Efficient Concurrent Oblivious Transfer in Super-Polynomial-Simulation Security

Susumu Kiyoshima[1], Yoshifumi Manabe[1,2], and Tatsuaki Okamoto[1,3]

[1] Graduate School of Informatics, Kyoto University, Japan
kiyoshima@ai.soc.i.kyoto-u.ac.jp
[2] NTT Communication Science Laboratories, Japan
manabe.yoshifumi@lab.ntt.co.jp
[3] NTT Secure Platform Laboratories, Japan
okamoto.tatsuaki@lab.ntt.co.jp

Abstract. In this paper, we show a concurrent oblivious transfer protocol in super-polynomial-simulation (SPS) security. Our protocol does not require any setup and does not assume any independence among the inputs. In addition, our protocol is efficient since it does not use any inefficient primitives such as general zero-knowledge proofs for all NP statements. This is the first concurrent oblivious transfer protocol that achieves both of these properties simultaneously. The security of our protocol is based on the decisional Diffie-Hellman (DDH) assumption.

1 Introduction

1.1 Background

Oblivious Transfer. *Oblivious transfer protocols* [30] have been extensively studied in cryptography due to their usefulness in protocol constructions. Oblivious transfer protocols[4] enable the *receiver* to receive one of two values from the *sender*. The sender cannot know which value the receiver received, whereas the receiver can learn only one value and cannot learn anything about the other value. Numerous protocols have been constructed using oblivious transfer protocols. In fact, oblivious transfer is *complete* for secure computation, i.e., we can compute any function securely given an oblivious transfer protocol [17, 18].

Oblivious transfer protocols against malicious adversaries can be obtained by transforming oblivious transfer protocols against semi-honest adversaries to protocols against malicious adversaries using the protocol compiler of Goldreich et al. [13]. However, the protocols obtained in this way are highly inefficient since they use general zero-knowledge proofs for NP statements. As a result, the task of constructing efficient oblivious transfer protocols, which are indispensable for practical purposes, has attracted much attention. Efficient "fully-simulatable"[5]

[4] In this paper, we consider 1-out-of-2 oblivious transfer protocols.
[5] If we consider the half-simulation definition [23], there exist many highly-efficient protocols, e.g., [1, 22].

oblivious transfer protocols are shown in [15, 19, 20]. In addition, there exist black-box transformations, which do not use general zero-knowledge proofs, from semi-honest oblivious transfer to malicious oblivious transfer [8, 16, 26]

Concurrent Security. All of the above protocols achieve only *stand-alone security*, i.e., they are secure only when a single instance of the protocols is executed at a time. More realistic and desirable security is *concurrent security*, which guarantees that the protocol remains secure even when several instances of the protocol are executed at the same time in an arbitrary schedule.

Unfortunately, in the standard model (with adaptively-chosen inputs and no trusted setup), we cannot construct concurrent oblivious transfer protocols with black-box simulation [21]. As a result, existent concurrent oblivious transfer protocols have been constructed in other models. For example, as noted in [21], the concurrent oblivious transfer of [11] circumvents the impossibility result by considering a model where all the inputs in all the executions are independent of each other. *Universally composable* (UC) oblivious transfer protocols [9, 14, 27] achieve UC security, which implies concurrent security, in models with setups such as *common reference strings* (CRS). Although these models are reasonable in some situations, it is desirable to achieve concurrent security in the standard model.

Super-Polynomial-Simulation Security. *Super-polynomial-simulation* (SPS) security [24, 28] enables us to achieve concurrent security in the standard model. SPS security is a relaxed notion of security in the *simulation paradigm*. Before explaining further about SPS security, we introduce the simulation paradigm.

In the simulation paradigm, we define the *real world* and the *ideal world*. In the real world, the parties carry out a task (or *functionality*) by executing a protocol. In the ideal world, the parties carry out the task by interacting with an incorruptible trusted third party called the *ideal functionality*. Then, the protocol is said to be secure if for any adversary who can perform some attacks in the real world there exists an adversary (or *simulator*) who can perform essentially the same attacks in the ideal world. In the case of oblivious transfer, we define the ideal functionality as follows. The ideal functionality $\mathcal{F}$ receives m_0 and m_1 from the sender and $\sigma \in \{0, 1\}$ from the receiver. Then, $\mathcal{F}$ sends m_σ to the receiver. Clearly, $\mathcal{F}$ carries out the task in a perfectly-secure fashion. Then, the security in the simulation paradigm means that, if some attacks can be performed on the protocol by the adversary, essentially the same attacks can be performed even on $\mathcal{F}$ by the simulator.

In standard security definitions of the simulation paradigm, we restrict the running time of the simulator to polynomial time. In SPS security, we relax this security definition by allowing the simulator to run in super-polynomial time. Thus, SPS security guarantees that, if the adversary can perform some attacks in the real world, the simulator can perform essentially the same attacks *in super-polynomial time*. Although SPS security is a relaxed notion of security, it guarantees sufficient security *if the ideal functionality is information-theoretically se-*

cure, i.e., if the ideal functionality is secure against computationally-unbounded adversaries. Clearly, the above oblivious transfer ideal functionality is information-theoretically secure.

SPS security was introduced to construct constant-round concurrent zero-knowledge proofs [24, 25]. SPS security was also used in the UC framework, and it was shown that there exist protocols that compute any functionality in the standard model [2, 6, 12, 29]. Hence, using these protocols, we can construct concurrent oblivious transfer protocols in the standard model. However, the protocols obtained in this way are inefficient, since they use general zero-knowledge proofs for all NP statements. Therefore, for practical purposes, we believe that more work is needed on efficient concurrent oblivious transfer protocols in the standard model.

1.2 Our Result

In this paper, we present a concurrently-secure oblivious transfer protocol secure under SPS security. Our protocol does not require any setup and does not assume any independence among the inputs. In addition, our protocol is efficient since it does not use any inefficient primitives such as general zero-knowledge proofs for all NP statements. To the best of our knowledge, our protocol is the first concurrent oblivious transfer protocol that achieves both of these properties simultaneously. The security of our protocol is based on the decisional Diffie-Hellman (DDH) assumption.

Our Technique. Here, we give a brief overview of our protocol.

We construct our protocol and prove its security in the *UC-SPS framework* [12,29]. The UC-SPS framework is the same as the UC framework [3] except that in the UC-SPS framework we allow the simulator to run in super-polynomial time.

Our protocol is based on the UC oblivious transfer of [27], which is secure in the CRS model. In the protocol of [27], the CRS is $(g_0, h_0, g_1, h_1) \in \mathbb{G}^4$ for cyclic group $\mathbb{G}$. The protocol of [27] has the following properties.

- If (g_0, h_0, g_1, h_1) is a non-DDH tuple, the sender can break the receiver's security with trapdoor $(\log_{g_0} h_0, \log_{g_1} h_1)$.
- If (g_0, h_0, g_1, h_1) is a DDH tuple, the receiver can break the sender's security with trapdoor $\log_{g_0} g_1$.

In [27], the simulator is constructed using these two properties.

In our protocol, the receiver chooses group $\mathbb{G}$ and its elements $g_0, h_0, g_1 \in \mathbb{G}$. Then, the sender and the receiver execute a coin-toss protocol and generate a random element $h_1 \in \mathbb{G}$. Finally, the sender and receiver execute the oblivious transfer protocol of [27] using (g_0, h_0, g_1, h_1) as the CRS. We note that, because of the security of the coin-toss protocol, (g_0, h_0, g_1, h_1) is a non-DDH tuple with overwhelming probability. Then, our protocol has the following properties.

– Super-polynomial-time senders can break the receiver's security by computing trapdoor $(\log_{g_0} h_0, \log_{g_1} h_1)$ in super-polynomial time.
– Super-polynomial-time receivers can let (g_0, h_0, g_1, h_1) be a DDH tuple by cheating in the coin-toss protocol in super-polynomial time. Then, the receivers can break the sender's security with trapdoor $\log_{g_0} g_1$.

Then, we construct a simulator using these two properties.

Although the idea of our protocol is relatively simple, the security proof is not so simple. The reason is that the simulator runs in super-polynomial time whereas we assume only an assumption for polynomial-time adversaries. Therefore, we cannot use simple reduction to prove the indistinguishability between the real-world execution (which runs in polynomial time) and the ideal-world execution (which runs in super-polynomial time). To overcome this problem, we use the technique of [12]. The idea is that we define a hybrid execution in which we use rewinding instead of the super-polynomial power. Then, since the running time of the hybrid execution is polynomial time, we can use the DDH assumption to prove the indistinguishability between the real execution and the hybrid execution. In contrast, since we can show the indistinguishability between the hybrid execution and the ideal execution without any computational assumption, the super-polynomial-time simulator does not cause any problem.

2 Preliminaries

2.1 Notations

Let $\mathbb{N}$ denote the set of all positive integers. For any $q \in \mathbb{N}$, let $\mathbb{Z}_q$ denote the set $\{0, \ldots, q-1\}$. For any set X, let $x \xleftarrow{\mathsf{U}} X$ denote that x is an element of X chosen uniformly at random. For any random variable X, let $x \xleftarrow{\mathsf{R}} X$ denote that x is a value chosen at random according to the probability distribution of X. For any randomized algorithm Algo, let $\mathsf{Algo}(x)$ denote a random variable for the output of Algo on input x with a uniformly-chosen random tape. For any random variable X, let $\mathsf{Algo}(X)$ denote a random variable for the output of Algo on input $x \xleftarrow{\mathsf{R}} X$ with a uniformly-chosen random tape.

Let λ denote a security parameter. Let $\epsilon(\lambda)$ denote an arbitrary negligible function in λ. That is, for any constant $c > 0$, there exists $N \in \mathbb{N}$ such that for any $n > N$ we have $\epsilon(n) < 1/n^c$. For any probability ensembles $\mathcal{X} = \{X_k\}_{k \in \mathbb{N}}$ and $\mathcal{Y} = \{Y_k\}_{k \in \mathbb{N}}$, let $\mathcal{X} \overset{\mathsf{c}}{\approx} \mathcal{Y}$ denote that $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable. That is, we have $\mathcal{X} \overset{\mathsf{c}}{\approx} \mathcal{Y}$ if and only if for any probabilistic polynomial-time distinguisher $\mathcal{D}$ we have

$$|\Pr[\mathcal{D}(X_\lambda) = 1] - \Pr[\mathcal{D}(Y_\lambda) = 1]| < \epsilon(\lambda)$$

for a sufficiently large λ.

2.2 The Assumption

In this paper, we use the DDH assumption. Let GenG be a probabilistic polynomial-time algorithm that, on input 1^λ, outputs a description of cyclic group $\mathbb{G}$, its order q, and generator $g \in \mathbb{G}$. Then, the DDH assumption on GenG is defined as follows.

Definition 1 (DDH assumption). *We say that the DDH assumption holds on* GenG *if for any probabilistic polynomial-time algorithm* $\mathcal{A}$ *we have*

$$\left| \Pr\left[\mathcal{A}(\mathbb{G},q,g,g^x,g^y,g^{xy})=1 \,\middle|\, \begin{array}{l}(\mathbb{G},q,g) \xleftarrow{\mathsf{R}} \mathsf{GenG}(1^\lambda);\\ x,y \xleftarrow{\mathsf{U}} \mathbb{Z}_q;\end{array}\right] - \Pr\left[\mathcal{A}(\mathbb{G},q,g,g^x,g^y,g^z)=1 \,\middle|\, \begin{array}{l}(\mathbb{G},q,g) \xleftarrow{\mathsf{R}} \mathsf{GenG}(1^\lambda);\\ x,y,z \xleftarrow{\mathsf{U}} \mathbb{Z}_q;\end{array}\right] \right| < \epsilon(\lambda).$$

2.3 UC Framework

In this section, we briefly review the UC framework. For full details, see [3].

The model for protocol execution consists of *environment* $\mathcal{Z}$, *adversary* $\mathcal{A}$, and the parties running protocol π. In the execution of the protocol, the environment $\mathcal{Z}$ is first invoked on external input z. Environment $\mathcal{Z}$ adaptively gives inputs to the parties and receives outputs from them. In addition, $\mathcal{Z}$ communicates freely with $\mathcal{A}$ throughout the execution of the protocol. On inputs from $\mathcal{Z}$, the parties execute π by sending messages to each other. Adversary $\mathcal{A}$ sees all communications between the parties and controls the schedule of the communications. In addition, adversary $\mathcal{A}$ can *corrupt* some parties. After corruption, $\mathcal{A}$ receives all internal information of the corrupted parties. Moreover, from now on, $\mathcal{A}$ can fully control the corrupted parties. In this paper, we assume that there exist authenticated communication channels[6]. Thus, the adversary cannot change the contents of messages sent by the honest parties. In addition, in this paper we consider only static adversaries. In other words, we assume that the adversary corrupts parties only at the beginning of the protocol execution. The protocol execution ends when $\mathcal{Z}$ outputs a bit. Let $\mathsf{Exec}_{\pi,\mathcal{A},\mathcal{Z}}(\lambda,z)$ denote a random variable for the output of $\mathcal{Z}$ on security parameter $\lambda \in \mathbb{N}$ and input $z \in \{0,1\}^*$ with a uniformly-chosen random tape. Let $\mathsf{Exec}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\mathsf{Exec}_{\pi,\mathcal{A},\mathcal{Z}}(\lambda,z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$.

The security of protocol π is defined using the *ideal protocol*. In the execution of the ideal protocol, all parties simply hand their inputs to *ideal functionality* $\mathcal{F}$. Ideal functionality $\mathcal{F}$ carries out the desired task securely and gives outputs to the parties. The parties simply forward these outputs to $\mathcal{Z}$. Let *dummy parties* denote the parties in the ideal protocol. Adversary $\mathcal{S}$ in the execution of the ideal protocol is often called the *simulator*. Let $\pi(\mathcal{F})$ denote the ideal protocol for functionality $\mathcal{F}$. Let $\mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the ensemble $\mathsf{Exec}_{\pi(\mathcal{F}),\mathcal{S},\mathcal{Z}}$.

[6] This is not essential since authentication can be realized by a protocol, given a standard authentication infrastructure [4].

Then, the security of π is defined by comparing the execution of π (referred to as the *real world*) and the execution of $\pi(\mathcal{F})$ (referred to as the *ideal world*).

Definition 2 (UC-realization). *Let π be a protocol and $\mathcal{F}$ be an ideal functionality. We say that π **UC-realizes** $\mathcal{F}$ if for any adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that for any environment $\mathcal{Z}$ we have*

$$\mathsf{Exec}_{\pi,\mathcal{A},\mathcal{Z}} \stackrel{c}{\approx} \mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \ .$$

2.4 UC-SPS Framework

The UC-SPS framework [2, 12, 29] is the same as the UC framework except that we allow the simulator to run in super-polynomial time. The running time of the other machines is implicitly assumed to be polynomial time.

The UC realization is generalized naturally to the UC-SPS framework as follows.

Definition 3 (UC-SPS-realization). *Let π be a protocol and $\mathcal{F}$ be an ideal functionality. We say that π **UC-SPS-realizes** $\mathcal{F}$ if for any adversary $\mathcal{A}$ there exists a super-polynomial-time simulator $\mathcal{S}$ such that for any environment $\mathcal{Z}$ we have*

$$\mathsf{Exec}_{\pi,\mathcal{A},\mathcal{Z}} \stackrel{c}{\approx} \mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \ .$$

We note that, in general, the UC theorem [3] does not hold in the UC-SPS framework. Thus, stand-alone security in the UC-SPS framework does not imply concurrent security.

3 Concurrent Oblivious Transfer Protocol

In this section, we construct a concurrently-secure oblivious transfer protocol in the UC-SPS framework and prove its security.

As noted in Section 2.4, we cannot use the UC theorem in the UC-SPS framework to prove concurrent security. We therefore prove concurrent security by defining the concurrent oblivious transfer ideal functionality $\mathcal{F}_{\mathsf{cOT}}$ and proving that our protocol UC-SPS-realizes $\mathcal{F}_{\mathsf{cOT}}$. The concurrent oblivious transfer ideal functionality $\mathcal{F}_{\mathsf{cOT}}$, which is based on the (stand-alone) oblivious transfer ideal functionality [5], is shown in Fig. 1 [7]. Functionality $\mathcal{F}_{\mathsf{cOT}}$ captures concurrent security since, with a single run of $\mathcal{F}_{\mathsf{cOT}}$, the sender can send several values to the receiver. Thus, by constructing a protocol that UC-SPS-realizes $\mathcal{F}_{\mathsf{cOT}}$, we obtain a concurrent oblivious transfer protocol. Here, *ssid* in $\mathcal{F}_{\mathsf{cOT}}$ is the *subsession ID*, which is used to distinguish among the different subsessions that take place within a single run of $\mathcal{F}_{\mathsf{cOT}}$. We note that $\mathcal{F}_{\mathsf{cOT}}$ is different from the

[7] We say that functionality $\mathcal{F}$ generates *delayed output* v to party P if $\mathcal{F}$ first sends to $\mathcal{S}$ a note that it is ready to generate an output to P and, after $\mathcal{S}$ replies to the note, $\mathcal{F}$ sends v to P. If the output is *private*, then v is not mentioned in this note [3].

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

$\mathcal{F}_{\mathsf{cOT}}$ proceeds as follows, running with sender P_S, receiver P_R, and simulator $\mathcal{S}$.

- Upon receiving input $(\mathsf{Send}, sid, ssid, m_0, m_1)$ from P_S, if message $(\mathsf{Send}, sid, ssid, \ldots)$ was previously received, then do nothing. Else if $sid = (P_S, P_R, sid')$ for some sid' and P_R, then record $(ssid, m_0, m_1)$ and send $(\mathsf{Input}_S, sid, ssid)$ to $\mathcal{S}$. Furthermore, if a value $(ssid, \sigma)$ is recorded, then generate private delayed output $(\mathsf{Output}, sid, ssid, m_\sigma)$ to P_R.
- Upon receiving input $(\mathsf{Receive}, sid, ssid, \sigma)$ for $\sigma \in \{0, 1\}$ from P_R, if message $(\mathsf{Receive}, sid, ssid, \ldots)$ was previously received, then do nothing. Else if $sid = (P_S, P_R, sid')$ for some sid' and P_S, then record $(ssid, \sigma)$ and send $(\mathsf{Input}_R, sid, ssid)$ to $\mathcal{S}$. Furthermore, if a value $(ssid, m_0, m_1)$ is recorded, then generate private delayed output $(\mathsf{Output}, sid, ssid, m_\sigma)$ to P_R.

Fig. 1. The concurrent oblivious transfer functionality $\mathcal{F}_{\mathsf{cOT}}$.

multi-session oblivious transfer functionality $\hat{\mathcal{F}}_{\mathsf{OT}}$ [7, 27], with which any party can concurrently send messages to other parties. Unlike $\hat{\mathcal{F}}_{\mathsf{OT}}$, here only a fixed party P_S can interact with $\mathcal{F}_{\mathsf{cOT}}$ as a sender[8], and as a result $\mathcal{F}_{\mathsf{cOT}}$ does not capture any kind of non-malleability.

3.1 Protocols

First, we show a challenge-response based extractable commitment scheme $\langle C, R\rangle$, and then we show our concurrent oblivious transfer protocol Π_{OT}, which uses $\langle C, R\rangle$ as a primitive.

Extractable Commitment Scheme $\langle C, R\rangle$. Let Com be a non-interactive perfectly-binding commitment scheme[9]. Then the extractable commitment scheme $\langle C, R\rangle$, which is used in literature such as [12, 26, 28], is defined as follows.

Commit Phase. Sender C commits to element a of group $\mathbb{G}$ for receiver R as follows.

(1) $C \Rightarrow R$: For each $i \in \{1, 2, \ldots, k = \omega(\log \lambda)\}$, C chooses $\alpha_i \xleftarrow{\mathsf{U}} \mathbb{G}$ and computes $A_i^{(0)} \xleftarrow{\mathsf{R}} \mathsf{Com}(\alpha_i)$ and $A_i^{(1)} \xleftarrow{\mathsf{R}} \mathsf{Com}(a\alpha_i^{-1})$. Then C sends these $\{(A_i^{(0)}, A_i^{(1)})\}_{i=1}^k$ to R.
(2) $R \Rightarrow C$: Receiver R chooses $r_1, \ldots, r_k \xleftarrow{\mathsf{U}} \{0, 1\}$ and sends them to C.
(3) $C \Rightarrow R$: Sender C opens all of $\{A_i^{(r_i)}\}_{i=1}^k$ to R.

[8] In this paper, we define $\mathcal{F}_{\mathsf{cOT}}$ in such a way that only a single sender and a single receiver can interact with $\mathcal{F}_{\mathsf{cOT}}$. Our protocol remains secure even when we modify $\mathcal{F}_{\mathsf{cOT}}$ so that (a) a single sender and multiple receivers can interact with $\mathcal{F}_{\mathsf{cOT}}$ or (b) multiple senders and a single receiver can interact with $\mathcal{F}_{\mathsf{cOT}}$.

[9] We can construct an efficient non-interactive perfectly-binding commitment scheme under the DDH assumption using ElGamal encryption.

Open Phase. Sender C sends a, and opens all of $\{(A_i^{(0)}, A_i^{(1)})\}_{i=1}^k$ to R.

It is known that $\langle C, R\rangle$ is a perfectly-binding commitment scheme [26].

Concurrent Oblivious Transfer Protocol Π_{OT}. Our concurrent oblivious transfer protocol Π_{OT} is described below. Here, we use algorithm GenG described in Section 2.2.

When the input of the sender is $(\mathsf{Send}, sid, ssid, m_0, m_1)$ and the input of the receiver is $(\mathsf{Receive}, sid, ssid, \sigma)$, sender P_S and receiver P_R do the following. (For simplicity, we assume $m_0, m_1 \in \{0,1\}$. It is easy to modify our protocol so that the sender can send any $m_0, m_1 \in \{0,1\}^{O(\log \lambda)}$. In addition, if there is an efficiently-decodable encoding scheme from $\{0,1\}^\lambda$ to $\mathbb{G}$ for any $\mathbb{G} \xleftarrow{\mathsf{R}} \mathsf{GenG}(1^\lambda)$, the sender can send any $m_0, m_1 \in \{0,1\}^\lambda$.)

(1) $P_R \Rightarrow P_S$: Receiver P_R computes $(\mathbb{G}, q, g_0) \xleftarrow{\mathsf{R}} \mathsf{GenG}(1^\lambda)$. Next, P_R chooses $x, y \xleftarrow{\mathsf{U}} \mathbb{Z}_q$ and sets $h_0 := g_0^x$, $g_1 := g_0^y$. Then P_R sends $(sid, ssid, \mathbb{G}, q, g_0, h_0, g_1)$ to P_S.

(2) $P_S \Leftrightarrow P_R$: Sender P_S chooses $a \xleftarrow{\mathsf{U}} \mathbb{G}$. Then P_S commits to a for P_R using $\langle C, R\rangle$. In other words, P_S and P_R do the following.

  (2.1) $P_S \Rightarrow P_R$: For each $i \in \{1, 2, \ldots, k = \omega(\log \lambda)\}$, P_S chooses $\alpha_i \xleftarrow{\mathsf{U}} \mathbb{G}$ and computes $A_i^{(0)} \xleftarrow{\mathsf{R}} \mathsf{Com}(\alpha_i)$ and $A_i^{(1)} \xleftarrow{\mathsf{R}} \mathsf{Com}(a\alpha_i^{-1})$. Then P_S sends $(sid, ssid, (A_1^{(0)}, A_1^{(1)}), \ldots, (A_k^{(0)}, A_k^{(1)}))$ to P_R.

  (2.2) $P_R \Rightarrow P_S$: Receiver P_R chooses $r_1, \ldots, r_k \xleftarrow{\mathsf{U}} \{0,1\}$ and sends $(sid, ssid, r_1, \ldots, r_k)$ to P_S.

  (2.3) $P_S \Rightarrow P_R$: Sender P_S opens all of $\{A_i^{(r_i)}\}_{i=1}^k$ to P_R. If P_S fails to open one of these commitments, P_R aborts the protocol.

(3) $P_R \Rightarrow P_S$: Receiver P_R chooses $b \xleftarrow{\mathsf{U}} \mathbb{G}$ and sends $(sid, ssid, b)$ to P_S.

(4) $P_S \Rightarrow P_R$: Sender P_S opens the commitment of $\langle C, R\rangle$ in step (2). If P_S fails to open the commitment, P_R aborts the protocol.

(5) P_S and P_R set $h_1 := ab$.

(6) $P_R \Rightarrow P_S$: Receiver P_R chooses $r \xleftarrow{\mathsf{U}} \mathbb{Z}_q$ and sets $g := g_\sigma^r, h := h_\sigma^r$. Then P_R sends $(sid, ssid, g, h)$ to P_S.

(7) $P_S \Rightarrow P_R$: For each $i \in \{0,1\}$, sender P_S chooses $s_i, t_i \xleftarrow{\mathsf{U}} \mathbb{Z}_q$, sets $(u_i, v_i) := (g_i^{s_i} h_i^{t_i}, g^{s_i} h^{t_i})$, and sets $c_i := (u_i, v_i g_0^{m_i})$. Then, P_S sends $(sid, ssid, c_0, c_1)$ to P_R.

(8) Receiver P_R parses c_σ as $(c_{\sigma,0}, c_{\sigma,1})$. Next, P_R sets $\tilde{m}_\sigma := 1$ if $c_{\sigma,1}/c_{\sigma,0}^r = g_0$ and sets $\tilde{m}_\sigma := 0$ otherwise. Then, P_R outputs $(\mathsf{Output}, sid, ssid, \tilde{m}_\sigma)$.

3.2 Security Proof

In this section, we prove the following theorem.

Theorem 4. *Assume that the DDH assumption holds. Then, protocol Π_{OT} UC-SPS-realizes $\mathcal{F}_{cOT}$.*

Proof. We need to show that for any adversary $\mathcal{A}$ there exists a super-polynomial-time simulator $\mathcal{S}$ such that for any environment $\mathcal{Z}$ we have

$$\mathsf{Exec}_{\Pi_{\mathsf{OT}},\mathcal{A},\mathcal{Z}} \overset{\mathsf{c}}{\approx} \mathsf{Ideal}_{\mathcal{F}_{\mathsf{cOT}},\mathcal{S},\mathcal{Z}} \ . \tag{1}$$

In the real world, the sender sends several values concurrently using Π_{OT}. The schedule of the message delivery is determined by adversary $\mathcal{A}$. In the ideal world, the sender sends several values using $\mathcal{F}_{\mathsf{cOT}}$. A single run of $\mathcal{F}_{\mathsf{cOT}}$ consists of several subsessions, where a single value is sent in each subsession.

First, we show the description of simulator $\mathcal{S}$ for adversary $\mathcal{A}$. Simulator $\mathcal{S}$ internally invokes $\mathcal{A}$ and forwards every message from $\mathcal{Z}$ to the internal $\mathcal{A}$. For each message that internal $\mathcal{A}$ outputs to $\mathcal{Z}$, simulator $\mathcal{S}$ simply forwards it to external $\mathcal{Z}$. Furthermore, $\mathcal{S}$ internally simulates a real world with $\mathcal{A}$ as follows.

Case 1. Corrupted P_S and Honest P_R. Since internal $\mathcal{A}$ behaves as the sender on behalf of corrupted P_S, simulator $\mathcal{S}$ needs to interact with $\mathcal{A}$ as the receiver. In addition, $\mathcal{S}$ needs to extract both of the sender's values and send them to $\mathcal{F}_{\mathsf{cOT}}$. Toward this, $\mathcal{S}$ does the following for each subsession.

- Simulator $\mathcal{S}$ starts the subsession in the same way as honest P_R does. That is, $\mathcal{S}$ computes $(\mathbb{G}, q, g_0) \overset{\mathsf{R}}{\leftarrow} \mathsf{GenG}(1^\lambda)$, chooses $x, y \overset{\mathsf{U}}{\leftarrow} \mathbb{Z}_q$, sets $h_0 := g_0^x$, $g_1 := g_0^y$, and sends $(\mathbb{G}, q, g_0, h_0, g_1)$ to internal $\mathcal{A}$.
- Upon receiving $\{(A_i^{(0)}, A_i^{(1)})\}_{i=1}^k$ from $\mathcal{A}$, simulator $\mathcal{S}$ chooses $r'_1, \ldots, r'_k \overset{\mathsf{U}}{\leftarrow} \{0,1\}$ and extracts the committed values of $\{A_i^{(r'_i)}\}_{i=1}^k$ by breaking the hiding property of Com in super-polynomial time. Then, $\mathcal{S}$ chooses $r_1, \ldots, r_k \overset{\mathsf{U}}{\leftarrow} \{0,1\}$ and sends them to $\mathcal{A}$ in the same way as honest P_R does.
- If $\mathcal{A}$ opens the commitments of Com correctly in response to challenge $r_1, \ldots, r_k$, simulator $\mathcal{S}$ extracts committed value a of $\langle C, R\rangle$ by combining these opened values with the above extracted values[10]. Then $\mathcal{S}$ sends $b := a^{-1} g_0^{xy}$ to $\mathcal{A}$. Here, if $\mathcal{S}$ finds out that commitment $\{(A_i^{(0)}, A_i^{(1)})\}_{i=1}^k$ of $\langle C, R\rangle$ is invalid when $\mathcal{S}$ tries to extract a, simulator $\mathcal{S}$ sends $b \overset{\mathsf{U}}{\leftarrow} \mathbb{G}$ instead.
- When $\mathcal{A}$ opens the commitment of $\langle C, R\rangle$, simulator $\mathcal{S}$ verifies its validity in the same way as honest P_R does.
- Simulator $\mathcal{S}$ chooses $r \overset{\mathsf{U}}{\leftarrow} \mathbb{Z}_q$, sets $(g, h) := (g_1^r, h_1^r)$, and sends (g, h) to $\mathcal{A}$.
- Upon receiving $(c_0, c_1) = ((c_{0,0}, c_{0,1}), (c_{1,0}, c_{1,1}))$ from $\mathcal{A}$, simulator $\mathcal{S}$ sets $\tilde{m}_i := 1$ for each $i \in \{0,1\}$ if $c_{i,1}/c_{i,0}^{ry^{1-i}} = g_0$ and sets $\tilde{m}_i := 0$ otherwise. Then, simulator $\mathcal{S}$ sends $(\mathsf{Send}, sid, ssid, \tilde{m}_0, \tilde{m}_1)$ to $\mathcal{F}_{\mathsf{cOT}}$.

In summary, $\mathcal{S}$ extracts committed value a of $\langle C, R\rangle$ in super-polynomial time and sets $b := a^{-1} g_0^{xy}$. This will let $h_1 := ab = g_0^{xy}$. Then, we have

$$(g, h) = (g_1^r, h_1^r) = (g_0^{ry}, h_0^{ry}) \ .$$

Simulator $\mathcal{S}$ sets $\tilde{m}_i := 1$ for each $i \in \{0,1\}$ if $c_{i,1}/c_{i,0}^{ry^{1-i}} = g_0$ and sets $\tilde{m}_i := 0$ otherwise. Then, $\mathcal{S}$ sends $(\tilde{m}_0, \tilde{m}_1)$ to $\mathcal{F}_{\mathsf{cOT}}$.

[10] Since the probability that $(r_1, \ldots, r_k) = (r'_1, \ldots, r'_k)$ holds is negligible, we simply assume $(r_1, \ldots, r_k) \neq (r'_1, \ldots, r'_k)$ in what follows.

Case 2. Honest P_S and Corrupted P_R. Since internal $\mathcal{A}$ behaves as the receiver on behalf of corrupted P_R, simulator $\mathcal{S}$ needs to communicate with $\mathcal{A}$ as the sender knowing only one of the two values that honest P_S sent to $\mathcal{F}_{\mathsf{cOT}}$. Toward this, $\mathcal{S}$ does the following.

Upon receiving $(\mathsf{Receive}, sid, ssid, \sigma)$ from $\mathcal{Z}$ and $(\mathsf{Input}_S, sid, ssid)$ from $\mathcal{F}_{\mathsf{cOT}}$, simulator $\mathcal{S}$ interacts with $\mathcal{A}$ as the honest sender from step (1) to step (6). Upon receiving (g, h) from $\mathcal{A}$, simulator $\mathcal{S}$ checks whether or not (g_0, h_0, g, h) is a DDH tuple in super-polynomial time. Next, $\mathcal{S}$ sets $\tilde{\sigma} := 0$ if (g_0, h_0, g, h) is a DDH tuple and sets $\tilde{\sigma} := 1$ otherwise. Then, $\mathcal{S}$ sends $(\mathsf{Receive}, sid, ssid, \tilde{\sigma})$ to $\mathcal{F}_{\mathsf{cOT}}$. Upon receiving $(\mathsf{Output}, sid, ssid, m)$ from $\mathcal{F}_{\mathsf{cOT}}$, simulator $\mathcal{S}$ carries out step (7) by letting $m_{\tilde{\sigma}} := m$ and $m_{1-\tilde{\sigma}} \xleftarrow{\mathsf{U}} \{0,1\}$.

Case 3. Honest P_S and Honest P_R. Simulator $\mathcal{S}$ interacts with $\mathcal{A}$ both as the sender and as the receiver. As the sender, $\mathcal{S}$ behaves honestly with input $(m_0 = 0, m_1 = 0)$. As the receiver, $\mathcal{S}$ behaves honestly with input $\sigma = 0$.

Next, we show that, if the above simulator $\mathcal{S}$ is used, we have (1) for each case.

Analysis of Case 1. We need to show that for any probabilistic polynomial-time distinguisher $\mathcal{D}$ and any polynomial p, we have

$$|\Pr[\mathcal{D}(\mathsf{Exec}_{\Pi_{\mathsf{OT}},\mathcal{A},\mathcal{Z}}(\lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{Ideal}_{\mathcal{F}_{\mathsf{cOT}},\mathcal{S},\mathcal{Z}}(\lambda)) = 1]| < \frac{1}{p(\lambda)} \ , \quad (2)$$

for a sufficiently large λ.

Let ℓ be an upper bound of the number of subsessions and let $\delta(\lambda) := 3\ell \cdot p(\lambda)$. We define the indices of the subsessions based on the order in which the messages of step (2.2) appear in the interaction between P_S and P_R. That is, the message of step (2.2) of subsession 1 appears before the message of step (2.2) of subsession 2, and the message of step (2.2) of subsession 2 appears before the message of step (2.2) of subsession 3, and so on.

To prove that we have (2), we use a hybrid argument by defining machines $B_0, \ldots, B_{2\ell+1}$. First, we describe the idea behind our argument. In the ideal world, simulator $\mathcal{S}$ extracts committed value a of $\langle C, R\rangle$ in step (2) of each subsession. Let us call this committed value a the *trapdoor secret* of each subsession. Now, machine B_0 internally executes the real-world protocol and machine $B_{2\ell+1}$ internally executes the ideal-world protocol. In the sequence of hybrid machines, we change B_0 into $B_{2\ell+1}$ step by step by increasing the number of subsessions of which the trapdoor secrets are extracted. That is, we will define $B_{2(i-1)}$ $(i = 1, \ldots, \ell)$ so that the trapdoor secrets of subsession j $(j = 1, \ldots, i-1)$ are extracted and used as in the ideal world. Then, we will define B_{2i-1} by modifying $B_{2(i-1)}$ in such a way that the trapdoor secret of subsession i is also extracted (but not used). Each hybrid machine records these extracted trapdoor secrets in a list, $\mathsf{a\text{-}List}$. We note that the hybrid machines, except $B_{2\ell+1}$, are designed so that they do not use their super-polynomial power to extract the

trapdoor secrets[11]. Instead, they use polynomial-time rewinding and extract the trapdoor secrets using the extractability of $\langle C, R \rangle$[12].

Now, let us define hybrid machines $B_0, \ldots, B_{2\ell+1}$. First, we introduce some notations. The hybrid machines, except $B_{2\ell+1}$, internally execute the real-world protocol repeatedly with different randomness. That is, they internally invoke machines such as $\mathcal{Z}$ and $\mathcal{A}$, execute the protocol, rewind all the machines, execute the protocol again, rewind all the machines again, and so on. We let a *thread* denote a single execution of the protocol. A thread begins when internal $\mathcal{Z}$ is invoked, and the thread ends when internal $\mathcal{Z}$ outputs a bit. Each hybrid machine outputs whatever internal $\mathcal{Z}$ outputs in the last thread. Let us call this last thread the *main thread* of each hybrid machine.

Machine B_0. As its main thread, machine B_0 internally executes the real-world protocol by internally invoking $\mathcal{Z}$, $\mathcal{A}$, P_S, and P_R. Machine B_0 simply outputs whatever the internal $\mathcal{Z}$ outputs.

Machine B_{2i-1} *(*$i = 1, \ldots, \ell$*).* First, B_{2i-1} runs in the same way as $B_{2(i-1)}$, but B_{2i-1} does not output (and does not halt) even after the main thread of $B_{2(i-1)}$ ends. At the time, the trapdoor secret of subsession j ($j = 1, \ldots, i-1$) on the main thread of $B_{2(i-1)}$ is extracted and recorded in the $\mathsf{a\text{-}List}$. After the main thread of $B_{2(i-1)}$, machine B_{2i-1} rewinds this main thread[13] and executes it δ times with the same randomness except in step (2.2) of subsession i. Let us call these δ threads the *look-ahead threads*. In each look-ahead thread, challenge $r_1, \ldots, r_k$ in step (2.2) of subsession i is chosen fleshly.

In the case that $\mathcal{A}$ opens the commitments of Com correctly in step (2.3) of subsession i in the main thread of $B_{2(i-1)}$ and in at least one of the δ look-ahead threads, B_{2i-1} extracts trapdoor secret a of subsession i by combining the opened values of these two threads. Then, B_{2i-1} adds a pair (i, a) to the $\mathsf{a\text{-}List}$. If B_{2i-1} finds out that the commitment of $\langle C, R \rangle$ is invalid when it tries to extract a, then B_{2i-1} adds $(i, \perp)$ to the $\mathsf{a\text{-}List}$ instead.

In the case that $\mathcal{A}$ does not open the commitments of Com correctly in step (2.3) of subsession i in all δ look-ahead threads but opens them correctly in the main thread of $B_{2(i-1)}$, machine B_{2i-1} outputs $\perp$ and halts. Let us call this event $\mathsf{RewindAbort}_i$.

After all look-ahead threads end, if $\mathsf{RewindAbort}_i$ does not occur, B_{2i-1} executes the main thread of $B_{2(i-1)}$ once again with exactly the same randomness. This thread is the main thread of B_{2i-1}. The output of B_{2i-1} is whatever internal $\mathcal{Z}$ outputs in this thread.

Remark 5. We note that B_{2i-1} can execute each look-ahead thread without any problem such as recursive rewinding. To see this, observe that each look-ahead

[11] If hybrid machines are super-polynomial-time machines, it is difficult to show the indistinguishability between the outputs of hybrid machines based on assumptions for polynomial-time adversaries.

[12] The technique of replacing the super-polynomial power with the polynomial-time rewinding is used in [6, 12].

[13] That is, B_{2i-1} rewinds all the machines such as $\mathcal{Z}$ and $\mathcal{A}$.

thread proceeds in exactly the same way as the main thread of $B_{2(i-1)}$ until step (2.2) of subsession i, since the randomness used in this part are the same. In particular, the message of step (2.1) in subsession j ($j = 1, \ldots, i-1$) in each look-ahead thread is the same as the message in the main thread of $B_{2(i-1)}$. This means that the trapdoor secret of subsession j ($j = 1, \ldots, i-1$) in each look-ahead thread is the same as the trapdoor secret in the main thread of $B_{2(i-1)}$. Thus, the values that are extracted and recorded in the $\mathsf{a\text{-}List}$ before the rewinding are valid even after the rewinding. Therefore, B_{2i-1} can also use them in the look-ahead threads.

Machine B_{2i} ($i = 1, \ldots, \ell$). B_{2i} runs in the same way as B_{2i-1} except that, in step (3) of subsession i in the main thread, internal P_R sets $b := a^{-1} g_0^{xy}$ if (i, a) is recorded in the $\mathsf{a\text{-}List}$ for $a \neq \perp$. In the case of $a = \perp$, internal P_R sets $b \stackrel{\mathsf{U}}{\leftarrow} \mathbb{G}$ as in B_{2i-1}.

Machine $B_{2\ell+1}$. $B_{2\ell+1}$ internally executes the ideal-world protocol by internally invoking $\mathcal{Z}$, $\mathcal{S}$, the dummy party P_S and P_R. Machine $B_{2\ell+1}$ outputs whatever the internal $\mathcal{Z}$ outputs.

Next, we show the indistinguishability among the outputs of hybrid machines. Below, we let $\mathsf{Exec}_i(\lambda)$ denote the random variable for the output of machine B_i.

$B_{2(i-1)}$ and B_{2i-1} ($i = 1, \ldots, \ell$). If $\mathsf{RewindAbort}_i$ does not occur in B_{2i-1}, the output of $B_{2(i-1)}$ and the output of B_{2i-1} are identical since their main threads are the same. $\mathsf{RewindAbort}_i$ occurs in B_{2i-1} if $\mathcal{A}$ does not open the commitments in step (2.3) on subsession i in all δ look-ahead threads but opens them correctly in the main thread. Since $\mathcal{A}$ opens these commitments correctly in each look-ahead thread with the same probability as in the main thread, we can show that $\mathsf{RewindAbort}_i$ occurs in B_{2i-1} with probability at most $1/\delta$. Thus, for any probabilistic polynomial-time distinguisher $\mathcal{D}$, we have

$$\left|\Pr\left[\mathcal{D}(\mathsf{Exec}_{2(i-1)}(\lambda)) = 1\right] - \Pr\left[\mathcal{D}(\mathsf{Exec}_{2i-1}(\lambda)) = 1\right]\right| \leq \frac{1}{\delta(\lambda)} \ . \tag{3}$$

B_{2i-1} and B_{2i} ($i = 1, \ldots, \ell$). B_{2i} is the same as B_{2i-1} except that B_{2i} sets $b := a^{-1} g_0^{xy}$ instead of $b \stackrel{\mathsf{U}}{\leftarrow} \mathbb{G}$ in step (3) of subsession i on the main thread. Thus, from the DDH assumption, for any probabilistic polynomial-time distinguisher $\mathcal{D}$, we have

$$\left|\Pr\left[\mathcal{D}(\mathsf{Exec}_{2i-1}(\lambda)) = 1\right] - \Pr\left[\mathcal{D}(\mathsf{Exec}_{2i}(\lambda)) = 1\right]\right| < \epsilon(\lambda) \ . \tag{4}$$

$B_{2\ell}$ and $B_{2\ell+1}$. In $B_{2\ell}$, all the trapdoor secrets are extracted and used as in $B_{2\ell+1}$. Machine $B_{2\ell}$ uses rewinding to extract the trapdoor secrets, whereas machine $B_{2\ell+1}$ uses its super-polynomial power. In order to show the indistinguishability, it suffices to show that the receiver's outputs and the computed trapdoor secrets in $B_{2\ell}$ are the same as in $B_{2\ell+1}$.

First, we show the indistinguishability between $B_{2\ell}$ and $B_{2\ell+1}$ under the condition that $\mathsf{RewindAbort}_i$ does not occur in $B_{2\ell}$ for all i. In this case, in each subsession, trapdoor secret a that $B_{2\ell}$ records in the $\mathsf{a\text{-}List}$ and trapdoor secret a that $\mathcal{S}$ computes in $B_{2\ell+1}$ are identically distributed. To see this, observe that in both machines we can think that the trapdoor secret a is computed by combining two responses of $\langle C, R\rangle$ for two different challenges. In addition, since we have

$$(g, h) = (g_1^r, h_1^r) = (g_0^{ry}, h_0^{ry}) \qquad \text{(since we have } h_1 = g_0^{xy}) \ ,$$

the receiver outputs the same value in $B_{2\ell}$ and $B_{2\ell+1}$. Therefore, we conclude that the view of $\mathcal{Z}$ in the main threads of $B_{2\ell}$ and the view of $\mathcal{Z}$ in $B_{2\ell+1}$ are identical if $\mathsf{RewindAbort}_i$ does not occur in $B_{2\ell}$ for all i.

Next, we compute the probability that $\mathsf{RewindAbort}_i$ occurs in $B_{2\ell}$ for some i. From (3) and (4), we have

$$|\Pr[\mathcal{D}(\mathsf{Exec}_0(\lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{Exec}_{2\ell}(\lambda)) = 1]| \leq \frac{\ell}{\delta(\lambda)} + \epsilon(\lambda) \ , \tag{5}$$

for any probabilistic polynomial-time distinguisher $\mathcal{D}$. Since $\mathsf{RewindAbort}_i$ does not occur in B_0 for all i, we conclude that $\mathsf{RewindAbort}_i$ occurs in $B_{2\ell}$ for some i with probability at most $\ell/\delta(\lambda) + \epsilon(\lambda)$.

Combining the above, we conclude that for any probabilistic polynomial-time distinguisher $\mathcal{D}$ we have

$$|\Pr[\mathcal{D}(\mathsf{Exec}_{2\ell}(\lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{Exec}_{2\ell+1}(\lambda)) = 1]| \leq \frac{\ell}{\delta(\lambda)} + \epsilon(\lambda) \ . \tag{6}$$

Finishing the Analysis of Case 1. From (5) and (6), for any probabilistic polynomial-time distinguisher $\mathcal{D}$, we have

$$|\Pr[\mathcal{D}(\mathsf{Exec}_0(\lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{Exec}_{2\ell+1}(\lambda)) = 1]| \leq \frac{2\ell}{\delta(\lambda)} + \epsilon(\lambda) \ .$$

By substituting $\mathsf{Exec}_0(\lambda) = \mathsf{Exec}_{\Pi_{\mathsf{OT}},\mathcal{A},\mathcal{Z}}(\lambda)$, $\mathsf{Exec}_{2\ell+1}(\lambda) = \mathsf{Ideal}_{\mathcal{F}_{\mathsf{cOT}},\mathcal{S},\mathcal{Z}}(\lambda)$ and $\delta(\lambda) = 3\ell \cdot p(\lambda)$, we have (2).

Analysis of Case 2. In the real world, honest sender P_S interacts with $\mathcal{A}$ (via the corrupted receiver) using m_0 and m_1, which P_S received as an input from $\mathcal{Z}$. In the ideal world, simulator $\mathcal{S}$ interacts with internal $\mathcal{A}$ honestly using m_0 and m_1, where $m_{\tilde{\sigma}}$ is received from $\mathcal{F}_{\mathsf{cOT}}$ and $m_{1-\tilde{\sigma}}$ is chosen uniformly at random. Thus, in the view of $\mathcal{Z}$, the only possible difference between the real world and the ideal world is the value of $c_{1-\tilde{\sigma}} = (u_{1-\tilde{\sigma}}, v_{1-\tilde{\sigma}} g_0^{m_{1-\tilde{\sigma}}})$. In what follows, we let $\mu := 1 - \tilde{\sigma}$.

First, we show the indistinguishability under the condition that (g_0, h_0, g_1, h_1) is a non-DDH tuple in each subsession both in the real world and in the ideal world. In this case, at least one of (g_0, h_0, g, h) and (g_1, h_1, g, h) is also a non-DDH tuple in each subsession. From the definition of $\tilde{\sigma}$, this means that (g_μ, h_μ, g, h)

is a non-DDH tuple. That is, there exist $\alpha, \beta, \gamma \in \mathbb{Z}_q$ such that $(h_\mu, g, h) = (g_\mu^\alpha, g_\mu^\beta, g_\mu^\gamma)$ and $\alpha\beta \neq \gamma$. Using this, we can show that v_μ is uniformly random for $\mathcal{Z}$. To see this, observe that we have

$$\begin{aligned} u_\mu &= g_\mu^{s_\mu} h_\mu^{t_\mu} = g_\mu^{s_\mu + \alpha t_\mu} \quad , \\ v_\mu &= g^{s_\mu} h^{t_\mu} = g_\mu^{\beta s_\mu + \gamma t_\mu} \end{aligned}$$

for random s_μ and t_μ, and the expressions $s_\mu + \alpha t_\mu$ and $\beta s_\mu + \gamma t_\mu$ are linearly independent combinations of s_μ and t_μ when $\alpha\beta \neq \gamma$. Thus, the distribution of c_μ is independent of m_μ. Therefore, the view of $\mathcal{Z}$ is identically distributed in the real world and the ideal world.

Next, we compute the probability that (g_0, h_0, g_1, h_1) is a DDH tuple in some subsessions. Below, we show that this probability is negligible in the real world. In this case, since simulator $\mathcal{S}$ interacts with internal $\mathcal{A}$ honestly (with uniformly chosen m_μ) and the computation of (g_0, h_0, g_1, h_1) is independent of the message m_μ, we conclude that this probability is also negligible in the ideal world.

Then, we show that (g_0, h_0, g_1, h_1) is a DDH tuple with negligible probability in the real world. Let us consider the following experiment $\mathsf{Exp}_i^{\mathcal{B}}(\lambda)$ for the hiding property of $\langle C, R \rangle$. First, adversary $\mathcal{B}$ sends $(a_{0,0}, a_{0,1}, a_{1,0}, a_{1,1})$ to the challenger. Then, the challenger commits to $a_{i,0}$ and $a_{i,1}$ for $\mathcal{B}$ by invoking $\langle C, R \rangle$ sequentially. Finally, $\mathcal{B}$ outputs bit i', which is the output of $\mathsf{Exp}_i^{\mathcal{B}}(\lambda)$. Advantage $\mathsf{Adv}_{\mathcal{B}}(\lambda)$ of $\mathcal{B}$ is

$$\mathsf{Adv}_{\mathcal{B}}(\lambda) := \left| \Pr\left[\mathsf{Exp}_0^{\mathcal{B}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}_1^{\mathcal{B}}(\lambda) = 1\right] \right| \quad .$$

Using the hiding property of $\langle C, R \rangle$, we can show that we have $\mathsf{Adv}_{\mathcal{B}}(\lambda) < \epsilon(\lambda)$ for any $\mathcal{B}$. Below, we show that, if in the real world (g_0, h_0, g_1, h_1) is a DDH tuple in some subsession j^* with probability $1/\lambda^c$ for some constant $c > 0$, we can construct adversary $\mathcal{B}^*$ such that $\mathsf{Adv}_{\mathcal{B}^*}(\lambda)$ is non-negligible, which contradicts the hiding property of $\langle C, R \rangle$.

Adversary $\mathcal{B}^*$ chooses $j \xleftarrow{\mathsf{U}} \{1, \ldots, \ell\}$ (here, ℓ is the upper bound of the number of subsessions), and internally executes the real-world execution until step (2.1) of subsession j. Let $(sid, ssid_j, \mathbb{G}_j, q_j, g_{j,0}, h_{j,0}, g_{j,1})$ be the message of step (1) in subsession j. Then, $\mathcal{B}^*$ chooses $a_{0,0}, a_{0,1}, a_{1,0}, a_{1,1} \xleftarrow{\mathsf{U}} \mathbb{G}_j$ and sends them to the challenger. When the challenger starts $\langle C, R \rangle$, adversary $\mathcal{B}^*$ forwards it to the internal execution as step (2) of subsession j. We call this internal execution exec_0. Let $(sid, ssid_j, b_0)$ be the message of step (3) in subsession j of exec_0. Next, $\mathcal{B}^*$ rewinds exec_0 to step (2) of subsession j. Then, $\mathcal{B}^*$ receives the next commitment of $\langle C, R \rangle$ from the challenger and forwards it to the rewound internal execution as step (2) of subsession j. We call this second execution exec_1. Let $(sid, ssid_j, b_1)$ be the message of step (3) in subsession j of exec_1. Then, $\mathcal{B}^*$ outputs 1 if and only if $b_0/b_1 = a_{0,0}^{-1}/a_{0,1}^{-1}$ holds.

Let ρ be a partial transcript such that step (2) of subsession j^* begins immediately after ρ in the real execution. Then, from the average argument, it holds

that

$$\Pr\left[(g_0, h_0, g_1, h_1) \text{ is a DDH tuple in subsession } j^* \mid \rho \text{ occurs}\right] \geq \frac{1}{2\lambda^c}$$

with probability at least $1/2\lambda^c$ over the choice of ρ.

In $\mathcal{B}^*$, we have $j = j^*$ with probability $1/\ell$. In addition, in $\mathsf{Exp}_0^{\mathcal{B}^*}(\lambda)$, we have

$$\Pr\left[b_0 = a_{0,0}^{-1} g_{j,0}^{x_j y_j} \bigwedge b_1 = a_{0,1}^{-1} g_{j,0}^{x_j y_j} \middle| j = j^*\right] \geq \frac{1}{2\lambda^c} \cdot \left(\frac{1}{2\lambda^c}\right)^2 \quad ,$$

where $x_j := \log_{g_{j,0}} h_{j,0}$ and $y_j := \log_{g_{j,0}} g_{j,1}$. Thus, we have $\Pr\left[\mathsf{Exp}_0^{\mathcal{B}^*}(\lambda) = 1\right] \geq 1/(8\ell\lambda^{3c})$. On the other hand, since no information about $a_{0,0}$ and $a_{0,1}$ is fed into exec_0 and exec_1 in $\mathsf{Exp}_1^{\mathcal{B}^*}(\lambda)$, we have $\Pr\left[\mathsf{Exp}_1^{\mathcal{B}^*}(\lambda) = 1\right] \leq 1/|\mathbb{G}| < \epsilon(\lambda)$. Therefore, we have $\mathsf{Adv}_{\mathcal{B}^*}(\lambda) \geq 1/\mathsf{poly}(\lambda)$. Since this contradicts the hiding property of $\langle C, R\rangle$, we conclude that the probability that (g_0, h_0, g_1, h_1) is a DDH tuple in some subsession is negligible in the real world.

Combining the above, we conclude that (1) holds in Case 2.

Analysis of Case 3. First, the outputs of the honest receiver in the real world are the same as in the ideal world. This is because, in each subsession of the real world, the receiver outputs 1 if and only if it holds that

$$g_0 = \frac{c_{\sigma,1}}{c_{\sigma,0}^r} = \frac{v_\sigma g_0^{m_\sigma}}{u_\sigma^r} = \frac{g^{s_\sigma} h^{t_\sigma} g_0^{m_\sigma}}{(g_\sigma^{s_\sigma} h_\sigma^{t_\sigma})^r} = g_0^{m_\sigma} \quad .$$

Thus, to show the indistinguishability, it suffices to show that $\mathcal{Z}$ cannot tell whether it interacts with $\mathcal{A}$ in the real world or it interacts with the internal $\mathcal{A}$ (of $\mathcal{S}$) in the ideal world. Toward this, let us consider the following hybrid.

Hybrid H_0 is the same as the ideal world except that, in each subsession, simulator $\mathcal{S}$ uses honest parties' inputs m_0, m_1, and σ instead of 0. Note that the view of $\mathcal{Z}$ in H_0 is the same as in the real world.

Hybrid H_1 is the same as H_0 except that $\mathcal{S}$ sets $\sigma := 1$ in each subsession. The view of $\mathcal{Z}$ in H_1 is indistinguishable from the one in H_0 since, from the DDH assumption, $(g_0, h_0, g_1, h_1, g_0^r, h_0^r)$ and $(g_0, h_0, g_1, h_1, g_1^r, h_1^r)$ are indistinguishable for $\mathcal{Z}$.

Hybrid H_2 is the same as H_1 except that $\mathcal{S}$ sets $m_0 := 0$ in each subsession. The view of $\mathcal{Z}$ in H_2 is identical with the one in H_1 except with negligible probability since, from the same argument as in Case 2, the distribution of c_0 in each subsession is independent of the value of m_0 except with negligible probability.

Hybrid H_3 is the same as H_2 except that $\mathcal{S}$ sets $\sigma := 0$ in each subsession. From the same argument as in H_1, the view of $\mathcal{Z}$ in H_3 is indistinguishable from the one in H_2.

Hybrid H_4 is the same as H_3 except that $\mathcal{S}$ sets $m_1 := 0$ in each subsession. From the same argument as in H_2, the view of $\mathcal{Z}$ in H_4 is identical with the one in H_3 except with negligible probability.

Since H_4 is the same as the ideal world, it holds that the view of $\mathcal{Z}$ in the real world is indistinguishable from the one in the ideal world.

Combining the above, we conclude that (1) holds in Case 3.

Since we have (1) for all three cases, we conclude that protocol Π_{OT} UC-SPS-realizes $\mathcal{F}_{\mathsf{cOT}}$. □

4 Conclusion

This paper showed a concurrently-secure oblivious transfer protocol in the SPS security without any setup. Our protocol is efficient since it does not use any inefficient primitive such as general zero-knowledge proofs for all NP statements. Therefore, our protocol may be useful for practical purposes.

It should be noted that, unlike many previous studies on SPS security, we considered only concurrent security and do not considered other security notions such as *non-malleability* [10] and UC security. Thus, our protocol achieves somewhat restricted security. However, we believe that concurrent security is sufficient for various settings such as a network in which one party is a server and the others are clients. It would be interesting to improve our protocol so that non-malleability or the UC security is also guaranteed.

References

1. Aiello, W., Ishai, Y., Reingold, O.: Priced oblivious transfer: How to sell digital goods. In: Pfitzmann, B. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 2045, pp. 119–135. Springer (2001)
2. Barak, B., Sahai, A.: How to play almost any mental game over the net - concurrent composition via super-polynomial simulation. In: FOCS. pp. 543–552. IEEE Computer Society (2005)
3. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS. pp. 136–145. IEEE Computer Society (2001)
4. Canetti, R.: Universally composable signature, certification, and authentication. In: CSFW. pp. 219–233. IEEE Computer Society (2004)
5. Canetti, R., Fischlin, M.: Universally composable commitments. In: Kilian, J. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 2139, pp. 19–40. Springer (2001)
6. Canetti, R., Lin, H., Pass, R.: Adaptive hardness and composable security in the plain model from standard assumptions. In: FOCS. pp. 541–550. IEEE Computer Society (2010)
7. Canetti, R., Rabin, T.: Universal composition with joint state. In: Boneh, D. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 2729, pp. 265–281. Springer (2003)

8. Choi, S.G., Dachman-Soled, D., Malkin, T., Wee, H.: Simple, black-box constructions of adaptively secure protocols. In: Reingold, O. (ed.) TCC. Lecture Notes in Computer Science, vol. 5444, pp. 387–402. Springer (2009)
9. Damgård, I., Nielsen, J.B., Orlandi, C.: Essentially optimal universally composable oblivious transfer. In: Lee, P.J., Cheon, J.H. (eds.) ICISC. Lecture Notes in Computer Science, vol. 5461, pp. 318–335. Springer (2008)
10. Dolev, D., Dwork, C., Naor, M.: Nonmalleable cryptography. SIAM J. Comput. 30(2), 391–437 (2000)
11. Garay, J.A., MacKenzie, P.D.: Concurrent oblivious transfer. In: FOCS. pp. 314–324. IEEE Computer Society (2000)
12. Garg, S., Goyal, V., Jain, A., Sahai, A.: Concurrently secure computation in constant rounds. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT. Lecture Notes in Computer Science, vol. 7237, pp. 99–116. Springer (2012)
13. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A.V. (ed.) STOC. pp. 218–229. ACM (1987)
14. Green, M., Hohenberger, S.: Universally composable adaptive oblivious transfer. In: Pieprzyk, J. (ed.) ASIACRYPT. Lecture Notes in Computer Science, vol. 5350, pp. 179–197. Springer (2008)
15. Green, M., Hohenberger, S.: Practical adaptive oblivious transfer from simple assumptions. In: Ishai, Y. (ed.) TCC. Lecture Notes in Computer Science, vol. 6597, pp. 347–363. Springer (2011)
16. Haitner, I., Ishai, Y., Kushilevitz, E., Lindell, Y., Petrank, E.: Black-box constructions of protocols for secure computation. SIAM J. Comput. 40(2), 225–266 (2011)
17. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer - efficiently. In: Wagner, D. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 5157, pp. 572–591. Springer (2008)
18. Kilian, J.: Founding cryptography on oblivious transfer. In: Simon, J. (ed.) STOC. pp. 20–31. ACM (1988)
19. Kurosawa, K., Nojima, R., Phong, L.T.: Efficiency-improved fully simulatable adaptive OT under the DDH assumption. In: Garay, J.A., Prisco, R.D. (eds.) SCN. Lecture Notes in Computer Science, vol. 6280, pp. 172–181. Springer (2010)
20. Lindell, A.Y.: Efficient fully-simulatable oblivious transfer. In: Malkin, T. (ed.) CT-RSA. Lecture Notes in Computer Science, vol. 4964, pp. 52–70. Springer (2008)
21. Lindell, Y.: Lower bounds and impossibility results for concurrent self composition. J. Cryptology 21(2), 200–249 (2008)
22. Naor, M., Pinkas, B.: Efficient oblivious transfer protocols. In: Kosaraju, S.R. (ed.) SODA. pp. 448–457. ACM/SIAM (2001)
23. Naor, M., Pinkas, B.: Computationally secure oblivious transfer. J. Cryptology 18(1), 1–35 (2005)
24. Pass, R.: Simulation in quasi-polynomial time, and its application to protocol composition. In: Biham, E. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 2656, pp. 160–176. Springer (2003)
25. Pass, R., Venkitasubramaniam, M.: On constant-round concurrent zero-knowledge. In: Canetti, R. (ed.) TCC. Lecture Notes in Computer Science, vol. 4948, pp. 553–570. Springer (2008)
26. Pass, R., Wee, H.: Black-box constructions of two-party protocols from one-way functions. In: Reingold, O. (ed.) TCC. Lecture Notes in Computer Science, vol. 5444, pp. 403–418. Springer (2009)

27. Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: Wagner, D. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 5157, pp. 554–571. Springer (2008)
28. Prabhakaran, M., Rosen, A., Sahai, A.: Concurrent zero knowledge with logarithmic round-complexity. In: FOCS. pp. 366–375. IEEE Computer Society (2002)
29. Prabhakaran, M., Sahai, A.: New notions of security: achieving universal composability without trusted setup. In: Babai, L. (ed.) STOC. pp. 242–251. ACM (2004)
30. Rabin, M.O.: How to exchange secrets by oblivious transfer. Tech. rep., TR-81, Harvard Aiken Computation Laboratory (1981)