Too Simple to be UC-Secure: On the UC-Insecurity of the "Simplest Protocol for Oblivious Transfer" of Chou and Orlandi

Ziya Alper Genç, Vincenzo Iovino, and Alfredo Rial

University of Luxembourg
ziya.genc@uni.lu, vinciovino@gmail.com, alfredo.rial@uni.lu

Abstract. In 2015, Chou and Orlandi presented an oblivious transfer protocol that already drew a lot of attention both from theorists and practitioners due to its extreme simplicity and high efficiency. Chou and Orlandi claimed that their protocol is UC-secure under dynamic corruptions, which is a very strong security guarantee. Unfortunately, in this work we point out a serious *flaw* in their security proof. Moreover, we show that their protocol *cannot* be proven UC-secure even under static corruptions unless some computational assumption, which we conjecture to hold, is false.
Keywords: oblivious transfer, universal composability.

1 Introduction

Oblivious Transfer. In an oblivious transfer (OT) protocol, a sender receives as input messages $M_1, \ldots M_N$ and a receiver receives as input indices $\sigma_1, \ldots, \sigma_k \in [1, N]$. At the end of the protocol, the receiver outputs $M_{\sigma_1}, \ldots, M_{\sigma_k}$ and learns nothing about the other messages. The sender does not learn anything about the indices.

OT was introduced by Rabin [**?**] and generalized by Even, Goldreich and Lempel [**?**] and Brassard, Crépeau and Robert [**?**]. (The notion of OT was also developed independently by Wiesner in the 1970's but published only later [**?**].) OT has a lot of applications and it is at the core of multi-party computation [**?**,**?**,**?**].

Chou and Orlandi's OT Protocol. Chou and Orlandi (CO) [**?**] present a novel OT protocol and claim that it is universally composable (UC) [**?**] under dynamic corruptions. Their protocol has the advantages of being extremely simple and efficient in comparison to known protocols that offer the same security level. They also report an implementation of the protocol that is orders of magnitude more efficient than previous ones. The work of CO has already gained some popularity both from theorists and practitioners and has so far been cited 21 times according to Google Scholar.

CO present a 1-out-of-2 OT protocol and extend it to a 1-out-of-n OT protocol in a straightforward manner. For the purpose of this work, which focuses

on negative results about the security of the CO protocol, it suffices to analyze the 1-out-of-2 OT protocol. We note that our negative results also apply to the 1-out-of-n OT protocol.

The 1-out-of-2 OT protocol by CO is depicted in Figure 1. To run the protocol, Alice (the sender) and Bob (the receiver) have first to agree on a group $\mathbb{G}$ and a generator g of prime order p. In the first message, Alice samples a random element a in $\mathbb{Z}_p$ and sends $A = g^a$ to Bob. Bob picks random b in $\mathbb{Z}_p$ and, depending on his index $c \in [0, 1]$, sends either $B = g^b$ or $B = Ag^b$ to Alice. Then, Alice derives two keys k_0 and k_1 from $(B)^a$ and $(B/A)^a$ respectively. Alice encrypts the messages M_0 and M_1 by using the keys k_0 and k_1 respectively. Bob can derive the key k_R from A^b, which allows Bob to obtain M_c. However, it is computationally hard for him to compute the key that allows the obtention of M_{1-c}.

The protocol uses as building block a symmetric-key encryption scheme given by two algorithms Enc and Dec. CO claim that, if the encryption scheme satisfies a property that they call *robustness*, then the resulting OT protocol is UC-secure under dynamic corruptions in the random oracle (RO) model [**?**]. They present a definition of robustness and claim a construction for a robust symmetric-key encryption scheme (see Section 3).

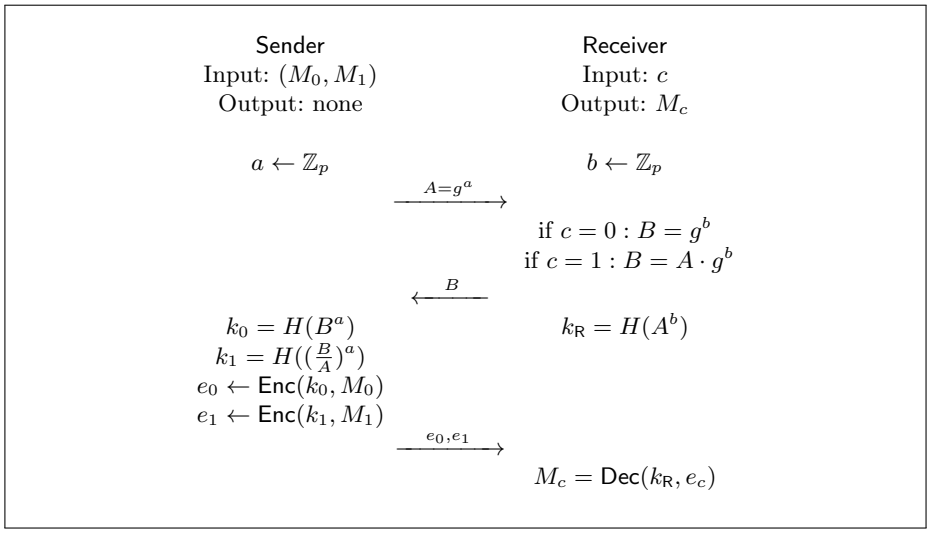

Fig. 1. Chou and Orlandi's 1-out-of-2 OT Protocol.

1.1 Our Results

In this paper, we present several issues in the work of CO, which range from minor mistakes to a *proof* that, if some computational assumption holds, then the CO's OT protocol *cannot* be proven UC-secure, even under static corruptions. We summarize the issues we found in the work of CO, from the least to the most serious, as follows.

Shortcomings in the definition of OT functionality. The ideal functionality for OT described by CO has several shortcomings, which we describe in Section 2.

Unclear construction for robust encryption. We cast doubts on the construction for *robust encryption* proposed by CO in Section 3.

Incorrect security proof. In Section 4, we report mistakes in the security proof given by CO for their OT protocol, both for the case of sender corruption and for the case of receiver corruption. These mistakes show that the simulators described by CO for those cases are incorrect, even when considering static corruptions.

Impossibility of proving UC security. The mistakes in the security proof of CO cannot be corrected. In Section 5, we show that the CO's OT protocol cannot be proven UC-secure (in the random oracle model). More concretely, we show that, when the sender is corrupt, there exists an adversary and an environment such that a correct simulation cannot be provided unless the simulator is able to solve a number-theoretic problem that we assume to be computationally hard.

In a nutshell, our impossibility result states the following. In the case of sender corruption, the simulator needs to be able to extract the messages M_0 and M_1 from the adversary in order to send them to the ideal functionality. We show that there exists an adversary such that, if a simulator that correctly extracts the messages from this adversary exists, then this simulator can be used to solve a computationally hard problem. Consequently, a simulator for the CO protocol cannot be provided and thus the protocol is not UC secure.

To the best of our knowledge, all the ideal functionalities for 1-out-of-2 OT in the literature require that the sender sends two messages M_0 and M_1 to the functionality. Indeed, this seems a basic requirement, because otherwise the functionality cannot send to the receiver the message M_c for her choice of index c. Therefore, an OT protocol that realizes any existing ideal functionality for OT must allow the construction of a simulator that extracts from a corrupt sender the messages M_0 and M_1 in order to send them to the functionality. In conclusion, our impossibility result applies to all OT functionalities and thus is independent of the shortcomings in the ideal functionality for OT defined by CO, which we describe in Section 2. We remark that the impossibility result holds even if the scheme is instantiated with a correct robust symmetric-key encryption scheme.

We assume that the simulator has full control of the random oracle (i.e., we are in the random oracle model) and this makes our impossibility result as strong as possible.

2 On the CO's Ideal Functionality for OT

CO defines an ideal functionality for m parallel executions of 1-out-of-N oblivious transfer. Because in this work we focus on negative results that show that the CO's OT protocol cannot be proven UC-secure, for simplicity we analyze the ideal functionality when $m = 1$ and $n = 2$. We remark that the shortcomings mentioned here apply to any values of m and n.

The ideal functionality for OT described by CO has several shortcomings, which we describe as follows.

- The ideal functionality for OT described by CO does not communicate with the simulator. This means that the ideal functionality for OT in CO cannot be realized at all. The reason is that, in the security proof, it is impossible to design a simulator for the case in which both the sender and the receiver are honest. As can be seen, in this case the simulator does not receive anything from the functionality, and thus it cannot provide a simulation.
- The description of the ideal functionality is incomplete. For example, the functionality expects to receive two l-bit messages from the sender and one bit c from the receiver. However, the behavior of the functionality is not specified if the inputs from the sender or from the receiver are not in the right domains. For example, if the messages input by the sender are not l-bit, what should the functionality do?
  It turns out that this question is very relevant for the mistakes in the security proof by CO, which we describe in Section 4, and for our impossibility result in Section 5. In Appendix B, we describe two ideal functionalities for OT and we show the behavior of the functionalities when the input is not in the right domain. When the sender is honest, the functionalities send an error message $\perp$ to the sender if the input is not in the right domain. When the sender is corrupt, a functionality accepts an error message $\perp$ from the sender or interprets any message not in the right domain as $\perp$, and sends $\perp$ to the receiver.
- The functionality defined by CO does not impose any restriction on the order in which the sender and the receiver send their inputs to the ideal functionality. I.e., the functionality allows the sender to send the messages before the receiver sends her input bit and viceversa.
  It turns out that this is a problem to prove secure the OT protocol by CO. In the OT protocol by CO, the receiver has to decide his input bit in order to compute the second message of the protocol. The sender decides what messages he inputs in order to compute the third message. In the security proof for the case of sender corruption, the simulator needs to extract the messages from the adversary. The simulator cannot perform such extraction until receiving the third message of the protocol from the adversary. However,

to receive this third message, the simulator has to send before the second message to the adversary. The problem here is that the simulator does not know whether the receiver has already input his bit to the functionality, because the functionality does not tell the sender that the receiver has sent her input. Consequently, the simulator does not know whether it can send the second message to the adversary, and so it cannot provide a correct simulation.

In the security proof by CO, this problem is overlooked. The simulator described by CO *always* sends a second message to the adversary, regardless of whether the receiver has already input a bit to the functionality, and therefore the simulator is not correct. The reason is that the environment can distinguish between the real world and the ideal world as follows. If the receiver gets $c \in \perp$ as input, in the real world, the honest receiver does not send any message, and thus the adversary does not receive any message from the honest receiver. However, in the ideal world, the simulator sends a message to the adversary. Therefore, the environment can distinguish real and ideal world.

This problem in the proof by CO was already observed by Li and Micciancio [**?**]. Li and Micciancio provide an alternative ideal functionality that informs the sender when the receiver sends her input bit. They also show a simulator that corrects this problem. They claim that their simulator is correct. However, in Section 4, we show that Li and Micciancio overlooked other problems in the simulator by CO, and thus their simulator is not correct either. We analyze more in detail the contribution by Li and Micciancio in Section 6.

In Appendix B, we describe two ideal functionalities for OT. The first functionality is for the case in which the sender inputs the messages before the receiver inputs her bit. This functionality tells the receiver that the sender has sent his input. The second functionality is for the case in which the receiver inputs her bit before the sender inputs his messages. This functionality tells the sender that the receiver has input her bit. This second functionality would be the one that the protocol by CO could realize. However, we show that the CO protocol has other problems that prevent it from being proven UC-secure.

In Figure 2, we show a functionality for 1-out-of-2 OT for static corruptions. This functionality takes into account the problems raised above regarding inputs in the wrong domain and the order of inputs from sender and receiver. We note that this functionality, like the one by CO, skips many details, such as the communication with the simulator and many other elements that are necessary in the UC framework (session identifiers, ...). We describe a fully specified ideal functionality for OT in Appendix B.

We would like to stress that our impossibility result holds for any existing 1-out-of-2 OT functionality, because all existing functionalities require the sender to send the messages to the functionality. Therefore, our impossibility result is

independent of the shortcomings described in this section, and also holds for a correct 1-out-of-2 OT functionality.

Ideal $\binom{2}{1}$-OT Functionality

The functionality waits for some input c from the receiver R and before receiving any input from R will ignore any input coming from the sender S. When the input c is received from R, if $c \notin \{0,1\}$, the functionality sends an error message $\perp$ to the receiver, otherwise it notifies the sender by transmitting a special symbol OK. The functionality will ignore further inputs from R.
After receiving $c \in \{0,1\}$ from R, if the functionality did not abort, the functionality waits for either a pair of messages (M_0, M_1) or an error message $\perp$ from the sender S. If the input of S is $\perp$ or a non-valid pair of messages (in the message space) the functionality sends $\perp$ to the receiver, otherwise it sends M_c to the receiver. The functionality ignores further inputs from S.

Fig. 2. Ideal functionality for 1-out-of-2 OT.

3 On the CO's Definition and Construction for Robust Encryption

The OT protocol by CO uses as building block an encryption scheme given by an encryption algorithm Enc and a decryption algorithm Dec. CO do not explicitly specify a key generation algorithm but they define a key space K from which the keys have to be sampled. CO require two properties for this encryption scheme. First, they require the scheme to be non-committing. Second, they require the scheme to satisfy what they call *robustness*.

If S is a set of keys from K, let $V_{S,e} \subseteq S$ be the subset of valid keys for a given ciphertext, i.e., the keys in S such that $\mathsf{Dec}(k, e) \neq \perp$. The CO's definition of robustness states that for all ciphertexts e adversarially generated by a PPT algorithm $\mathcal{A}$ on input a set S randomly chosen among the subsets of K, then $|V_{S,e}| \leq 1$, except with negligible probability.

CO do not make the probability space explicit. Our interpretation is that the probability is taken over the choices of S. Informally speaking, robustness requires that, for a given ciphertext, there is at most one key such that the result of decryption is not $\perp$.

CO propose the following scheme as an example of robust encryption scheme. The message space of the scheme is $\{0,1\}^l$ and both the key space and the ciphertext space are $\{0,1\}^{l+\lambda}$. The encryption algorithm $\mathsf{Enc}(k, m)$ parses k as (α, β) and outputs the ciphertext $e = (m \ \mathsf{XOR} \ \alpha, \beta)$. The decryption algorithm $\mathsf{Dec}(k, e)$ parses $k = (\alpha, \beta)$ and $e = (e_1, e_2)$ and, if $e_2 \neq \beta$, outputs $\perp$, else outputs $m = e_1 \ \mathsf{XOR} \ \alpha$.

CO claim that the above scheme is robust. It is easy to see that, if the keys are not picked randomly, this is false. For instance, given a ciphertext $e = (\alpha, \beta)$, the keys $k_1 = (\alpha, \beta)$ and $k_2 = (\alpha \ \mathsf{XOR} \ \gamma, \beta)$ for some $\gamma \neq 0^l$ are different but induce the decryption algorithm to return respectively 0^l and γ.

In the OT protocol by CO, the keys are computed by a random oracle, which forces keys to be random. We think that, for the scheme described by CO to be robust, a key generation algorithm needs to be specified that is able to force keys to be random. Using a random oracle can be useful for this purpose. Nevertheless, we would like to stress that the problems in the security proof by CO, which we describe in Section 4, and our impossibility result in Section 5 hold when the protocol is instantiated with *any* robust encryption scheme.

4 On the Incorrectness of CO's Security Proof for Their OT Protocol

In this section, we analyze the security proof provided by CO both for the case of sender corruption and for the case of receiver corruption. We show that the simulators described by CO for both cases are incorrect. For simplicity, we analyze the instantiation of the protocol as a 1-out-of-2 OT scheme, but we remark that the mistakes we found also hold for the case of m parallel executions of 1-out-of-n OT for other values of m and n.

Sender corruption. The simulator needs to extract the messages from the corrupt sender in order to send them to the ideal functionality. To do this, when the corrupt sender makes a random oracle query, the simulator described by CO picks a random key, stores it and replies the query with this random key. After that, when the corrupt sender sends the ciphertexts, the simulator tries to decrypt the ciphertexts (e_0, e_1) by using all the stored keys until the result of one of the decryptions is not $\perp$. If the result of decryption is $\perp$ in all cases, then the message is set to $\perp$.

The problem in this simulator is the following. The corrupt sender can submit an oracle query on input $X \neq B^a$ (resp. $Y \neq (\frac{B}{A})^a$) and compute the ciphertexts e_0 (resp. e_1) using key $k'_0 = H(X)$ (resp. $k'_1 = H(Y)$). Then the simulator would decrypt using (k'_0, k'_1) and obtain messages different from $\perp$. However, the honest receiver in the real world would obtain $\perp$ because the oracle query made by the receiver is for the correct value $Z = A^b$, and so the key k_{R} that the honest receiver obtains is different from both k'_0 and k'_1.

CO argue that their simulator is correct thanks to the robustness of the encryption scheme. They claim that, because there is only one key that, for any ciphertext, decrypts the cipherteext to a message different from $\perp$, then the message decrypted by the simulator and the one obtained by the honest receiver have to be equal. However, this is untrue. The problem is that the corrupt sender can compute a ciphertext with a key different from the correct key used by the honest receiver. I.e., the corrupt sender can send a random oracle query for an incorrect value and then compute a ciphertext by using the key obtained for this

query. In this case, the honest receiver obtains $\perp$, but the simulator decrypts the ciphertext to a message different from $\perp$ by using the key that was sent to the corrupt sender to answer the random oracle query for an incorrect value.

To fix the simulator, we would need a mechanism that allows the simulator to check whether a random oracle query from the corrupt sender is for a correct value, i.e., $X = B^a$ or $Y = (\frac{B}{A})^a$, or not. In Section 5, we show that the simulator cannot perform this check for both X and Y unless the simulator can solve a hard number-theoretic problem. More formally, we prove that, if a correct simulator exists for a certain environment and adversary, then the simulator can be used to solve this number-theoretic problem. Consequently, the security proof in CO cannot be patched in any way.

Receiver corruption. We point out that the security proof by CO is also incorrect for the case of a corrupt receiver. We show two problems in the simulator.

When the receiver is corrupt, the simulator must extract the bit c in order to send it to the ideal functionality. The simulator described by CO uses the random oracle queries sent by the adversary in order to extract the bit c. The problem is that, in the protocol, the receiver only needs to send random oracle queries in order to decrypt the ciphertexts, which is the last step of the protocol. The simulator described by CO assumes that the random oracle queries are sent by the corrupt receiver before the simulator sends the ciphertexts to the corrupt receiver. This is incorrect.

We show that there exists an adversary that allows the environment to distinguish between real and ideal world. This adversary simply behaves like the honest receiver, except that it does not send the random oracle query for A^b until it receives the ciphertexts (e_0, e_1). In the real world, the adversary receives the first message A from the honest sender, sends the second message B to the honest sender, receives (e_0, e_1) from the honest sender, runs a random oracle query for A^b to get the key k_{R}, uses k_{R} to decrypt the ciphertext e_c corresponding to her choice c and finally outputs the message M_c. In the ideal world, the adversary receives the first message A from the simulator, sends the second message B to the simulator, and after that waits indefinitely for the ciphertexts (e_0, e_1), which the simulator cannot send because, to compute (e_0, e_1), it needs to send c to the functionality in order to receive the message M_c. Therefore, the environment can distinguish between the real world and the ideal world.

Although we do not show an impossibility result for the construction of this simulator (because showing one impossibility result is enough), we note that fixing this problem does not seem possible. Intuitively, for the simulator to be correct, the simulator must be able to extract the bit c from the second message of the protocol, i.e., B. Because B is a random value, performing this extraction does not seem possible.

A second problem in the simulator by CO is the following. CO claim that the probability that the simulator fails is negligible because, if an adversary makes the simulator fail, that adversary can be used to break the CDH assumption. The reduction to the CDH assumption is given in Lemma 2 in [**?**]. This reduction needs to run the adversary three times to send different values in the CDH

instance as the first message of the OT protocol in order to solve the CDH problem for that instance. This means that the adversary needs to be rewound, which is not allowed in the UC framework.

5 Impossibility of the UC-security of the CO's OT protocol

Before showing our impossibility result, we introduce the computational assumption it is based on. The assumption is an interactive assumption for discrete-log groups.

Assumption 1. Let $\mathcal{G}$ be a group generator that on input a security parameter λ outputs a group description $(\mathbb{G}, p, g)$. To state the assumption we define the following game between a challenger and an adversary $\mathcal{A}$. The challenger runs $\mathcal{G}$ on input the security parameter λ to get a group description $(\mathbb{G}, p, g)$ for a group $\mathbb{G}$ of prime order p with generator g. The challenger picks a random value a from $\mathbb{Z}_p$. The adversary $\mathcal{A}$ receives as input the group description $(\mathbb{G}, p, g)$ and g^a. The adversary returns to the challenger a group element B. The challenger aborts if $B = g^a$. Otherwise B draws a bit b at random and proceeds as follows:

- If $b = 0$, set $Z_0 = B^a$ and $Z_1 = B^a / g^{a^2}$.
- If $b = 1$, draw randomly another bit d and proceed as follows.
  - If $d = 0$, set Z_0 as a random element in $\mathbb{G}$ and $Z_1 = B^a / g^{a^2}$.
  - If $d = 1$, set $Z_0 = B^a$ and set Z_1 as a random element in $\mathbb{G}$.

  The challenger sends the pair (Z_0, Z_1) to the adversary. The adversary outputs its guess b'. The adversary wins the game if $b' = b$.

Following standard terminology, we define $\mathsf{Adv}_1^{\mathcal{A}}(\lambda)$ to be the advantage of Adv in winning the above game. We say that Assumption 1 holds for generator $\mathcal{G}$ if for all probabilistic polynomial-time algorithms $\mathcal{A}$, $\mathsf{Adv}_1^{\mathcal{A}}(\lambda)$ is a negligible function of λ.

Discussion on the assumption. We conjecture that this assumption holds for some generator $\mathcal{G}$ for which standard Diffie-Hellman-like assumptions hold. As can be seen, the hardness of the assumption is based on the hardness of computing g^{a^2}. Our assumption is non-standard and as such is a very strong assumption. Nonetheless, we believe that even basing our impossibility result on this sort of assumption furnishes evidence of the implausibility of the UC-security of the CO's protocol. In fact, any proof of UC-security of the CO's protocol would necessarily have to entail a refutation of our assumption. Thus, coming up with a UC-security proof of the CO's OT protocol would require substantial new techniques.

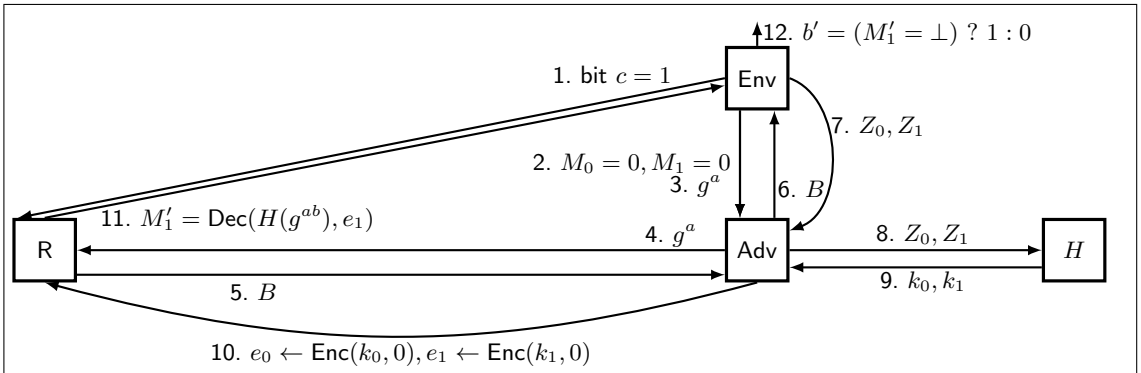

Fig. 3. Interaction in the real world between adversary $\mathcal{A}$ and environment Env and the parties explained pictorially. The messages in the interaction are exchanged in sequence from 1 to 12. Env chooses a random value a and computes values g^a for an instance of Assumption 1 (here and in the Figure we do not explicitly mention the group parameters $\mathcal{G}, g, p$). Furthermore, Env chooses two random bits b and d and after receiving value B in message 6, it computes (Z_0, Z_1) as in Assumption 1. For simplicity we assume that the sender and the adversary are one entity and that corruptions are not explicitly modeled; e.g., message 2 is supposed to go from the environment to the sender. The expression $b' = (M_1' = \perp)\ ?\ 1 : 0$ is a shorthand for "if $M_1' = \perp$ set $b' = 1$ else set $b' = 0$" and represents the output of the environment, i.e., an alleged guess for b.

UC-security. We assume that the reader is familiar with the notion of UC-security [**?**], in particular we refer to the full version [**?**]. We provide a summary in Appendix A.

Theorem 1 If Assumption 1 holds, the CO's OT protocol instantiated with an arbitrary robust encryption scheme is *not* UC-secure (in the random oracle model).

Proof. Canetti [**?**] shows that the standard definition of UC is equivalent to one in which the simulator can depend on the code of the environment, i.e., in which the quantification is of the type "for any adversary and environment there exists a simulator.." (rather than "for any adversary there exists a simulator such that for any environment...").

Therefore, to show that the CO's OT protocol cannot be proven UC-secure, it suffices to define a specific adversary $\mathcal{A}$ and environment Env such that the following holds. If there exists a simulator Sim such that Env cannot tell whether it is interacting with the CO's protocol and $\mathcal{A}$ or with the ideal OT functionality and Sim, then that simulator can be used to break Assumption 1. Here, in the random oracle model the simulator has full control of the random oracle.

We now define the adversary $\mathcal{A}$ and environment Env used in our impossibility result. We invite the reader to refer to Figure 3 for a pictorial description of the interaction between $\mathcal{A}$, Env and the parties running the CO's protocol (i.e., a "real world" execution) is detailed pictorially. We assume that the adversary

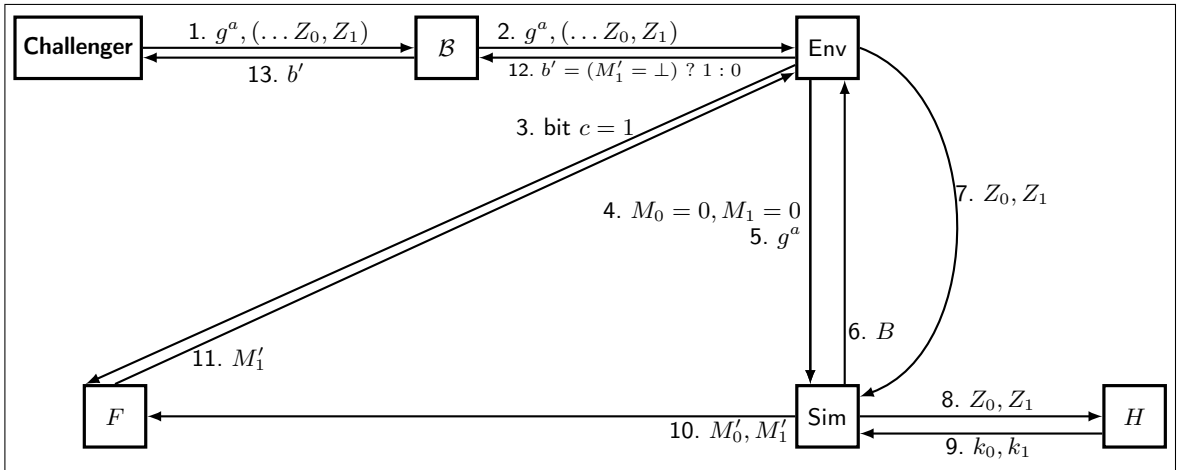

Fig. 4. Our reduction explained pictorially. Adversary $\mathcal{B}$ plays against the challenger of Assumption 1. $\mathcal{B}$ writes the values g^a on the work tape of Env and starts the execution of Env from the point in which such values are computed. The messages in the interaction are exchanged in sequence from 1 to 13. For simplicity we assume that the dummy sender and the simulator are one entity and that corruptions and delays are not explicitly modeled; e.g., message 4 is supposed to go from the environment to the dummy sender and from that to the functionality (see Theorem 1 for more details). Note also that, being the sender corrupted, it is the simulator (implicitly controlling the dummy sender) to decide when to hand the messages to the functionality. The challenger chooses a random bit b. After receiving the value B from Sim, Env forwards $\mathcal{B}$ to the challenger that returns Z_0, Z_1 to Env computed as in Assumption 1; in order not to complicate the picture, we do not make explicit these last steps in the figure and we write "$(\dots Z_0, Z_1)$" to mean that such values are not sent in step 1 and 2 (but only later) and we do not show the transmission of B from Env to $\mathcal{B}$ and from $\mathcal{B}$ to the challenger. The expression $b' = (M_1' = \perp)\ ?\ 1 : 0$ is a shorthand for "if $M_1' = \perp$ set $b' = 1$ else set $b' = 0$" and represents the output of the environment that $\mathcal{B}$ outputs as its own alleged guess for b. The random oracle H is controlled by the simulator.

corrupts the sender at the beginning of the protocol. The environment Env and the adversary Adv are next defined by describing the flow of messages exchanged by the interactive Turing machines.

Flow of messages for the environment Env.

- Env chooses values random value $a \in \mathbb{Z}_p$ (we are implicitly assuming that it also chooses a random group instance $\mathcal{G}, p, g$ but for not overburden the details henceforth and in the Figure 3 we omit such details) for an instance of Assumption 1 and computes g^a. Furthermore Env chooses two random bits b and d.
- Env hands input $c = 1$ to the (honest) receiver R. This corresponds to step 1 in Figure 3.

- Env hands the messages $M_0 = 0$ and $M_1 = 1$ to the sender. This corresponds to step 2 in Figure 3. Note that in step 2 of the figure such message is sent to the adversary that is assumed to control the sender. Formally in UC the inputs should flow from the environment to the parties and the corruptions are modeled explicitly at protocol level but in this work, for sake of simplicity, we do not explicitly deal with corruptions and these details.
- Env receives a value B from the adversary Adv. This corresponds to step 6 in Figure 3.
- Env uses the bits b and d and the value B to compute the values Z_0 and Z_1 as in Assumption 1 (it can compute such values because it knows a) and sends (Z_0, Z_1) to Adv. This corresponds to step 7 in Figure 3.
- Env receives message M_1'. If $M_1' = \perp$, Env outputs 1 otherwise outputs 0.

Flow of messages for the adversary Adv.

- Adv, impersonating the sender (recall that we do not explicitly model corruptions), receives messages $M_0 = 0$ and $M_1 = 0$ from the environment Env. This corresponds to step 2 in Figure 3.
- Adv receives value g^a from Env. This corresponds to step 3 in Figure 3.
- Adv forwards g^a to the (honest) receiver R. This corresponds to step 4 in Figure 3.
- Adv receives value B from R. This corresponds to step 5 in Figure 3.
- Adv forwards B to Env. This corresponds to step 6 in Figure 3.
- Adv receives values Z_0 and Z_1 from Env. This corresponds to step 7 in Figure 3.
- Adv makes queries Z_0 (resp. Z_1) to the random oracle H and receives answers k_0 (resp. k_1). This corresponds to steps 8 and 9 in Figure 3.
- Adv computes and sends ciphertexts $e_0 \leftarrow \mathsf{Enc}(k_0, 0)$ and $e_1 \leftarrow \mathsf{Enc}(k_1, 0)$ to R. This corresponds to step 10 in Figure 3.

Fact 2 Observe now that the value M_1' sent in message 11 from the receiver to the environment is equal, with all except negligible probability, to $\perp$ if $b = 1$ and $d = 1$ and to 0 otherwise; this is because, by the robustness of the encryption scheme, e_1 is computed with key $k_1 = H(Z_1)$ but decrypted with key $H(g^{ab})$ and the latter keys differ w.v.h.p if $b = 1$ and $d = 1$ (i.e., if Z is a random group element) and are equal otherwise (i.e., if $Z = B^a/g^{a^2}$). This implies that the bit b' output in message 12 by Env is equal, with all except negligible probability, to 1 if $b = 1$ and $d = 1$ and to 0 otherwise.

Note that Fact 2 holds for *any* robust encryption scheme.

Towards a contradiction, we assume that the CO's OT protocol is UC-secure. Hence, there exists a simulator Sim such that Env cannot tell whether it is interacting with the CO's protocol and $\mathcal{A}$ or with the ideal OT functionality and Sim.

By hypothesis, the output of the environment when interacting with Adv and when interacting with Sim are computationally indistinguishable. Let us call Real the interaction between Env, Adv and the real parties (i.e., the real experiment) and Ideal the interaction between $\mathsf{Env}, \mathsf{Sim}$ and the ideal functionality (i.e., the ideal experiment).

We construct an adversary $\mathcal{B}$ against Assumption 1 interacting with the challenger of the assumption. First of all, $\mathcal{B}$ receives an instance of Assumption 1 and executes Env from the point in which Env generates the value of the assumption g^a. That is, $\mathcal{B}$ runs Env and when Env computes such value, $\mathcal{B}$ overwrites the aforementioned values with the one coming from the challenger of the assumption. $\mathcal{B}$ forwards to the challenger of the assumption the value B sent by Env and after receiving the pair (Z_0, Z_1) forwards it to Env and finally forwards to the challenger the bit b' output by Env.

Observe that the values coming from the challenger and overwritten by $\mathcal{B}$ on the work tape of Env are identically distributed to the ones computed by the environment.

Note also the above Env gives two outputs, firstly the element B and then the bit b'. This is in contrast with the standard UC setting; in fact in the standard UC setting the environment is not allowed to communicate during the execution with parties external either from the ones of the protocol or the adversary. This is only for simplicity and later we will show how to adapt it to the standard UC setting.

Let H_1 be an experiment between Env, $\mathcal{A}$, $\mathcal{B}$, the real parties and the challenger of assumption analogous to an execution between $\mathsf{Env}, \mathcal{A}$ and the real parties but modified in the same way (i.e., replacing the values of the assumption computed by Env with the values coming from the challenger of Assumption 1). It is easy to see that the output of Real is identically distributed to the output of H_1.

Let H_2 be an experiment identical to Ideal but modified in the same way (i.e., replacing the values of the assumption computed by Env with the values coming from the challenger of Assumption 1) and where the random oracle H is controlled by the simulator (i.e., H is programmable by the simulator). Analogously, the output of Ideal is identically distributed to the output of H_2.

The interaction between $\mathcal{B}$, Env, Sim and the ideal functionality F in experiment H_2 is depicted in Figure 4. As we assume that the adversary in the real world corrupts the sender at the beginning of the protocol, we are implicitly assuming that the simulator does the same.

We remark that, in order not to overburden the presentation with too many details, in Figure 4 we let message 4 flow from Env to the simulator whereas formally in UC the message should be sent from the environment to the dummy sender and from it to the functionality. In the full details of UC, since we are

in the case of sender corruption (that, as we said, we also model implicitly), the simulator would then have the possibility of replacing the input of the sender as done in Step 10 of Figure 4.

Moreover, we do not make explicit the OK message from the functionality to the sender/simulator as described in our definition of the OT functionality, and we do not show the transmission of B from Env to $\mathcal{B}$ and from $\mathcal{B}$ to the challenger. See the caption of the Figure for more details.

As $\mathsf{Real} \equiv H_1, \mathsf{Ideal} \equiv H_2, \mathsf{Real} \approx_c \mathsf{Ideal}$ and Fact 2 holds, then the bit b' sent in message 12 of Figure 4 from Env to $\mathcal{B}$ is equal, with all except negligible probability, to 1 if $b = 1$ and $d = 1$, and to 0 otherwise. Therefore, the guess b' output by $\mathcal{B}$ is correct with all except negligible probability whenever $b = 1$ and $d = 1$. Therefore, the advantage of the reduction B to break Assumption 1 is non-negligibly greater than $1/2$. This means that Assumption 1 would not hold, which is a contradiction.

In the argument above we defined an environment Env that returns two outputs (i.e., B and b') used for the reduction. This diverges from the standard UC setting. To adapt the previous reasoning to the standard UC setting (i.e., in which the environment returns only one output and cannot communicate with parties different than the one of the protocol and the adversary), the algorithm Env can be adapted as follows. The environment is changed so as to be executed on an auxiliary input consisting either of the group parameters and the element g^a or an element B and in addition a bit z that indicates whether the environment is in the first or second "round".

If $z = 0$ the environment is in the first round, and does the computation as the algorithm Env above except that the group parameters and g^a are taken as input (instead of being computed by the algorithm itself) and outputs the value B received by Adv.[1] If $z = 1$ the environment is in the second round, and is executed on input the pair (Z_0, Z_1) computed using the bits b, d and the exponent a as in Assumption 1 and the *same* random tape as in the first round and outputs a bit b'.

Then, it is easy to see that the previous adversary $\mathcal{B}$ can be easily adapted to use such algorithm Env to break Assumption 1 as argued before.

6 Related work

Li and Micciancio [**?**] raise *different* security issues about CO's OT protocol in comparison to ours. Li and Micciancio find out a problem shared by OT protocols in which the receiver chooses his bit before the sender chooses the messags. The problem is that, in the typical OT functionality, the sender is not told whether the receiver sends a choice $c \in \{0, 1\}$ or not. To prove secure this type of OT protocol, the functionality for OT can be modified so that the functionality tells the sender whether c has been sent by the receiver.

[1] The output of the environment would be then non-binary but Canetti [**?**] shows that UC with environments with binary outputs is equivalent to UC with environments with non-binary outputs.

The following quote is from Li and Micciancio:

> "We simply use the equational framework to model and analyze the protocol as described in the original paper of CO. We had noticed that this work appeared to use the traditional OT definition, and our original goal was to show that the protocol does not satisfy it, but it can be proved secure if the OT ideal functionality is revised as in the OT transformation case. We still find it rather surprising that the protocol cannot be proved secure according to either definition."

Li and Micciancio try to prove the security of the OT protocol by CO using the modified OT functionality. They claim that, with the modified functionality, sender security can be proven, but that then receiver security does not hold anymore. In particular, they provide a simulator for the case of sender corruption to realize the modified OT functionality and say "we leave the verification that the simulator is indeed correct to the reader." As we show in this paper, this simulator is not correct either because it does not extract the messages from the corrupt sender correctly. The following quote is from Li and Micciancio:

> "We remark that what we mean by cannot be proved secure is that for any candidate simulator (within the model) there exists a distinguishing environment that can tell the real system and ideal system apart. We do not have a direct attack to the protocol. So, by no means our results should be interpreted as a cryptanalysis of [**?**], and, in fact, we believe that the protocol provides some meaningful form of security. Also, our results do not point to any specific bug in the (informal) proof in the original paper [**?**]."

In contrast, we show several bugs in the proof by CO. In addition, we show an impossibility result on the ability to provide a correct proof.

7 Conclusions and future directions

In this work, we pointed out several issues of the Chou and Orlandi's OT protocol, culminating in a formal proof that the aforementioned protocol cannot be proven UC-secure (unless some computational assumption that we conjecture to hold is false). This negative result raises the question of whether the CO's protocol can be patched or tweaked to meet UC-security or whether it can be proven secure according to less stringent security notions. While we believe that the answer to the second question might be positive, we have to stress that a positive answer to the first question can be considered satisfactory only if a potential corrected protocol retained approximately the same efficiency benefits of the CO's OT protocol.

A Universally Composable Security

The universal composability framework [**?**] is a framework for defining and analyzing the security of cryptographic protocols so that security is retained under

arbitrary composition with other protocols. The security of a protocol is defined by means of an ideal protocol that carries out the desired task. In the ideal protocol, all parties send their inputs to an ideal functionality $\mathcal{F}$ for the task. The ideal functionality locally computes the outputs of the parties and provides each party with its prescribed output.

The security of a protocol φ is analyzed by comparing the view of an environment $\mathcal{Z}$ in a real execution of φ against that of $\mathcal{Z}$ in the ideal protocol defined in $\mathcal{F}_\varphi$. The environment $\mathcal{Z}$ chooses the inputs of the parties and collects their outputs. In the real world, $\mathcal{Z}$ can communicate freely with an adversary $\mathcal{A}$ who controls both the network and any corrupt parties. In the ideal world, $\mathcal{Z}$ interacts with dummy parties, who simply relay inputs and outputs between $\mathcal{Z}$ and $\mathcal{F}_\varphi$, and a simulator $\mathcal{S}$. We say that a protocol φ securely realizes $\mathcal{F}_\varphi$ if $\mathcal{Z}$ cannot distinguish the real world from the ideal world, i.e., $\mathcal{Z}$ cannot distinguish whether it is interacting with $\mathcal{A}$ and parties running protocol φ or with $\mathcal{S}$ and dummy parties relaying to $\mathcal{F}_\varphi$.

A.1 Notation

A binary distribution ensemble $X = \{X(k,a)\}_{k\in\mathbb{N},a\in\{0,1\}^*}$ and another ensemble $Y = \{Y(k,a)\}_{k\in\mathbb{N},a\in\{0,1\}^*}$ are indistinguishable ($X \approx Y$) if for any $c,d \in \mathbb{N}$ there exists $k_0 \in \mathbb{N}$ such that for all $k > k_0$ and all $a \in \cup_{\kappa\leq k^d}\{0,1\}^\kappa$, $|\Pr[\,]X(k,a) = 1] - \Pr[\,]Y(k,a) = 1]| < k^{-c}$. Let $\mathrm{REAL}_{\varphi,\mathcal{A},\mathcal{Z}}(k,a)$ denote the distribution given by the output of $\mathcal{Z}$ when executed on input a with $\mathcal{A}$ and parties running φ, and let $\mathrm{IDEAL}_{\mathcal{F}_\varphi,\mathcal{S},\mathcal{Z}}(k,a)$ denote the output distribution of $\mathcal{Z}$ when executed on input a with $\mathcal{S}$ and dummy parties relaying to $\mathcal{F}_\varphi$. We say that the protocol φ securely realizes $\mathcal{F}_\varphi$ if, for all polynomial-time $\mathcal{A}$, there exists a polynomial-time $\mathcal{S}$ such that, for all polynomial-time $\mathcal{Z}$, $\mathrm{REAL}_{\varphi,\mathcal{A},\mathcal{Z}} \approx \mathrm{IDEAL}_{\mathcal{F}_\varphi,\mathcal{S},\mathcal{Z}}$.

A protocol $\varphi^{\mathcal{G}}$ securely realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model when φ is allowed to invoke the ideal functionality $\mathcal{G}$. Therefore, for any protocol ψ that securely realizes the functionality $\mathcal{G}$, the composed protocol φ^ψ, which is obtained by replacing each invocation of an instance of $\mathcal{G}$ with an invocation of an instance of ψ, securely realizes $\mathcal{F}$.

A.2 Conventions

When describing ideal functionalities, we use the following conventions:

Interface Naming Convention. An ideal functionality can be invoked by using one or more interfaces. The name of a message in an interface consists of three fields separated by dots. The first field indicates the name of the functionality and is the same in all interfaces of the functionality. This field is useful for distinguishing between invocations of different functionalities in a hybrid protocol that uses two or more different functionalities. The second field indicates the kind of action performed by the functionality and is the same in all messages that the functionality exchanges within the same interface. The third field distinguishes between the messages that belong to the same interface, and can take

six different values. A message $*.*.\mathsf{ini}$ is the incoming message received by the functionality, i.e., the message through which the interface is invoked. A message $*.*.\mathsf{end}$ is the outgoing message sent by the functionality, i.e., the message that ends the execution of the interface. The message $*.*.\mathsf{sim}$ is used by the functionality to send a message to the simulator, and the message $*.*.\mathsf{rep}$ is used to receive a message from the simulator. The message $*.*.\mathsf{req}$ is used by the functionality to send a message to the simulator to request the description of algorithms from the simulator, and the message $*.*.\mathsf{alg}$ is used by the simulator to send the description of those algorithms to the functionality.

Network vs. local communication. The identity of an interactive Turing machine (ITM) instance (ITI) consists of a party identifier *pid* and a session identifier *sid*. A set of parties in an execution of a system of ITMs is a protocol instance if they have the same session identifier *sid*. ITIs can pass direct inputs to and outputs from "local" ITIs that have the same *pid*. An ideal functionality $\mathcal{F}$ has $pid = \perp$ and is considered local to all parties. An instance of $\mathcal{F}$ with the session identifier *sid* only accepts inputs from and passes outputs to machines with the same session identifier *sid*. Some functionalities require the session identifier to have some structure. Those functionalities check whether the session identifier possesses the required structure in the first message that invokes the functionality. For the subsequent messages, the functionality implicitly checks that the session identifier equals the session identifier used in the first message. Communication between ITIs with different party identifiers must take place over the network. The network is controlled by the adversary, meaning that he can arbitrarily delay, modify, drop, or insert messages.

B Ideal Functionalities for Oblivious Transfer with Static Corruptions

The functionality for OT given in CO is not realizable. The reason is that the functionality does not communicate with the simulator at all. Additionally, the functionality for OT in CO is not fully specified. For example, the behavior of the functionality when the sender sends messages that do not belong to the message space is not specified.

Here we provide a functionality for OT with static corruptions that is realizable and whose behavior is fully specified. First, we describe the typical OT functionality in which the receiver does not need to wait for the receiver to send his input to the functionality. As noted by Li and Micciancio, the protocol by CO cannot realize this functionality. The problem is that, in order to simulate an honest receiver towards a corrupt sender, the simulator needs to know whether the receiver has sent his input to the functionality. Second, we show a second functionality in which the sender has to wait for the receiver's input.

First Functionality $\mathcal{F}_{\text{OT}}$

$\mathcal{F}_{\text{OT}}$ is parameterized by a message space $\mathcal{M}$.

1. On input $(\mathsf{ot.send.ini}, sid, m_0, m_1)$ from a party $\mathcal{T}$:
   - If $sid \neq (\mathcal{T}, \mathcal{R}, sid')$, or if $m_0 \notin \mathcal{M}$, or if $m_1 \notin \mathcal{M}$, or if there is a tuple $(sid, m_0, m_1, 0)$ stored, send $(\mathsf{ot.send.end}, sid, \perp)$ to $\mathcal{T}$.
   - Store $(sid, m_0, m_1, 0)$.
   - Send $(\mathsf{ot.send.sim}, sid)$ to $\mathcal{S}$.

S. On input $(\mathsf{ot.send.rep}, sid)$ from the simulator $\mathcal{S}$:
   - If $(sid, m_0, m_1, 0)$ is not stored or if $(sid, m_0, m_1, 1)$ is already stored, ignore the message.
   - Store $(sid, m_0, m_1, 1)$ and parse sid as $(\mathcal{T}, \mathcal{R}, sid')$.
   - Send $(\mathsf{ot.send.end}, sid, \text{ready})$ to $\mathcal{R}$.

2. On input $(\mathsf{ot.receive.ini}, sid, c)$ from $\mathcal{R}$:
   - If $sid \neq (\mathcal{T}, \mathcal{R}, sid')$, or if $c \notin [0, 1]$, or if $(sid, m_0, m_1, 1)$ is not stored, or if $(sid, c, 0)$ is already stored, then send $(\mathsf{ot.receive.end}, sid, \perp)$ to $\mathcal{R}$.
   - Store $(sid, c, 0)$.
   - Send $(\mathsf{ot.receive.sim}, sid)$ to $\mathcal{S}$.

S. On input $(\mathsf{ot.receive.rep}, sid, d)$ from the simulator $\mathcal{S}$:
   - If $(sid, c, 0)$ is not stored or if $(sid, c, 1)$ is already stored, ignore the message.
   - Store $(sid, c, 1)$.
   - If $\mathcal{T}$ is not corrupt, send $(\mathsf{ot.receive.end}, sid, m_c)$ to OT. Else, if $d = OK$, then send $(\mathsf{ot.receive.end}, sid, m_c)$ to OT, else send $(\mathsf{ot.receive.end}, sid, \text{fail})$ to $\mathcal{R}$.

Second Functionality $\mathcal{F}_{\text{OT}}$

$\mathcal{F}_{\text{OT}}$ is parameterized by a message space $\mathcal{M}$.

1. On input $(\mathsf{ot.receive.ini}, sid, c)$ from $\mathcal{R}$:
   - If $sid \neq (\mathcal{T}, \mathcal{R}, sid')$, or if $c \notin [0, 1]$, or if $(sid, c, 0)$ is already stored, then send $(\mathsf{ot.receive.end}, sid, \perp)$ to $\mathcal{R}$.
   - Store $(sid, c, 0)$.
   - Send $(\mathsf{ot.receive.sim}, sid)$ to $\mathcal{S}$.

S. On input $(\mathsf{ot.receive.rep}, sid)$ from the simulator $\mathcal{S}$:
   - If $(sid, c, 0)$ is not stored or if $(sid, c, 1)$ is already stored, ignore the message.

– Store $(sid, c, 1)$.
– Send $(\mathsf{ot.receive.end}, sid, \text{ready})$ to $\mathcal{T}$.

2. On input $(\mathsf{ot.send.ini}, sid, m_0, m_1)$ from a party $\mathcal{T}$:
   – If $\mathcal{T}$ is not corrupt do the following.
     • If $sid \neq (\mathcal{T}, \mathcal{R}, sid')$, or $m_0 \notin \mathcal{M}$, or $m_1 \notin \mathcal{M}$, or $(sid, c, 1)$ is not stored, or if there is a tuple $(sid, m_0, m_1, 0)$ stored, send $(\mathsf{ot.send.end}, sid, \perp)$ to $\mathcal{T}$, else continue with the next steps.
   – If $\mathcal{T}$ is corrupt do the following.
     • If $sid \neq (\mathcal{T}, \mathcal{R}, sid')$, or $(sid, c, 1)$ is not stored, or if there is a tuple $(sid, m_0, m_1, 0)$ stored, send $(\mathsf{ot.send.end}, sid, \perp)$ to $\mathcal{T}$.
     • Else, if $m_0 \notin \mathcal{M}$, set $m_0 = \perp$, and if $m_1 \notin \mathcal{M}$, set $m_1 = \perp$. Continue with the next steps.
   – Store $(sid, m_0, m_1, 0)$.
   – Send $(\mathsf{ot.send.sim}, sid)$ to $\mathcal{S}$.

S. On input $(\mathsf{ot.send.rep}, sid)$ from the simulator $\mathcal{S}$:
   – If $(sid, m_0, m_1, 0)$ is not stored or if $(sid, m_0, m_1, 1)$ is already stored, ignore the message.
   – Store $(sid, m_0, m_1, 1)$ and parse sid as $(\mathcal{T}, \mathcal{R}, sid')$.
   – Send $(\mathsf{ot.send.end}, sid, m_c)$ to $\mathcal{R}$.