A Compiler of Two-Party Protocols for Composable and Game-Theoretic Security, and Its Application to Oblivious Transfer*

Shota Goto[1], and Junji Shikata[1,2]

[1] Graduate School of Environment and Information Sciences, Yokohama National University, Japan

[2] Institute of Advanced Sciences, Yokohama National University, Japan

goto-shota-zx@ynu.jp, shikata@ynu.ac.jp

Abstract

In this paper, we consider the following question: Does composing protocols having game-theoretic security result in a secure protocol in the sense of game-theoretic security? In order to discuss the composability of game-theoretic properties, we study security of cryptographic protocols in terms of the universal composability (UC) and game theory simultaneously. The contribution of this paper is the following: (i) We propose a compiler of two-party protocols in the local universal composability (LUC) framework such that it transforms any two-party protocol secure against semi-honest adversaries into a protocol secure against malicious adversaries in the LUC framework; (ii) We consider the application of our compiler to oblivious transfer (OT) protocols, by which we obtain a construction of OT meeting both UC security and game-theoretic-security.

Keywords: game-theoretic security, universal composability, local universal composability, oblivious transfer.

1 Introduction

1.1 Background

In recent years, game-theoretic security of cryptographic protocols has been studied. Generally, cryptographic security is defined so as to guarantee some basic concrete properties when participants follow the designed algorithms, even if facing an adversarial behavior. In contrast, game-theoretic security is defined such that, by considering behaviors of rational participants in a protocol whose goal is to achieve their best satisfactions, following the specifications of the protocol honestly is the most reasonable for the rational participants. This security notion enables us to design protocols more realistically. In this way, these concepts capture situations from different perspectives and it seems that there is great difference between the cryptographic security and game-theoretic security. Up to date, there are several works aiming at bridging the gaps between the two kinds of security [9, 17, 18, 19, 13]. Recently, Asharov et al. [4] studied two-party protocols in the fail-stop model in terms of game-theoretic security and showed how the notion of game theory can capture cryptographic properties. Furthermore, the game-theoretic security for oblivious transfer [14] and bit commitment [15] has been studied in the malicious model.

In addition to cryptographic security and game-theoretic security, composable security has also been studied in order to guarantee security of protocols even if they are composed with other ones.

*This is a full version of the paper which appears in IMACC 2015.

The previous frameworks of this line of research are based on the ideal-world/real-world paradigm, and the paradigm underlies universal composability (UC) by Canetti [6] and reactive simulatability by Backes, Pfitzmann and Waidner [5]. In addition, a simple paradigm for composable security was given by Maurer [21], and this approach is called constructive cryptography. In this paper, we utilize the UC framework [6] to consider composable security of cryptographic protocols, since this approach has been utilized in discussing composability of protocols in many papers.

In this paper, we consider the following question: Does composing protocols having game-theoretic security result in a *secure* protocol in the sense of game-theoretic security? In order to discuss the composability of game-theoretic properties, we need to consider protocols having both universally composable (UC) and game-theoretic security. Although the UC framework achieves guarantee of composability of protocols, the framework models the attacker as a centralized entity so that it can capture only the situation that the attacker is like a dictator and corrupted parties are all cooperative. For this reason, some other formalisms have been proposed in [20, 3, 1, 16, 22, 2, 8]. In these formalizations, the centralized adversary is shattered to plural adversaries and each of them is limited to obtain only local information. This modeling seems to be more realistic than existing ones and can capture many settings that are not captured by centralized adversary approach. In particular, we focus on the local universal composability (LUC) framework in [8] in this paper, and we try to answer the question mentioned above.

1.2 Our Approach

In this paper, we study security of cryptographic protocols in terms of composability and game theory simultaneously. In particular, we consider realizing a compiling mechanism which transforms a protocol that is not game-theoretically secure into a protocol that achieves the composable and game-theoretic security. Although the UC framework is a powerful theory to consider composability of protocols, it cannot cover game-theoretic security since the UC framework considers a centralized adversary and cannot deal with protocols as games among plural rational participants. However, if we switch the framework to the local universal composability (LUC) framework [8], we can analyze protocols in terms of game-theoretic security by clarifying which strategy is in Nash equilibrium.

Besides the LUC framework, there is also a well-established framework with a composition theorem and an application to game theory, called collusion-preserving (CP) framework [2]. The reason for our choice of the LUC framework over the CP framework is that, the compiler of two-party protocols which we focus on in this paper was originally proposed on the basis of the UC framework [7]. On that point, choosing the LUC framework whose modeling is a direct extension of the UC framework enables us to discuss the whole aspect of the compiler simply and similarly to the case of UC.

Furthermore, we refer to a connection between the LUC framework and game-theoretic security. At first sight, one may think that these two notions are not well connected, since there is a difference in the requirement for security definitions: game-theoretic security requires that all participants can get the highest utility when each of them acts honestly, while LUC security requires the indistinguishability between the real-world and the ideal-world. However, there is an important point common to these two notions, namely, all participants are allowed to behave in a malicious (or rational) way. Considering this point, if we define an ideal functionality in the LUC framework accurately so that it captures the correct actions which each participant should essentially take, LUC security will satisfy the desirable property that following the protocol specifications honestly is the most reasonable for rational participants. However, in general, defining an ideal functionality in such a reasonable way may be a hard work, if we target complicated protocols where participants communicate intricately. If we can do it in such a way, we can say that LUC security implies game-theoretic security. As an illustration, in this paper we target oblivious transfer (OT), since its functionality is traditional and relatively simple. Specifically, we explicitly formalize the functionality of OT in the LUC framework in a way mentioned above, and our resulting OT protocol will be proven to be secure even in terms of game-theoretic security.

1.3 Our Results

(i) A compiler for both UC security and game-theoretic security. First, we propose a compiler of two-party protocols in the LUC framework such that it transforms any two-party protocol secure against semi-honest adversaries into a protocol secure against malicious adversaries in the LUC framework. Our compiler is constructed based on the compiler of [7] in the UC framework. In other words, we try to adapt the compiler of [7] to the LUC framework. For doing it, we define a commit-and-prove functionality, denoted by $\hat{\mathcal{F}}_{CP}$, which is a slight modification of the commit-and-prove functionality $\mathcal{F}_{CP}$ in the UC framework. And, we show that the compiled protocol is secure against malicious adversaries in the $\hat{\mathcal{F}}_{CP}$-hybrid model in the LUC framework (in Theorem 2 in Section 3.2).
(ii) Application of the compiler to oblivious transfer. Second, we consider the application of our compiler to oblivious transfer (OT) protocols. Since, OT is an important primitive for secure multi-party computation, it is worth exploring a practical construction. In particular, we consider the construction of the OT protocol, denoted by SOT, in [7, 11, 12] which UC-realizes the OT functionality in static and semi-honest adversarial model. For the protocol SOT, we show that:

(1) SOT LUC-realizes $\hat{\mathcal{F}}_{OT}$ in the presence of semi-honest and static adversaries, where $\hat{\mathcal{F}}_{OT}$ is the OT functionality in the LUC framework (in Theorem 3 in Section 4.2);

(2) SOT is not game-theoretically secure in the presence of rational parties (in Theorem 4 in Section 4.3); and

(3) The compiled protocol of SOT by our compiler is game-theoretically secure in the presence of rational parties (in Theorem 4 in Section 4.3).

Since the functionality of OT is relatively simple, we will be able to define it in the LUC framework so that (3) follows from (i) and (1). However, we directly prove (3) in terms of the game theory in order to confirm that the compiled protocol of SOT actually meets game-theoretic security, and the analysis of the compiled protocol from the viewpoint of the game theory enables us to see how Nash equilibrium is achieved in it.

2 Preliminaries

2.1 Framework of Universally Composable Security

In this section, we provide an overview of the universal composability framework (UC framework for short) in [6]. This framework allows us to define the security properties of given tasks, as follows. First, the process of executing a protocol with a realistic adversary is formalized. Next, an ideal process is formalized. In this process, the parties hand their inputs to a trusted party that is programmed to capture the appropriate functionality and obtain their outputs from it with no interaction. A protocol is said to securely realize an ideal functionality if the process of executing the protocol amounts to emulating the ideal process. The models for protocol execution and ideal process are as follows.

Protocol execution. The model of protocol execution includes the parties running the given protocol π and an adversary $\mathcal{A}$, and an environment $\mathcal{Z}$. The actions of all parties may depend on a security parameter $k \in \mathbf{N}$ and are polynomial in k. In each activation, only a single participant whose tape is written is activated and the activations are sequential.

First, $\mathcal{Z}$ is activated with a message $z \in \{0,1\}^*$ on its input tape. In each activation, it may write information on the input tape of one of the parties or of the adversary and may read the output tapes of all parties. In addition, it may invoke a new party that runs the protocol. When the adversary is activated, it may read its own tapes or the outgoing communication tapes of all parties. Furthermore, it may either deliver a message to some party or send information to $\mathcal{Z}$, or corrupt a party. When a party is activated, it follows its code and may write an output on its output tape or a message on its outgoing communication tape. Once the environment halts, the protocol execution ends. Let $\mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ be the random variable taking output of $\mathcal{Z}$ when interacting with an adversary $\mathcal{A}$ and parties $P_1,\ldots,P_n$ running protocol π on a security parameter k and an input z. Let $\mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ be the ensemble $\{\mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

Ideal process. The model of ideal process includes a simulator $\mathcal{S}$ and an environment $\mathcal{Z}$, an ideal functionality $\mathcal{F}$ and dummy parties. As in the protocol execution, the ideal process consists of a sequence of activations. Specifically, the activation of the environment is performed in the same way as described above.

When the simulator $\mathcal{S}$ is activated, it may read its own input tape and the messages sent from $\mathcal{F}$, and may send a message to $\mathcal{F}$. In addition, it may corrupt a party. When the ideal functionality $\mathcal{F}$ is activated, it reads its incoming communication tape and may send messages to the simulator $\mathcal{S}$. Eventually, it provides parties with an output. When a dummy party is activated with an input, it sends its input to the ideal functionality $\mathcal{F}$, and then receives an output. Finally, it outputs the received value automatically as its own output. As in the protocol execution, the ideal process ends when the environment halts. Let $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)$ be the random variable taking the output of $\mathcal{Z}$ when interacting in the ideal process with the simulator $\mathcal{S}$ and the ideal functionality $\mathcal{F}$ on a security parameter k and an input z. Let $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ be the ensemble $\{\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

Definition 1. *Two binary distribution ensembles X and Y are computationally indistinguishable (written as $X \overset{c}{\approx} Y$), if for any $c \in \mathbf{N}$ there exists $k_0 \in \mathbf{N}$ such that for all $k > k_0$ and for all a we have $|\Pr(X(k,a)=1) - \Pr(Y(k,a)=1)| < k^{-c}$.*

Definition 2. *Let $n \in \mathbf{N}$. Let $\mathcal{F}$ be an ideal functionality and let π be an n-party protocol. We say that π UC-realizes $\mathcal{F}$, if for any adversary $\mathcal{A}$ there exists an ideal-process simulator $\mathcal{S}$ such that for any environment $\mathcal{Z}$, $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$*

The hybrid model and the composition theorem. The hybrid model is identical to the model of protocol execution except the following difference: The parties running the protocol come to have an access to multiple copies of an ideal functionality $\mathcal{F}$ (the $\mathcal{F}$-hybrid model) and the communication between the parties and each one of the copies of $\mathcal{F}$ is done in the same way as in the ideal process. Let $\mathrm{EXEC}^{\mathcal{F}}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ be the random variable taking the output of $\mathcal{Z}$ when interacting in the $\mathcal{F}$-hybrid model with an adversary $\mathcal{A}$ and parties $P_1,\ldots,P_n$ running protocol π on a security parameter k and an input z. $\mathrm{EXEC}^{\mathcal{F}}_{\pi,\mathcal{A},\mathcal{Z}}$ denotes the ensemble as well.

In general, the composition theorem says that, if p is a protocol which UC-realizes an ideal functionality $\mathcal{F}$ in the $\mathcal{G}$-hybrid model for an ideal functionality $\mathcal{G}$, the composed protocol π^p running in the $\mathcal{G}$-hybrid model emulates the protocol π in the $\mathcal{F}$-hybrid model.

Theorem 1 (Composition Theorem [6])**.** *Let $\mathcal{F},\mathcal{G},\mathcal{I}$ be ideal functionalities. Let π be an n-party protocol in the $\mathcal{F}$-hybrid model, and let p be an n-party protocol that UC-realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model. Then for any adversary $\mathcal{A}$ in the $\mathcal{G}$-hybrid model there exists an simulator $\mathcal{S}$ in the $\mathcal{F}$-hybrid model such that for any environment $\mathcal{Z}$, $\mathrm{EXEC}^{\mathcal{G}}_{\pi^p,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathrm{EXEC}^{\mathcal{F}}_{\pi,\mathcal{S},\mathcal{Z}}$.*

The model of adversaries. In general, a protocol is designed by considering a model of adversaries, which depends on to what extent the designer wants to achieve security against the adversaries. We outline the model of corruptions and adversarial behaviors as follows.

1. The model of corruptions.
   (a) **Static corruption model.** The set of parties who are to be corrupted by an adversary is fixed at the beginning of the computation and no more corruptions will be happen after that.
   (b) **Adaptive corruption model.** In contrast to the static corruption model, an adversary is allowed to corrupt parties at any time throughout the computation.
2. The model of adversarial behaviors.
   (a) **Semi-honest adversarial model.** Even if parties are corrupted, they follow the specification of the protocol. Therefore, the adversary is restricted only to get read access to the states of corrupted parties.
   (b) **Malicious adversarial model.** Once the adversary corrupts parties, they follow all the instruction of the adversary. In particular, the adversary can make the corrupted parties deviate from the specification of the protocol.

2.2 Framework of Local Universally Composable Security

The notion of the local universal composability (LUC for short) was proposed by Canetti and Vald [8]. Roughly speaking, instead of setting a single adversary as in the UC framework, there can be plural local adversaries who can corrupt only a single party according to their party IDs. In the ideal process, the simulator is also shattered to plural local simulators, therefore, the simulation is done by relying only on each entity's local information. We describe the LUC model of protocol execution as follows, and aside from some modifications, the underlying computational model is identical to the UC model.

Protocol execution in the LUC framework. At first, a set $\mathcal{P}$ of party IDs and session ID, denoted by *pid* and *sid* respectively, are chosen by the environment. This is different from the UC model where the party IDs can be chosen arbitrarily during protocol execution. Next, the adversaries are invoked with identity $id = ((i,j), \perp)$ and denoted by $\mathcal{A}_{(i,j)}$ for ordered pairs $(i,j) \in \mathcal{P}^2$. The purpose of this modeling is to capture locality properly. Each local adversary comes to take charge of a different side of the communication line, and can interfere with the parties' communication only via this line. This means that the centralized adversary no longer exists and many situations, in real-life, where an adversary can only rely on restricted information are capturable.

Once an adversary $\mathcal{A}_{(i,j)}$ is activated, it can send information to $\mathcal{Z}$ or deliver a message to a party with $pid = i$ where the sender's *pid* must be j. The adversary is also allowed to corrupt parties with $pid = i$ throughout the computation. An important point is that adversaries cannot communicate each other directly and their communications must be done through the environment $\mathcal{Z}$ (or an ideal functionality if any). This formalization enables us to represent different subsets of adversaries, if there exists a trusted party (an ideal functionality in the hybrid model) and it provides a specific communication interface.

Once a party is activated, it basically follows its code and may write an output on its output tape or send a message to the adversary where the *pid* of the adversary must be sender's $pid = i$ and receiver's $pid = j$. As in the UC model, the protocol execution ends when the environment halts. Let $\mathrm{LREAL}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ be a random variable taking the output of $\mathcal{Z}$ and $\mathrm{LREAL}_{\pi,\mathcal{A},\mathcal{Z}}$ be the ensemble as in the UC model. The random variable $\mathrm{LIDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z)$ and the ensemble $\mathrm{LIDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ in the ideal process are defined as well.

Delayed output. A delayed output is an output which is scheduled by the adversaries. Let P_0, P_1 be the party-identities, and $A_{(0,1)}, A_{(1,0)}$ denote the appropriate adversaries. A functionality $\mathcal{F}$ is said to send a delayed output m to P_i by sending (**Output**, P_i, p) first to $A_{(1-i,i)}$, where p denotes optional parameters and after receiving (**Approve**, P_i, p), then $\mathcal{F}$ sends (**Output**, P_i, p) to $A_{(i,1-i)}$. If $A_{(i,1-i)}$ approves of the delivery, $\mathcal{F}$ writes m to the subroutine output tape of P_i. A *public* delayed output reveals the message to the adversaries, while a *private* delayed output does not.

Then, we have the following definition in the LUC model as well as that of the UC model.

Definition 3 ([8]). *Let $n \in \mathbf{N}$. Let $\mathcal{F}$ be an ideal functionality and let π be an n-party protocol. We say that π LUC-realizes $\mathcal{F}$, if for every adversary $\mathcal{A}$ there exists an ideal-process simulator $\mathcal{S}$ such that for any environment $\mathcal{Z}$,* $\mathrm{LIDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \mathrm{LREAL}_{\pi,\mathcal{A},\mathcal{Z}}$

In addition, we can have the results about the hybrid model and the composition theorem in the LUC model which are very similar to those in the UC model (i.e., Theorem 1). For details, see [8].

3 A Compiler in the LUC Framework

In this section, we analyze the protocol-compiler of [7] (i.e., the compiler in the UC model) in the LUC framework. At the beginning, we describe it, and then we point out that it does not work well in the LUC framework in general, and show a condition that it works well even in LUC framework.

3.1 Previous Compiler in the UC Framework

In order to transform a protocol into one that is secure against malicious adversaries, it is necessary to enforce malicious corrupted parties to follow the prescribed protocol in a semi-honest way. Canetti et al. [7] proposed a universally composable compiler based on the work of [12]. The compiler uses the commit-and-prove functionality $\mathcal{F}_{CP}$ which is defined so that only correct statements are received by a receiver and incorrect statements are rejected. In a nutshell, the committer commits its input value w as a witness and forwards a statement x to the verifier by using $\mathcal{F}_{CP}$. The statement x is received by the verifier only when $R(x, w)$ holds, where R is a predetermined relation. In the compiled protocol, there are two copies of the functionality, one for the case where P_1 is the committer and the other one for the case where P_2 is the committer, denoted by $\mathcal{F}^1_{CP}$ and $\mathcal{F}^2_{CP}$ respectively, and these are identified by session-identifiers sid_1 and sid_2. The definition of the ideal functionality $\mathcal{F}_{CP}$ and the protocol-compiler Comp() are given as follows.

Functionality $\mathcal{F}_{CP}$

$\mathcal{F}_{CP}$, which is running with a committer C, a receiver V and an adversary $\mathcal{S}$, and is parameterized by a value k and a relation R, proceeds as follows:

- **Commit phase.** Upon receiving a message (**commit**, *sid*, w) from C where $w \in \{0,1\}^k$, append the value w to the list $\overline{w}$, and send the message (**receipt**, *sid*) to V and $\mathcal{S}$. (Initially, the list $\overline{w}$ is empty.)

- **Prove phase.** Upon receiving a message (**CP-prover**, *sid*, x) from C where $x \in \{0,1\}^{poly(k)}$, compute $R(x, \overline{w})$: If $R(x, \overline{w}) = 1$, then send V and S the message (**CP-proof**, *sid*, x). Otherwise, ignore the message.

Figure 1: The commit-and-prove functionality in the UC model.

Description of Comp($\cdot$): A party P_1 proceeds as follows (the code for a party P_2 is analogous).

1. **Random tape generation.** When activating Comp(π) for a protocol π for the first time with a session-identifier *sid*, the party P_1 (and P_2) proceeds as follows.

   (a) *Choosing a random tape for P_1.*

      i. P_1 chooses $r_1^1 \in_R \{0,1\}^k$ and sends (**commit**,sid_1,r_1^1) to $\mathcal{F}_{CP}^1$. Then, P_2 receives a (**receipt**, sid_1), and P_2 chooses $r_1^2 \in_R \{0,1\}^k$ and sends (sid, r_1^2) to P_1.

      ii. When P_1 receives a message (sid, r_1^2) from P_2, it sets $r_1 := r_1^1 \oplus r_1^2$ (r_1 serves as P_1's random tape for execution of π).

   (b) *Choosing a random tape for P_2.*

      i. P_1 waits to receive a message (**receipt**,sid_2) from $\mathcal{F}_{CP}^2$ (this occurs after P_2 sends a commit message (**commit**,sid_2,r_2^2) to $\mathcal{F}_{CP}^2$). It then chooses $r_2^1 \in_R \{0,1\}^k$ and sends (sid,r_2^1) to P_2.

2. **Activation due to a new input.** When activated with an input (sid, x), the party P_1 proceeds as follows.

   (a) *Input commitment.* P_1 sends (**commit**,sid_1,x) to $\mathcal{F}_{CP}^1$ and adds x to the list of inputs $\overline{x}$ (this list is initially empty and contains P_1's inputs from all the previous activations of π).

   (b) *Protocol computation.* Let $\overline{m}_1$ be the series of π-messages that P_1 received from P_2 in all the activations of π until now ($\overline{m}_1$ is initially empty). P_1 runs the code of π on its input list $\overline{x}$, messages $\overline{m}_1$, and the random tape r_1 (as generated above).

   (c) *Outgoing message transmission.* For any outgoing message m that π instructs P_1 to send to P_2, P_1 sends (**CP-prover**,sid_1,(m, r_1^2, $\overline{m}_1$)) to $\mathcal{F}_{CP}^1$ where the relation R_π for $\mathcal{F}_{CP}^1$ is defined as follows:

   $$R_\pi = \{((m, r_1^2, \overline{m}_1), (\overline{x}, r_1^1)) \mid m = \pi(\overline{x}, r_1^1 \oplus r_1^2, \overline{m}_1)\}.$$

   In this step, P_1 proves that m is truely the correct message generated by π with the input list $\overline{x}$, the random tape $r_1 = r_1^1 \oplus r_1^2$, and the series of incoming π-messages $\overline{m}_1$.

3. **Activation due to incoming message.** When activated with an incoming message (**CP-proof**, sid_2, (m, r_2^1, $\overline{m}_2$)) from $\mathcal{F}_{CP}^2$, P_1 first verifies that the following conditions hold ($\mathcal{F}_{CP}^2$ is parameterized by the same relation R_π as $\mathcal{F}_{CP}^1$):

   (a) r_2^1 is the string that P_1 sent to P_2 in the step of 1-(b)-i above.

   (b) $\overline{m}_2$ equals the series of π-messages received by P_2 from P_1 in all the activations until now.

   If the conditions do not hold, then P_1 ignores the message. Otherwise, P_1 appends m to its list of incoming π-messages $\overline{m}_1$ and proceeds as in the steps 2-(b) and 2-(c).

4. **Output.** Whenever π generates an output, Comp(π) generates the same output.

In the UC framework, if a protocol π has UC security against semi-honest adversaries, the compiled protocol Comp(π)-$\mathcal{F}_{CP}$ is proved to be secure against malicious adversaries. The proof can be shown by the following steps: First, let $\mathcal{A}$ be a malicious adversary against the compiled protocol Comp(π) in the $\mathcal{F}_{CP}$-hybrid model and let $\mathcal{A}'$ be a semi-honest adversary against the plain protocol π, then $\mathcal{A}'$ runs a simulated copy of $\mathcal{A}$ internally and interacts with π. Recall that $\mathcal{A}'$ follows the specification of the protocol π, on the other hand, $\mathcal{A}$ is allowed to behave arbitrarily. However, the malicious adversary $\mathcal{A}$ cannot cheat since each message sent throughout the protocol is verified by $\mathcal{F}_{CP}$ so that it has no choice but to behave in a semi-honest manner. For this reason, the semi-honest adversary $\mathcal{A}'$ can simulate the behavior of $\mathcal{A}$ by delivering a message only when $\mathcal{A}$ sends a correct message. In other words, from the view of the environment $\mathcal{Z}$, it is impossible to distinguish whether it is interacting with Comp(π) and $\mathcal{A}$ in the $\mathcal{F}_{CP}$-hybrid model, or with the plain protocol π and $\mathcal{A}'$. In the circumstances, it is shown that the compiled protocol UC-realizes the target functionality in the malicious model.

3.2 A Compiler in the LUC framework

To utilize the compiler Comp($\cdot$) in the LUC framework, we need to similarly complete the simulation mentioned in the previous subsection even in the LUC framework. However, we cannot do that without any modification on the existing process. The reason of this impossibility lies in the difference between the models of UC and LUC. In the UC model, communications between parties are mediated by the centralized adversary and it directly delivers a message to recipients. In contrast, in the LUC model, plural adversaries mediate communications and messages are supposed to go through the environment in the process. That means the environment $\mathcal{Z}$ can tell whether parties communicate each other through the ideal functionality, since if messages were delivered by the ideal functionality, they would not go through the environment. Therefore, by focusing on this point, the simulation will be distinguishable.

Based on the above point, we consider switching the interacting process of an original protocol π from the one totally controlled by the environment to the one which uses a subroutine so that the problem does not occur. Specifically, we consider a message transmission functionality, denoted by $\hat{\mathcal{F}}_{MT}$, below. Note that this functionality can be realized in the LUC framework, though the functionality is originally considered in the UC framework [6].

Functionality $\hat{\mathcal{F}}_{MT}$

$\hat{\mathcal{F}}_{MT}$, which is running with parties $P_1, \dots, P_n$ and adversaries $\mathcal{S}_{(i,j)}$, where $(i,j) \in \mathcal{P}^2$ $(i \neq j)$ and $\mathcal{P}$ is the set of identities, proceeds as follows:

- Upon receiving a message (**Send**, sid, m, P_j) from a party P_i, send a public delayed output (**Send**, sid, m, P_i) to the party P_j.

- Upon receiving a message (**Deliver**, m, (l,k)) from an adversary $\mathcal{S}_{(i,j)}$, where $l,k,i,j \in \mathcal{P}$, send the message (**Delivered**, m, (i,j)) to the adversary $\mathcal{S}_{(l,k)}$.

Figure 2: The message transmission functionality in the LUC model.

Subsequently, we show an adjusting point in regards to the commit-and-prove functionality required for constructing the protocol compiler. In the UC model, we can use the ideal functionality $\mathcal{F}_{CP}$ since it has been proved that there exists a protocol which UC-realizes it. However, to use such a functionality in the LUC model, we first need to show an existence of a protocol which LUC-realizes it. In this paper, we adopt the notion of the merger functionality in [8]. In short, we modify the functionality of [7] (in the UC model) artificially so that the protocol will LUC-realize the resulting functionality. Unfortunately, this modification allows adversaries to communicate freely when the modified ideal functionality, denoted by $\hat{\mathcal{F}}_{CP}$, is used as a subroutine of other protocols (hybrid model). Concerning this point, if we demand collusion-freeness for designing protocols, we cannot adopt this method. However, such a property is not needed in this work. Generally, in two-party protocols, if both parties are corrupted by the corresponding adversaries respectively and they coordinate their actions, the mechanism of protocol compiler seems to be totally unnecessary. Considering that, we should focus on the case where both adversaries are not cooperative. (The situation either P_1 or P_2 is corrupted by the corresponding adversary can be covered by the protocol compiler in the UC model, however, the situation both parties are corrupted by different adversaries cannot be covered except if we consider it in the LUC framework.) Therefore, the most important point is whether the simulation can be completed in this framework. First, we propose an ideal functionality $\hat{\mathcal{F}}_{CP}$ as follows.

Functionality $\hat{\mathcal{F}}_{CP}$

$\hat{\mathcal{F}}_{CP}$, which is running with a committer C, a receiver V and adversaries $\mathcal{S}_{(C,V)}$ and $\mathcal{S}_{(V,C)}$, and being parameterized by a value k and a relation R, proceeds as follows:

• **Commit phase.** Upon receiving a message (**commit**, *sid*,w) from C, where $w \in \{0,1\}^k$, append the value w to the list $\overline{w}$, and send a public delayed output (**receipt**, *sid*) to V. (Initially, the list $\overline{w}$ is empty.)

• **Prove phase.** Upon receiving a message (**CP-prover**, *sid*,x) from C where $x \in \{0,1\}^{poly(k)}$, compute $R(x, \overline{w})$; If $R(x, \overline{w}) = 1$, then send a public delayed output (**CP-proof**, *sid*,x) to V; Otherwise, ignore the message.

• Upon receiving a message (**Deliver**, m, (j,i)) from the adversary $\mathcal{S}_{(i,j)}$, if $\mathcal{S}_{(i,j)}, \mathcal{S}_{(j,i)} \in \{\mathcal{S}_{(C,V)}, \mathcal{S}_{(V,C)}\}$, send the message (**Delivered**, m, (i,j)) to the adversary $\mathcal{S}_{(j,i)}$.

Figure 3: The commit-and-prove functionality in the LUC model.

Then, we can show the following results.

Theorem 2. *Let π be a two-party protocol and let* Comp(π-$\hat{\mathcal{F}}_{MT}$) *be the protocol obtained by applying the compiler to π in the $\hat{\mathcal{F}}_{MT}$-hybrid model. Then, for every malicious adversary $\mathcal{A}$ that interacts with* Comp(π-$\hat{\mathcal{F}}_{MT}$) *in the $\hat{\mathcal{F}}_{CP}$-hybrid model there exists a semi-honest adversary $\mathcal{A}'$ that interacts with π-$\hat{\mathcal{F}}_{MT}$, such that for every environment $\mathcal{Z}$,*

$$\mathrm{LREAL}_{\pi\text{-}\hat{\mathcal{F}}_{\mathrm{MT}},\mathcal{A}',\mathcal{Z}} \equiv \mathrm{LEXEC}^{\hat{\mathcal{F}}_{\mathrm{CP}}}_{\mathrm{Comp}(\pi\text{-}\hat{\mathcal{F}}_{\mathrm{MT}}),\mathcal{A},\mathcal{Z}}.$$

Proof. Let $\mathcal{A}'_{(1,2)}$ be an adversary for P_1's side and $\mathcal{A}'_{(2,1)}$ be an adversary for P_2's side. As in the UC case, $\mathcal{A}'_{(1,2)}$ and $\mathcal{A}'_{(2,1)}$ run a simulated copy of $\mathcal{A}_{(1,2)}$ and $\mathcal{A}_{(2,1)}$ respectively, and their actions are utilized as a guide for the interaction with π-$\hat{\mathcal{F}}_{MT}$ and $\mathcal{Z}$. We regard the communications of $\mathcal{A}'_{(1,2)}$ and $\mathcal{A}'_{(2,1)}$ with $\mathcal{Z}$ and π-$\hat{\mathcal{F}}_{MT}$ as an external communication, and the communications of $\mathcal{A}'_{(1,2)}$ and $\mathcal{A}'_{(2,1)}$ with the corresponding simulated $\mathcal{A}'_{(1,2)}$ or $\mathcal{A}'_{(2,1)}$ as an internal communication. $\mathcal{A}'_{(1,2)}$ and $\mathcal{A}'_{(2,1)}$ proceed as follows.

– **Simulating the communication with $\mathcal{Z}$.** Every input coming from $\mathcal{Z}$ is sent to the corresponding simulated adversary $\mathcal{A}_{(1,2)}$ or $\mathcal{A}_{(2,1)}$ as if coming from their own environment. In the same way, every output from internal adversaries is treated as an output of corresponding simulator.

– **Simulating the random tape generation phase.** We consider the following cases below.

1. **Both parties are honest**: We describe the simulation for the P_1's random tape generation (the simulation for P_2 is analogous). $\mathcal{A}'_{(1,2)}$ begins by passing the message (**receipt**, sid_1) to $\mathcal{A}_{(1,2)}$ as if coming from $\hat{\mathcal{F}}^1_{CP}$, and after $\mathcal{A}_{(1,2)}$ approved, $\mathcal{A}'_{(1,2)}$ delivers this message to $\mathcal{A}'_{(2,1)}$ using $\hat{\mathcal{F}}_{MT}$. Similarly, $\mathcal{A}'_{(2,1)}$ passes the message (**receipt**, sid_1) to $\mathcal{A}_{(2,1)}$ and if it approves, then chooses a random r_1^2 and passes the value to $\mathcal{A}_{(2,1)}$ as if coming from P_2. Furthermore, confirming that $\mathcal{A}_{(2,1)}$ delivers this value to P_1 using $\hat{\mathcal{F}}_{MT}$, $\mathcal{A}'_{(2,1)}$ actually delivers it to $\mathcal{A}'_{(1,2)}$ using $\hat{\mathcal{F}}_{MT}$. Finally, $\mathcal{A}'_{(1,2)}$ receives an approval from $\mathcal{A}_{(1,2)}$.

2. P_1 **is honest and** P_2 **is corrupted**: At first, we consider the generation of P_1's random tape. The simulation proceeds as in the case 1. $\mathcal{A}'_{(2,1)}$ receives the message (**receipt**, sid_1), then passes it to $\mathcal{A}_{(2,1)}$. If $\mathcal{A}_{(2,1)}$ delivers r_1^2 to P_2 using $\hat{\mathcal{F}}_{MT}$, $\mathcal{A}'_{(2,1)}$ actually delivers it to $\mathcal{A}'_{(1,2)}$ using $\hat{\mathcal{F}}_{MT}$. The rest of the process is the same as in the case 1. Next, we consider the generation of P_2's random tape. $\mathcal{A}'_{(2,1)}$ obtains the message (**commit**, sid_2, r_2^2) from $\mathcal{A}_{(2,1)}$ which sends it to $\hat{\mathcal{F}}^2_{CP}$ on behalf of P_2 in execution of Comp(π-$\hat{\mathcal{F}}_{MT}$). Now, as the direction of this simulation, we must let the random tape of internal P_2 equal the random tape of P_2 in external execution of π-$\hat{\mathcal{F}}_{MT}$ so that $\mathcal{A}_{(2,1)}$ is forced to use the same randomness throughout the computation. For that reason, $\mathcal{A}'_{(2,1)}$ delivers the random tape of external P_2, denoted by r_2, to $\mathcal{A}'_{(1,2)}$ using $\hat{\mathcal{F}}_{MT}$. Then $\mathcal{A}'_{(1,2)}$ simulates $\mathcal{A}_{(1,2)}$'s behavior as in the case 1, and then delivers $r_2^1 = r_2$ to $\mathcal{A}'_{(2,1)}$. Finally, $\mathcal{A}'_{(2,1)}$ sets $r_2^1 = r_2 \oplus r_2^2$ and passes it to $\mathcal{A}_{(2,1)}$.

3. P_1 **is corrupted and** P_2 **is honest**: This case can be simulated analogously to the previous one. That is, the random tape of internal P_1 corrupted by $\mathcal{A}_{(1,2)}$ becomes to be equal to that of external P_1 corrupted by $\mathcal{A}'_{(1,2)}$.

4. **Both parties are corrupted**: Similarly, this case can be simulated by applying simultaneously the simulators of the cases 2 and 3 above.

– **Simulating an activation due to a new input.** We describe the simulation from P_1's side (the simulation for P_2 is analogous).

1. P_1 **is not corrupted**: $\mathcal{A}'_{(1,2)}$ learns the fact that external P_1 is given a new input when it receives an approval request from $\hat{\mathcal{F}}_{MT}$. Then, $\mathcal{A}'_{(1,2)}$ passes the message (**receipt**, sid_1) to $\mathcal{A}_{(1,2)}$ as if coming from $\hat{\mathcal{F}}^1_{CP}$ and after receiving an approval from $\mathcal{A}_{(1,2)}$, $\mathcal{A}'_{(1,2)}$ delivers the same message to $\mathcal{A}'_{(2,1)}$ using $\hat{\mathcal{F}}_{MT}$. Subsequently, $\mathcal{A}'_{(2,1)}$ proceeds the rest process by checking whether $\mathcal{A}'_{(2,1)}$ approves or not.

2. P_1 **is corrupted**: $\mathcal{A}'_{(1,2)}$ receives the message (**commit**, sid_1, x) from $\mathcal{A}_{(1,2)}$. Then, $\mathcal{A}'_{(1,2)}$ adds x to its list $\overline{x}$ and passes (**receipt**, sid_1) to $\mathcal{A}_{(1,2)}$, as if coming from $\hat{\mathcal{F}}^1_{CP}$. After receiving an approval from $\mathcal{A}_{(1,2)}$, $\mathcal{A}'_{(1,2)}$ sets the input tape of external P_1 being equal to x (Note that a semi-honest adversary is allowed to modify the input values of corrupted parties, which is mentioned in [7] and this definition is due to the fact that there is no difference in terms of security between the case where the semi-honest adversary can modify a corrupted party's input value and the case where it cannot). Furthermore, $\mathcal{A}'_{(1,2)}$ delivers (**receipt**, sid_1) to $\mathcal{A}'_{(2,1)}$ using $\hat{\mathcal{F}}_{MT}$ and $\mathcal{A}'_{(2,1)}$ proceeds the rest process as in the case 1.

– **Dealing with π-$\hat{\mathcal{F}}_{MT}$ messages sent externally by uncorrupted parties.** When external P_1 who is not corrupted sends a message m to P_2 using $\hat{\mathcal{F}}_{MT}$, $\mathcal{A}'_{(1,2)}$ internally passes $\mathcal{A}_{(1,2)}$ the message

(**CP-proof**, sid_1, $(m,\ r_1^2,\ \overline{m}_1)$) as $\mathcal{A}_{(1,2)}$ expects to receive from $\hat{\mathcal{F}}_{CP}^1$), where r_1^2 is the value used in the P_1's random tape generation phase, and $\overline{m}_1$ is the series of all messages P_1 received in the execution of π-$\hat{\mathcal{F}}_{MT}$ so far. Similarly, if P_2 sends a message m to P_1, $\mathcal{A}'_{(2,1)}$ would pass the message (**CP-proof**, sid_2, $(m,\ r_2^1,\ \overline{m}_2)$) to $\mathcal{A}_{(2,1)}$. Each simulator delivers a message to the recipients only when the internal adversary approves of the message delivery.

– **Dealing with** Comp(π-$\hat{\mathcal{F}}_{MT}$) **messages sent internally by corrupted parties.** Consider the case where P_1 is corrupted. When $\mathcal{A}_{(1,2)}$ sends the message (**CP-prover**, sid_1, $(m,\ r'^2_1,\ \overline{m}'_1)$) to $\hat{\mathcal{F}}_{CP}^1$, $\mathcal{A}'_{(1,2)}$ can verify that $\overline{m}'_1 = \overline{m}_1$ and $r_1^2 = r'^2_1$, besides, $m = \pi\text{-}\hat{\mathcal{F}}_{MT}(\overline{x}, r_1^1 \oplus r_1^2, \overline{m}_1)$, since P_1 is corrupted so that $\mathcal{A}'_{(1,2)}$ can obtain all the information needed for these checking. If no error is found, $\mathcal{A}'_{(1,2)}$ passes (**CP-proof**, sid_1, $(m,\ r'^2_1,\ \overline{m}'_1)$) to $\mathcal{A}_{(1,2)}$ as if coming from $\hat{\mathcal{F}}_{CP}^1$. Then, when $\mathcal{A}_{(1,2)}$ approves of delivering this message, $\mathcal{A}'_{(1,2)}$ delivers m to $\mathcal{A}'_{(2,1)}$ using $\hat{\mathcal{F}}_{MT}$. After that, $\mathcal{A}'_{(2,1)}$ passes (**CP-proof**,...) message to $\mathcal{A}_{(2,1)}$, and regardless of whether P_2 is corrupted or not, $\mathcal{A}'_{(2,1)}$ approves of message delivering only when $\mathcal{A}_{(2,1)}$ approves. With this, the simulation is completed and the simulation for P_2 is analogous. □

Corollary 1. *Let $\mathcal{F}$ be a two-party functionality and let π be a non-trivial protocol that LUC-realizes $\mathcal{F}$ in the presence of semi-honest adversaries. Then,* Comp(π-$\hat{\mathcal{F}}_{MT}$) *is a non-trivial protocol that LUC-realizes $\mathcal{F}$ in the $\hat{\mathcal{F}}_{CP}$-hybrid model and in the presence of malicious adversaries.*

4 Oblivious Transfer with UC and Game-Theoretic Security

4.1 Oblivious Transfer in the UC Framework

The oblivious transfer [10, 23] is a two-party cryptographic functionality implemented by a sender T who has input $x_1, x_2, \ldots, x_l$ and a receiver R who has input $i \in \{1, 2, \ldots, l\}$. When they follow the given specifications correctly, R receives the message x_i such that R cannot obtain any more information, while T obtains no information about the selection of R. We describe the ideal functionality $\mathcal{F}_{OT}$ in [7], and the protocol SOT (1-out-of-l) for the static and semi-honest adversarial model in [7, 11, 12] as follows.

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

$\mathcal{F}_{OT}$, which is parameterized with an integer l, and running with an sender T, a receiver R and an adversary $\mathcal{S}$, proceeds as follows:

• Upon receiving a message (**sender**, sid, $x_1, \ldots, x_l$) from T, where $x_i \in \{0,1\}^m$, record the tuple $(x_1, \ldots, x_l)$.

• Upon receiving a message (**receiver**, sid, i) from R, where $i \in \{1, \ldots, l\}$, send (sid, x_i) to R and (sid) to $\mathcal{S}$, and halt.

Figure 4: Functionality of oblivious transfer in the UC model.

Protocol SOT

Proceed with a security parameter k as follows.

• Given input (**sender**, sid, $x_1, \ldots, x_l$), the party T chooses a trapdoor permutation f over $\{0,1\}^k$, together with its inverse f^{-1}, and sends (sid, f) to the receiver R. (The permutation f is chosen uniformly from a given family of trapdoor permutations.)

• Given input (**receiver**, sid, i), and having received (sid, f) from T, the receiver R chooses $y_1, \ldots, y_{i-1}, r, y_{i+1}, \ldots, y_l \in_R \{0,1\}^k$, computes $y_i = f(r)$, and sends (sid, $y_1, \ldots, y_l$) to T.

• Having received (sid, $y_1, \ldots, y_l$) from R, the sender T sends (sid, $x_1 \oplus B(f^{-1}(y_1)), \ldots, x_l \oplus B(f^{-1}(y_l))$) to R, where $B(\cdot)$ is a hard-core predicate for f.

• Having received (sid, $b_1, \ldots, b_l$) from T, the receiver R outputs (sid, $b_i \oplus B(r)$).

Figure 5: A static and semi-honest oblivious transfer protocol.

4.2 Oblivious Transfer in the LUC Framework

For game-theoretic analysis, we consider realizing functionality of oblivious transfer in the LUC framework. To do so, we first investigate whether some modification will be needed in changing the framework from UC to LUC as follows: We consider the case that we use the previous ideal functionality $\mathcal{F}_{OT}$ for the protocol simulation in the LUC framework. If sender T is not corrupted, in the ideal process, the corresponding dummy party T passes its own input value to $\mathcal{F}_{OT}$ automatically at the first step. Then, after receiving the value from T, $\mathcal{F}_{OT}$ records it and enters a waiting state. Following that, the environment $\mathcal{Z}$ is activated next and it is supposed to activate the receiver R with an input value. On the other hand, in the real life protocol execution, the process of the first message delivery is as follows. At first, the sender T passes its input value to the corresponding adversary, denoted by $\mathcal{A}_{(T,R)}$, and then $\mathcal{A}_{(T,R)}$ delivers the value to the opponent adversary, denoted by $\mathcal{A}_{(R,T)}$, through the environment $\mathcal{Z}$. Finally, the receiver R receives the value from $\mathcal{A}_{(R,T)}$. Therefore, the environment $\mathcal{Z}$ can obviously tell whether it is facing the ideal process or the real life protocol execution, since there is great difference between the two situations.

For this reason, we need to modify the definition of the previous ideal functionality of OT by changing the framework so that the difference mentioned above does not arise. We describe the modified ideal functionality $\hat{\mathcal{F}}_{OT}$ as follows.

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

$\hat{\mathcal{F}}_{OT}$, which is parameterized with an integer l and running with a sender T, a receiver R and adversaries $\mathcal{S}_{(T,R)}$ and $\mathcal{S}_{(R,T)}$, proceeds as follows:

• Upon receiving a message (**sender**, sid, $x_1, \ldots, x_l$) from T, where $x_i \in \{0,1\}^m$, record the tuple $(x_1, \ldots, x_l)$. If the message from R has already been recorded, then send a private delayed output (sid, x_i) to R, and halt. Otherwise, send a public delayed output (**receipt**, sid) to R.

• Upon receiving a message (**receiver**, sid, i) from R, where $i \in \{1, \ldots, l\}$, record the value i. If the message from T has already been recorded, then send a private delayed output (sid, x_i) to R, and halt. Otherwise, send a public delayed output (**receipt**, sid) to T.

• Upon receiving a message (**Deliver**, m, (j,i)) from the adversary $\mathcal{S}_{(i,j)}$, if $\mathcal{S}_{(i,j)}, \mathcal{S}_{(j,i)} \in \{\mathcal{S}_{(T,R)}, \mathcal{S}_{(R,T)}\}$, send the message (**Delivered**, m, (i,j)) to the adversary $\mathcal{S}_{(j,i)}$.

Figure 6: Functionality of oblivious transfer in the LUC model.

Then, we show that the protocol SOT meets the following security.

Theorem 3. *Suppose that f in the protocol* SOT *is an enhanced trapdoor permutation*[1]. *Then,* SOT *LUC-realizes $\hat{\mathcal{F}}_{OT}$ in the presence of semi-honest and static adversaries.*

Proof. As in the UC case, $\mathcal{S}_{(T,R)}$ and $\mathcal{S}_{(R,T)}$ run a simulated copy of $\mathcal{A}_{(T,R)}$ and $\mathcal{A}_{(R,T)}$ respectively, and their actions are utilized as a guide for the interaction with $\hat{\mathcal{F}}_{OT}$ and $\mathcal{Z}$. $\mathcal{S}_{(T,R)}$ and $\mathcal{S}_{(R,T)}$ proceed as follows.

– **Simulating the communication with $\mathcal{Z}$.** Every input coming from $\mathcal{Z}$ is sent to the corresponding simulated adversary $\mathcal{A}_{(T,R)}$ or $\mathcal{A}_{(R,T)}$ as if coming from their own environment. In the same way, every output from internal adversaries is treated as an output of corresponding simulator.

– **Simulating the case where no party is corrupted.** At first, the simulator $\mathcal{S}_{(T,R)}$ is activated by receiving the message (**receipt**, *sid*) from $\hat{\mathcal{F}}_{OT}$ ($\mathcal{S}_{(T,R)}$ is demanded for approving of the message delivery). Then, $\mathcal{S}_{(T,R)}$ randomly chooses a trapdoor permutation f over $\{0,1\}^k$ with its inverse f^{-1} and passes (*sid*, f) to the simulated adversary $\mathcal{A}_{(T,R)}$. When $\mathcal{A}_{(T,R)}$ delivers the message to the environment $\mathcal{Z}$, $\mathcal{S}_{(T,R)}$ actually delivers it to the opponent simulator $\mathcal{S}_{(R,T)}$ through $\mathcal{Z}$. Following that, $\mathcal{S}_{(R,T)}$ activates $\mathcal{S}_{(T,R)}$ by using $\hat{\mathcal{F}}_{OT}$, and $\mathcal{S}_{(T,R)}$ approves of $\hat{\mathcal{F}}_{OT}$'s message delivery at this timing. Then, $\mathcal{S}_{(R,T)}$ is activated again with a request for an approval from $\hat{\mathcal{F}}_{OT}$. $\mathcal{S}_{(R,T)}$ approves after confirming that $\mathcal{A}_{(R,T)}$ delivers (*sid*, f) to R. Next, the dummy party R receives (**receiver**, *sid*, i) from $\mathcal{Z}$ as an input and sends it to $\hat{\mathcal{F}}_{OT}$. Then, $\mathcal{S}_{(T,R)}$ is activated with a request for an approval from $\hat{\mathcal{F}}_{OT}$. At this timing, $\mathcal{S}_{(T,R)}$ approves of $\hat{\mathcal{F}}_{OT}$'s message delivery. After that, $\mathcal{S}_{(R,T)}$ is activated with a request for an approval similar to the process of $\mathcal{S}_{(T,R)}$. Then, $\mathcal{S}_{(R,T)}$ chooses $y_1, \dots, y_l \in \{0,1\}^k$ and passes these values to $\mathcal{A}_{(R,T)}$. After confirming that $\mathcal{A}_{(R,T)}$ delivers the message (*sid*, $y_1, \dots, y_l$) to $\mathcal{Z}$, $\mathcal{S}_{(R,T)}$ actually delivers it to $\mathcal{S}_{(T,R)}$ through $\mathcal{Z}$. Similarly, $\mathcal{S}_{(T,R)}$ simulates $\mathcal{A}_{(T,R)}$ delivering the message to T internally. Following that, $\mathcal{S}_{(T,R)}$ chooses $b_1, \dots, b_l$ uniformly, and passes the message (*sid*, $b_1, \dots, b_l$) to $\mathcal{A}_{(T,R)}$. If $\mathcal{A}_{(T,R)}$ delivers the message correctly, then $\mathcal{S}_{(T,R)}$ actually delivers it to $\mathcal{S}_{(R,T)}$ through $\mathcal{Z}$. Finally, $\mathcal{S}_{(R,T)}$ concludes the simulation by confirming that $\mathcal{A}_{(R,T)}$ delivers the message to R and approving of $\hat{\mathcal{F}}_{OT}$'s message delivery.

– **Simulating the case where only the sender T is corrupted.** $\mathcal{S}_{(T,R)}$ begins by sending the message (**sender**, *sid*, $x_1, \dots, x_l$) to $\hat{\mathcal{F}}_{OT}$ and receives a request for an approval of message delivery. Before approving, $\mathcal{S}_{(T,R)}$ activates $\mathcal{A}_{(T,R)}$ with an input value and receives (*sid*, f) that $\mathcal{A}_{(T,R)}$ is supposed to deliver R in a real life protocol execution. Furthermore, $\mathcal{S}_{(T,R)}$ delivers the message to $\mathcal{S}_{(R,T)}$ through $\mathcal{Z}$ as in the above case, and once activated next, it approves the message delivery of $\hat{\mathcal{F}}_{OT}$. The following process is analogous to the above case. Next, after R sends its input to $\hat{\mathcal{F}}_{OT}$, $\mathcal{S}_{(T,R)}$ is activated with a request for approving of $\hat{\mathcal{F}}_{OT}$'s message delivery. At the time when $\mathcal{S}_{(T,R)}$ receives (*sid*, $y_1, \dots, y_l$), it passes the message to $\mathcal{A}_{(T,R)}$. Subsequently, after receiving (*sid*, $b_1, \dots, b_l$) from $\mathcal{A}_{(T,R)}$, $\mathcal{S}_{(T,R)}$ delivers it to $\mathcal{S}_{(R,T)}$ through $\mathcal{Z}$. Finally, $\mathcal{S}_{(R,T)}$ concludes the simulation in the same way as in the above case.

– **Simulating the case where only the receiver R is corrupted.** The simulation proceeds similar to the case where no party is corrupted until $\mathcal{S}_{(R,T)}$, controlling R, is activated with an input (**receiver**, *sid*, i). Following that, $\mathcal{S}_{(R,T)}$ passes it to $\mathcal{A}_{(R,T)}$ and receives (*sid*, $y_1, \dots, y_l$). Furthermore, $\mathcal{S}_{(R,T)}$ delivers the message to $\mathcal{S}_{(T,R)}$ through $\mathcal{Z}$. After receiving that message, $\mathcal{S}_{(T,R)}$ delivers (*sid*, $b_1, \dots, b_l$) to $\mathcal{S}_{(R,T)}$ through $\mathcal{Z}$, and $\mathcal{S}_{(R,T)}$ obtains f^{-1} by using $\hat{\mathcal{F}}_{OT}$. Then, $\mathcal{S}_{(R,T)}$

[1]The enhanced trapdoor permutation has the property that a random element generated by the domain sampler is hard to invert, even given the random coins used by the sampler. Note that any trapdoor permutation over $\{0,1\}^k$ is clearly enhanced, since this domain can be easily and directly sampled.

sends (**receiver**, sid, i) to $\hat{\mathcal{F}}_{OT}$ and subsequently both simulators approve of the message delivery ($\mathcal{S}_{(R,T)}$ receives x_i). Next, $\mathcal{S}_{(R,T)}$ sets $b_i = x_i \oplus B(f^{-1}(y_i))$ and passes $(sid,\ b_1, \ldots, b_l)$ to $\mathcal{A}_{(R,T)}$. Finally, $\mathcal{S}_{(R,T)}$ concludes the simulation by outputting x_i when $\mathcal{A}_{(R,T)}$ does so.

– **Simulating the case where both parties are corrupted.** This case can be simulated by applying simultaneously the simulators of each case where only one of the parties is corrupted.

□

4.3 Analysis of Game-Theoretic Security

Next, we consider the case where rational parties implement the protocol SOT in the case $l = 2$ as in [14], since 1-out-of-2 OT is simple and fundamental. As already mentioned, SOT is designed for the semi-honest adversarial model so that its security does not concern the behaviors of rational parties. We investigate whether SOT is game-theoretically secure, before and after being compiled, respectively.

First, we define utility functions for two-message OT protocols similar to the work of [14]. In [14], Higo et al. studied the game-theoretic concepts of two-message OT protocols with reasonable utility functions, so it seems to be appropriate to follow the previous definitions. For doing it, we consider sender's (i.e., T's) and receiver's (i.e., R's) preferences as follows:

- T does not prefer the receiver R to know the input bit x_{1-i}, where the index of the receiver's selection is $i \in \{0, 1\}$ (This explains the case where R obtains x_0 and x_1 simultaneously),
- T prefers to complete the protocol execution,
- T prefers to know the input index of the receiver's selection $i \in \{0, 1\}$; and
- R does not prefer the sender S to know its input index of the selection,
- R prefers to complete the protocol execution,
- R prefers to know the other sender's input bit x_{1-i}.

Then, a formal definition of utility functions is given as follows.

Definition 4 (Utility functions). *Let π be an OT protocol having a sender T with inputs $x_0, x_1 \in \{0, 1\}$ and a receiver R with an input $i \in \{0, 1\}$. Let $\alpha_T, \beta_T, \gamma_T, \alpha_R, \beta_R, \gamma_R$ be positive constants. The utility functions U_T for T and U_R for R are defined by*

$$\begin{aligned}
U_T \;:=\;& -\alpha_T \cdot \big(\Pr\{x' = x_{1-i} \mid guess_R(T(x_0, x_1), R(i)) = x'\} - 1/2\big) \\
& +\beta_T \cdot (\Pr\{fin(T(x_0.x_1), R(i)) = 1\} - 1) \\
& +\gamma_T \cdot \big(\Pr\{i' = i \mid guess_T(T(x_0, x_1), R(i)) = i'\} - 1/2\big), \\
U_R \;:=\;& -\alpha_R \cdot \big(\Pr\{i' = i \mid guess_T(T(x_0, x_1), R(i)) = i'\} - 1/2\big) \\
& +\beta_R \cdot (\Pr\{fin(T(x_0.x_1), R(i)) = 1\} - 1) \\
& +\gamma_R \cdot \big(\Pr\{x' = x_{1-i} \mid guess_R(T(x_0, x_1), R(i)) = x'\} - 1/2\big),
\end{aligned}$$

where $guess_T(\cdot)$ and $guess_R(\cdot)$ mean guessing by T and R, respectively, for the opponent's private value, and $fin(\cdot)$ represents the completion of the protocol execution: $fin(\cdot) = 1$ if the protocol satisfies the specifications correctly; otherwise $fin(\cdot) = 0$.

In addition, as in the work of [4, 14, 15], we consider Nash equilibrium as the solution concept in terms of the game theory.

Definition 5 (Nash equilibrium)**.** *For a pair of utility functions* (U_T, U_R)*, we say that a pair of strategies* (σ_T, σ_R) *is in Nash equilibrium, if for every pair of strategies* (σ_T^*, σ_R^*)*, it holds that* $U_T(\sigma_T, \sigma_R) \geq U_T(\sigma_T^*, \sigma_R) - negl(n)$ *and* $U_R(\sigma_T, \sigma_R) \geq U_R(\sigma_T, \sigma_R^*) - negl(n)$*.*

Definition 6 (Game-theoretic security for OT)**.** *Let* π *be an OT protocol having a sender* T *and a receiver* R*. Let* σ_T *and* σ_R *be strategies planned to follow all the specifications of* π*, respectively. We say that* π *is game-theoretically secure, if the pair of strategies* (σ_T, σ_R) *is in Nash equilibrium with respect to the pair of utility functions* (U_T,U_R)*.*

Then, we can show the game-theoretic security of SOT before/after application of the compiler in the LUC model below.

Theorem 4. *The protocol* SOT *is not game-theoretically secure in the presence of rational parties, however, the compiled protocol* Comp(SOT-$\hat{\mathcal{F}}_{MT}$) *in the* $\hat{\mathcal{F}}_{CP}$*-hybrid model is game-theoretically secure in the presence of rational parties.*

Proof. First, we show that the plain protocol SOT is not secure. Once both parties are allowed to behave rationally, this protocol becomes quite imbalanced. If the receiver R attempts to enhance its own utility more than that of the case where it acts honestly, it takes action such as the following. In the step where R is supposed to choose $y_{1-i}, r \in_R \{0,1\}^k$ and computes $y_i = f(r)$, it also applies f for generating y_{1-i}. For this, R can obviously obtain the T's private value x_{1-i} in addition to x_i unless the sender T aborts the protocol execution. (Note that we take no account of the case where each party changes its own input value, since it seems reasonable to assume so. Furthermore even if that occurs, the result is not affected essentially.) In addition, since y_{1-i} and r are randomly chosen, R's dishonest behavior is not detectable. Thus, this results in increasing the value $\gamma_R \cdot (\Pr\{x' = x_{1-i} \mid guess_R(T(x_0, x_1), R(i)) = x'\} - 1/2)$.

On the sender T's side, he/she would think that R does wrong absolutely. However, T has only two choices, either following the specifications of the protocol or aborting, since T obtains no information from the received values y_0, y_1 and there is no way to benefit in the subsequent process. The selection depends on to which T gives much weight *the completion of the protocol* or *protecting the secret.* If T prefers the completion of the protocol, it results in decreasing the value $-\alpha_T \cdot (\Pr\{x' = x_{1-i} \mid guess_R(T(x_0, x_1), R(i)) = x'\} - 1/2)$ and increasing the value $\beta_T \cdot (\Pr\{fin(T(x_0.x_1), R(i)) = 1\} - 1)$ in comparison with the latter. On the contrary, if T prefers to protect the secret and chooses to abort, it results in increasing the value $-\alpha_T \cdot (\Pr\{x' = x_{1-i} \mid guess_R(T(x_0, x_1), R(i)) = x'\} - 1/2)$ and decreasing the value $\beta_T \cdot (\Pr\{fin(T(x_0.x_1), R(i)) = 1\} - 1)$ in comparison with the former. From the above discussion, at least the pair of strategies (σ_T, σ_R) is not in Nash equilibrium.

Next, we show that the compiled protocol Comp(SOT-$\hat{\mathcal{F}}_{MT}$)-$\hat{\mathcal{F}}_{CP}$ is secure. Regarding the dishonest actions of R mentioned above, R cannot enhance its own utility even if applying f for generating y_0 and y_1. Since the functionality $\hat{\mathcal{F}}_{CP}$ rejects incorrect messages, the protocol execution would never be completed. Therefore, it results in decreasing the value $\beta_R \cdot (\Pr\{fin(T(x_0.x_1), R(i)) = 1\} - 1)$ compared to that of the case where R follows the protocol specifications. On the T's side, there is no need to worry about the R's dishonest actions, and hence T can obtain the highest utility by following the protocol honestly. Similarly to the R's case, if T chooses to deviate from the protocol, T's total utility obviously decreases. Thus, the pair of strategies (σ_T, σ_R) is in Nash equilibrium.

□

5 Concluding Remarks

In this paper, we have proposed a compiler of two-party protocols in the LUC framework such that it transforms any two-party protocol secure against semi-honest adversaries into a protocol secure against malicious adversaries. Then, we have shown the application of our compiler to an oblivious transfer protocol to achieve a primitive with both UC and game-theoretic security. We emphasize that our main purpose was to address how protocols with security in the game-theoretic model can be composed to obtain an overall game-theoretically secure protocol. In this sense, our result is successful and the constructed protocol has desirable properties.

An interesting line for future work is to address whether this resulting protocol carries over to the general multi-party computation protocols as a building block in the game-theoretic setting.

Acknowledgments. We would like to thank anonymous referees for their helpful comments. This work was partially supported by JSPS KAKENHI Grant Number 15H02710, and it was partially conducted under the auspices of the MEXT Program for Promoting the Reform of National Universities.

References

[1] J. Alwen, J. Katz, Y. Lindell, G. Persiano, A. Shelat, and I. Visconti. Collusion-free multiparty computation in the mediated model. In *Advances in Cryptology — CRYPTO 2009*, pages 524–540, 2009.

[2] J. Alwen, J. Katz, U. Maurer, and V. Zikas. Collusion-preserving computation. In *Advances in Cryptology — CRYPTO 2012*, pages 124–143, 2012.

[3] J. Alwen, A. Shelat, and I. Visconti. Collusion-free protocols in the mediated model. In *Advances in Cryptology — CRYPTO 2008*, pages 497–514, 2008.

[4] G. Asharov, R. Canetti, and C. Hazay. Towards a game theoretic view of secure computation. In *Advances in Cryptology — EUROCRYPT 2011*, pages 426–445, 2011.

[5] M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. In *IACR Cryptology ePrint Archive*, 2003.

[6] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science (FOCS 2001)*, pages 136–145, 2001.

[7] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing (STOC 2002)*, pages 494–503, 2002.

[8] R. Canetti and M. Vald. Universally composable security with local adversaries. In *8th International Conference on Security and Cryptography for Networks (SCN 2012)*, pages 281–301, 2012.

[9] Y. Dodis, S. Halevi, and T. Rabin. A cryptographic solution to a game theoretic problem. In *Advances in Cryptology — CRYPTO 2000*, pages 112–130, 2000.

[10] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[11] O. Goldreich. *The Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, 2004.

[12] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *19th Annual ACM Symposium on Theory of Computing (STOC'87)*, pages 218–229, 1987.

[13] R. Gradwohl, N. Livne, and A. Rosen. Sequential rationality in cryptographic protocols. In *51th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010)*, pages 623–632, 2010.

[14] H. Higo, K. Tanaka, A. Yamada, and K. Yasunaga. A game-theoretic perspective on oblivious transfer. In *17th Australasian Conference on Information Security and Privacy (ACISP 2012)*, pages 29–42, 2012.

[15] H. Higo, K. Tanaka, and K. Yasunaga. Game-theoretic security for bit commitment. In *8th International Workshop on Security (IWSEC 2013)*, pages 303–318, 2013.

[16] S. Izmalkov, M. Lepinski, and S. Micali. Perfect implementation. *Games and Economic Behavior*, 71(1):121–140, 2011.

[17] S. Izmalkov, S. Micali, and M. Lepinski. Rational secure computation and ideal mechanism design. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005)*, pages 585–595, 2005.

[18] J. Katz. Bridging game theory and cryptography: Recent results and future directions. In *Fifth IACR Theory of Cryptography Conference (TCC 2008)*, pages 251–272, 2008.

[19] G. Kol and M. Naor. Cryptography and game theory: Designing protocols for exchanging information. In *Fifth IACR Theory of Cryptography Conference (TCC 2008)*, pages 320–339, 2008.

[20] M. Lepinski, S. Micali, and A. Shelat. Collusion-free protocols. In *37th Annual ACM Symposium on Theory of Computing (STOC 2005)*, pages 543–552, 2005.

[21] U. Maurer. Constructive cryptography – a primer. In *Financial Cryptography*, page 1, 2010.

[22] U. Maurer and R. Renner. Abstract cryptography. In *Second Symposium on Innovations in Computer Science (ICS 2011)*, pages 1–21, 2011.

[23] M. O. Rabin. How to exchange secrets with oblivious transfer. Technical Report TR-81, Aiken Computation Lab., Harvard University, 1981.