2$
$x \leftarrow \mathbb{Z}_q; (B_1, B_2) := (g^x G^b, h^x H^b)$
Abort if seed_2 is programmed
$(G, H) := \mathcal{G}_{\mathsf{rpRO1}}(\mathsf{sid}, `R'||\mathsf{seed}_2)$
Output $m_0 := \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||B_1^r B_2^s)$
$m_1 := \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||(\frac{B_1}{G})^r(\frac{B_2}{H})^s)$
Output $m_b := \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||z^x)$

Figure 20: 2-round EOT protocol $\Pi_{\mathsf{EOT\text{-}GrpRO}}$ in the $\{\mathcal{G}_{\mathsf{rpRO}}, \mathcal{F}_{\mathsf{Auth}}\}$-hybrid world, where $\mathcal{G}_{\mathsf{rpRO}} = \{\mathcal{G}_{\mathsf{rpRO1}}, \mathcal{G}_{\mathsf{rpRO2}}\}$. Let $\mathcal{G}_{\mathsf{rpRO1}} : \{0,1\}^* \to \mathbb{G}^2$ and $\mathcal{G}_{\mathsf{rpRO2}} : \{0,1\}^* \to \{0,1\}^\lambda$ be random oracles.

Let $\mathcal{G}_{\mathsf{rpRO1}} : \{0,1\}^* \to \mathbb{G} \times \mathbb{G}$ and $\mathcal{G}_{\mathsf{rpRO2}} : \{0,1\}^* \to \{0,1\}^\lambda$. Here we assume authenticated channels $\mathcal{F}_{\mathsf{Auth}}$ are available to all parties.

Protocol Description. Our protocol $\Pi_{\mathsf{EOT\text{-}GrpRO}}$ works as follows. We first present the case where both sender and receiver are honest in Figure 20. More precisely, the sender first selects $\mathsf{seed}_1 \leftarrow \{0,1\}^\lambda$ and computes $(g,h) := \mathcal{G}_{\mathsf{rpRO1}}(\mathsf{sid}, `S'||\mathsf{seed}_1)$ where 'S' is the public identifier of the sender. Then the sender computes $z := g^r h^s$ where g is a common group generator and $r, s \leftarrow \mathbb{Z}_q$. After that, the sender sends (seed_1, z) to the receiver via $\mathcal{F}_{\mathsf{Auth}}$. If the receiver receives (seed_1, z) from $\mathcal{F}_{\mathsf{Auth}}$, the receiver first checks if seed_1 is programed. If so, the receiver simply aborts. Otherwise, the receiver selects $\mathsf{seed}_2 \leftarrow \{0,1\}^\lambda$ and computes $(g,h) := \mathcal{G}_{\mathsf{rpRO1}}(\mathsf{sid}, `S'||\mathsf{seed}_1)$, $(G,H) := \mathcal{G}_{\mathsf{rpRO1}}(\mathsf{sid}, `R'||\mathsf{seed}_2)$ where 'R' is the public identifier of the receiver, and $(B_1, B_2) := (g^x G^b, h^x H^b)$ where $x \leftarrow \mathbb{Z}_q$. After that, the receiver sends $(\mathsf{seed}_2, B_1, B_2)$ to the sender via $\mathcal{F}_{\mathsf{Auth}}$. Finally, the receiver outputs $m_b := \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||z^x)$. If the sender receives $(\mathsf{seed}_2, B_1, B_2)$ from $\mathcal{F}_{\mathsf{Auth}}$, the sender first checks if $\mathsf{seed}_2$ is programed. If so, the sender simply aborts. Otherwise, the sender computes $(G,H) := \mathcal{G}_{\mathsf{rpRO1}}(\mathsf{sid}, `R'||\mathsf{seed}_2)$ and outputs $m_0 := \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||B_1^r B_2^s)$ and $m_1 := \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||(\frac{B_1}{G})^r(\frac{B_2}{H})^s)$. We then talk about the general solution for the honest party when the other party may get corrupted: if the honest party obtains the other party's message from $\mathcal{F}_{\mathsf{Auth}}$ and the obtained message is in the correct form, the honest party will continue the protocol according to Figure 20; otherwise, the honest party simply aborts. The security of the protocol has been stated in Theorem 6.

Theorem 6. *Assume the DDH assumption holds in group $\mathbb{G}$. Let $\mathcal{G}_{\mathsf{rpRO1}} : \{0,1\}^\lambda \to \mathbb{G} \times \mathbb{G}$ and $\mathcal{G}_{\mathsf{rpRO2}} : \mathbb{G} \to \{0,1\}^\lambda$ be the random oracles. The protocol $\Pi_{\mathsf{EOT\text{-}GrpRO}}$ depicted in Figure 20 GUC-realizes the functionality $\mathcal{F}_{\mathsf{EOT}}$ depicted in Figure 8 in the $\{\mathcal{G}_{\mathsf{rpRO}}, \mathcal{F}_{\mathsf{Auth}}\}$-hybrid world against adaptive malicious corruption, where $\mathcal{G}_{\mathsf{rpRO}} = \{\mathcal{G}_{\mathsf{rpRO1}}, \mathcal{G}_{\mathsf{rpRO2}}\}$.*

Proof. We leave the formal proof in Appendix C.5. □

Efficiency. We consider the efficiency of our protocol $\Pi_{\mathsf{EOT\text{-}GrpRO}}$ in the amortized setting here, just like we did in Section 3. Our protocol $\Pi_{\mathsf{EOT\text{-}GrpRO}}$ requires 5 exponentiations w.r.t. computation and 2 group elements w.r.t. communication; while the state-of-the-art 2-round GrpRO-based OT protocol in [CSW20] requires the same computation and extra 2λ bits string w.r.t. communication compared to our protocol. We emphasize that our protocol $\Pi_{\mathsf{EOT\text{-}GrpRO}}$ achieves GUC security; whereas the protocol proposed in [CSW20] achieves only UC security.

Acknowledgment. We thank anonymous reviewers of Eurocrypt 2023 for their helpful and constructive comments.

References

[BFSK11] Christina Brzuska, Marc Fischlin, Heike Schröder, and Stefan Katzenbeisser. Physically uncloneable functions in the universal composition framework. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 51–70. Springer, Heidelberg, August 2011.

[BM90] Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and applications. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 547–557. Springer, Heidelberg, August 1990.

[BPRS17] Megha Byali, Arpita Patra, Divya Ravi, and Pratik Sarkar. Fast and universally-composable oblivious transfer and commitment scheme with adaptive security. Cryptology ePrint Archive, Report 2017/1165, 2017. https://eprint.iacr.org/2017/1165.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CDG$^+$18] Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 280–312. Springer, Heidelberg, April / May 2018.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 61–85. Springer, Heidelberg, February 2007.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, Heidelberg, August 1994.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998.

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 597–608. ACM Press, November 2014.

[CK02] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 337–351. Springer, Heidelberg, April / May 2002.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

[CO15] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 40–58. Springer, Heidelberg, August 2015.

[CP93] David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer, Heidelberg, August 1993.

[CSW20] Ran Canetti, Pratik Sarkar, and Xiao Wang. Efficient and round-optimal oblivious transfer and commitment with adaptive security. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 277–308. Springer, Heidelberg, December 2020.

[Dam02] Ivan Damgård. On Σ-protocols. *Lecture Notes, University of Aarhus, Department for Computer Science*, page 84, 2002. https://www.cs.au.dk/~ivan/Sigma.pdf.

[DD20] Bernardo David and Rafael Dowsley. Efficient composable oblivious transfer from CDH in the global random oracle model. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, *CANS 20*, volume 12579 of *LNCS*, pages 462–481. Springer, Heidelberg, December 2020.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DKLs18] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *2018 IEEE Symposium on Security and Privacy*, pages 980–997. IEEE Computer Society Press, May 2018.

[DSW08] Yevgeniy Dodis, Victor Shoup, and Shabsi Walfish. Efficient constructions of composable commitments and zero-knowledge proofs. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 515–535. Springer, Heidelberg, August 2008.

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Heidelberg, August 1984.

[Fis05] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Heidelberg, August 2005.

[FKMV12] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 60–79. Springer, Heidelberg, December 2012.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[GIR20] Ziya Alper Genç, Vincenzo Iovino, and Alfredo Rial. "the simplest protocol for oblivious transfer" revisited. *Information Processing Letters*, 161:105975, 2020.

[GIS18a] Sanjam Garg, Yuval Ishai, and Akshayaram Srinivasan. Two-round MPC: Information-theoretic and black-box. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 123–151. Springer, Heidelberg, November 2018.

[GIS18b] Sanjam Garg, Yuval Ishai, and Akshayaram Srinivasan. Two-round MPC: Information-theoretic and black-box. Cryptology ePrint Archive, Report 2018/909, 2018. https://eprint.iacr.org/2018/909.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 339–358. Springer, Heidelberg, May / June 2006.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Heidelberg, April 2008.

[GS17] Sanjam Garg and Akshayaram Srinivasan. Garbled protocols and two-round MPC from bilinear maps. In Chris Umans, editor, *58th FOCS*, pages 588–599. IEEE Computer Society Press, October 2017.

[HL17] Eduard Hauck and Julian Loss. Efficient and universally composable protocols for oblivious transfer from the CDH assumption. Cryptology ePrint Archive, Report 2017/1011, 2017. https://eprint.iacr.org/2017/1011.

[Kat07] Jonathan Katz. On achieving the "best of both worlds" in secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 11–20. ACM Press, June 2007.

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

[KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer, Heidelberg, March 2013.

[Ks22] Yashvanth Kondi and abhi shelat. Improved straight-line extraction in the random oracle model with applications to signature aggregation. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 279–309. Springer, Heidelberg, December 2022.

[LM18] Baiyu Li and Daniele Micciancio. Equational security proofs of oblivious transfer protocols. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 527–553. Springer, Heidelberg, March 2018.

[MR19a] Daniel Masny and Peter Rindal. Endemic oblivious transfer. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 309–326. ACM Press, November 2019.

[MR19b] Daniel Masny and Peter Rindal. Endemic oblivious transfer. Cryptology ePrint Archive, Report 2019/706, 2019. https://eprint.iacr.org/2019/706.

[MRS17] Payman Mohassel, Mike Rosulek, and Alessandra Scafuro. Sublinear zero-knowledge arguments for RAM programs. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 501–531. Springer, Heidelberg, April / May 2017.

[Oka93] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 31–53. Springer, Heidelberg, August 1993.

[Pas03] Rafael Pass. On deniability in the common reference string and random oracle model. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 316–337. Springer, Heidelberg, August 2003.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.

[ZZZR22] Zhelei Zhou, Bingsheng Zhang, Hong-Sheng Zhou, and Kui Ren. GUC-secure commitments via random oracles: New impossibility and feasibility. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 129–158. Springer, Heidelberg, December 2022.

A Additional Preliminaries

A.1 Pedersen Commitment

The Pedersen commitment is one of the most common trapdoor commitment scheme [Ped92]. In order to formally define the Pedersen commitment, we first introduce the trapdoor commitment and then describe how to instantiate it with Pedersen commitment.

A trapdoor commitment scheme $\Pi = (\Pi.\mathsf{KeyGen}, \Pi.\mathsf{Commit}, \Pi.\mathsf{ComVer}, \Pi.\mathsf{Equiv})$ allows the committer to compute the commitment c to any value m using the commitment key ck and the randomness r. Later, the committer can open c to m by sending the the opening d to the receiver who verifies it. Furthermore, if the committer somehow obtains the trapdoor key td with respect to ck, it can open the previous commitment c to any other message $\tilde{m} \neq m$. Formally, the trapdoor commitment has the following algorithms:

- $(\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ takes input as the security parameter λ, and outputs a commitment key ck and a trapdoor key td.
- $(c, d) := \mathsf{Commit}(\mathsf{ck}, m; r)$ takes input as a commitment key ck, a message m and a randomness r. It outputs the commitment c and the opening d. We assume that there exists a deterministic algorithm that can extract m from d. When r is not important, we use $\mathsf{Commit}(\mathsf{ck}, m)$ for simplicity.
- $b := \mathsf{ComVer}(\mathsf{ck}, c, d)$ takes input as a commitment key ck, and a commitment-opening pair (c, d). It outputs a bit b indicating acceptance ($b = 1$) or rejection ($b = 0$).
- $(\tilde{d}, \tilde{r}) := \mathsf{Equiv}(\mathsf{ck}, \mathsf{td}, c, d, \tilde{m})$ takes input as a commitment key ck, a trapdoor key td, a commitment c and its opening d, and an arbitrary message $\tilde{m}$ for which equivocation is required. It outputs the new opening $\tilde{d}$ and a new randomness $\tilde{r}$ such that $(c, \tilde{d}) = \mathsf{Commit}(\mathsf{ck}, \tilde{m}; \tilde{r})$.

The trapdoor commitment requires the following properties: perfect correctness, perfect hiding, computational binding and trapdoor property. Perfect correctness means that any commitment produced by the honest committer can always be verified. Perfect hiding means that the commitment c reveals nothing about the message m. Computational binding means that it is infeasible for the PPT committer to output the commitment c that can be opened in two different ways. Trapdoor property means that given the trapdoor key td, one can open a previously constructed commitment c that corresponds to the message m to any other message $\tilde{m} \neq m$. Formally, we have the following definition:

Definition 15. *We say a scheme* $\Pi = (\Pi.\mathsf{KeyGen}, \Pi.\mathsf{Commit}, \Pi.\mathsf{ComVer}, \Pi.\mathsf{Equiv})$ *is a trapdoor commitment scheme if the following conditions hold:*

- ***(Perfect Correctness)*** *For any message m, we say it is perfect correct if*

$$\Pr[(\mathsf{ck},\mathsf{td}) \leftarrow \mathsf{KeyGen}(1^\lambda); (c,d) \leftarrow \mathsf{Commit}(\mathsf{ck},m) : \mathsf{ComVer}(\mathsf{ck},c,d) = 1] = 1$$

- ***(Perfect Hiding)*** *We say it is perfect hiding if for any adversary $\mathcal{A}$*

$$\Pr\left[\begin{array}{l}(\mathsf{ck},\mathsf{td}) \leftarrow \mathsf{KeyGen}(1^\lambda); (m_0,m_1,\mathsf{st}) \leftarrow \mathcal{A}(\mathsf{ck});\\ b \leftarrow \{0,1\}; (c,d) \leftarrow \mathsf{Commit}(\mathsf{ck},m_b); b' \leftarrow \mathcal{A}(c,\mathsf{st})\end{array} : b = b'\right] = \frac{1}{2}$$

- ***(Computational Binding)*** *We say it is computational binding if for any PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that*

$$\Pr\left[\begin{array}{l}(\mathsf{ck},\mathsf{td}) \leftarrow \mathsf{KeyGen}(1^\lambda);\\ (c,d_0,d_1) \leftarrow \mathcal{A}(\mathsf{ck})\end{array} : \begin{array}{l}\mathsf{ComVer}(\mathsf{ck},c,d_0) = \mathsf{ComVer}(\mathsf{ck},c,d_1) = 1\\ \wedge\, d_0 \neq d_1\end{array}\right] \leq \mathsf{negl}(\lambda)$$

- ***(Trapdoor Property)*** *For any message pair $(m,\tilde{m})$, we say it has trapdoor property if*

$$\begin{array}{l}\left\{(\mathsf{ck},c,d,r) \,\middle|\, (\mathsf{ck},\mathsf{td}) \leftarrow \mathsf{KeyGen}(1^\lambda); (c,d) := \mathsf{Commit}(\mathsf{ck},m;r)\right\}\\ \overset{c}{\approx} \left\{(\mathsf{ck},c,d,r) \,\middle|\, \begin{array}{l}(\mathsf{ck},\mathsf{td}) \leftarrow \mathsf{KeyGen}(1^\lambda); (c,\tilde{d}) := \mathsf{Commit}(\mathsf{ck},\tilde{m};\tilde{r});\\ (d,r) := \mathsf{Equiv}(\mathsf{ck},\mathsf{td},c,\tilde{d},\tilde{r})\end{array}\right\}\end{array}$$

Now we show how to instantiate the trapdoor commitment scheme with the famous Pedersen commitment. We present the formal description of the Pedersen commitment in Figure 21. It is well-known that the Pedersen commitment is a trapdoor commitment scheme under the Discrete Logarithm (DL) assumption.

Pedersen Commitment

- $\mathsf{KeyGen}(1^\lambda)$: It outputs $\mathsf{ck} := (\mathbb{G},p,q,g,h)$ and $\mathsf{td} := \alpha$, where $q = \frac{p-1}{2}$ and p are primes, $\mathbb{G}$ is a subgroup of order q of $\mathbb{Z}_p^*$, g and $h = g^\alpha$ are the generators of $\mathbb{G}$.
- $\mathsf{Commit}(\mathsf{ck},m;r)$: It outputs $c := g^m h^r \mod p$ and $d := (m,r)$.
- $\mathsf{ComVer}(\mathsf{ck},c,d)$: It checks if $c = g^m h^r$ holds. If so, it outputs 1; otherwise, outputs 0.
- $\mathsf{Equiv}(\mathsf{ck},\mathsf{td},c,d,\tilde{m})$: It computes $\tilde{r} := \frac{m-\tilde{m}}{\alpha} + r \mod q$, and outputs $\tilde{d} := (\tilde{m},\tilde{r})$ and $\tilde{r}$.

Figure 21: Pedersen Commitment

A.2 ElGamal Encryption

The ElGamal encryption is a pseudorandom Public Key Encryption (PKE) system [GOS06]. Analogously to the Section A.1, we first introduce the pseudorandom PKE system, then describe how to instantiate it with the ElGamal encryption.

A pseudorandom PKE system $\Pi = (\Pi.\mathsf{KeyGen}, \Pi.\mathsf{Enc}, \Pi.\mathsf{Dec})$ allows anyone to compute an encryption E of the message m using the public key pk and the randomness r. The one who has the decryption key dk can compute m from E. Formally, the pseudorandom PKE system has the following algorithms:

- $(\mathsf{pk},\mathsf{dk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ takes input as the security parameter λ, and outputs a public key pk and a decryption key dk.

- $E := \mathsf{Enc}(\mathsf{pk},m;r)$ takes input as a public key pk, a message m and a randomness r. It outputs the ciphertext E. When r is not important, we use $\mathsf{Enc}(\mathsf{pk},m)$ for simplicity.

- $m \leftarrow \mathsf{Dec}(\mathsf{pk},\mathsf{dk},E)$ takes input as a public key pk, a decryption key dk and a ciphertext E. It outputs the plaintext message m.

The pseudorandom PKE system requires the following properties: perfect correctness, semantic security. Perfect correctness is trivial. Semantic security means that the ciphertext E reveals nothing about the message m. Formally, we have the following definition:

Definition 16. *We say a scheme* $\Pi = (\Pi.\mathsf{KeyGen}, \Pi.\mathsf{Enc}, \Pi.\mathsf{Dec})$ *is a pseudorandom PKE system if the following conditions hold:*

- ***(Perfect Correctness)*** *For any message* m*, we say it is perfect correct if*

$$\Pr\left[\begin{array}{l}(\mathsf{pk},\mathsf{dk}) \leftarrow \mathsf{KeyGen}(1^\lambda); E \leftarrow \mathsf{Enc}(\mathsf{pk}, m);\\ m' \leftarrow \mathsf{Dec}(\mathsf{pk},\mathsf{dk},E)\end{array} \;:\; m = m'\right] = 1$$

- ***(Semantic Security)*** *We say it is semantic secure if for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that*

$$\left|\Pr\left[\begin{array}{l}(\mathsf{pk},\mathsf{dk}) \leftarrow \mathsf{KeyGen}(1^\lambda); (m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{ck});\\ b \leftarrow \{0,1\}; E \leftarrow \mathsf{Enc}(\mathsf{ck}, m_b); b' \leftarrow \mathcal{A}(E, \mathsf{st})\end{array} \;:\; b = b'\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)$$

Here we show how to instantiate the pseudorandom PKE system with the well-known ElGamal encryption. We present the formal description of the ElGamal encryption in Figure 22. It is well-known that the ElGamal encryption is a pseudorandom PKE system under the DDH assumption.

ElGamal Encryption

- $\mathsf{KeyGen}(1^\lambda)$: It outputs $\mathsf{pk} := (\mathbb{G}, p, q, g, h)$ and $\mathsf{dk} := \alpha$, where $q = \frac{p-1}{2}$ and p are primes, $\mathbb{G}$ is a subgroup of order q of $\mathbb{Z}_p^*$, g and $h = g^\alpha$ are the generators of $\mathbb{G}$.
- $\mathsf{Enc}(\mathsf{pk}, m; r)$: It computes $(u, v) := (g^r \mod p, g^m h^r \mod p)$ and outputs $E := (u, v)$.
- $\mathsf{Dec}(\mathsf{ck}, \mathsf{dk}, E)$: It computes $\frac{v}{u^\alpha}$ and searches exhaustively for m.

Figure 22: ElGamal Encryption

B Building Blocks of (Straight-Line Extractable) NIZK/NIWH Arguments

In this section, we introduce the building blocks of our (straight-line extractable) NIZK/NIWH arguments. More precisely, we first introduce several Sigma-protocols that are previously mentioned in our instantiations. We then introduce the randomized Fischlin transform [Ks22] which compiles a Sigma-protocol into a straight-line extractable NIZK (resp. NIWH) arguments in the RO (resp. observable RO) model.

B.1 Concrete Examples of Sigma-Protocols

Schnorr's protocol [Sch90]. The Schnorr's protocol aims to prove the knowledge of a discrete logarithm. Let $\mathcal{R}_{\mathsf{DL}} := \{((g, h), w) \mid h = g^w\}$. We present the Sigma-protocol for relation $\mathcal{R}_{\mathsf{DL}}$ in Figure 23.

Schnorr's Protocol

Common Input: Statement $x := (\mathbb{G}, p, q, g, h)$, where p, q describe the group $\mathbb{G}$ and $g, h \in \mathbb{G}$.
Private Input of the Prover: Witness w such that $h = g^w$.

- $\mathsf{P1}(x, w)$: Select $r \leftarrow \mathbb{Z}_q$ and output $a := g^r \mod p, \mathsf{st} := r$.
- $\mathsf{V1}(1^\lambda)$: Select $e \leftarrow \mathbb{Z}_q$ and output e.
- $\mathsf{P2}(x, w, e, \mathsf{st})$: Compute $z := r + e \cdot w \mod q$ and output z.
- $\mathsf{Verify}(x, a, e, z)$: Check if $g^z = a \cdot h^e$ holds. If so, output 1; otherwise, output 0.

Figure 23: Sigma-Protocol for Relation $\mathcal{R}_{\mathsf{DL}}$

Chaum-Pedersen protocol [CP93]. The Chaum-Pedersen protocol aims to prove the knowledge of a DDH tuple. Let $\mathcal{R}_{\mathsf{DDH}} := \{((g, h, u, v), w) \mid u = g^w \ \wedge \ v = h^w\}$. We present the Sigma-protocol for relation $\mathcal{R}_{\mathsf{DDH}}$ in Figure 24.

Chaum-Pedersen Protocol

Common Input: Statement $x := (\mathbb{G}, p, q, g, h, u, v)$, where p, q describe the group $\mathbb{G}$ and $g, h, u, v \in \mathbb{G}$.
Private Input of the Prover: Witness w such that $u = g^w$ and $v = h^w$.

- $\mathsf{P1}(x, w)$: Select $r \leftarrow \mathbb{Z}_q$, compute $a_0 := g^r \mod p, a_1 := h^r \mod p$ and output $a := (a_0, a_1), \mathsf{st} := r$.
- $\mathsf{V1}(1^\lambda)$: Select $e \leftarrow \mathbb{Z}_q$ and output e.
- $\mathsf{P2}(x, w, e, \mathsf{st})$: Compute $z := r + e \cdot w \mod q$ and output z.
- $\mathsf{Verify}(x, a, e, z)$: Check if $g^z = a_0 \cdot u^e$ and $h^z = a_1 \cdot v^e$ hold. If so, output 1; otherwise, output 0.

Figure 24: Sigma-Protocol for Relation $\mathcal{R}_{\mathsf{DDH}}$

Okamoto's protocol [Oka93]. The Okamoto's protocol aims to prove the knowledge of a Pedersen commitment's opening. Let $\mathcal{R}_{\mathsf{Com}} := \{((g, h, C), (m, r)) \mid C = g^m h^r\}$. We present the Sigma-protocol for relation $\mathcal{R}_{\mathsf{Com}}$ in Figure 25.

Okamoto's Protocol

Common Input: Statement $x := (\mathbb{G}, p, q, g, h, C)$, where p, q describe the group $\mathbb{G}$ and $g, h, C \in \mathbb{G}$.
Private Input of the Prover: Witness $w := (m, r)$ such that $C = g^m h^r$.

- $\mathsf{P1}(x, w)$: Select $s, t \leftarrow \mathbb{Z}_q$, compute $a := g^s h^t \mod p$ and output $a, \mathsf{st} := (s, t)$.
- $\mathsf{V1}(1^\lambda)$: Select $e \leftarrow \mathbb{Z}_q$ and output e.
- $\mathsf{P2}(x, w, e, \mathsf{st})$: Compute $z_0 := s + e \cdot m \mod q, z_1 := t + e \cdot r \mod q$ and output $z := (z_0, z_1)$.
- $\mathsf{Verify}(x, a, e, z)$: Check if $g^{z_0} h^{z_1} = a \cdot C^e$ holds. If so, output 1; otherwise, output 0.

Figure 25: Sigma-Protocol for Relation $\mathcal{R}_{\mathsf{Com}}$

OR-Composition

Common Input: Statement $x := (x_0, x_1)$.
Private Input of the Prover: Witness w such that $(x_b, w) \in \mathcal{R}_b$.

- $\mathsf{P1}(x, w)$: Compute $(a_b, \mathsf{st}) \leftarrow \Pi_b.\mathsf{P1}(x_b, w)$. Select $s_{1-b} \leftarrow \Pi_b.\mathsf{V1}(1^\lambda)$ and compute $(a_{1-b}, z_{1-b}) \leftarrow \Pi_{1-b}.\mathsf{Sim}(x_{1-b}, s_{1-b})$. Output $a := (a_0, a_1), \mathsf{st}$.
- $\mathsf{V1}(1^\lambda)$: Select $e \leftarrow \mathbb{Z}_q$ and output e.
- $\mathsf{P2}(x, w, e, \mathsf{st})$: Set $s_b := e \oplus s_{1-b}$, compute $z_b \leftarrow \Pi_b.\mathsf{P2}(x_b, w, s_b, \mathsf{st})$ and output $z := (s_0, z_0, s_1, z_1)$.
- $\mathsf{Verify}(x, a, e, z)$: Check if $e = s_0 \oplus s_1$ and $\Pi_0.\mathsf{Verify}(x_0, a_0, s_0, z_0) = \Pi_1.\mathsf{Verify}(x_1, a_1, s_1, z_1) = 1$ hold. If so, output 1; otherwise, output 0.

Figure 26: Sigma-Protocol for Relation $\mathcal{R}_0 \vee \mathcal{R}_1$

OR-composition [CDS94]. Let (x_0, x_1) be a pair of the statements.The OR-composition of the Σ-protocol allows the prover to prove that it 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 Sigma-protocol for relation $\mathcal{R}_0$ and Π_1 be the one for relation $\mathcal{R}_1$. We present the Sigma-protocol for relation $\mathcal{R}_0 \vee \mathcal{R}_1$ in Figure 26.

B.2 Randomized Fischlin Transform

Recently, Kondi and shelat revisited the famous Fischlin transform [Fis05], which complies a Sigma-protocol into a straight-line extractable NIZK (resp. NIWH) arguments in the RO (resp. observable RO) model, and improved it in both efficiency and applicability [Ks22]. We call their construction *randomized Fischlin transform* as the original Fischlin transform is somewhat deterministic and suffers from a "replay" attack that breaks the witness-indistinguishability property of the OR-composition of Sigma-protocols [Ks22]. We review the randomized Fischlin transform $\Pi_{\mathsf{Ran\text{-}Fis}}$ in Figure 27.

Randomized Fischlin Transform

Common Input: Statement x.
Private Input of the Prover: Witness w such that $(x, w) \in \mathcal{R}$.
Parameter: r, ℓ, t such that r is an even number, $r \cdot \ell = 2^\lambda$ and $t = \ulcorner \log \lambda \urcorner \cdot \ell$.
Primitive: Sigma-protocol Π_Σ for relation $\mathcal{R}$.
Random Oracle: $\mathcal{F}_{\mathsf{RO}} : \{0,1\}^* \to \{0,1\}^{2\ell}$.

```
• Prove^{F_RO}(x, w):
    1. For $i \in [r]$: compute $(a_i, \mathsf{st}_i) \leftarrow \Pi_\Sigma.\mathsf{P1}(x, w)$.
    2. Set $a := (a_i)_{i \in [r]}$.
    3. For $i \in [r/2]$: do the following:
        (a) Set $\xi_i := \emptyset$ and $\mathsf{flag} := 0$.
        (b) Sample $e_i \leftarrow \Pi_\Sigma.\mathsf{V1}(1^\lambda) \setminus \xi_i$ and compute $z_i \leftarrow \Pi_\Sigma.\mathsf{P2}(x, w, e_i, \mathsf{st}_i)$.
        (c) For $j \in [2^{2\ell}]$:
            i. Set $\xi_{i+r/2} := \emptyset$.
            ii. Sample $e_{i+r/2} \leftarrow \Pi_\Sigma.\mathsf{V1}(1^\lambda) \setminus \xi_{i+r/2}$ and compute $z_{i+r/2} \leftarrow \Pi_\Sigma.\mathsf{P2}(x, w, e_{i+r/2}, \mathsf{st}_{i+r/2})$.
            iii. If $\mathcal{F}_{\mathsf{RO}}(a, i, e_i, z_i) \neq \mathcal{F}_{\mathsf{RO}}(a, i + r/2, e_{i+r/2}, z_{i+r/2})$, repeat Step 3(c)ii; else, set $\mathsf{flag} := 1$ and break the loop.
        (d) If $\mathsf{flag} \neq 1$, repeat Step 3b.
    4. Output $\pi := (a_i, e_i, z_i)_{i \in [r]}$.
• Verify^{F_RO}(x, π):
    1. Output 1 if the following conditions hold:
        (a) For $i \in [r/2]$: Check if $\mathcal{F}_{\mathsf{RO}}(a, i, e_i, z_i) = \mathcal{F}_{\mathsf{RO}}(a, i + r/2, e_{i+r/2}, z_{i+r/2})$ holds.
        (b) For $i \in [r]$: Check if $\Pi_\Sigma.\mathsf{Verify}(x, a_i, e_i, z_i) = 1$ holds.
```

Figure 27: Randomized Fischlin Transform $\Pi_{\mathsf{Ran\text{-}Fis}}$ from [Ks22]

Recall that, the Fischlin transform requires the Sigma-protocol to additionally satisfy quasi-unique responses [Fis05]. Roughly speaking, this means that no PPT prover can compute a statement x and a, e, z, z' such that (a, e, z) and (a, e, z') are both accepting transcripts for x. Kondi and shelat showed that a simpler property, i.e., *strong special soundness*, would be sufficient for their purpose. More precisely, they defined a variant of Sigma-protocols, i.e., *strong special sound Sigma-protocols* and proved that their transform can be applied to these Sigma-protocols securely. We present the definition of strong special sound Sigma-protocols as follows.

Definition 17. *Fix an NP relation $\mathcal{R}$ and its associate language $\mathcal{L}$. We say a protocol $\Pi = (\Pi.\mathsf{P1}, \Pi.\mathsf{V1}, \Pi.\mathsf{P2}, \Pi.\mathsf{Verify}, \Pi.\mathsf{Ext}, \Pi.\mathsf{Sim})$ is a strong special sound Sigma-protocol for relation $\mathcal{R}$ if it satisfies completeness and SHVZK that have been defined in Definition 5 and the following property:*

1. ***(Strong Special Soundness)** For any $x \in \mathcal{L}$, we say it is a strong special sound if for any PPT adversary $\mathcal{A}$,*

$$\Pr\left[\begin{array}{l}(T := (a, e, z), T' := (a, e', z')) \leftarrow \mathcal{A}(x); \\ w \leftarrow \mathsf{Ext}(x, T, T')\end{array} : \begin{array}{l}\mathsf{Verify}(x, T) = \mathsf{Verify}(x, T') = 1 \\ \wedge\, T \neq T' \;\wedge\; (x, w) \in \mathcal{R}\end{array}\right] = 1$$

In the main body of our paper, we apply randomized Fischlin transform to Schnorr's protocol [Sch90] and Okamoto's protocol [Oka93] to instantiate straight-line extractable NIZK (resp. NIWH) arguments in the RO (resp. observable RO) model. It is easy to see that both Schnorr's protocol and Okamoto's protocol are strong special sound Sigma-protocols. Thus the security of our instantiations is guaranteed.

C Additional Security Proofs

C.1 Proof of Theorem 1

Theorem 1. *Assume the DDH assumption holds in group $\mathbb{G}$. Let $\mathcal{F}_{\mathsf{RO1}} : \{0,1\}^* \to \mathbb{G}$ and $\mathcal{F}_{\mathsf{RO2}} : \{0,1\}^* \to \{0,1\}^\lambda$ be the random oracles. Let Π_{NIZK} be an NIZK argument in the $\mathcal{F}_{\mathsf{RO3}}$-hybrid world. Let Π_{sleNIZK} be a straight-line extractable*

NIZK argument in the $\mathcal{F}_{\mathsf{RO}4}$-hybrid world. The protocol $\Pi_{\mathsf{EOT\text{-}RO}}$ depicted in Figure 13 UC-realizes the functionality $\mathcal{F}_{\mathsf{EOT}}$ depicted in Figure 8 in the $\{\mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{Syn}}\}$-hybrid world against static malicious corruption, where $\mathcal{F}_{\mathsf{RO}} = \{\mathcal{F}_{\mathsf{RO}i}\}_{i\in[4]}$.

Proof. We now prove the security of our protocol $\Pi_{\mathsf{EOT\text{-}RO}}$ by showing it is a UC-secure realization of $\mathcal{F}_{\mathsf{EOT}}$. We describe the workflow of $\mathcal{S}$ in the ideal-world with $\mathcal{F}_{\mathsf{EOT}}$, and give a proof that the simulation in the ideal world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{EOT}},\mathcal{S},\mathcal{Z}}$ is computationally indistinguishable from a real world execution $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{RO}},\mathcal{F}_{\mathsf{Syn}}}_{\Pi_{\mathsf{EOT\text{-}RO}},\mathcal{A},\mathcal{Z}}$ for any PPT adversary $\mathcal{A}$ and any PPT environment $\mathcal{Z}$. Note that, the simulator $\mathcal{S}$ simply forwards the communication between $\mathcal{A}$ and $\mathcal{Z}$.

The sender is honest while the receiver is statically corrupted. Here the simulator $\mathcal{S}$ needs to extract the choice bit of the receiver from the message sent from R^*. Note that, the simulator $\mathcal{S}$ needs to send its message on behalf of the honest sender before seeing the adversary $\mathcal{A}$'s message. We describe the strategy of $\mathcal{S}$ as follows:

- Generate (z, π_{DL}) honestly, and send (z, π_{DL}) to R^*.
- Whenever R^* invokes $\mathcal{F}_{\mathsf{RO}1}$ on candidate seed values, select different $\alpha \in \mathbb{Z}_q$ and return $h = g^\alpha$ as the output of $\mathcal{F}_{\mathsf{RO}1}$.
- Wait for R^* to send $(\mathsf{seed}, u, v, \pi_{\mathsf{ENC}})$, then do the following:
  - Abort if $\Pi_{\mathsf{NIZK}}.\mathsf{Verify}^{\mathcal{F}_{\mathsf{RO}4}}((g, h, u, v), \pi_{\mathsf{ENC}}) = 0$ for $\mathcal{R}_{\mathsf{ENC}}$.
  - If $u = v^\alpha$, set $b := 0$; else if $u = (\frac{v}{h})^\alpha$, set $b := 1$; else, abort.
  - Compute m_0, m_1 honestly, and send $(\textsc{Receive}, \mathsf{sid}, S, R^*, b)$ to $\mathcal{F}_{\mathsf{EOT}}$ on behalf of the dummy corrupted receiver. Return $(\textsc{Proceed}, \mathsf{sid}, S, m_b)$ to $\mathcal{F}_{\mathsf{EOT}}$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, S)$.
- If R^* does not send its message, pick a random bit b and send $(\textsc{Receive}, \mathsf{sid}, S, R^*, b)$ to $\mathcal{F}_{\mathsf{EOT}}$ on behalf of the dummy corrupted receiver. Return $(\textsc{No}, \mathsf{sid}, S)$ to $\mathcal{F}_{\mathsf{EOT}}$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, S)$.

We prove the indistinguishability through the following hybrid experiments:

- $\mathcal{H}_0$: This is the real world execution $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{RO}},\mathcal{F}_{\mathsf{Syn}}}_{\Pi_{\mathsf{EOT\text{-}RO}},\mathcal{A},\mathcal{Z}}$.
- $\mathcal{H}_1$: Same as $\mathcal{H}_0$, except that $\mathcal{S}$ selects different $\alpha \in \mathbb{Z}_q$ and returns $h = g^\alpha$ as the output of $\mathcal{F}_{\mathsf{RO}1}$ whenever R^* invokes $\mathcal{F}_{\mathsf{RO}1}$ on candidate seed values. Indistinguishability follows from the fact that the tuple (g, h) is randomly selected in both $\mathcal{H}_0$ and $\mathcal{H}_1$.
- $\mathcal{H}_2$: Same as $\mathcal{H}_1$, except that $\mathcal{S}$ aborts when π_{ENC} is not valid. Indistinguishability follows from the fact that honest sender would always abort when π_{ENC} is not valid.
- $\mathcal{H}_3$: Same as $\mathcal{H}_2$, except that $\mathcal{S}$ aborts when it fails to extract $b \in \{0, 1\}$.

  Lemma 2. *Assume the DDH assumption holds in $\mathbb{G}$. Let Π_{NIZK} be an NIZK argument in the RO model. Hybrid $\mathcal{H}_3$ is computationally indistinguishable from $\mathcal{H}_1$.*

  Proof. Since the proof π_{ENC} has already been verified, the tuple (u, v) should be an ElGamal encryption of a bit $b \in \{0, 1\}$. In this case, due to the computational soundness of Π_{NIZK}, the simulator $\mathcal{S}$ fails to extract $b \in \{0, 1\}$ only at a negligible probability. In conclusion, $\mathcal{H}_3$ is computationally indistinguishable from $\mathcal{H}_2$. □

- $\mathcal{H}_4$: Same as $\mathcal{H}_3$, except that $\mathcal{S}$ extracts $b \in \{0, 1\}$, computes m_0, m_1 honestly, sends $(\textsc{Receive}, \mathsf{sid}, S, R^*, b)$ to $\mathcal{F}_{\mathsf{EOT}}$ on behalf of the dummy corrupted receiver, and returns $(\textsc{Proceed}, \mathsf{sid}, S, m_b)$ to $\mathcal{F}_{\mathsf{EOT}}$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, S)$. The simulator $\mathcal{S}$ aborts when R^* computes both m_0 and m_1.

  Lemma 3. *Assume the CDH assumption holds in $\mathbb{G}$, then $\mathcal{H}_4$ is computationally indistinguishable from $\mathcal{H}_3$.*

  Proof. The simulator $\mathcal{S}$ aborts when R^* computes m_0 and m_1 by setting $b = 0$ $(b = 1)$ and querying $z^r, \frac{z^r}{h^s}$ $(z^r h^s)$ to $\mathcal{F}_{\mathsf{RO}2}$. We discuss the case where R^* sets $b = 0$ and queries $z^r, \frac{z^r}{h^s}$ to $\mathcal{F}_{\mathsf{RO}2}$ here, and the other case is similar.

  We observe that if there exists such a R^*, then there is a reduction $\mathcal{B}$ which breaks the CDH assumption. The reduction $\mathcal{B}$ interacts with the CDH game challenger $\mathcal{C}$ and receives (A_1, A_2, A_3) from $\mathcal{C}$, and we assume that

$A_1 := g, A_2 := g^t, A_3 := g^s$. Then $\mathcal{B}$ simulates $\mathcal{F}_{\mathsf{RO1}}, \mathcal{F}_{\mathsf{RO2}}, \mathcal{F}_{\mathsf{RO3}}, \mathcal{F}_{\mathsf{RO4}}$ and starts the protocol $\Pi_{\mathsf{EOT\text{-}RO}}$ with R^* by running R^* internally as a black-box. Note that $\mathcal{B}$ has full control over $\{\mathcal{F}_{\mathsf{RO}i}\}_{i\in[4]}$. First $\mathcal{B}$ sets $z := A_3$ and simulates the straight-line extractable NIZK proof π_{DL} by programming the random oracles $\mathcal{F}_{\mathsf{RO3}}$, and sends z, π_{DL} to R^*. When R^* queries a λ-bit string seed_i to $\mathcal{F}_{\mathsf{RO1}}$, $\mathcal{B}$ returns $A_2 \cdot g^{a_i}$ where $a_i \leftarrow \mathbb{Z}_q$. Upon receiving $(\mathsf{seed}, u, v, \pi_{\mathsf{ENC}})$ from R^*, $\mathcal{B}$ looks up the query-answer table of $\mathcal{F}_{\mathsf{RO1}}$ and finds the index j such that $\mathsf{seed} = \mathsf{seed}_j$ and $h = A_2 \cdot g^{a_j}$. Finally, $\mathcal{B}$ randomly selects two queries q_1, q_2 made to $\mathcal{F}_{\mathsf{RO2}}$ by R^* and sends $A_4 := \frac{q_1}{q_2 A_3^{a_j}}$ to $\mathcal{C}$. If there are Q queries made to $\mathcal{F}_{\mathsf{RO2}}$ by R^* in total, then $A_4 = \frac{q_1}{q_2 A_3^{a_j}} = \frac{z^r}{\frac{z^r}{h^s} A_3^{a_j}} = \frac{A_2^s \cdot g^{s a_j}}{A_3^{a_j}} = g^{st}$ happens at probability $\frac{1}{2\cdot\binom{Q}{2}}$. Therefore, $\mathcal{B}$ wins the CDH game at probability $\frac{1}{2\cdot\binom{Q}{2}}$ which is non-negligible. In conclusion, $\mathcal{H}_4$ is computationally indistinguishable from $\mathcal{H}_3$. □

- $\mathcal{H}_5$: Same as $\mathcal{H}_4$, except that when R^* does not send its message, the simulator $\mathcal{S}$ picks a random bit b and sends $(\textsc{Receive}, \mathsf{sid}, S, R^*, b)$ to $\mathcal{F}_{\mathsf{EOT}}$ on behalf of the dummy corrupted receiver, and returns $(\textsc{No}, \mathsf{sid}, S)$ to $\mathcal{F}_{\mathsf{EOT}}$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, S)$. Indistinguishability follows from the fact that the honest sender will abort in both ideal world and the real world when the malicious receiver refuses to continue the protocol.

Hybrid $\mathcal{H}_5$ is identical to the ideal world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{EOT}},\mathcal{S},\mathcal{Z}}$. In conclusion, when the sender S is honest and the receiver R^* is statically corrupted, $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{RO}},\mathcal{F}_{\mathsf{Syn}}}_{\Pi_{\mathsf{EOT\text{-}RO}},\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}_{\mathcal{F}_{\mathsf{EOT}},\mathcal{S},\mathcal{Z}}$ holds.

The receiver is honest while the sender is statically corrupted. Here the simulator $\mathcal{S}$ needs to extract the secret randomness of the sender from the message sent from S^* and computes both messages. Note that, the simulator $\mathcal{S}$ needs to send its message on behalf of the honest receiver before seeing the adversary $\mathcal{A}$'s message. We describe the strategy of $\mathcal{S}$ as follows:

- Select a random bit $b \in \{0, 1\}$ and generate $(\mathsf{seed}, u, v, \pi_{\mathsf{ENC}})$ honestly. Send $(\mathsf{seed}, u, v, \pi_{\mathsf{ENC}})$ to S^*.
- Wait for S^* to send $(\mathsf{seed}, z, \pi_{\mathsf{DL}})$, then do the following:
  - Abort if $\Pi_{\mathsf{sleNIZK}}.\mathsf{Verify}^{\mathcal{F}_{\mathsf{RO3}}}((g, z), \pi_{\mathsf{DL}}) = 0$ for $\mathcal{R}_{\mathsf{DL}}$.
  - Invoke the straight-line extractor $\Pi_{\mathsf{sleNIZK}}.\mathsf{Ext}$ to obtain s such that $z = g^s$ (note that, the proof π_{DL} is verified, and the simulator $\mathcal{S}$ can see the query-answer list of $\mathcal{F}_{\mathsf{RO3}}$ since $\mathcal{F}_{\mathsf{RO3}}$ is simulated by $\mathcal{S}$, and thus $\mathcal{S}$ is able to invoke the extractor algorithm).
  - Compute $m_0 := \mathcal{F}_{\mathsf{RO2}}(\mathsf{sid}, \text{'}S^*\text{'}||v^r)$ and $m_1 := \mathcal{F}_{\mathsf{RO2}}(\mathsf{sid}, \text{'}S^*\text{'}||(\frac{v}{h})^r)$.
  - Send $(\textsc{Send}, \mathsf{sid}, S^*, R)$ to $\mathcal{F}_{\mathsf{EOT}}$ on behalf of the dummy corrupted sender. Return $(\textsc{Proceed}, \mathsf{sid}, R, m_0, m_1)$ to $\mathcal{F}_{\mathsf{EOT}}$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, R)$.
- If S^* does not send its message, send $(\textsc{Send}, \mathsf{sid}, S, R^*)$ to $\mathcal{F}_{\mathsf{EOT}}$ on behalf of the dummy corrupted sender. Return $(\textsc{No}, \mathsf{sid}, R)$ to $\mathcal{F}_{\mathsf{EOT}}$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, R)$.

We prove the indistinguishability through the following hybrid experiments:

- $\mathcal{H}_0$: This is the real world execution $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{RO}},\mathcal{F}_{\mathsf{Syn}}}_{\Pi_{\mathsf{EOT\text{-}RO}},\mathcal{A},\mathcal{Z}}$.
- $\mathcal{H}_1$: Same as $\mathcal{H}_0$, except that $\mathcal{S}$ aborts when π_{DL} is not valid. Indistinguishability follows from the fact that honest receiver would always abort when π_{DL} is not valid.
- $\mathcal{H}_2$: Same as $\mathcal{H}_1$, except that $\mathcal{S}$ extracts s by invoking the straight-line extractor $\Pi_{\mathsf{sleNIZK}}.\mathsf{Ext}^{\mathcal{F}_{\mathsf{RO3}}}$ and computes m_0, m_1. Indistinguishability follows from the straight-line extractability of Π_{sleNIZK}.
- $\mathcal{H}_3$: Same as $\mathcal{H}_2$, except that $\mathcal{S}$ sends $(\textsc{Send}, \mathsf{sid}, S^*, R)$ to $\mathcal{F}_{\mathsf{EOT}}$ on behalf of the dummy corrupted sender, and returns $(\textsc{Proceed}, \mathsf{sid}, R, m_0, m_1)$ to $\mathcal{F}_{\mathsf{EOT}}$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, R)$. The simulator $\mathcal{S}$ aborts when S^* extracts b from the messages sent by $\mathcal{S}$.

Lemma 4. *Assume the DDH assumption holds in $\mathbb{G}$. Let Π_{NIZK} be an NIZK argument in the RO model. Hybrid $\mathcal{H}_3$ is computationally indistinguishable from $\mathcal{H}_1$. Hybrid $\mathcal{H}_3$ is computationally indistinguishable from $\mathcal{H}_2$.*

Proof. We show that any PPT S^* cannot extract b from (u, v) or π_{ENC} except with a negligible probability. It is easy to see that (u, v) is the ElGammal encryption of b, so b is computationally hidden in (u, v). The zero-knowledge property of Π_{NIZK} guarantees that no PPT S^* can learn any information about b from π_{ENC}. Therefore, $\mathcal{S}$ aborts at a negligible probability. In conclusion, $\mathcal{H}_3$ is computationally indistinguishable from $\mathcal{H}_2$. □

- $\mathcal{H}_4$: Same as $\mathcal{H}_3$, except that when S^* does not send its message, the simulator $\mathcal{S}$ sends $(\textsc{Send}, \mathsf{sid}, S, R^*)$ to $\mathcal{F}_{\mathsf{EOT}}$ on behalf of the dummy corrupted sender, and returns $(\textsc{No}, \mathsf{sid}, R)$ to $\mathcal{F}_{\mathsf{EOT}}$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, R)$.

Hybrid $\mathcal{H}_4$ is identical to the ideal world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{EOT}},\mathcal{S},\mathcal{Z}}$. In conclusion, when the receiver R is honest and the sender S^* is statically corrupted, $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{RO}},\mathcal{F}_{\mathsf{Syn}}}_{\Pi_{\mathsf{EOT\text{-}RO}},\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}_{\mathcal{F}_{\mathsf{EOT}},\mathcal{S},\mathcal{Z}}$ holds. □

C.2 Proof of Theorem 2

Theorem 2. *The protocol Π_{Com} depicted in Figure 14 UC-realizes the functionality $\mathcal{F}_{\mathsf{Com}}$ depicted in Figure 9 with unconditional security in the $\{\mathcal{F}_{\mathsf{EOT}}, \mathcal{F}_{\mathsf{Auth}}\}$-hybrid world against static malicious corruption.*

Proof. We now prove the security of our protocol Π_{Com} by showing it is a UC-secure realization of $\mathcal{F}_{\mathsf{Com}}$. We describe the workflow of $\mathcal{S}$ in the ideal-world with $\mathcal{F}_{\mathsf{Com}}$, and give a proof that the simulation in the ideal world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{Com}},\mathcal{S},\mathcal{Z}}$ is statistically indistinguishable from a real world execution $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{EOT}},\mathcal{F}_{\mathsf{Auth}}}_{\Pi_{\mathsf{Com}},\mathcal{A},\mathcal{Z}}$ for any adversary $\mathcal{A}$ and any environment $\mathcal{Z}$. Note that, the simulator $\mathcal{S}$ simply forwards the communication between $\mathcal{A}$ and $\mathcal{Z}$.

The committer is honest while the receiver is statically corrupted. Here the simulator $\mathcal{S}$ needs to generate an equivocate commitment message without knowing b. We describe the strategy of $\mathcal{S}$ as follows:

- Committing Phase:
  - Simulate $\mathcal{F}_{\mathsf{EOT}}$. Upon receiving $(\textsc{Proceed?}, \mathsf{sid}, C, R^*)$ from $\mathcal{F}_{\mathsf{Com}}$, send $(\textsc{Proceed?}, \mathsf{sid}, C)$ to the adversary on behalf of $\mathcal{F}_{\mathsf{EOT}}$. Upon receiving $(\textsc{Proceed}, \mathsf{sid}, R^*, C, m_0, m_1)$ from the adversary, output $(\textsc{Received}, \mathsf{sid}, R^*, C, m_0, m_1)$ to R^* on behalf of $\mathcal{F}_{\mathsf{EOT}}$, and return $(\textsc{Proceed}, \mathsf{sid}, C, R^*)$ to $\mathcal{F}_{\mathsf{Com}}$; Upon receiving $(\textsc{No}, \mathsf{sid}, C)$ from the adversary, let the honest committer abort and return $(\textsc{No}, \mathsf{sid}, C)$ to $\mathcal{F}_{\mathsf{Com}}$.
- Opening Phase:
  - Upon receiving $(\textsc{Proceed?}, \mathsf{sid}, R^*, C)$ from $\mathcal{F}_{\mathsf{Com}}$, return $(\textsc{Proceed}, \mathsf{sid}, R^*, C)$. Upon receiving $(\textsc{Decommit}, \mathsf{sid}, C, R^*, b)$ from $\mathcal{F}_{\mathsf{Com}}$, send (b, m_b) to R^* on behalf of the honest committer.

We prove the indistinguishability through the following hybrid experiments:

- $\mathcal{H}_0$: This is the real world execution $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{EOT}},\mathcal{F}_{\mathsf{Auth}}}_{\Pi_{\mathsf{Com}},\mathcal{A},\mathcal{Z}}$.
- $\mathcal{H}_1$: Same as $\mathcal{H}_0$, except that $\mathcal{S}$ obtains (m_0, m_1) by playing the role of the $\mathcal{F}_{\mathsf{EOT}}$ and outputs (m_0, m_1) as the commitment message to R^*, when the adversary agrees to proceed the protocol. The simulator $\mathcal{S}$ aborts when R^* extracts b from (m_0, m_1). Perfect indistinguishability holds since b is independent of m_0, m_1; in other words, any malicious malicious R^* cannot learn b.
- $\mathcal{H}_2$: Same as $\mathcal{H}_1$, except that $\mathcal{S}$ lets the honest committer abort when the adversary refuses to continue the protocol. Perfect indistinguishability holds since the honest committer will abort in both ideal world and real world when the malicious party refuse to continue the protocol.
- $\mathcal{H}_3$: Same as $\mathcal{H}_2$, except that $\mathcal{S}$ sends (b, m_b) to R^* after receiving $(\textsc{Decommit}, \mathsf{sid}, C, R, b)$ from $\mathcal{F}_{\mathsf{Com}}$. Perfect indistinguishability holds since $\mathcal{S}$ does not modify anything.

Hybrid $\mathcal{H}_3$ is identical to the ideal world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{Com}},\mathcal{S},\mathcal{Z}}$. In conclusion, when the committer C is honest and the receiver R^* is statically corrupted, $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{EOT}},\mathcal{F}_{\mathsf{Auth}}}_{\Pi_{\mathsf{Com}},\mathcal{A},\mathcal{Z}} \equiv \mathsf{EXEC}_{\mathcal{F}_{\mathsf{Com}},\mathcal{S},\mathcal{Z}}$ holds.

The receiver is honest while the committer is statically corrupted. Here the simulator $\mathcal{S}$ needs to extract b from the commitment message sent by the malicious C^*. We describe the strategy of $\mathcal{S}$ as follows:

- Committing Phase:
  - Simulate $\mathcal{F}_{\mathsf{EOT}}$. Upon receiving and $(\textsc{Receive}, \mathsf{sid}, R, C^*, b)$ from C^*, sends $(\textsc{Proceed?}, \mathsf{sid}, R)$ to the adversary on behalf of $\mathcal{F}_{\mathsf{EOT}}$.
  - Upon receiving $(\textsc{Proceed}, \mathsf{sid}, R, m_b)$ from the adversary, sample an uniformly random $m_{1-b} \leftarrow \{0,1\}^\lambda$ and output $(\textsc{Received}, \mathsf{sid}, R, C^*, m_b)$ to C^* on behalf of $\mathcal{F}_{\mathsf{EOT}}$. Send $(\textsc{Commit}, \mathsf{sid}, C^*, R, b)$ to $\mathcal{F}_{\mathsf{Com}}$ on behalf of the dummy corrupted committer. Return $(\textsc{Proceed}, \mathsf{sid}, C^*, R)$ to $\mathcal{F}_{\mathsf{Com}}$ when $\mathcal{F}_{\mathsf{Com}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, C^*, R)$.
  - Upon receiving $(\textsc{No}, \mathsf{sid}, R)$ from the adversary, pick a random bit b and send $(\textsc{Commit}, \mathsf{sid}, C^*, R, b)$ to $\mathcal{F}_{\mathsf{Com}}$ on behalf of the dummy corrupted committer. Return $(\textsc{No}, \mathsf{sid}, C^*, R)$ to $\mathcal{F}_{\mathsf{Com}}$ when $\mathcal{F}_{\mathsf{Com}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, C^*, R)$.
- Opening Phase:
  - Wait for C^* to send (b', m). If $b' = 1 - b$ and $m = m_{1-b}$ hold or C^* does not send its message, return $(\textsc{No}, \mathsf{sid}, C^*, R)$ to $\mathcal{F}_{\mathsf{Com}}$ when $\mathcal{F}_{\mathsf{Com}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, C^*, R)$; otherwise, return $(\textsc{Proceed}, \mathsf{sid}, C^*, R)$ to $\mathcal{F}_{\mathsf{Com}}$.

We prove the indistinguishability through the following hybrid experiments:

- $\mathcal{H}_0$: This is the real world execution $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{EOT}}, \mathcal{F}_{\mathsf{Auth}}}_{\Pi_{\mathsf{Com}}, \mathcal{A}, \mathcal{Z}}$.
- $\mathcal{H}_1$: Same as $\mathcal{H}_0$, except that $\mathcal{S}$ obtains (b, m_b) by playing the role of the $\mathcal{F}_{\mathsf{EOT}}$ and outputs m_b to C^*, when the adversary agrees to proceed the protocol. Perfect indistinguishability holds since $\mathcal{S}$ does not modify anything.
- $\mathcal{H}_2$: Same as $\mathcal{H}_1$, except that $\mathcal{S}$ lets the honest receiver abort when the adversary refuses to continue the protocol. Perfect indistinguishability holds since the honest receiver will abort in both ideal world and real world when the malicious party refuses to continue the protocol.
- $\mathcal{H}_3$: Same as $\mathcal{H}_2$, except that $\mathcal{S}$ lets the honest receiver abort when C^* sends $(1 - b, m_{1-b})$ or C^* does not send its message. For the former case, indistinguishability follows from the fact that the message m_{1-b} is sampled uniformly, thus the probability of the adversary guessing the m_{1-b} correctly is $2^{-\lambda}$, which is negligible. For the latter case, indistinguishability follows from the fact that the honest receiver will abort in both ideal world and real world when the malicious party refuses to continue the protocol. In conclusion, $\mathcal{H}_3$ is statistically indistinguishable from $\mathcal{H}_2$.

Hybrid $\mathcal{H}_3$ is identical to the ideal world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{Com}}, \mathcal{S}, \mathcal{Z}}$. In conclusion, when the receiver R is honest while the committer C^* is statically corrupted, $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{EOT}}, \mathcal{F}_{\mathsf{Auth}}}_{\Pi_{\mathsf{Com}}, \mathcal{A}, \mathcal{Z}}$ is statistically indistinguishable $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{Com}}, \mathcal{S}, \mathcal{Z}}$. $\square$

C.3 Proof of Theorem 3

Theorem 3. *The protocol Π_{Coin} depicted in Figure 16 UC-realizes the functionality $\mathcal{F}_{\mathsf{Coin}}$ depicted in Figure 5 with unconditional security in the $\{\mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{Auth}}\}$-hybrid world against static malicious corruption.*

Proof. We now prove the security of our protocol Π_{Coin} by showing it is a UC-secure realization of $\mathcal{F}_{\mathsf{Coin}}$. We describe the workflow of $\mathcal{S}$ in the ideal-world with $\mathcal{F}_{\mathsf{Coin}}$, and give a proof that the simulation in the ideal world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{Coin}}, \mathcal{S}, \mathcal{Z}}$ is perfectly indistinguishable from a real world execution $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{Auth}}}_{\Pi_{\mathsf{Coin}}, \mathcal{A}, \mathcal{Z}}$ for any adversary $\mathcal{A}$ and any environment $\mathcal{Z}$. Note that, the simulator $\mathcal{S}$ simply forwards the communication between $\mathcal{A}$ and $\mathcal{Z}$.

The player 1 is honest while the player 2 is statically corrupted. Here the simulator $\mathcal{S}$ needs to generate an equivocate commitment message and later open it to any message. We describe the strategy of $\mathcal{S}$ as follows:

- Round 1: Simulate $\mathcal{F}_{\mathsf{Com}}$. Send $(\textsc{Proceed?}, \mathsf{sid}, P_1, P_2^*)$ to the adversary on behalf of $\mathcal{F}_{\mathsf{Com}}$. Upon receiving $(\textsc{Proceed}, \mathsf{sid}, P_1, P_2^*)$ from the adversary, output $(\textsc{Receipt}, \mathsf{sid}, P_1, P_2^*)$ to P_2^* on behalf of $\mathcal{F}_{\mathsf{Com}}$. Upon receiving $(\textsc{No}, \mathsf{sid}, P_1, P_2^*)$ from the adversary, return $(\textsc{No}, \mathsf{sid}, P_1)$ (resp. $(\textsc{No}, \mathsf{sid}, P_2^*)$) to $\mathcal{F}_{\mathsf{Coin}}$ when $\mathcal{F}_{\mathsf{Coin}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, P_1)$ (resp. $(\textsc{Proceed?}, \mathsf{sid}, P_2^*)$).
- Round 2: Wait for P_2^* to send m_2. If P_2^* does not send the message, return $(\textsc{No}, \mathsf{sid}, P_1)$ (resp. $(\textsc{No}, \mathsf{sid}, P_2^*)$) to $\mathcal{F}_{\mathsf{Coin}}$ when $\mathcal{F}_{\mathsf{Coin}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, P_1)$ (resp. $(\textsc{Proceed?}, \mathsf{sid}, P_2^*)$).

- Round 3: Return $(\textsc{Proceed}, \mathsf{sid}, P_1)$ (resp. $(\textsc{Proceed}, \mathsf{sid}, P_2^*)$) to $\mathcal{F}_{\mathsf{Coin}}$ when $\mathcal{F}_{\mathsf{Coin}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, P_1)$ (resp. $(\textsc{Proceed?}, \mathsf{sid}, P_2^*)$). Upon receiving $(\textsc{Tossed}, \mathsf{sid}, P_1, P_2^*, r)$ from $\mathcal{F}_{\mathsf{Coin}}$, compute $m_1 := r \oplus m_2$, and output $(\textsc{Decommit}, \mathsf{sid}, P_1, P_2^*, m_1)$ to P_2^* on behalf of $\mathcal{F}_{\mathsf{Com}}$.

We prove the indistinguishability through the following hybrid experiments:

- $\mathcal{H}_0$: This is the real world execution $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{Auth}}}_{\Pi_{\mathsf{Com}}, \mathcal{A}, \mathcal{Z}}$.
- $\mathcal{H}_1$: Same as $\mathcal{H}_0$, except that $\mathcal{S}$ plays the role of the $\mathcal{F}_{\mathsf{Com}}$ and outputs $(\textsc{Receipt}, \mathsf{sid}, P_1, P_2^*)$ to P_2^*, when the adversary agrees to proceed the protocol. Perfect indistinguishability holds since in both $\mathcal{H}_0$ and $\mathcal{H}_1$, the malicious P_2^* would receive $(\textsc{Receipt}, \mathsf{sid}, P_1, P_2^*)$ from $\mathcal{F}_{\mathsf{Com}}$.
- $\mathcal{H}_2$: Same as $\mathcal{H}_1$, except that $\mathcal{S}$ lets the honest player 1 abort when the adversary refuses to continue the protocol. Perfect indistinguishability holds since the honest player 1 will abort in both ideal world and real world when the malicious player 2 refuses to continue the protocol.
- $\mathcal{H}_3$: Same as $\mathcal{H}_2$, except that $\mathcal{S}$ computes $m_1 := r \oplus m_2$, where m_2 is sent by P_2^* and r is sent by $\mathcal{F}_{\mathsf{Coin}}$, and outputs $(\textsc{Decommit}, \mathsf{sid}, P_1, P_2^*, m_1)$ to P_2^* on behalf of $\mathcal{F}_{\mathsf{Com}}$. Perfect indistinguishability holds since $\mathcal{F}_{\mathsf{Com}}$ is simulated by $\mathcal{S}$.

Hybrid $\mathcal{H}_3$ is identical to the ideal world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{Coin}}, \mathcal{S}, \mathcal{Z}}$. In conclusion, when the the player 1 P_1 is honest while the player 2 P_2^* is statically corrupted, $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{Auth}}}_{\Pi_{\mathsf{Coin}}, \mathcal{A}, \mathcal{Z}} \equiv \mathsf{EXEC}_{\mathcal{F}_{\mathsf{Coin}}, \mathcal{S}, \mathcal{Z}}$ holds.

The player 2 is honest while the player 1 is statically corrupted. Here the simulator $\mathcal{S}$ needs to extract m_1 from the commitment message sent by P_1. We describe the strategy of $\mathcal{S}$ as follows:

- Round 1: Simulate $\mathcal{F}_{\mathsf{Com}}$, and receive $(\textsc{Commit}, \mathsf{sid}, P_1^*, P_2, m_1)$ from P_1^* on behalf of $\mathcal{F}_{\mathsf{Com}}$. If P_1^* does not send the message, return $(\textsc{No}, \mathsf{sid}, P_1^*)$ (resp. $(\textsc{No}, \mathsf{sid}, P_2)$) to $\mathcal{F}_{\mathsf{Coin}}$ when $\mathcal{F}_{\mathsf{Coin}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, P_1^*)$ (resp. $(\textsc{Proceed?}, \mathsf{sid}, P_2)$).
- Round 2: Return $(\textsc{Proceed}, \mathsf{sid}, P_1^*)$ to $\mathcal{F}_{\mathsf{Coin}}$ when $\mathcal{F}_{\mathsf{Coin}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, P_1^*)$. Upon receiving $(\textsc{Tossed}, \mathsf{sid}, P_1^*, P_2, r)$ from $\mathcal{F}_{\mathsf{Coin}}$, compute and send $m_2 := r \oplus m_1$ to P_1^*.
- Round 3: Wait for P_1^* to open the commitment. If P_1^* does not open the commitment, return $(\textsc{No}, \mathsf{sid}, P_2)$ when $\mathcal{F}_{\mathsf{Coin}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, P_2)$; else, return $(\textsc{Proceed}, \mathsf{sid}, P_2)$.

We prove the indistinguishability through the following hybrid experiments:

- $\mathcal{H}_0$: This is the real world execution $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{Auth}}}_{\Pi_{\mathsf{Coin}}, \mathcal{A}, \mathcal{Z}}$.
- $\mathcal{H}_1$: Same as $\mathcal{H}_0$, except that when the adversary agrees to proceed the protocol, the simulator $\mathcal{S}$ obtains m_1 by playing the role of the $\mathcal{F}_{\mathsf{Com}}$, and sends $m_2 := r \oplus m_1$ to P_1^* where r is sent by $\mathcal{F}_{\mathsf{Coin}}$ and r is uniformly random. Perfect indistinguishability holds since m_2 in both $\mathcal{H}_1$ and $\mathcal{H}_0$ is uniformly random.
- $\mathcal{H}_2$: Same as $\mathcal{H}_1$, except that if P_1^* does not open the commitment, $\mathcal{S}$ will $(\textsc{No}, \mathsf{sid}, P_2)$ when $\mathcal{F}_{\mathsf{Coin}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, P_2)$. Perfect indistinguishability holds since the honest P_2 would always aborts if the malicious P_1^* does not open the commitment.
- $\mathcal{H}_3$: Same as $\mathcal{H}_2$, except that if P_1^* opens the commitment, $\mathcal{S}$ will return $(\textsc{Proceed}, \mathsf{sid}, P_2)$ when $\mathcal{F}_{\mathsf{Coin}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, P_2)$. Perfect indistinguishability holds since the protocol participants would receive r as final output in both $\mathcal{H}_3$ and $\mathcal{H}_2$.

Hybrid $\mathcal{H}_3$ is identical to the ideal world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{Coin}}, \mathcal{S}, \mathcal{Z}}$. In conclusion, when the player 2 P_2 is honest while the player 1 P_1^* is statically corrupted, $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{Auth}}}_{\Pi_{\mathsf{Coin}}, \mathcal{A}, \mathcal{Z}} \equiv \mathsf{EXEC}_{\mathcal{F}_{\mathsf{Coin}}, \mathcal{S}, \mathcal{Z}}$ holds. □

C.4 Proof of Theorem 4

Theorem 4. *Assume the CDH assumption holds in group $\mathbb{G}$. Let $\mathcal{G}_{\mathsf{roRO1}} : \{0,1\}^\lambda \to \mathbb{G}$ and $\mathcal{G}_{\mathsf{roRO2}} : \mathbb{G} \to \{0,1\}^\lambda$ be the random oracles. Let Π^S_{sleNIWH} be a straight-line extractable NIWH argument in the $\mathcal{G}_{\mathsf{roRO3}}$-hybrid world. Let Π^R_{sleNIWH} be a straight-line extractable NIWH argument in the $\mathcal{G}_{\mathsf{roRO4}}$-hybrid world. The protocol $\Pi_{\mathsf{EOT\text{-}GroRO}}$ depicted in Figure 17 GUC-realizes the functionality $\mathcal{F}_{\mathsf{tEOT}}$ depicted in Figure 18 in the $\{\mathcal{G}_{\mathsf{roRO}}, \mathcal{F}_{\mathsf{Syn}}\}$-hybrid world against static malicious corruption, where $\mathcal{G}_{\mathsf{roRO}} = \{\mathcal{G}_{\mathsf{roRO}i}\}_{i \in [4]}$.*

Proof. We now prove the security of our protocol $\Pi_{\mathsf{EOT\text{-}GroRO}}$ by showing it is a GUC-secure realization of $\mathcal{F}_{\mathsf{tEOT}}$. We only need to prove that $\Pi_{\mathsf{EOT\text{-}GroRO}}$ EUC-realizes $\mathcal{F}_{\mathsf{tEOT}}$ with respect to the shared functionality $\mathcal{G}_{\mathsf{roRO}}$, where $\mathcal{G}_{\mathsf{roRO}} = (\{\mathcal{G}_{\mathsf{roRO}i}\}_{i\in[4]})$. Therefore, we describe the workflow of $\mathcal{S}$ in the ideal-world with $\mathcal{F}_{\mathsf{tEOT}}$, and give a proof that the simulation in the ideal world execution $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{roRO}}}_{\mathcal{F}_{\mathsf{tEOT}},\mathcal{S},\mathcal{Z}}$ is computationally indistinguishable from a real world execution $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{roRO}},\mathcal{F}_{\mathsf{Syn}}}_{\Pi_{\mathsf{EOT\text{-}GrpRO}},\mathcal{A},\mathcal{Z}}$ for any PPT adversary $\mathcal{A}$ and any PPT $\mathcal{G}_{\mathsf{roRO}}$-constrained environment $\mathcal{Z}$. Note that, the simulator $\mathcal{S}$ simply forwards the communication between $\mathcal{A}$ and $\mathcal{Z}$.

The sender is honest while the receiver is statically corrupted. Here the simulator $\mathcal{S}$ needs to extract the choice bit of the receiver from the message sent from malicious receiver R^*. Note that, the simulator $\mathcal{S}$ needs to send its message on behalf of the honest sender before seeing the adversary $\mathcal{A}$'s message. We describe the strategy of $\mathcal{S}$ as follows:

- Generate (z, π_{DL}) honestly, and send (z, π_{DL}) to R^*.
- Wait for R^* to send $(\mathsf{seed}, B, \pi_{\mathsf{Com}})$, then do the following:
  - Compute $h := \mathcal{G}_{\mathsf{roRO1}}(\mathsf{sid}, `R'||\mathsf{seed})$.
  - Abort if $\Pi^{R}_{\mathsf{sleNIWH}}.\mathsf{Verify}^{\mathcal{G}_{\mathsf{roRO4}}}((B, g, h), \pi_{\mathsf{Com}}) = 0$ for $\mathcal{R}_{\mathsf{Com}}$.
  - Request illegitimate queries $\mathcal{Q}_{\mathsf{sid}}$ from $\mathcal{F}_{\mathsf{tEOT}}$.
  - Invoke the straight-line extractor $\Pi^{R}_{\mathsf{sleNIWH}}.\mathsf{Ext}$ to obtain (b, r) such that $B = g^r h^b$ (note that, the proof π_{Com} is verified, and the simulator $\mathcal{S}$ obtains the RO queries and answers posed by the adversary, and thus $\mathcal{S}$ is able to invoke the extractor algorithm).
  - If $b \notin \{0, 1\}$, act as the honest sender to until the end of the protocol.
  - Else, compute m_0, m_1 honestly, and send $(\textsc{Receive}, \mathsf{sid}, S, R^*, b)$ to $\mathcal{F}_{\mathsf{tEOT}}$ on behalf of the dummy corrupted receiver. Return $(\textsc{Proceed}, \mathsf{sid}, S, m_b)$ to $\mathcal{F}_{\mathsf{tEOT}}$ when $\mathcal{F}_{\mathsf{tEOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, S)$.
- If R^* does not send its message, pick a random bit b and send $(\textsc{Receive}, \mathsf{sid}, S, R^*, b)$ to $\mathcal{F}_{\mathsf{tEOT}}$ on behalf of the dummy corrupted receiver. Return $(\textsc{No}, \mathsf{sid}, S)$ to $\mathcal{F}_{\mathsf{tEOT}}$ when $\mathcal{F}_{\mathsf{tEOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, S)$.

We prove the indistinguishability through the following hybrid experiments:

- $\mathcal{H}_0$: This is the real world execution $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{roRO}},\mathcal{F}_{\mathsf{Syn}}}_{\Pi_{\mathsf{EOT\text{-}GroRO}},\mathcal{A},\mathcal{Z}}$.
- $\mathcal{H}_1$: Same as $\mathcal{H}_0$, except that $\mathcal{S}$ aborts when π_{Com} is not valid. Indistinguishability follows from the fact that honest sender would always abort when π_{Com} is not valid.
- $\mathcal{H}_2$: Same as $\mathcal{H}_1$, except that $\mathcal{S}$ obtains b by invoking the straight-line extractor $\Pi^{R}_{\mathsf{sleNIWH}}.\mathsf{Ext}^{\mathcal{G}_{\mathsf{roRO4}}}$. Indistinguishability follows from the straight-line extractability of Π_{sleNIWH}.
- $\mathcal{H}_3$: Same as $\mathcal{H}_2$, except that $b \notin \{0, 1\}$ and $\mathcal{S}$ acts as the honest sender to until the end of the protocol. The simulator $\mathcal{S}$ aborts when R^* computes either m_0 or m_1.

  Lemma 5. *Assume the CDH assumption holds in $\mathbb{G}$, then $\mathcal{H}_3$ is computationally indistinguishable from $\mathcal{H}_2$.*

  Proof. Here we argue that when $b \notin \{0, 1\}$, the malicious receiver R^* can compute m_0 or m_1 with only negligible probability. Note that, the malicious R^* compute m_0 (m_1) by querying $z^r h^{s\cdot b}$ ($z^r h^{s(b-1)}$) to $\mathcal{G}_{\mathsf{roRO2}}$. We discuss the case where R^* computes m_0 by querying $z^r h^{sb}$ to $\mathcal{G}_{\mathsf{roRO2}}$ here, and the other case is similar.

  More precisely, we observe that if there exists such a R^*, then there is a reduction $\mathcal{B}$ which breaks the CDH assumption. The reduction $\mathcal{B}$ interacts with the CDH game challenger $\mathcal{C}$ and receives (A_1, A_2, A_3) from $\mathcal{C}$, and we assume that $A_1 := g, A_2 := g^t, A_3 := g^s$. Then $\mathcal{B}$ simulates $\{\mathcal{G}_{\mathsf{roRO}i}\}_{i\in[4]}$ and starts the protocol $\Pi_{\mathsf{EOT\text{-}GroRO}}$ with R^* by running R^* internally as a black-box. Note that $\mathcal{B}$ has full control over $\{\mathcal{G}_{\mathsf{roRO}i}\}_{i\in[4]}$. First $\mathcal{B}$ sets $z := A_3$ and simulates the straight-line extractable NIWH argument π_{DL} by programming the random oracles, and sends z, π_{DL} to R^*. When R^* queries a λ-bit string seed_i to $\mathcal{G}_{\mathsf{roRO1}}$, $\mathcal{B}$ returns $A_2 \cdot g^{a_i}$ where $a_i \xleftarrow{\$} \mathbb{Z}_q$. Upon receiving $(B, \pi_{\mathsf{Com}}, \mathsf{seed})$ from R^*, $\mathcal{B}$ looks up the query-answer table of $\mathcal{G}_{\mathsf{roRO1}}$ and finds the index j such that $\mathsf{seed} = \mathsf{seed}_j$ and $h = A_2 \cdot g^{a_j}$. After that, $\mathcal{B}$ invokes the straight-line extractor $\Pi^{R}_{\mathsf{sleNIWH}}.\mathsf{Ext}^{\mathcal{G}_{\mathsf{roRO4}}}$ to obtain (b, r) such that $B = g^r h^b$. Finally, $\mathcal{B}$ randomly selects a query q made to $\mathcal{G}_{\mathsf{rpRO2}}$ by R^* and sends $A_4 := (\frac{q}{A_3^{r+b\cdot a_j}})^{b^{-1}}$ to $\mathcal{C}$. If there are Q queries made to $\mathcal{G}_{\mathsf{roRO2}}$ by R^* in total, then

$A_4 = (\frac{q}{A_3^{r+b\cdot a_j}})^{b^{-1}} = (\frac{z^r h^{s\cdot b}}{z^r z^{b\cdot a j}})^{b^{-1}} = (\frac{A_2^{s\cdot b}\cdot g^{s\cdot b\cdot a_j}}{g^{s\cdot b\cdot a_j}})^{b^{-1}} = (A_2^{s\cdot b})^{b^{-1}} = g^{st}$ happens at probability $\frac{1}{Q}$. Therefore, $\mathcal{B}$ wins the CDH game at probability $\frac{1}{Q}$ which is non-negligible. In conclusion, $\mathcal{H}_3$ is computationally indistinguishable from $\mathcal{H}_2$. □

- $\mathcal{H}_4$: Same as $\mathcal{H}_3$, except that $\mathcal{S}$ extracts $b \in \{0,1\}$, computes m_0, m_1 honestly, sends $(\textsc{Receive}, \mathsf{sid}, S, R^*, b)$ to the $\mathcal{F}_{\mathsf{tEOT}}$ on behalf of the dummy corrupted receiver, and return $(\textsc{Proceed}, \mathsf{sid}, S, m_b)$ to $\mathcal{F}_{\mathsf{tEOT}}$ when $\mathcal{F}_{\mathsf{tEOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, S)$. The simulator $\mathcal{S}$ aborts when R^* computes both m_0 and m_1.

Lemma 6. *Assume the CDH assumption holds in $\mathbb{G}$, $\mathcal{H}_4$ is computationally indistinguishable from $\mathcal{H}_3$.*

Proof. The simulator $\mathcal{S}$ aborts when R^* computes m_0 and m_1 by setting $b = 0$ ($b = 1$) and querying $z^r, \frac{z^r}{h^s}$ ($z^r h^s$) to $\mathcal{G}_{\mathsf{roRO2}}$. We discuss the case where R^* sets $b = 0$ and queries $z^r, \frac{z^r}{h^s}$ to $\mathcal{G}_{\mathsf{roRO2}}$ here, and the other case is similar.

We observe that if there exists such a R^*, then there is a reduction $\mathcal{B}$ which breaks the CDH assumption. The reduction $\mathcal{B}$ interacts with the CDH game challenger $\mathcal{C}$ and receives (A_1, A_2, A_3) from $\mathcal{C}$, and we assume that $A_1 := g, A_2 := g^t, A_3 := g^s$. Then $\mathcal{B}$ simulates $\{\mathcal{G}_{\mathsf{roRO}i}\}_{i\in[4]}$ and starts the protocol $\Pi_{\mathsf{EOT\text{-}GroRO}}$ with R^* by running R^* internally as a black-box. Note that $\mathcal{B}$ has full control over $\{\mathcal{G}_{\mathsf{roRO}i}\}_{i\in[4]}$. First $\mathcal{B}$ sets $z := A_3$ and simulates the straight-line extractable NIWH argument π_{DL} by programming the random oracles, and sends z, π_{DL} to R^*. When R^* queries a λ-bit string seed_i to $\mathcal{G}_{\mathsf{roRO1}}$, $\mathcal{B}$ returns $A_2 \cdot g^{a_i}$ where $a_i \overset{\$}{\leftarrow} \mathbb{Z}_q$. Upon receiving $(B, \pi_{\mathsf{Com}}, \mathsf{seed})$ from R^*, $\mathcal{B}$ looks up the query-answer table of $\mathcal{G}_{\mathsf{roRO1}}$ and finds the index j such that $\mathsf{seed} = \mathsf{seed}_j$ and $h = A_2 \cdot g^{a_j}$. Finally, $\mathcal{B}$ randomly selects two queries q_1, q_2 made to $\mathcal{G}_{\mathsf{rpRO2}}$ by R^* and sends $A_4 := \frac{q_1}{q_2 A_3^{a_j}}$ to $\mathcal{C}$. If there are Q queries made to $\mathcal{G}_{\mathsf{roRO2}}$ by R^* in total, then $A_4 = \frac{q_1}{q_2 A_3^{a_j}} = \frac{z^r}{\frac{z^r}{h^s} A_3^{a_j}} = \frac{A_2^s \cdot g^{s a_j}}{A_3^{a_j}} = g^{st}$ happens at probability $\frac{1}{2\cdot\binom{Q}{2}}$. Therefore, $\mathcal{B}$ wins the CDH game at probability $\frac{1}{2\cdot\binom{Q}{2}}$ which is non-negligible. In conclusion, $\mathcal{H}_4$ is computationally indistinguishable from $\mathcal{H}_3$. □

- $\mathcal{H}_5$: Same as $\mathcal{H}_4$, except that when R^* does not send its message, the simulator $\mathcal{S}$ picks a random bit b and sends $(\textsc{Receive}, \mathsf{sid}, S, R^*, b)$ to $\mathcal{F}_{\mathsf{tEOT}}$ on behalf of the dummy corrupted receiver, and returns $(\textsc{No}, \mathsf{sid}, S)$ to $\mathcal{F}_{\mathsf{tEOT}}$ when $\mathcal{F}_{\mathsf{tEOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, S)$. Indistinguishability follows from the fact that the honest sender will abort in both ideal world and the real world when the malicious receiver refuses to continue the protocol.

Hybrid $\mathcal{H}_5$ is identical to the ideal world execution $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{roRO}}}_{\mathcal{F}_{\mathsf{tEOT}},\mathcal{S},\mathcal{Z}}$. In conclusion, when the sender S is honest and the receiver R^* is statically corrupted, $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{roRO}},\mathcal{F}_{\mathsf{Syn}}}_{\Pi_{\mathsf{EOT\text{-}GroRO}},\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}^{\mathcal{G}_{\mathsf{roRO}}}_{\mathcal{F}_{\mathsf{tEOT}},\mathcal{S},\mathcal{Z}}$ holds.

The receiver is honest while the sender is statically corrupted. Here the simulator $\mathcal{S}$ needs to compute both m_0 and m_1. Note that, the simulator $\mathcal{S}$ needs to send its message on behalf of the honest receiver before seeing the adversary $\mathcal{A}$'s message. We describe the strategy of $\mathcal{S}$ as follows:

- Select $b \leftarrow \{0,1\}$, generate $(\mathsf{seed}, B, \pi_{\mathsf{Com}})$ honestly, and send $(\mathsf{seed}, B, \pi_{\mathsf{Com}})$ to R^*.
- Wait for S^* to send (z, π_{DL}), then do the following:
  - Abort if $\Pi^S_{\mathsf{sleNIWH}}.\mathsf{Verify}^{\mathcal{G}_{\mathsf{roRO3}}}((g,z), \pi_{\mathsf{DL}}) = 0$ for $\mathcal{R}_{\mathsf{DL}}$.
  - Request illegitimate queries $\mathcal{Q}_{\mathsf{sid}}$ from $\mathcal{F}_{\mathsf{tEOT}}$.
  - Invoke the straight-line extractor $\Pi^S_{\mathsf{sleNIWH}}.\mathsf{Ext}$ to obtain s such that $z = g^s$ (note that, the proof π_{DL} is verified, and the simulator $\mathcal{S}$ obtains the RO queries and answers posed by the adversary, and thus $\mathcal{S}$ is able to invoke the extractor algorithm).
  - Compute $m_0 := \mathcal{G}_{\mathsf{roRO2}}(\mathsf{sid}, `S^*'||B^s)$ and $m_1 := \mathcal{G}_{\mathsf{roRO2}}(\mathsf{sid}, `S^*'||(\frac{B}{h})^s)$.
  - Send $(\textsc{Send}, \mathsf{sid}, S^*, R)$ to $\mathcal{F}_{\mathsf{tEOT}}$ on behalf of the dummy corrupted sender. Return $(\textsc{Proceed}, \mathsf{sid}, R, m_0, m_1)$ to $\mathcal{F}_{\mathsf{tEOT}}$ when $\mathcal{F}_{\mathsf{tEOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, R)$.
- If S^* does not send its message, send $(\textsc{Send}, \mathsf{sid}, S, R^*)$ to $\mathcal{F}_{\mathsf{tEOT}}$ on behalf of the dummy corrupted sender. Return $(\textsc{No}, \mathsf{sid}, R)$ to $\mathcal{F}_{\mathsf{tEOT}}$ when $\mathcal{F}_{\mathsf{tEOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, R)$.

We prove the indistinguishability through the following hybrid experiments:

- $\mathcal{H}_0$: This is the real world execution $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{roRO}}, \mathcal{F}_{\mathsf{Syn}}}_{\Pi_{\mathsf{EOT\text{-}GroRO}}, \mathcal{A}, \mathcal{Z}}$.
- $\mathcal{H}_1$: Same as $\mathcal{H}_0$, except that $\mathcal{S}$ aborts when π_{DL} is not valid. Indistinguishability follows from the fact that honest receiver would always abort when π_{DL} is not valid.
- $\mathcal{H}_2$: Same as $\mathcal{H}_1$, except that $\mathcal{S}$ extracts s by invoking the straight-line extractor $\Pi^{\mathcal{S}}_{\mathsf{sleNIWH}}.\mathsf{Ext}^{\mathcal{G}_{\mathsf{roRO3}}}$ and computes m_0, m_1. Indistinguishability follows from the straight-line extractability of Π_{sleNIWH}.
- $\mathcal{H}_3$: Same as $\mathcal{H}_2$, except that $\mathcal{S}$ sends $(\textsc{Send}, \mathsf{sid}, S^*, R)$ to $\mathcal{F}_{\mathsf{tEOT}}$ on behalf of the dummy corrupted sender, and returns $(\textsc{Proceed}, \mathsf{sid}, R, m_0, m_1)$ to $\mathcal{F}_{\mathsf{tEOT}}$ when $\mathcal{F}_{\mathsf{tEOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, R)$. The simulator $\mathcal{S}$ aborts when S^* extracts b from the messages sent by $\mathcal{S}$.

  Lemma 7. *If Π_{sleNIWH} be a straight-line extractable NIWH argument in the RO model, then $\mathcal{H}_3$ is computationally indistinguishable from $\mathcal{H}_2$.*

  Proof. We only have to show that any PPT S^* cannot extract b from B or π_1. It is easy to see that B is the Pedersen commitment of b, so b is perfectly hidden in B due to the perfect hiding property. The witness hiding property of Π_{sleNIWH} guarantees that any PPT S^* cannot extract b from π_{Com}. Therefore, $\mathcal{S}$ aborts at a negligible probability. In conclusion, $\mathcal{H}_3$ is computationally indistinguishable from $\mathcal{H}_2$. □

- $\mathcal{H}_4$: Same as $\mathcal{H}_3$, except that when S^* does not send its message, the simulator $\mathcal{S}$ sends $(\textsc{Send}, \mathsf{sid}, S, R^*)$ to $\mathcal{F}_{\mathsf{tEOT}}$ on behalf of the dummy corrupted sender, and returns $(\textsc{No}, \mathsf{sid}, R)$ to $\mathcal{F}_{\mathsf{tEOT}}$ when $\mathcal{F}_{\mathsf{tEOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, R)$.

Hybrid $\mathcal{H}_4$ is identical to the ideal world execution $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{roRO}}}_{\mathcal{F}_{\mathsf{tEOT}}, \mathcal{S}, \mathcal{Z}}$. In conclusion, when the receiver R is honest and the sender S^* is statically corrupted, $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{roRO}}, \mathcal{F}_{\mathsf{Syn}}}_{\Pi_{\mathsf{EOT\text{-}GroRO}}, \mathcal{A}, \mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}^{\mathcal{G}_{\mathsf{roRO}}}_{\mathcal{F}_{\mathsf{tEOT}}, \mathcal{S}, \mathcal{Z}}$ holds. □

C.5 Proof of Theorem 6

Theorem 6. *Assume the DDH assumption holds in group $\mathbb{G}$. Let $\mathcal{G}_{\mathsf{rpRO1}} : \{0,1\}^\lambda \to \mathbb{G} \times \mathbb{G}$ and $\mathcal{G}_{\mathsf{rpRO2}} : \mathbb{G} \to \{0,1\}^\lambda$ be the random oracles. The protocol $\Pi_{\mathsf{EOT\text{-}GrpRO}}$ depicted in Figure 20 GUC-realizes the functionality $\mathcal{F}_{\mathsf{EOT}}$ depicted in Figure 8 in the $\{\mathcal{G}_{\mathsf{rpRO}}, \mathcal{F}_{\mathsf{Auth}}\}$-hybrid world against adaptive malicious corruption, where $\mathcal{G}_{\mathsf{rpRO}} = \{\mathcal{G}_{\mathsf{rpRO1}}, \mathcal{G}_{\mathsf{rpRO2}}\}$.*

Proof. We now prove the security of our protocol $\Pi_{\mathsf{EOT\text{-}GrpRO}}$ by showing it is a GUC-secure realization of $\mathcal{F}_{\mathsf{EOT}}$. We only need to prove that $\Pi_{\mathsf{EOT\text{-}GrpRO}}$ EUC-realizes $\mathcal{F}_{\mathsf{EOT}}$ with respect to the shared functionality $\mathcal{G}_{\mathsf{rpRO}}$, where $\mathcal{G}_{\mathsf{rpRO}} = \{\mathcal{G}_{\mathsf{rpRO1}}, \mathcal{G}_{\mathsf{rpRO2}}\}$. Therefore, we describe the workflow of $\mathcal{S}$ in the ideal-world with $\mathcal{F}_{\mathsf{EOT}}$, and give a proof that the simulation in the ideal world execution $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{rpRO}}}_{\mathcal{F}_{\mathsf{EOT}}, \mathcal{S}, \mathcal{Z}}$ is computationally indistinguishable from a real world execution $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{rpRO}}, \mathcal{F}_{\mathsf{Auth}}}_{\Pi_{\mathsf{EOT\text{-}GrpRO}}, \mathcal{A}, \mathcal{Z}}$ for any PPT adversary $\mathcal{A}$ and any PPT $\mathcal{G}_{\mathsf{rpRO}}$-constrained environment $\mathcal{Z}$. Note that, the simulator $\mathcal{S}$ simply forwards the communication between $\mathcal{A}$ and $\mathcal{Z}$. We first argue the static security, then discuss the adaptive corruptions of the parties.

The sender is honest while the receiver is statically corrupted. Here the simulator $\mathcal{S}$ needs to extract the choice bit of the receiver from the message sent in the second round. We describe the strategy of $\mathcal{S}$ as follows:

- Round 1:
  - Select $\mathsf{seed}_1 \leftarrow \{0,1\}^\lambda$; $\alpha \leftarrow \mathbb{Z}_q$.
  - Sample a generator g of group $\mathbb{G}$ and compute $h := g^\alpha$.
  - Send $(\textsc{Program}, \mathsf{sid}, `S' || \mathsf{seed}_1, (g, h))$ to $\mathcal{G}_{\mathsf{rpRO1}}$.
  - Select $r, s \leftarrow \mathbb{Z}_q$ and compute $z := g^r h^s$. Send (seed_1, z) to R^*.
- Round 2:
  - Return $(\textsc{IsProgramed}, \mathsf{sid}, 0)$ when R^* invokes $\mathcal{G}_{\mathsf{rpRO1}}$ on $(\textsc{IsProgramed}, \mathsf{sid}, `S' || \mathsf{seed}_1)$.
  - Wait for R^* to send $(\mathsf{seed}_2, B_1, B_2)$. If R^* does not send the message, return $(\textsc{No}, \mathsf{sid}, S)$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, S)$.
- Local Computation:

– Invoke $\mathcal{G}_{\mathsf{rpRO1}}$ on $(\textsc{IsPrograMed}, \mathsf{sid}, `S'||\mathsf{seed}_2)$ and abort if it returns $(\textsc{IsPrograMed}, \mathsf{sid}, 1)$.
– Compute $(G, H) := \mathcal{G}_{\mathsf{rpRO1}}(\mathsf{sid}, `R'||\mathsf{seed}_2)$.
– If $B_2 = B_1^\alpha$, set $b := 0$; else if $\frac{B_2}{H} = (\frac{B_1}{G})^\alpha$, set $b := 1$; else, set $b := \perp$.
– Compute m_0, m_1 honestly. If $b = \perp$, do nothing; else, send $(\textsc{Receive}, \mathsf{sid}, S, R^*, b)$ to $\mathcal{F}_{\mathsf{EOT}}$ on behalf of the dummy corrupted receiver, and return $(\textsc{Proceed}, \mathsf{sid}, S, m_b)$ to $\mathcal{F}_{\mathsf{EOT}}$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, S)$.

We prove the indistinguishability through the following hybrid experiments:

- $\mathcal{H}_0$: This is the real world execution $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{rpRO}}, \mathcal{F}_{\mathsf{Auth}}}_{\Pi_{\mathsf{EOT\text{-}GrpRO}}, \mathcal{A}, \mathcal{Z}}$.

- $\mathcal{H}_1$: Same as $\mathcal{H}_0$, except that $\mathcal{S}$ programs $\mathcal{G}_{\mathsf{rpRO1}}$ on seed_1, so $\mathcal{S}$ knows α s.t. $h = g^\alpha$. Indistinguishability between the hybrids follows from (i) the tuple (g, h) is randomly selected in both $\mathcal{H}_0$ and $\mathcal{H}_1$, and (ii) $\mathcal{S}$ would return $(\textsc{IsPrograMed}, \mathsf{sid}, 0)$ when R^* invokes $\mathcal{G}_{\mathsf{rpRO1}}$ on $(\textsc{IsPrograMed}, \mathsf{sid}, `S'||\mathsf{seed}_1)$.

- $\mathcal{H}_2$: Same as $\mathcal{H}_1$, except that when R^* does not send the message in the second round, the simulator $\mathcal{S}$ will return $(\textsc{No}, \mathsf{sid}, S)$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, S)$. Perfect indistinguishability holds since the honest sender will always abort when the malicious receiver refuses to continue the protocol.

- $\mathcal{H}_3$: Same as $\mathcal{H}_2$, except that $\mathcal{S}$ extracts $b \notin \{0, 1\}$ and just does nothing.

  Lemma 8. *Assume the DL assumption holds in $\mathbb{G}$, then $\mathcal{H}_2$ is computationally indistinguishable from $\mathcal{H}_1$.*

  Proof. We first consider the case where the choice bit of R^* is 0, but $B_2 \neq B_1^\alpha$. This would happen when $B_1 = g^{x_1}, B_2 = h^{x_2}$ for $x_1 \neq x_2$. We then argue that any PPT malicious receiver R^* cannot obtain m_0, thus the indistinguishability holds. If we assume that $h = g^\alpha$, then $B_1^r B_2^s = g^{rx_1} h^{sx_2} = g^{rx_1 + \alpha s x_2}, z = g^r h^s = g^{r + \alpha s}$. Since the DL assumption holds in $\mathbb{G}$, any PPT malicious receiver R^* cannot know r, s from z. Thus it is impossible for R^* to compute $B_1^r B_2^s$ given only z, x_1, x_2. Therefore, any PPT R^* cannot compute m_0. The case where the choice bit of R^* is 1 but $\frac{B_2}{H} \neq (\frac{B_1}{G})^\alpha$ is similar. In conclusion, $\mathcal{H}_3$ is computationally indistinguishable from $\mathcal{H}_2$. □

- $\mathcal{H}_4$: Same as $\mathcal{H}_3$, except that $\mathcal{S}$ extracts $b \in \{0, 1\}$, computes m_0, m_1 honestly, sends $(\textsc{Receive}, \mathsf{sid}, S, R^*, b)$ to $\mathcal{F}_{\mathsf{EOT}}$ on behalf of the dummy corrupted receiver, and returns $(\textsc{Proceed}, \mathsf{sid}, S, m_b)$ to $\mathcal{F}_{\mathsf{EOT}}$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, S)$. The simulator $\mathcal{S}$ aborts when R^* computes m_{1-b}.

  Lemma 9. *Assume DDH assumption holds in $\mathbb{G}$, then $\mathcal{H}_3$ is computationally indistinguishable from $\mathcal{H}_2$.*

  Proof. We first consider the case where (g, h, G, H) is a DDH tuple. In this case, R^* can compute m_0 and m_1 easily. Since $\mathcal{S}$ has checked that seed_2 is not programmed by R^*, the probability of (g, h, G, H) being a DDH tuple is negligible.

  We then consider the case where (g, h, G, H) is not a DDH tuple, and R^* computes both m_0 and m_1 by setting $b = 0$ $(b = 1)$ and querying z^x, $\frac{z^x}{G^r H^s}$ $(z^x G^r H^s)$ to $\mathcal{G}_{\mathsf{rpRO2}}$. We discuss the case where R^* sets $b = 0$ and queries z^x, $\frac{z^x}{G^r H^s}$ to $\mathcal{G}_{\mathsf{rpRO2}}$ here, and the other case is similar. We observe that if there exists such a R^*, then there is a reduction $\mathcal{B}$ which breaks the CDH assumption. The reduction $\mathcal{B}$ interacts with the CDH game challenger $\mathcal{C}$ and receives (A_1, A_2, A_3) from $\mathcal{C}$, and we assume that $A_1 := g, A_2 := g^t, A_3 := g^r$. Then $\mathcal{B}$ simulates $\mathcal{G}_{\mathsf{rpRO1}}, \mathcal{G}_{\mathsf{rpRO2}}$ and starts the protocol $\Pi_{\mathsf{EOT\text{-}GrpRO}}$ with R^* by running R^* internally as a black-box. Note that $\mathcal{B}$ has full control over $\mathcal{G}_{\mathsf{rpRO1}}, \mathcal{G}_{\mathsf{rpRO2}}$. In round 1, $\mathcal{B}$ samples s, seed_1, h randomly, programs the $\mathcal{G}_{\mathsf{rpRO1}}$ such that it can output (g, h) on input seed_1 , and sends $z := A_3 \cdot h^s$ to R^*. In round 2, when R^* queries seed_2 to $\mathcal{G}_{\mathsf{rpRO1}}$, $\mathcal{B}$ samples H randomly and returns $(G := A_2, H)$ as the output of $\mathcal{G}_{\mathsf{rpRO1}}$. Finally, $\mathcal{B}$ randomly selects two queries q_1, q_2 made to $\mathcal{G}_{\mathsf{rpRO2}}$ by R^* and sends $A_4 := \frac{q_1}{q_2 H^s}$ to $\mathcal{C}$. If there are Q queries made to $\mathcal{G}_{\mathsf{rpRO2}}$ by R^* in total, then $A_4 = \frac{q_1}{q_2 H^s} = \frac{z^x}{\frac{z^x}{G^r H^s} H^s} = G^r = g^{tr}$ happens at probability $\frac{1}{2 \cdot \binom{Q}{2}}$. Therefore, $\mathcal{B}$ wins the CDH game at probability $\frac{1}{2 \cdot \binom{Q}{2}}$ which is non-negligible. In conclusion, $\mathcal{H}_4$ is computationally indistinguishable from $\mathcal{H}_3$. □

Hybrid $\mathcal{H}_4$ is identical to the ideal world execution $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{rpRO}}}_{\mathcal{F}_{\mathsf{EOT}},\mathcal{S},\mathcal{Z}}$. In conclusion, when the sender S is honest and the receiver R^* is statically corrupted, $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{rpRO}},\mathcal{F}_{\mathsf{Auth}}}_{\Pi_{\mathsf{EOT\text{-}GrpRO}},\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}^{\mathcal{G}_{\mathsf{rpRO}}}_{\mathcal{F}_{\mathsf{EOT}},\mathcal{S},\mathcal{Z}}$ holds.

The receiver is honest while the sender is statically corrupted. Here the simulator $\mathcal{S}$ needs to compute both m_0 and m_1. We describe the strategy of $\mathcal{S}$ as follows:

- Round 1: Wait for S^* to send (seed_1, z). If S^* does not send the message, return $(\textsc{No}, \mathsf{sid}, R)$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, R)$.
- Round 2:
  - Invoke $\mathcal{G}_{\mathsf{rpRO1}}$ on $(\textsc{IsProgramed}, \mathsf{sid}, `S'||\mathsf{seed}_1)$ and abort if it returns $(\textsc{IsProgramed}, \mathsf{sid}, 1)$.
  - Compute $(g, h) := \mathcal{G}_{\mathsf{rpRO1}}(\mathsf{sid}, `S'||\mathsf{seed}_1)$.
  - Select $\mathsf{seed}_2 \leftarrow \{0,1\}^\lambda$ and $t \leftarrow \mathbb{Z}_q$, and compute $G := g^t, H \leftarrow h^t$.
  - Send $(\textsc{Program}, \mathsf{sid}, `R'||\mathsf{seed}_2, (G, H))$ to $\mathcal{G}_{\mathsf{rpRO1}}$.
  - Select $x \leftarrow \mathbb{Z}_q$, compute $B_1 := g^x$ and $H_2 := h^x$, and $(\mathsf{seed}_1, B_1, B_2)$ to S^*.
- Local Computation:
  - Return $(\textsc{IsProgramed}, \mathsf{sid}, 0)$ when S^* invokes $\mathcal{G}_{\mathsf{rpRO1}}$ on $(\textsc{IsProgramed}, \mathsf{sid}, `R'||\mathsf{seed}_2)$.
  - Compute $m_0 := \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||z^x), m_1 := \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||z^{x-t})$. Send $(\textsc{Send}, \mathsf{sid}, S^*, R)$ to $\mathcal{F}_{\mathsf{EOT}}$ on behalf of the dummy corrupted sender, and return $(\textsc{Proceed}, \mathsf{sid}, R, m_0, m_1)$ to $\mathcal{F}_{\mathsf{EOT}}$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, R)$.

We prove the indistinguishability through the following hybrid experiments:

- $\mathcal{H}_0$: This is the real world execution $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{rpRO}},\mathcal{F}_{\mathsf{Auth}}}_{\Pi_{\mathsf{EOT\text{-}GrpRO}},\mathcal{A},\mathcal{Z}}$.
- $\mathcal{H}_1$: Same as $\mathcal{H}_0$, except that when S^* does not send the message in the second round, the simulator $\mathcal{S}$ will return $(\textsc{No}, \mathsf{sid}, R)$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, R)$. Perfect indistinguishability holds since the honest receiver will always abort when the malicious sender refuses to continue the protocol.
- $\mathcal{H}_2$: Same as $\mathcal{H}_1$, except that $\mathcal{S}$ programs $\mathcal{G}_{\mathsf{rpRO1}}$ on seed_2, so $\mathcal{S}$ knows t s.t. $G = g^t, H = h^t$; furthermore, the simulator $\mathcal{S}$ would return $(\textsc{IsProgramed}, \mathsf{sid}, 0)$ when S^* invokes $\mathcal{G}_{\mathsf{rpRO1}}$ on $(\textsc{IsProgramed}, \mathsf{sid}, `R'||\mathsf{seed}_2)$.

  Lemma 10. *Assume the DDH assumption holds in $\mathbb{G}$, then $\mathcal{H}_1$ is computationally indistinguishable from $\mathcal{H}_0$.*

  Proof. The simulator $\mathcal{S}$ would return $(\textsc{IsProgramed}, \mathsf{sid}, 0)$ when the malicious sender S^* invokes $\mathcal{G}_{\mathsf{rpRO1}}$ on $(\textsc{IsProgramed}, \mathsf{sid}, `R'||\mathsf{seed}_2)$. Thus, the distinction is revealed if and only if S^* can distinguish the DDH tuple (g, h, g^t, h^t) from the non-DDH tuple. In other words, we observe that if there exists such a S^*, then there is a reduction $\mathcal{B}$ which breaks the DDH assumption. The reduction $\mathcal{B}$ interacts with the DDH game challenger $\mathcal{C}$ and receives (A_1, A_2, A_3, A_4) from $\mathcal{C}$. Then $\mathcal{B}$ simulates $\mathcal{G}_{\mathsf{rpRO1}}, \mathcal{G}_{\mathsf{rpRO2}}$ and starts the protocol $\Pi_{\mathsf{EOT\text{-}GrpRO}}$ with R^* by running S^* internally as a black-box. Note that $\mathcal{B}$ has full control over $\mathcal{G}_{\mathsf{rpRO1}}, \mathcal{G}_{\mathsf{rpRO2}}$. In round 1, when S^* queries seed_1 to $\mathcal{G}_{\mathsf{rpRO1}}$, $\mathcal{B}$ returns $(g := A_1, h := A_2)$ as the output of $\mathcal{G}_{\mathsf{rpRO1}}$. In round 2, $\mathcal{B}$ acts as an honest receiver except that programs the output of $\mathcal{G}_{\mathsf{rpRO1}}$ as $(T_1 := A_3, T_2 := A_4)$ on input seed_2. If R^* aborts the protocol, $\mathcal{B}$ sets $b := 1$ indicating (A_1, A_2, A_3, A_4) is a DDH tuple; else, $\mathcal{B}$ sets $b := 0$. Finally, $\mathcal{B}$ sends b to $\mathcal{C}$ and wins the game. In conclusion, $\mathcal{H}_2$ is computationally indistinguishable from $\mathcal{H}_1$. $\square$

- $\mathcal{H}_3$: Same as $\mathcal{H}_2$, except that $\mathcal{S}$ always sets $B_1 := g^x, B_2 := h^x$ and computes $m_0 := \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||z^x)$, and $m_1 := \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||z^{x-t})$. Indistinguishability follows from the fact that for $b \in \{0,1\}$, we can always compute $m_b := \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||z^{\alpha-bt})$.
- $\mathcal{H}_4$: Same as $\mathcal{H}_3$, except that $\mathcal{S}$ sends $(\textsc{Send}, \mathsf{sid}, S^*, R)$ to $\mathcal{F}_{\mathsf{EOT}}$ on behalf of the dummy corrupted sender, and returns $(\textsc{Proceed}, \mathsf{sid}, R, m_0, m_1)$ to $\mathcal{F}_{\mathsf{EOT}}$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, R)$. The simulator $\mathcal{S}$ aborts when S^* extracts b from the messages sent by $\mathcal{S}$. Indistinguishability follows from the perfect hiding property of the Pedersen commitment.

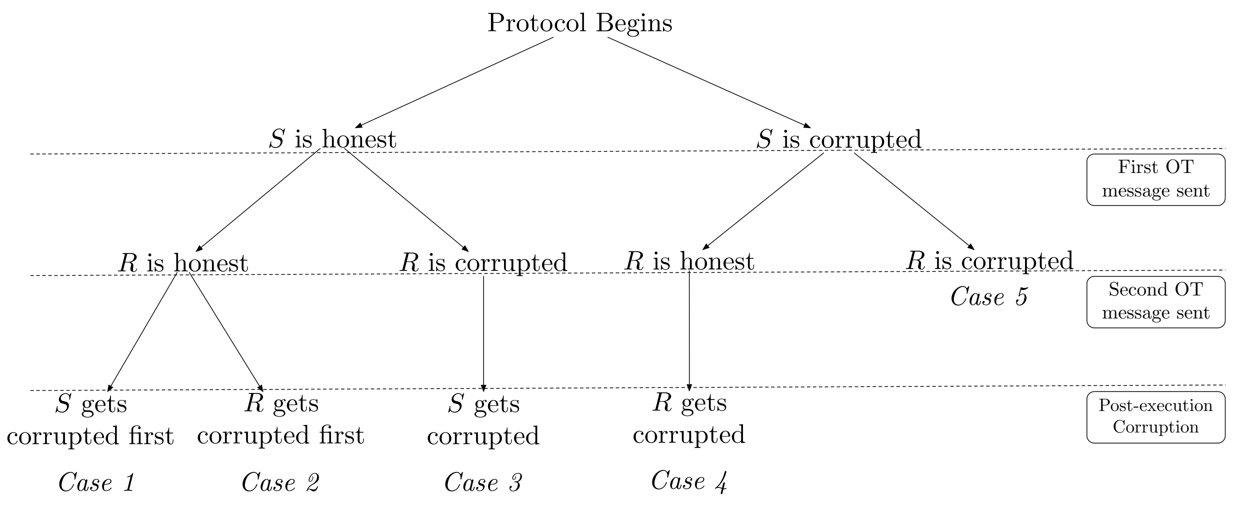


Figure 28: Simulation Cases for Adaptive Corruptions in Protocol $\Pi_{\mathsf{EOT\text{-}GrpRO}}$

Hybrid $\mathcal{H}_4$ is identical to the ideal world execution $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{rpRO}}}_{\mathcal{F}_{\mathsf{EOT}},\mathcal{S},\mathcal{Z}}$. In conclusion, when the receiver R is honest and the sender S^* is statically corrupted, $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{rpRO}},\mathcal{F}_{\mathsf{Auth}}}_{\Pi_{\mathsf{EOT\text{-}GrpRO}},\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}^{\mathcal{G}_{\mathsf{rpRO}}}_{\mathcal{F}_{\mathsf{EOT}},\mathcal{S},\mathcal{Z}}$ holds.

The proof of static security is completed, and we will discuss the cases for adaptive corruptions.

The sender/receiver gets corrupted adaptively. We present the different simulation cases in Figure 28. We present the according strategy of the simulator $\mathcal{S}$ as follows:

- Round 1:
  - *Case 1-3*: If S is honest, $\mathcal{S}$ performs the following:
    * Select $\mathsf{seed}_1 \leftarrow \{0,1\}^\lambda$; $\alpha \leftarrow \mathbb{Z}_q$.
    * Sample a generator g of group $\mathbb{G}$ and compute $h := g^\alpha$.
    * Send $(\textsc{Program}, \mathsf{sid}, `S' || \mathsf{seed}_1, (g,h))$ to $\mathcal{G}_{\mathsf{rpRO1}}$.
    * Select $r, s \leftarrow \mathbb{Z}_q$ and compute $z := g^r h^s$. Send (seed_1, z) to R.
  - *Case 4-5*: Else, $\mathcal{S}$ simply waits for S^* to send (seed_1, z). If S^* does not send the message, return $(\textsc{No}, \mathsf{sid}, R)$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, R)$.
- Round 2:
  - *Case 1,2*: If S and R are honest, $\mathcal{S}$ performs the following:
    * Select $\mathsf{seed}_2 \leftarrow \{0,1\}^\lambda$ and $t \leftarrow \mathbb{Z}_q$, and compute $G := g^t, H \leftarrow h^t$.
    * Send $(\textsc{Program}, \mathsf{sid}, `R' || \mathsf{seed}_2, (G,H))$ to $\mathcal{G}_{\mathsf{rpRO1}}$.
    * Select $x \leftarrow \mathbb{Z}_q$, compute $B_1 := g^x$ and $H_2 := h^x$, and $(\mathsf{seed}_1, B_1, B_2)$ to S.
  - *Case 3*: If S is honest and R^* is corrupted, $\mathcal{S}$ performs the following:
    * Return $(\textsc{IsProgramed}, \mathsf{sid}, 0)$ when R^* invokes $\mathcal{G}_{\mathsf{rpRO1}}$ on $(\textsc{IsProgramed}, \mathsf{sid}, `S' || \mathsf{seed}_1)$.
    * Wait for R^* to send $(\mathsf{seed}_2, B_1, B_2)$. If R^* does not send the message, return $(\textsc{No}, \mathsf{sid}, S)$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, S)$.
  - *Case 4*: If R is honest and S^* is corrupted, $\mathcal{S}$ performs the following:
    * Invoke $\mathcal{G}_{\mathsf{rpRO1}}$ on $(\textsc{IsProgramed}, \mathsf{sid}, `S' || \mathsf{seed}_1)$ and abort if it returns $(\textsc{IsProgramed}, \mathsf{sid}, 1)$.
    * Compute $(g,h) := \mathcal{G}_{\mathsf{rpRO1}}(\mathsf{sid}, `S' || \mathsf{seed}_1)$.
    * Select $\mathsf{seed}_2 \leftarrow \{0,1\}^\lambda$ and $t \leftarrow \mathbb{Z}_q$, and compute $G := g^t, H \leftarrow h^t$.
    * Send $(\textsc{Program}, \mathsf{sid}, `R' || \mathsf{seed}_2, (G,H))$ to $\mathcal{G}_{\mathsf{rpRO1}}$.
    * Select $x \leftarrow \mathbb{Z}_q$, compute $B_1 := g^x$ and $H_2 := h^x$, and $(\mathsf{seed}_1, B_1, B_2)$ to S^*.
  - *Case 5*: If S^* and R^* are corrupted, $\mathcal{S}$ ends the simulation.
- Local Computation:

- *Case 1,2*: If S and R are honest, $\mathcal{S}$ performs nothing.
- *Case 3*: If R^* is corrupted and S is honest, $\mathcal{S}$ performs the following:
  - Invoke $\mathcal{G}_{\mathsf{rpRO1}}$ on $(\textsc{IsProgramed}, \mathsf{sid}, `S'||\mathsf{seed}_2)$ and abort if it returns $(\textsc{IsProgramed}, \mathsf{sid}, 1)$.
  - Compute $(G, H) := \mathcal{G}_{\mathsf{rpRO1}}(\mathsf{sid}, `R'||\mathsf{seed}_2)$.
  - If $B_2 = B_1^{\alpha}$, set $b := 0$; else if $\frac{B_2}{H} = (\frac{B_1}{G})^{\alpha}$, set $b := 1$; else, set $b := \perp$.
  - Compute m_0, m_1 honestly. If $b = \perp$, do nothing; else, send $(\textsc{Receive}, \mathsf{sid}, S, R^*, b)$ to $\mathcal{F}_{\mathsf{EOT}}$ on behalf of the dummy corrupted receiver, and return $(\textsc{Proceed}, \mathsf{sid}, S, m_b)$ to $\mathcal{F}_{\mathsf{EOT}}$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, S)$.
- *Case 4*: If S^* is corrupted and R is honest, $\mathcal{S}$ performs the following:
  - Return $(\textsc{IsProgramed}, \mathsf{sid}, 0)$ when S^* invokes $\mathcal{G}_{\mathsf{rpRO1}}$ on $(\textsc{IsProgramed}, \mathsf{sid}, `R'||\mathsf{seed}_2)$.
  - Compute $m_0 := \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||z^x), m_1 := \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||z^{x-t})$, send $(\textsc{Send}, \mathsf{sid}, S^*, R)$ to $\mathcal{F}_{\mathsf{EOT}}$ on behalf of the dummy corrupted sender, and return $(\textsc{Proceed}, \mathsf{sid}, R, m_0, m_1)$ to $\mathcal{F}_{\mathsf{EOT}}$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Receive}, \mathsf{sid}, R)$.

- Post-Execution Corruption:
  - *Case 1,2*: If S and R are honest, $\mathcal{S}$ performs the following based on the sequence of corruption:
    - *Case 1*: If R is honest and S gets corrupted first, $\mathcal{S}$ obtains (m_0, m_1) from $\mathcal{F}_{\mathsf{EOT}}$ and programs $\mathcal{G}_{\mathsf{rpRO2}}$ such that $m_0 = \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||B_1^r B_2^s)$, $m_1 = \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||(\frac{B_1}{G})^r(\frac{B_2}{H})^s)$. The simulator $\mathcal{S}$ then reveals (r, s) as the honest sender's internal states. When R gets corrupted, $\mathcal{S}$ obtains b and provides $x' := x - b \cdot t$ as the honest receiver's internal state.
    - *Case 2*: If S is honest and R gets corrupted first, $\mathcal{S}$ obtains (b, m_b) from $\mathcal{F}_{\mathsf{EOT}}$ and programs $\mathcal{G}_{\mathsf{rpRO2}}$ such that $m_b = \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||B_1^r B_2^s(G^r H^s)^{-b})$. The simulator $\mathcal{S}$ then reveals $x' := x - b \cdot t$ as the honest receiver's internal state. When the sender S gets corrupted, $\mathcal{S}$ obtains (m_0, m_1) from $\mathcal{F}_{\mathsf{EOT}}$, programs $\mathcal{G}_{\mathsf{rpRO2}}$ such that $m_{1-b} = \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||B_1^r B_2^s(G^r H^s)^{b-1})$ and provides r, s as the honest sender's internal state.
  - *Case 3*: If R^* is corrupted and S gets corrupted, $\mathcal{S}$ obtains (m_0, m_1) from $\mathcal{F}_{\mathsf{EOT}}$, programs $\mathcal{G}_{\mathsf{rpRO2}}$ such that $m_{1-b} = \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||B_1^r B_2^s(G_1^r H_2^s)^{b-1})$ and provides r, s as the honest sender's internal state.
  - *Case 4*: If S^* is corrupted and R gets corrupted, $\mathcal{S}$ obtains b and provides $x' := x - b \cdot t$ as the honest receiver's internal state.

We prove the indistinguishability through the following hybrid experiments:

- $\mathcal{H}_0$: This is the real world execution $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{rpRO}}, \mathcal{F}_{\mathsf{Auth}}}_{\Pi_{\mathsf{EOT\text{-}GrpRO}}, \mathcal{A}, \mathcal{Z}}$.

- $\mathcal{H}_1$: Same as $\mathcal{H}_0$, except if S is honest before the first OT message is sent, the simulator $\mathcal{S}$ programs $\mathcal{G}_{\mathsf{rpRO1}}$ on seed_1, so $\mathcal{S}$ knows α s.t. $h = g^{\alpha}$. Indistinguishability has been proven in the static security proof. This happens in *Case 1-3* of Figure 28.

- $\mathcal{H}_2$: Same as $\mathcal{H}_1$, except if S and R are honest before the second OT message is sent, the simulator $\mathcal{S}$ programs $\mathcal{G}_{\mathsf{rpRO1}}$ on seed_2, so $\mathcal{S}$ knows t s.t. $G = g^t, H = h^t$. Indistinguishability follows from the DDH assumption as explained in Lemma 10. This happens in *Case 1,2* of Figure 28.

- $\mathcal{H}_3$: Same as $\mathcal{H}_2$, except if S and R are honest during the protocol and S gets corrupted first in post-execution and then R gets corrupted, in this case the simulator $\mathcal{S}$ obtains (m_0, m_1) from $\mathcal{F}_{\mathsf{EOT}}$ and programs $\mathcal{G}_{\mathsf{rpRO2}}$ such that $m_0 = \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||B_1^r B_2^s)$, $m_1 = \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||(\frac{B_1}{G})^r(\frac{B_2}{H})^s)$. The simulator $\mathcal{S}$ then reveals (r, s) as the honest sender's internal states. When R gets corrupted, $\mathcal{S}$ obtains b and provides $x' := x - b \cdot t$ as the honest receiver's internal state. This completes *Case 1* of Figure 28.

  Lemma 11. *Assume the CDH assumption holds in $\mathbb{G}$, then $\mathcal{H}_3$ is computationally indistinguishable from $\mathcal{H}_2$.*

  Proof. Note that, the simulator $\mathcal{S}$ runs the honest sender's algorithm to produce the sender's message. Therefore, the simulator $\mathcal{S}$'s revealed internal states r, s have the same distribution with the honest sender's internal states. On the other hand, due to the trapdoor property of Pedersen commitment, no PPT adversary can distinguish from the revealed internal state x' from the honest receiver's internal state.

Now the only thing left is to argue that the simulator $\mathcal{S}$ can program $\mathcal{G}_{\mathsf{rpRO2}}$ successfully, i.e., the external adversary $\mathcal{A}$ cannot query $B_1^r B_2^s$ or $(\frac{B_1}{G})^r(\frac{B_2}{H})^s$ to $\mathcal{G}_{\mathsf{rpRO2}}$ without corrupting any party and without knowing (r, s) or x. Here we only discuss the case where the PPT adversary $\mathcal{A}$ cannot query $B_1^r B_2^s$ to $\mathcal{G}_{\mathsf{rpRO2}}$, the case concerning $(\frac{B_1}{G})^r(\frac{B_2}{H})^s$ is similar. Formally, we observe that if such adversary $\mathcal{A}$ exists, then we can build a reduction $\mathcal{B}$ which breaks the CDH assumption. The reduction $\mathcal{B}$ interacts with the CDH game challenger $\mathcal{C}$ and receives (A_1, A_2, A_3) from $\mathcal{C}$, and we assume that $A_1 := g, A_2 := g^r, A_3 := g^x$. Then $\mathcal{B}$ simulates $\mathcal{G}_{\mathsf{rpRO1}}, \mathcal{G}_{\mathsf{rpRO2}}$ and starts the protocol $\Pi_{\mathsf{EOT\text{-}GrpRO}}$ with $\mathcal{A}$ by running $\mathcal{A}$ internally as a black-box. Note that $\mathcal{B}$ has full control over $\mathcal{G}_{\mathsf{rpRO1}}, \mathcal{G}_{\mathsf{rpRO2}}$. In round 1, $\mathcal{B}$ samples $s, \mathsf{seed}_1, \alpha$ randomly, computes $h := g^\alpha$ and programs the $\mathcal{G}_{\mathsf{rpRO1}}$ such that it can output (g, h) on input seed_1. Finally it sends $z := A_2 \cdot h^s$. In round 2, $\mathcal{B}$ generates seed_2, G, H honestly and sends $B_1 := A_3, B_2 := A_3^\alpha$. Finally, $\mathcal{B}$ randomly selects a queries q made to $\mathcal{G}_{\mathsf{rpRO2}}$ by $\mathcal{A}$ and sends $A_4 := \frac{q}{A_3^{s\alpha}}$ to $\mathcal{C}$. If there are Q queries made to $\mathcal{G}_{\mathsf{rpRO2}}$ by $\mathcal{A}$ in total, then $A_4 = \frac{q}{A_3^{s\alpha}} = \frac{B_1^r B_2^s}{B_2^s} = B_1^r = g^{rx}$ happens at probability $\frac{1}{Q}$. Therefore, $\mathcal{B}$ wins the CDH game at probability $\frac{1}{Q}$ which is non-negligible. In conclusion, if the CDH assumption holds, then the simulator $\mathcal{S}$ can program $\mathcal{G}_{\mathsf{rpRO2}}$ successfully. Therefore, we prove that $\mathcal{H}_3$ is computationally indistinguishable from $\mathcal{H}_2$. □

- $\mathcal{H}_4$: Same as $\mathcal{H}_3$, except if S and R are honest during the protocol and R gets corrupted first in post-execution and then S gets corrupted, in this case the simulator $\mathcal{S}$ obtains (b, m_b) from $\mathcal{F}_{\mathsf{EOT}}$ and programs $\mathcal{G}_{\mathsf{rpRO2}}$ such that $m_b = \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, \text{'}S\text{'}||B_1^r B_2^s(G^r H^s)^{-b})$. The simulator $\mathcal{S}$ then reveals $x' := x - b \cdot t$ as the honest receiver's internal state. When the sender S gets corrupted, $\mathcal{S}$ obtains (m_0, m_1) from $\mathcal{F}_{\mathsf{EOT}}$, programs $\mathcal{G}_{\mathsf{rpRO2}}$ such that $m_{1-b} = \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, \text{'}S\text{'}||B_1^r B_2^s(G^r H^s)^{b-1})$ and provides r, s as the honest sender's internal state. This completes *Case* 2 of Figure 28.

Lemma 12. *Assume the CDH assumption holds in $\mathbb{G}$, then $\mathcal{H}_4$ is computationally indistinguishable from $\mathcal{H}_3$.*

Proof. Here we only argue that the simulator can program the $\mathcal{G}_{\mathsf{rpRO2}}$ such that
$m_{1-b} = \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, \text{'}S\text{'}||B_1^r B_2^s(G^r H^s)^{b-1})$, i.e., the adversary $\mathcal{A}$ cannot query $B_1^r B_2^s(G^r H^s)^{b-1}$ to $\mathcal{G}_{\mathsf{rpRO2}}$ even after corrupting the receiver R. The rest is analogously to the Lemma 11.

We observe that if such adversary $\mathcal{A}$ exists, then we can build a reduction $\mathcal{B}$ which breaks the CDH assumption. The reduction $\mathcal{B}$ interacts with the CDH game challenger $\mathcal{C}$ and receives (A_1, A_2, A_3) from $\mathcal{C}$, and we assume that $A_1 := g, A_2 := g^r, A_3 := g^t$. Then $\mathcal{B}$ simulates $\mathcal{G}_{\mathsf{rpRO1}}, \mathcal{G}_{\mathsf{rpRO2}}$ and starts the protocol $\Pi_{\mathsf{EOT\text{-}GrpRO}}$ with $\mathcal{A}$ by running $\mathcal{A}$ internally as a black-box. Note that $\mathcal{B}$ has full control over $\mathcal{G}_{\mathsf{rpRO1}}, \mathcal{G}_{\mathsf{rpRO2}}$. In round 1, $\mathcal{B}$ samples $s, \mathsf{seed}_1, \alpha$ randomly, computes $h := g^\alpha$ and programs the $\mathcal{G}_{\mathsf{rpRO1}}$ such that it can output (g, h) on input seed_1. Finally it sends $z := A_2 \cdot h^s$. In round 2, $\mathcal{B}$ samples seed_2, x randomly and programs the $\mathcal{G}_{\mathsf{rpRO1}}$ such that it can output $(G := A_3, H := A_3^\alpha)$ on input seed_2. It then sends $B_1 := g^x h^b, B_2 := G^x H^b$. When R gets corrupted, $\mathcal{B}$ obtains b and reveals $x' := x - b \cdot \alpha$ as the internal states. Finally, $\mathcal{B}$ randomly selects two queries q_1, q_2 made to $\mathcal{G}_{\mathsf{rpRO2}}$ by $\mathcal{A}$ and sends $A_4 := \frac{q_1}{q_2 \cdot H^s}$ to $\mathcal{C}$. If there are Q queries made to $\mathcal{G}_{\mathsf{rpRO2}}$ by $\mathcal{A}$ in total, then $A_4 = \frac{B_1^r B_2^s}{B_1^r B_2^s (G^r H^s)^{-1} \cdot H^s} = \frac{G^r H^s}{H^s} = G^r = g^{rt}$ happens at probability $\frac{1}{2 \cdot \binom{Q}{2}}$. Therefore, $\mathcal{B}$ wins the CDH game at probability $\frac{1}{2 \cdot \binom{Q}{2}}$ which is non-negligible. In conclusion, if the CDH assumption holds, then the adversary $\mathcal{A}$ cannot query $B_1^r B_2^s(G^r H^s)^{b-1}$ to $\mathcal{G}_{\mathsf{rpRO2}}$ even after corrupting the receiver R. Therefore, we prove that $\mathcal{H}_4$ is computationally indistinguishable from $\mathcal{H}_3$. □

- $\mathcal{H}_5$: Same as $\mathcal{H}_4$, except if S is honest and R^* is corrupted before the second OT message is sent, if R^* does not send its message, the simulator $\mathcal{S}$ returns $(\textsc{No}, \mathsf{sid}, S)$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, S)$; else, the simulator $\mathcal{S}$ will extract the choice bit b of R^* and finish the rest of the simulation. Indistinguishability has been proven in the static security proof. This happens in *Case* 3 of Figure 28.

- $\mathcal{H}_6$: Same as $\mathcal{H}_5$, except if S is honest and R^* is corrupted before the second OT message is sent and finally S gets corrupted post-execution, the simulator $\mathcal{S}$ obtains (m_0, m_1) from $\mathcal{F}_{\mathsf{EOT}}$, programs $\mathcal{G}_{\mathsf{rpRO2}}$ such that $m_{1-b} = \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, \text{'}S\text{'}||B_1^r B_2^s(G_1^r H_2^s)^{b-1})$ and provides r, s as the honest sender's internal state. Indistinguishability follows from the DDH assumption (as explained in Lemma 9). This completes *Case* 3 of Figure 28.

- $\mathcal{H}_7$: Same as $\mathcal{H}_6$, except if S^* is statically corrupted and R is honest at the time of sending the second OT message, when S^* does not send its message, the simulator will return $(\textsc{No}, \mathsf{sid}, R)$ when $\mathcal{F}_{\mathsf{EOT}}$ sends $(\textsc{Proceed?}, \mathsf{sid}, R)$. Perfect indistinguishability holds since the honest receiver will always abort when the malicious sender refuses to continue the protocol.

- $\mathcal{H}_8$: Same as $\mathcal{H}_7$, except if S^* is statically corrupted and R is honest at the time of sending the second OT message, when S^*'s message arrives, the simulator $\mathcal{S}$ programs $\mathcal{G}_{\mathsf{rpRO1}}$ on seed_2, so $\mathcal{S}$ knows t s.t. $G = g^t, H = h^t$, and computes $B_1 := g^x, B_2 := h^x$; furthermore, the simulator $\mathcal{S}$ would return $(\textsc{IsProgramed}, \mathsf{sid}, 0)$ when S^* invokes $\mathcal{G}_{\mathsf{rpRO1}}$ on $(\textsc{IsProgramed}, \mathsf{sid}, `R'||\mathsf{seed}_2)$. Indistinguishability follows from the DDH assumption as explained in Lemma 10. This happens in *Case 4* of Figure 28.

- $\mathcal{H}_9$: Same as $\mathcal{H}_8$, except if S^* is statically corrupted and R is honest at the local computation phase, the simulator $\mathcal{S}$ computes $m_0 := \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||z^x)$, $m_1 := \mathcal{G}_{\mathsf{rpRO2}}(\mathsf{sid}, `S'||z^{x-t})$. Indistinguishability has been proven in the static security proof. This happens in *Case 4* of Figure 28.

- $\mathcal{H}_{10}$: Same as $\mathcal{H}_9$, except if S^* is statically corrupted and R get corrupted post-execution, the simulator obtains b and provides $x' := x - b \cdot t$ as the honest receiver's internal state. Indistinguishability follows from the trapdoor property of the Pedersen commitment. This completes *Case 4* of Figure 28.

- $\mathcal{H}_{11}$: Same as $\mathcal{H}_{10}$, except if S^* is statically corrupted and R^* is corrupted before the second OT message is sent, the simulator $\mathcal{S}$ simply halts. Indistinguishability follows from the fact that the distribution is identical in both hybrids since both two parties are controlled by the adversary before sending anything. This completes *Case 5* of Figure 28.

Hybrid $\mathcal{H}_{11}$ is identical to the ideal world execution $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{rpRO}}}_{\mathcal{F}_{\mathsf{EOT}},\mathcal{S},\mathcal{Z}}$. In conclusion, when the sender or the receiver gets corrupted adaptively, $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{rpRO}},\mathcal{F}_{\mathsf{Auth}}}_{\Pi_{\mathsf{EOT\text{-}GrpRO}},\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}^{\mathcal{G}_{\mathsf{rpRO}}}_{\mathcal{F}_{\mathsf{EOT}},\mathcal{S},\mathcal{Z}}$ holds.

The proof of adaptive security is completed. $\square$