Founding Cryptography on Smooth Projective Hashing

Bing Zeng[a]

[a]School of Software Engineering, South China University of Technology, Guangzhou, 510006, China

Abstract

Oblivious transfer (OT) is a fundamental primitive in cryptography. Halevi-Kalai OT (Halevi, S. and Y. Kalai (2012), Journal of Cryptology 25(1)), which is based on smooth projective hash(SPH), is a famous and the most efficient framework for 1-out-of-2 oblivious transfer (OT_1^2) against malicious adversaries in plain model. However, it does not provide simulation-based security. Thus, it is harder to use it as a building block in secure multiparty computation (SMPC) protocols. A natural question however, which so far has not been answered, is whether it can be can be made fully-simulatable. In this paper, we give a positive answer. Further, we present a fully-simulatable framework for general OT_t^n ($n, t \in \mathbb{N}$ and $n > t$). Our framework can be interpreted as a constant-round blackbox reduction of OT_t^n (or OT_1^2) to SPH. To our knowledge, this is the first such reduction. Combining Kilian's famous completeness result, we immediately obtain a black-box reduction of SMPC to SPH.

Keywords: oblivious transfer, secure multiparty computation, malicious adversaries, smooth projective hashing.

1. Introduction

1.1. Secure Oblivious Transfer

Oblivious transfer (OT), introduced in [1], is a fundamental cryptographic primitive allowing the secure multiparty computation (SMPC) of *any* computable function [2]. Beside this completeness result by Kilian, OT exhibits interests on its own since this primitive is generally used as a building block in a variety of cryptographic protocols: secure computation of the median [3], privacy-preserving data publishing [4], privacy-preserving set operations [5], electronic commerce [6], private mutual authentication [7], privacy preserving data mining [8], database search [9, 10], oblivious keyword search [11], oblivious polynomial evaluation [12], contract signing [13].

t-out-of-n oblivious transfer (OT_t^n) deals with the scenario where a sender holds n private values $m_1, m_2, \ldots, m_n$ and a receiver possesses t private indexes $i_1, i_2, \ldots, i_t$. The receiver expects to get the values $m_{i_1}, m_{i_2}, \ldots, m_{i_t}$ without leaking any information about which ones were chosen. On the other hand, the sender does not want the receiver to know anything but the t values queried about.

In SMPC, two categories of adversaries are usually considered: *semi-honest* (also called *honest-but-curious*) and *malicious*. In both cases, the adversary's goal is to learn more information using the transcript of the computation than what is inferred by his private input and the result of the computation. However, there is a fundamental difference between these two models: a semi-honest adversary always executes the steps of the SMPC protocol faithfully while a malicious enemy can arbitrarily deviate from them [14]. Obviously, security against malicious adversaries (SAMA) is higher than security against a semi-honest adversary and is closer to the reality. In this paper, we focus on SAMA. Many OT protocols with SAMA are known, e.g., [15, 6, 16, 17, 18, 19, 10, 20, 21].

A standard way of proving the security of a SMPC protocol is to use the ideal/real model paradigm. In this context, adversaries in the real world are demonstrated to be equivalent to enemies in the ideal world where the computation is executed by an incorruptible entity named *ideal functionality*. As the SMPC protocol (in the real world) simulates the ideal world, the security is said to be *fully-simulatable*. [17, 18, 19, 10, 20, 21] are shown to be fully-simulatable.

Halevi-Kalai OT [16] is a remarkable work among known protocols, because it is the most efficient framework for OT_1^2 against malicious adversaries in plain model. Indeed, it is an abstraction of highly efficient protocols of Naor

Email address: zeng.bing.zb@gmail.com (Bing Zeng)

and Pinkas [15] and Aiello et al [6]. However, Halevi-Kalai OT is non-simulatable. Thus, it is harder to use it as a building block in secure multi-party computation protocols. A concrete attack on non-simulatable OT shown by [9] is selective-failure attack where the sender causes a failure depending on the receiver's selection. A natural question however, which so far has not been answered, is whether Halevi-Kalai OT can be made fully-simulatable.

1.2. Our Contribution

In the following subsections, we are to provide detailed explanations of our contribution. The main points can be summarized as follows.

1. We present a positive answer to the question. Specially, we make Halevi-Kalai OT_1^2 fully-simulatable, then extends it to general case OT_t^n. Thus, we present a generally realizable framework for OT_t^n with fully-simulatable SAMA in plain model.
2. As a theoretical contribution, our framework is a constant-round blackbox reduction of OT_t^n to SPH. To our knowledge, this is the first such reduction. Combining Kilian's completeness result [2], we immediately obtain a black-box reduction of SMPC to SPH.

We tress that OT_t^n has its own interesting when compare it with OT_1^2. First, there are many applications for OT_t^n itself, e.g., [12, 9, 22, 10]. Second, when an efficient OT_t^n protocol is needed in practice, it is unknown how to construct it from a known OT_1^2 protocol. Specifically, there is not an efficient reduction OT_t^n to OT_1^2 on SAMA level is known except a prohibitively expensive reduction employing zero-knowledge proofs for $\mathcal{NP}$ [23]. Although there are some reductions of weaker security, e.g. [9], they do not provide simulation-based security. This justifies directly constructing fully-simulatable SAMA protocols for OT_t^n.

1.2.1. Cryptographic Approach

The SPH variant used in [16] was called *verifiably smooth projective hash family*. It deals with two types of instances (smooth and projective) which are computationally indistinguishable due to a property called *hard subset membership*. Nonetheless, another property called *verifiable smoothness* provides a way to verify whether at least one of a two instances is smooth. In the remaining of this paper, we are to denote verifiably smooth projective hash family with hard membership property by VSPH-HM.

For each instance x of each type, there are two kinds of keys (hash keys and projection keys). In addition, every projective instance holds a witness while no smooth instance does so. A hash value is computed from a hash key (i.e., $\mathsf{Hash}(x, hk)$) while a projection value is computed from a projection key and a witness (i.e., $\mathsf{pHash}(x, pk, w)$). For a smooth instance, the projection value reveals almost no knowledge about the hash value due to a property called *smoothness*. However, for a projective instance, the projection value equals the hash value. This fact is guaranteed by another property called *projection*.

Despite the notion of VSPH-HM can be used to deal with OT_1^n, it seems difficult to extend it to handle the general case OT_t^n. The reason is that, to hold verifiable smoothness, both types of instances have to be generated in a dependent way. This makes it difficult to design an algorithm checking that at least t of n arbitrary instances are smooth without leaking any information to the adversaries which could be used to distinguish smooth instances from projective instances. Therefore, even constructing a non-simulatable protocol for OT_t^n as in [16] seems difficult.

Another problem comes from the fact that, for a protocol using a VSPH-HM, it is impossible to gain simulation-based security in the case where only the receiver is corrupted. The reason is that, to extract the adversary's real input in this case, the simulator has to identify the projective part of a smooth-projective instance pair. However, this is computationally impossible because of the hard subset membership assumption.

Seeing the above difficulties, we define a new variant of SPH called *smooth projective hash family with distinguishability and hard subset membership* (SPH-DHM). See [24] for its instantiations. The most essential difference between the notions of SPH-DHM and VSPH-HM is that a SPH-DHM also provides witnesses to the smooth instances while verifiable smoothness is removed. Furthermore, we introduce the *distinguishability* property providing a way to differentiate smooth instances from projective ones when needed witnesses are given. This enables a SPH-DHM to generate both types of instances independently. Thus, the notion of SPH-DHM is tailored to treat OT_t^n.

We would like to recall that, in [16], the receiver learns the value it queried about via a projective instance. For a smooth-projective instance pair, if the witness pair is available, the simulator can identify the projective instance, and

hence the simulator can learn which value the adversary chooses (i.e. it learns the adversary's real input). Employing a cut-and-choose technique as in [19, 25], the simulator can see the witnesses by rewinding the adversary's computation. Our idea is for the receiver to "cut" some instance vectors (where each one contains t projective instances and $n - t$ smooth instances) and for the sender to "choose" some instance vectors at random to check their *legalities* (i.e., the sender checks that each vector indeed contains at least $n - t$ smooth instances). The receiver then sends the chosen instance vectors' witnesses. Combining the previous analysis, we can see that for a protocol constructed following this idea, the simulator can extract the adversary's real input, and hence simulation-based security can be gained in the case where only the receiver is corrupted.

In the case where only the sender is corrupted. The authors of [16] argue that their protocol gives a simulation-based security. Though their simulation idea is intuitively right, we find it not true. The main problem is that the input extraction is not completed in one shot. See Section 3.2 for the details. To solve this problem, we let both parties commonly choose instance vectors to open via a coin-tossing protocol. This gives opportunities to the simulator to know the choices of malicious adversaries and to bias the common choices. Then the simulator can cheat malicious adversaries and extract their input in one shot.

In [26], Zeng *et al.* presented a framework for OT_t^n secure against covert adversaries whose design is close to the protocol presented in this paper. In [27], Aumann and Lindell proved that, for any protocol, if its deterrence factor to covert adversaries was $\epsilon = 1 - \mu(k)$ (where $\mu(\cdot)$ is a negligible function of the security parameter), then the protocol was also secure against malicious adversaries. With this result in mind, one might naturally wonder whether it would be possible to get such a good deterrence factor for [26] via a simple parameter setting to ensure [26]'s security against malicious adversaries. Unfortunately, this straightforward idea does not allow for [26]'s formal security proof to hold. For example, in the case where only the sender is corrupted, following Theorem 23 and the proof of Lemma 24 in [26], the expected running time of the simulator would be $\binom{K}{K-g} = 1/(1-\epsilon)$, where K and g are statistical security parameters. If $\epsilon = 1 - \mu(k)$, then the expected running time $\binom{K}{K-g} = 1/\mu(k)$ is greater than any positive polynomial in the security parameter. As a consequence, in the current paper, to obtain security against malicious adversaries, the ideas behind formal security proofs are totally different from [26].

The above discussion gives examples emphasizing that, in the field of provable security, it is usually technically difficult and error-prone rather than straightforward to construct a scheme with higher security level from one with lower security level.

To summarize our approach at a high level, our basic idea is to use the notion of SPH-DHM and a cut-and-choose technique to construct a framework Π for OT_t^n with fully-simulatable SAMA. Our scheme can be depicted as follows:

1. Let K be a predetermined positive integer. The receiver generates a hash family parameter and "cuts" K vectors where each vector contains t projective instance-witness pairs and $n-t$ smooth instance-witness pairs. It shuffles each vector and sends the parameter and the shuffled *instance vectors* to the sender.
2. The sender checks that the hash family parameter is legal. Then, both parties commonly run a coin-tossing protocol to "choose" instance vectors to check their legalities.
3. To prove the chosen instance vectors' legalities, the receiver sends their witness vectors to the sender.
4. After the sender has checked the validity of those vectors, the receiver reorders each non-chosen instance vector using a permutation over $\{1, 2, \ldots, n\}$ based on its private indexes (representing the t elements it wants to obtain). Then, the receiver forwards all these permutations to the sender.
5. According to the permutations, the sender reorders every non-chosen instance vector. Then, it encrypts its private n values by XOR-ing them with the hash values of the non-chosen instance vectors. Finally, the sender sends the encryptions and projection keys of non-chosen instance vectors to the receiver.
6. The receiver computes the projection values of the non-chosen instances vectors and it XOR-es the projection values and the encryptions to gain the t values it sought.

1.2.2. Reducing SMPC to SPH

As a theoretical contribution, our protocol can be interpreted as a constant-round blackbox reduction of OT_t^n to SPH. Besides SPH, our protocol Π employs two cryptographic primitives: a perfectly hiding commitment and a perfectly binding commitment, which are used to toss coins. The coin-tossing can be carried out sequentially (bit by bit) [28], without using perfectly hiding commitments. Further, perfectly binding commitments can be built from

one-way functions [29], which are implied by SPH. We therefore obtain a $O(K)$ round black-box reduction of OT_t^n to SPH. Combining Kilian's completeness result [2], we also obtain a black-box reduction of SMPC to SPH.

1.2.3. Efficiency of the Construction

Our framework Π for OT_t^n costs 6 communication rounds. In practice, it expectedly costs $20\,n$ encryptions and $20\,t$ decryptions. In the particular case of OT_1^2, its communication rounds (respectively, expected computational overhead) is 3 times (respectively, 20 times) of [16]. Thus, it is practical.

1.3. Paper Organization

In the next section, we describe the notations used throughout our work and the security definition for OT_t^n. In Section 4, we define a new variant of smooth projective hash (i.e. SPH-DHM). Our framework Π for OT_t^n is exposed in Section 5 and its security is demonstrated in Section 6. In Section 7, we extend our framework from OT_1^2 to OT_t^n.

2. Preliminaries

Most notations and concepts mentioned in this section come from [29, 30, 31] and we tailored them to deal with OT_t^n.

2.1. Basic Notations and Definitions

We set the following notations for this paper:

- $\mathbb{N}$: set of natural numbers.
- k: *security parameter* where $k \in \mathbb{N}$. It is used to measure the security of the underlying computational assumptions (e.g., the DDH assumption).
- K: *statistical security parameter* where $K \in \mathbb{N}$. This is the number of instance vectors that the receiver "cuts", and so K defines the size of our "cut-and-choose" test. Thus, K is used to measure the probability that the adversary is not caught in cut-and-choose type checks (see Lemma 30). Note that this probability does not depend on any computational intractability assumption.
- $[n]$: the set $\{1, 2, \ldots, n\}$ where $n \in \mathbb{N}$.
- $\Psi = \{T \subseteq [n] : |T| = t\}$: the set of the receiver's all legal private inputs in t-out-of-n oblivious transfer.
- $\vec{x}\langle j\rangle$: the j-th entry of the vector $\vec{x}$.
- S_n: the set of all permutations of $[n]$ where $n \in \mathbb{N}$.
- $\sigma(\vec{x})$: the vector gained by shifting the i-th entry of the n-vector $\vec{x}$ to the $\sigma(i)$-th entry where $\sigma \in S_n$. In other words, $\sigma(\vec{x})$ denotes the vector $\vec{y}$ such that: $\forall i \in [n] \quad \vec{y}\langle\sigma(i)\rangle = \vec{x}\langle i\rangle$.
- $\mathsf{Poly}(.)$: an unspecified positive polynomial.
- $\{0,1\}^*$: set of all bitstrings.
- $\alpha \in_U D$: an element α chosen uniformly at random from a domain D.
- $\alpha \in_\chi D$: an element α chosen from a domain D according the probabilistic distribution χ.
- $|X|$: the cardinality of a finite set X.

Definition 1. *A (positive) function $\mu(.)$ is called* negligible in *k, if and only if:*

$$\forall \mathsf{Poly}(\cdot) > 0\ \exists k_0 \in \mathbb{N} : \forall k > k_0 \quad \mu(k) < 1/\mathsf{Poly}(k).$$

Definition 2. *A* probability ensemble

$$X \stackrel{def}{=} \{X(1^k, a)\}_{k\in\mathbb{N}, a\in\{0,1\}^*}$$

is an infinite sequence of random variables indexed by (k, a)*, where a represents various types of inputs used to sample the instances according to the distribution of the random variable* $X(1^k, a)$*.*

Definition 3. *A probability ensemble X is* polynomial-time constructible*, if there exists a* probabilistic polynomial-time *(PPT) sampling algorithm* Samp(.) *such that for any a, any k, the random variables* $\mathsf{Samp}(1^k, a)$ *and* $X(1^k, a)$ *are identically distributed.*

Definition 4. *Let X, Y be two probability ensembles. We say they are* computationally indistinguishable*, denoted by* $X \stackrel{c}{=} Y$*, if for any non-uniform PPT algorithm D with auxiliary input* $z = (z_k)_{k\in\mathbb{N}}$ *(where each* $z_k \in \{0,1\}^*$*), there exists a negligible function* $\mu(.)$ *such that for any sufficiently large k and any* $a \in \{0,1\}^*$*, it holds that*

$$|\text{Prob}(D(1^k, a, X(1^k, a), z_k) = 1) - \text{Prob}(D(1^k, a, Y(1^k, a), z_k) = 1)| \leqslant \mu(k).$$

Definition 5. *Let X, Y be two probability ensembles. They are said to be* statistically indistinguishable*, denoted by* $X \stackrel{s}{=} Y$*, if their statistical difference is negligible. More specifically, if there exists a negligible function* $\mu(.)$ *such that*

$$1/2 \cdot \sum_{\alpha\in\{0,1\}^*} |\text{Prob}(X(1^k, a) = \alpha) - \text{Prob}(Y(1^k, a) = \alpha)| = \mu(k).$$

Definition 6. *Let X, Y be two probability ensembles. They are said to be* identical*, denoted by* $X \equiv Y$*, if the distributions of* $X(1^k, a)$ *and* $Y(1^k, a)$ *are identical.*

Remark 7. *Let X, Y be two probability ensembles. We obviously have:* $X \equiv Y$ *implies* $X \stackrel{s}{=} Y$ *and* $X \stackrel{s}{=} Y$ *implies* $X \stackrel{c}{=} Y$*.*

2.2. OT_t^n *with Non-Adaptive SAMA*

The OT_t^n functionality is defined as follows.

$$\begin{array}{rcl} f : \mathbb{N} \times \{0,1\}^* \times \{0,1\}^* & \longrightarrow & \{0,1\}^* \times \{0,1\}^* \\ (1^k, \vec{m}, T) & \longmapsto & (\lambda, (\vec{m}\langle i\rangle)_{i\in T}) \end{array}$$

where:

- k is the security parameter,
- $\vec{m}$ is a vector of n values having identical bitlength,
- λ denotes the empty string,
- T is a set of t indexes from $[n]$,
- $(\vec{m}\langle i\rangle)_{i\in T}$ is the sequence of t values indexed by T.

In this functionality, the sender $\mathcal{S}$ privately holds the input $\vec{m}$ and receives no output while the receiver $\mathcal{R}$ privately owns the set T and receives $(\vec{m}\langle i\rangle)_{i\in T}$.

Before engaging the OT protocol, the adversary $\mathcal{A}$ corrupts the parties listed in the set $I \subseteq \{\mathcal{S}, \mathcal{R}\}$. As $\mathcal{A}$ is *non-adaptive*, the set I of corrupted players will not change until the computation ends. In our case, we consider that at most one party is corrupted by $\mathcal{A}$. Non-corrupted participants (i.e. honest parties) will faithfully follow the protocol's instructions. When $\mathcal{A}$ corrupts a party, $\mathcal{A}$ takes control over the party's actions and $\mathcal{A}$ gets aware of the party's communication and computation history. In particular, the input of the corrupt participant is known (and controlled) by $\mathcal{A}$. The objective of $\mathcal{A}$ is to gain some extra knowledge about the honest player's private input other than what is inferred by the result of the computation and $\mathcal{A}$'s protocol input.

We are now to recall how the ideal/real world paradigm works. For simplicity, we are to associate the value 1 with the sender $\mathcal{S}$ and the value 2 with the receiver $\mathcal{R}$. Thus: $\{\mathcal{S}, \mathcal{R}\} = \{1, 2\}$.

2.2.1. The Ideal World

In the ideal world, there is an incorruptible *trusted third party* (TTP) (named ideal functionality in Section 1). An execution of OT_t^n proceeds as follows.

- **Inputs.** All entities know the public security parameter k. The sender $\mathcal{S}$ holds $\vec{m}$. The receiver $\mathcal{R}$ holds T. The adversary $\mathcal{A}$ holds a name list $I \subseteq \{1, 2\}$, a randomness $r_{\mathcal{A}} \in \{0,1\}^*$ and an auxiliary input $z = (z_k)_{k\in\mathbb{N}}$, where $z_k \in \{0,1\}^*$.

  Before proceeding to the next stage, $\mathcal{A}$ corrupts parties listed in I and learns their inputs.

- **Sending inputs to the TTP.** Each honest party sends its input to the TTP. For each corrupted party, $\mathcal{A}$ sends a string to the TTP on behalf of the party. This string may be the input of the corrupted party, other input of the same length, or an early termination request Abort_i ($i \in I$).

Denote the inputs received by the TTP by $\vec{y} = (y_1, y_2)$ (note that $\vec{y}$ does not necessarily equal $(\vec{m}, T)$). If the TTP receive an Abort_i for some $i \in I$, it sends Abort_i to both parties and the ideal execution terminates.

Remark 8. *If the TTP received multiple aborting messages, then it means that both players have been corrupted by $\mathcal{A}$. This situation represents no security objective whatsoever since the adversary controls every entity.*

In the case no aborting message was sent to the TTP, the execution of the protocol proceeds to the next step.

- **TTP answering the adversary.** The TTP computes $f(y_1, y_2)$ and sends $\mathcal{A}$ the outputs $(f(y_1, y_2)\langle i\rangle)_{i\in I}$ of the corrupted parties.

- **TTP answering the honest parties.** $\mathcal{A}$ sends either Abort_i for some $i \in I$ or $\mathsf{Continue}$ to the TTP. If the TTP receives $\mathsf{Continue}$, then it sends the honest parties their results. Otherwise, it sends the honest parties Abort_i.

- **Outputs.** Each honest party always outputs the message obtained from the TTP. Each corrupted party outputs nothing. Instead, $\mathcal{A}$ outputs any arbitrary (PPT computable) function of the initial inputs of the corrupted parties, the auxiliary input, and the messages obtained from the TTP.

The output of the execution is defined/denoted by a 3-entry vector $\mathsf{Ideal}_{f,\mathcal{A}(z),I}(1^k, \vec{m}, T, r_{\mathcal{A}})$ written as:

$$\mathsf{Ideal}_{f,\mathcal{A}(z),I}(1^k, \vec{m}, T, r_{\mathcal{A}}) = (\mathsf{Ideal}_{f,\mathcal{A}(z),I}(1^k, \vec{m}, T, r_{\mathcal{A}})\langle 0\rangle, \mathsf{Ideal}_{f,\mathcal{A}(z),I}(1^k, \vec{m}, T, r_{\mathcal{A}})\langle 1\rangle, \mathsf{Ideal}_{f,\mathcal{A}(z),I}(1^k, \vec{m}, T, r_{\mathcal{A}})\langle 2\rangle)$$

where

- $\mathsf{Ideal}_{f,\mathcal{A}(z),I}(1^k, \vec{m}, T, r_{\mathcal{A}})\langle 0\rangle$ is $\mathcal{A}$'s output,
- $\mathsf{Ideal}_{f,\mathcal{A}(z),I}(1^k, \vec{m}, T, r_{\mathcal{A}})\langle 1\rangle$ is the sender's output,
- $\mathsf{Ideal}_{f,\mathcal{A}(z),I}(1^k, \vec{m}, T, r_{\mathcal{A}})\langle 2\rangle$ is the receiver's output.

2.2.2. The Real World

In the real world, we do not have any TTP and the two parties communicate with each other using an authenticated channel. Let Π be a protocol for OT_t^n. A execution of Π proceeds as follows.

- **Inputs.** They are identical to the inputs in the ideal world except that $\mathcal{S}$ (resp., $\mathcal{R}$) additionally holds a randomness r_1 (resp., r_2).

- **Computation.** Computing f is done via interactions between the sender and the receiver. Each honest party strictly follows the prescribed protocol Π. The corrupted parties follows $\mathcal{A}$'s instructions and may arbitrarily deviate from Π.

- **Outputs.** Each honest party always outputs what Π instructs. Each corrupted party outputs nothing. Instead, $\mathcal{A}$ outputs any arbitrary (PPT computable) function of the initial inputs of the corrupted parties, the auxiliary input, and the messages it sees during the execution of Π.

The output of the execution is defined/denoted by a 3-entry vector $\mathsf{Real}_{\Pi,I,\mathcal{A}(z_k)}(1^k, \vec{m}, T, r_{\mathcal{A}}, r_1, r_2)$ where the first, second and third entries are $\mathcal{A}$'s output, the sender's output and the receiver's output respectively similarly to $\mathsf{Ideal}_{f,\mathcal{A}(z),I}(1^k, \vec{m}, T, r_{\mathcal{A}})$.

2.2.3. Security Definition

Intuitively speaking, we say that protocol Π securely computes OT_t^n in the presence of malicious adversaries, if and only if, for any malicious adversary $\mathcal{A}$, what harm $\mathcal{A}$ can do in the real world is not more than in the ideal world. This intuition is formally captured by the following definition.

Definition 9. *Let f denote the functionality of OT_t^n. Let Π be a concrete protocol for OT_t^n. Let Ψ be a set of the receiver's all legal private inputs. We say* Π securely computes *f in the presence of malicious adversaries, if and only if for any non-uniform PPT adversary $\mathcal{A}$ with auxiliary input $z = (z_k)_{k\in\mathbb{N}}$ in the real world, there exists a non-uniform probabilistic expected polynomial-time adversary $\mathfrak{S}$ with the same auxiliary input in the ideal world such that, for any $I \subseteq [2]$, the following equation holds.*

$$\{\mathsf{Real}_{\Pi,I,\mathcal{A}(z_k)}(1^k,\vec{m},T)\}_{k\in\mathbb{N},\vec{m}\in(\{0,1\}^*)^n,T\in\Psi,z_k\in\{0,1\}^*} \overset{c}{=} \{\mathsf{Ideal}_{f,I,\mathfrak{S}(z_k)}(1^k,\vec{m},T)\}_{k\in\mathbb{N},\vec{m}\in(\{0,1\}^*)^n,T\in\Psi,z_k\in\{0,1\}^*}, \quad (1)$$

where the parameters input to the two probability ensembles are the same. The adversary $\mathfrak{S}$ is called the simulator *of the adversary $\mathcal{A}$.*

We point out that the security definitions presented in [30, 31] require the simulator $\mathfrak{S}$ to run in strictly polynomial-time but those from [32, 25, 19] allow $\mathfrak{S}$ to run in expected polynomial-time. Definition 9 follows the latter. We argue about our choice as follows. First, allowing the simulator to run in expected polynomial-time is essential for achieving (non-trivial) constant-round protocols (our framework Π has constant round complexity) as Barak and Lindell showed that there was no (non-trivial) constant-round ZK proof of argument having a strictly polynomial-time black-box simulator [33]. Second, in many cases (also when strictly polynomial-time simulators exist), the expected running time of the simulator provides a better bound than the worst-case running time [34].

2.3. Smooth Projective Hash

As said in Section 1, SPH was introduced to design chosen-ciphertext secure encryption schemes and Halevi and Tauman Kalai applied a variant of this cryptographic primitive to construct a protocol for OT.

Definition 10 ([16])**.** *A hash family $\mathcal{H}$ is defined by means of the following PPT algorithms $\mathcal{H} = (\mathsf{PG}, \mathsf{IS}, \mathsf{IT}, \mathsf{KG}, \mathsf{Hash}, \mathsf{pHash})$:*

- *Parameter generator* PG*: it takes a security parameter k as input and returns a hash parameter Λ: i.e. $\Lambda \leftarrow \mathsf{PG}(1^k)$.*
- *Instance sampler* IS*: it takes a security parameter k and a hash parameter Λ as input and returns a triple, i.e., $(\dot{x}, \dot{w}, \ddot{x}) \leftarrow \mathsf{IS}(1^k, \Lambda)$), where $\dot{x}$ is a projective instance, $\dot{w}$ is one of its witnesses, $\ddot{x}$ is a smooth instance.*
- *Instance-testing algorithm* IT*: it tests the parameters Λ and two strings x_0, x_1, i.e., $\mathsf{IT}(\Lambda, x_0, x_1) \in \{0, 1\}$. The intent is to test that at least one of x_0, x_1 is a smooth instance.*
- *Key generator* KG*: it takes a security parameter k, a hash parameter Λ and an instance x as input and outputs a hash-projection key pair (hk, pk): i.e., $(hk, pk) \leftarrow \mathsf{KG}(1^k, \Lambda)$.*
- *Hash algorithm* Hash*: it takes a security parameter k, a hash parameter Λ, an instance x and a hash key hk as input and outputs a value y: i.e., $y \leftarrow \mathsf{Hash}(1^k, \Lambda, x, hk)$.*
- *Projection algorithm* pHash*: it takes a security parameter k, a hash parameter Λ, an instance x, a projection key pk and a witness w of x as input and outputs a value y: i.e., $y \leftarrow \mathsf{pHash}(1^k, \Lambda, x, pk, w)$.*

The *smoothness* requires that for any $\ddot{x}$, its projection key and hash value are almost uniformly distributed. The *projection* requires that for any $\dot{x}$ and any its hash-projection key pair (hk, pk), its hash value equals its projection value. The *verifiable smoothness* requires that if $\mathsf{IT}(\Lambda, x_0, x_1) = 1$, then at least one of x_0, x_1 is a smooth instance. The *hard subset membership* requires the smooth instances $\ddot{x}$ and projective instances $\dot{x}$ are computationally indistinguishable. We let VSPH-HM denote the hash family which holds all properties mentioned here.

2.4. Commitment Scheme

In this section, we briefly introduce the cryptographic tool commitment scheme which will be used in our framework. For the strict definitions and the details, please see [29, 35].

Definition 11. *A* commitment scheme *is a two-party protocol involving two phases.*

- *Initial Inputs. At the beginning, all parties know the public security parameter k. The unbounded sender P_1 holds a randomness $r_1 \in \{0,1\}^*$, a value $m \in \{0,1\}^{\mathsf{Poly}(k)}$ to be committed to. The probabilistic polynomial time (PPT) receiver P_2 holds a randomness $r_2 \in \{0,1\}^*$.*
- *Commit Phase. P_1 computes a commitment, i.e., $\gamma \leftarrow \mathsf{Com}(1^k, m, r_1)$, then P_1 send γ to P_2.*
- *Reveal Phase. P_1 sends a de-commitment, which typically consists of (m, r_1), to P_2. Receiving de-commitment, P_2 checks its validity. Typically P_2 checks that $\gamma = \mathsf{Com}(1^k, m, r_1)$ holds. If de-commitment pass the check, P_2 accepts m.*

Definition 12. *A commitment scheme provides two security guarantees.*

- Hiding *prevents P_2 from the committed value m before reveal phase. That is, for any PPT P_2, any $m_1, m_2 \in \{0,1\}^{poly(k)}$, the probability ensembles describing the output of P_2 in two cases are computationally indistinguishable, i.e.,*
$$\{\langle P_1(m_1), P_2\rangle(1^k)\}_{k\in\mathbb{N}} \stackrel{c}{=} \{\langle P_1(m_1), P_2\rangle(1^k)\}_{k\in\mathbb{N}},$$
- *Binding prevents P_1 opening a commitment in two different ways. That is, for any unbounded P_1, any $m_1, m_2 \in \{0,1\}^{\mathsf{Poly}(k)}$ such that $m_1 \neq m_2$, the probability that P_2 accepts m_2 while the committed value is m_1 is 0, where the probability is taken only over the randomness used by P_2.*

Definition 12 indeed describes a type called perfectly binding of commitment scheme, which we will use in this paper. Another type we will use is perfectly hiding, which guarantees that for any (possibly unbounded) P_2 learns nothing about the committed value before reveal phase. For any commitment scheme, at most one of its two security guarantees is against unbounded adversaries. We use BC and HC to denote the commitment operations Com of perfectly binding type and perfectly hiding type, respectively.

3. Halevi-Kalai OT and Its Problems in Simulation-Based Proof

3.1. Halevi-Kalai OT

Let us recall Halevi-Kalai OT_1^2 protocol [16] first. It proceeds as follows:

- R1 (Receiver's step): $\mathcal{R}$ generates the hashing parameters Λ and samples random instances $(\dot{x}, \ddot{x}, w)$, where $\dot{x}$ is projective and w is its witness. $\mathcal{R}$ sets $x_b \leftarrow \dot{x}$ and $x_{3-b} \leftarrow \ddot{x}$ ($b \in \{1,2\}$). $\mathcal{R}$ sends (Λ, x_b, x_{3-b}).
- S1 (Sender's step): $\mathcal{S}$ verifies that at least one instance of (x_1, x_2) is smooth. If the test fails then the sender aborts. Otherwise the sender encrypts each message m_i via XOR-ing it with hash value of x_i. $\mathcal{S}$ sends ciphertext c_i along projection key of x_i.
- R2 (Receiver's step): $\mathcal{R}$ XOR-es ciphertext c_b with projection value of x_b and gets message m_b.

Halevi-Kalai OT_1^2 protocol meets privacy-based definition, which is a weaker notion than simulation-based definition. In this definition, no party should be able to distinguish two views generated based on distinct set of inputs for the other party but yield the same output.

Definition 13 ([16])**.** *A protocol is said to* privately implement oblivious transfer OT_1^2 *if the following conditions are satisfied:*

- *Receiver's Privacy: Denoted by $\mathcal{R}(1^n, b)$ the message sent by the honest receiver with input $(1^n, b)$. Then the ensembles $\{\mathcal{R}(1^n, 1)\}_{n\in\mathbb{N}}$ and $\{\mathcal{R}(1^n, 2)\}_{n\in N}$ are computationally indistinguishable; $\{\mathcal{R}(1^n, 1)\}_{n\in\mathbb{N}} \stackrel{c}{=} \{\mathcal{R}(1^n, 2)\}_{n\in N}$.*

- *Sender's Privacy: Denote by $\mathcal{S}(1^n, m_1, m_2, q)$ the response of the honest sender with input $(1^n, m_1, m_2)$ when the receiver' s first message is q. Then there is a negligible function μ such that for any $n > 0$, any three messages $m_1, m_2, m' \in \{0,1\}^{l(n)}$, and any message $q \in \{0,1\}^*$, it holds that*

$$\mathcal{S}(1^n, m_1, m_2, q) \stackrel{s}{=} \mathcal{S}(1^n, m_1, m', q) \vee \quad or$$
$$\mathcal{S}(1^n, m_1, m_2, q) \stackrel{s}{=} \mathcal{S}(1^n, m', m_2, q)$$

3.2. Simulation Problems

Halevi and Kalai discusses the possibility of simulation-based security of their OT_1^2. In the case that the receiver is corrupted, [16] admits that their security definition does not give a simulation-based guarantee. The reason is that the simulator can not extract the choice of a malicious receiver.

In the case that the sender is corrupted, Halevi-Kalai asserts that Definition 13 gives a simulation-based guarantee. We find that this is not true. Following their ideas [16, Sec. 3], the simulator should be constructed as follows.

1. The simulator $\mathfrak{S}$ invokes the adversary $\mathcal{A}$ as a subroutine.
2. Simulator $\mathfrak{S}$ plays the role of an honest receiver with private input 1, and extracts message m_1.
3. $\mathfrak{S}$ rewinds $\mathcal{A}$, plays the role of an honest receiver with private input 2, and extracts message m_2.
4. $\mathfrak{S}$ sends (m_1, m_2) to the TTP, and outputs what $\mathcal{A}$ outputs.

Since $\{\mathcal{R}(1^n, 1)\}_{n\in N} \stackrel{s}{=} \{\mathcal{R}(1^n, 2)\}_{n\in N}$, the views of adversary $\mathcal{A}$ in the real world and ideal world are statistically indistinguishable too. Combining with the fact that the sender $\mathcal{S}$ outputs nothing in both the real world and the ideal world, one may conclude that the two worlds are computationally indistinguishable. However, this simulation ignores two subtle problems.

P1 $\mathcal{A}$ may not always gives responses to $\mathfrak{S}$.

P2 (m_1, m_2) is not extracted in one shot.

To illustrate P1, consider the following. $\mathcal{A}$ may gives responses in the first extraction and refuses to respond in the second extraction. This is possible, because $\mathcal{A}$ receives distinct messages in two distinct extractions. If this is the case, $\mathfrak{S}$ can not extract m_2 and fails.

To illustrate P2, consider the following. Since $\mathcal{A}$ is malicious, it may choose distinct values in distinct extractions. For example, the adversary $\mathcal{A}$ follows the following strategy:

- in each execution, $\mathcal{A}$ chooses two random values $m_1, m_2 \in_U \{0,1\}^*$ as its real input and finally outputs them.

More concretely, let us assume that the real input of $\mathcal{A}$ in the first interaction and the second interaction are (a_1, a_2) and (b_1, b_2) respectively. We also assume that honest receiver R takes 1 as its input. Then we know,

$$\mathsf{Real} = ((a_1, a_2), \lambda, a_1) \quad \mathsf{Ideal} = ((b_1, b_2), \lambda, a_1)$$

where $(a_1, a_2), (b_1, b_2)$ are outputs of adversary $\mathcal{A}$ and simulator $\mathfrak{S}$, respectively. Considering the random choices of a_1, a_2, b_1, b_2, it holds with overwhelming probability that

$$a_1 \notin \{b_1, b_2\}.$$

That is, in the ideal world, the receiver get a value that is highly probable to be inconsistent with the output of $\mathcal{S}$. This distinguishes the real world from the ideal world.

More generally, we have Lemma 14. Since the proof is similar to the discussion above, we omit the details.

Lemma 14. *Let Π be a protocol that is supposed to implement a two-party functionality $f(x, y)$ such that the first party receives $f_1(x, y)$ and the second party receives $f_2(x, y)$. Let $\mathfrak{S}$ be the simulator of the case that the malicious adversary corrupts the first party. If simulator $\mathfrak{S}$ does not extract real input of the adversary in one shot, and there exists a value v such that makes $f_2(\cdot, v)$ injective, then $\mathfrak{S}$ does not provide black-box-simulation-based security.*

4. A New Smooth Projective Hash

Since previous definitions of SPH do not suffice for our application, we define another variant of SPH. See [24] for its instantiations.

Definition 15. *A (n, t)-hash family $\mathcal{H}$ is defined by means of the following PPT algorithms $\mathcal{H} = (\mathsf{PG}, \mathsf{IS}, \mathsf{pIS}, \mathsf{Check}, \mathsf{DI}, \mathsf{KG}, \mathsf{Hash}, \mathsf{pHash})$:*

- *Parameter generator* PG*: it takes a security parameter k as input and returns a hash parameter Λ: i.e. $\Lambda \leftarrow \mathsf{PG}(1^k)$.*
- *Checker* Check*: it takes a security parameter k and a hash parameter Λ as input and returns an indicator bit $b \in \{0, 1\}$: i.e. $b \leftarrow \mathsf{Check}(1^k, \Lambda)$. The objective is to check that Λ was correctly generated.*
- *Instance sampler* IS*: it takes a security parameter k and a hash parameter Λ as input and returns a vector $\vec{a} = ((\dot{x}_1, \dot{w}_1), \ldots, (\dot{x}_t, \dot{w}_t), (\ddot{x}_{t+1}, \ddot{w}_{t+1}), \ldots, (\ddot{x}_n, \ddot{w}_n))$ (i.e., $\vec{a} \leftarrow \mathsf{IS}(1^k, \Lambda)$) where each entry of $\vec{a}$ is an instance-witness pair with the first t pairs are projective and the last $n - t$ pairs are smooth.*
- *Projective instance sampler* pIS*: similar to* IS *with exception that all n instances are projective.*
- *Distinguisher* DI*: it takes a security parameter k, a hash parameter Λ and an instance-witness pair (x, w) as input and outputs an indicator value b: i.e., $b \leftarrow \mathsf{DI}(1^k, \Lambda, x, w)$. Its goal is to distinguish smooth instances and projective instances.*
- *Key generator* KG*: it takes a security parameter k, a hash parameter Λ and an instance x as input and outputs a hash-projection key pair (hk, pk): i.e., $(hk, pk) \leftarrow \mathsf{KG}(1^k, \Lambda, x)$.*
- *Hash algorithm* Hash*: it takes a security parameter k, a hash parameter Λ, an instance x and a hash key hk as input and outputs a value y: i.e., $y \leftarrow \mathsf{Hash}(1^k, \Lambda, x, hk)$.*
- *Projection algorithm* pHash*: it takes a security parameter k, a hash parameter Λ, an instance x, a projection key pk and a witness w of x as input and outputs a value y: i.e., $y \leftarrow \mathsf{pHash}(1^k, \Lambda, x, pk, w)$.*

Definition 16. *For a given hash parameter Λ, if* Check *outputs* 1*, then Λ is said to be* legal*; otherwise, it is said to be* illegal*.*

Remark 17. *It is obvious that any Λ generated by* PG *is legal.*

Definition 18. *Let $R = \{(x, w) : x, w \in \{0, 1\}^*\}$ be a relation. For a legal Λ, we define its* projective relation *as $\dot{R}_\Lambda = \{(\dot{x}, \dot{w}) : (\dot{x}, \dot{w})$ is generated by $\mathsf{IS}(1^k, \Lambda)\}$ and its* smooth relation *as $\ddot{R}_\Lambda = \{(\ddot{x}, \ddot{w}) : (\ddot{x}, \ddot{w})$ is generated by $\mathsf{IS}(1^k, \Lambda)\}$.*

Definition 19 (Distinguishability)**.** *For any legal hash parameter Λ, any instance-witness pair (x, w), we require:*

$$\mathsf{DI}(1^k, \Lambda, x, w) = \begin{cases} 0 & \text{if } (x, w) \in \dot{R}_\Lambda, \\ 1 & \text{if } (x, w) \in \ddot{R}_\Lambda, \\ 2 & \text{otherwise.} \end{cases}$$

Definition 20. *If R is a relation, then its* language *is defined as $L \stackrel{def}{=} \{x \in \{0, 1\}^* : \exists w((x, w) \in R)\}$.*

Let $\dot{L}_\Lambda$ and $\ddot{L}_\Lambda$ be the languages of relation $\dot{R}_\Lambda$ and relation $\ddot{R}_\Lambda$, respectively. The properties smoothness and projection to be defined next will ensure that $\dot{L}_\Lambda \cap \ddot{L}_\Lambda = \emptyset$ holds. That is, no instance can exhibit both smoothness and projection.

Definition 21 (Projection)**.** *For any hash parameter Λ generated by $\mathsf{PG}(1^k)$, any projective instance-witness pair $(\dot{x}, \dot{w})$ generated by $\mathsf{IS}(1^k, \Lambda)$, and any hash-projection key pair (hk, pk) generated by $\mathsf{KG}(1^k, \Lambda, \dot{x})$, it holds that*

$$\mathsf{Hash}(1^k, \Lambda, \dot{x}, hk) = \mathsf{pHash}(1^k, \Lambda, \dot{x}, pk, \dot{w}).$$

Definition 22. *For an instance-witness vector* $\vec{a} = ((x_1, w_1), \ldots, (x_n, w_n))$*, we define its* instance vector *as* $x(\vec{a}) \stackrel{def}{=} (x_1, \ldots, x_n)$ *and its* witness vector *as* $w(\vec{a}) \stackrel{def}{=} (w_1, \ldots, w_n)$.

Definition 23. *Fix a legal hash parameter* Λ*. If a vector* $\vec{a}$ *contain at least* $n - t$ *smooth instance-witness pairs, (i.e., at least* $n - t$ *pairs in* $\ddot{R}_\Lambda$*), then* $\vec{a}$ *is said to be* legal.

Remark 24. *Note that any* $\vec{a}$ *generated by* $\mathsf{IS}(1^k, \Lambda)$ *is legal, and the legality of any* $\vec{a}$ *that may be maliciously generated can be checked by invoking algorithm* DI *at most n times.*

Definition 25 (Smoothness)**.** *For any legal hash parameter* Λ*, any legal instance-witness vector* $\vec{a}$ *(without loss of generality, we assume that the last* $n - t$ *entries of* $\vec{a}$ *are smooth), any permutation* $\sigma \in S_n$*,* smoothness *holds if the two probability ensembles* $\mathsf{SM}_1 \stackrel{def}{=} \{\mathsf{SM}_1(1^k)\}_{k\in\mathbb{N}}$ *and* $\mathsf{SM}_2 \stackrel{def}{=} \{\mathsf{SM}_2(1^k)\}_{k\in\mathbb{N}}$*, specified as follows, are statistically indistinguishable: i.e.,* $\mathsf{SM}_1 \stackrel{s}{=} \mathsf{SM}_2$. Perfect smoothness *holds, if* $\mathsf{SM}_1 \equiv \mathsf{SM}_2$.

- $G_\Lambda \stackrel{def}{=} \{y : x \in \dot{L}_\Lambda \cup \ddot{L}_\Lambda, (hk, pk) \leftarrow \mathsf{KG}(1^k, \Lambda, x), y \leftarrow \mathsf{Hash}(1^k, \Lambda, x, hk)\}$ *is a set of all possible hash values.*
- *Algorithm* $\mathsf{SmGen}_1(1^k)$ *works as follows:*
  - $\vec{x} \leftarrow x(\vec{a})$.
  - *For each* $j \in [n]$*, perform:* $(hk_j, pk_j) \leftarrow \mathsf{KG}(1^k, \Lambda, \vec{x}\langle j\rangle)$, $y_j \leftarrow \mathsf{Hash}(1^k, \Lambda, \vec{x}\langle j\rangle, hk_j)$.
  - *Set* $\overrightarrow{pky} \leftarrow (pk_j, y_j)_{j\in[n]}$ *and output* $\overrightarrow{pky}$.
- *Algorithm* $\mathsf{SmGen}_2(1^k)$ *works as* $\mathsf{SmGen}_1(1^k)$ *except that for each* $j \in \{t+1, t+2, \ldots, n\}$, $y_j \in_U G_\Lambda$.
- *For* $i \in [2]$*, algorithm* $\mathsf{SM}_i(1^k)$ *works as follows:* $\overrightarrow{pky} \leftarrow SmGen_i(1^k)$, $\widetilde{\overrightarrow{pky}} \leftarrow \sigma(\overrightarrow{pky})$ *and output* $\widetilde{\overrightarrow{pky}}$.

Definition 26 (Hard Subset Membership)**.** *For any* $\sigma \in S_n$*, the two probability ensembles* $\mathsf{HS}_1 \stackrel{def}{=} \{\mathsf{HS}_1(1^k)\}_{k\in\mathbb{N}}$ *and* $\mathsf{HS}_2 \stackrel{def}{=} \{\mathsf{HS}_2(1^k)\}_{k\in\mathbb{N}}$*, specified as follows, are computationally indistinguishable, i.e.,* $\mathsf{HS}_1 \stackrel{c}{=} \mathsf{HS}_2$.

- *Algorithm* $\mathsf{HS}_1(1^k)$ *works as follows:* $\Lambda \leftarrow \mathsf{PG}(1^k)$, $\vec{a} \leftarrow \mathsf{IS}(1^k, \Lambda)$ *and outputs* $(\Lambda, x(\vec{a}))$.
- *Algorithm* $\mathsf{HS}_2(1^k)$ *operates as* $\mathsf{HS}_1(1^k)$ *except that it outputs* $(\Lambda, \sigma(x(\vec{a})))$.

Remark 27. *In this paper, for a projective instance* $\dot{x}$*, its witness* $\dot{w}$ *is mainly used to gain the instance's hash value while, for a smooth instance* $\ddot{x}$*, its witness* $\ddot{w}$ *serves as a proof of smoothness (i.e., a proof of* $\ddot{x} \in \ddot{L}_\Lambda$*). The distinguishability property guarantees that, given the needed witness-vector, projective instances and smooth instances are distinguishable.*

For notational simplicity, we denote a (n, t)-hash family $\mathcal{H}$ that holds properties smoothness, projection, distinguishability and hard subset membership (n, t)-SPH-DHM. Similarly, we denote a verifiably smooth projective hash family with hard subset membership property [16] VSPH-HM. As both our SPH-DHM notion and Halevi and Tauman Kalai's VSPH-HM are used to construct protocols for OT, it is necessary to discuss the differences between these two variants of SPH.

1. The major difference between these two notions is that a SPH-DHM not only provides a witness to each projective instance (as a VSPH-HM) but also to every smooth instance. For example, $(2, 1)$-SPH-DHM samples tuples of form $((\dot{x}_1, \dot{w}_1), (\ddot{x}_2, \ddot{w}_2))$ while VSPH-HM samples tuples of form $(\dot{w}, \dot{x}, \ddot{x})$.
2. The hard subset membership property of SPH-DHM is stronger than that of VSPH-HM, because the former requires that indistinguishability holds for multiple instances.
3. The key generation algorithm KG of a SPH-DHM takes an additional parameter: an instance x. This technical modification makes instantiating such a hash family easier as we will see in Section **??** using the LWE assumption.

4. The instance sampling algorithm IS of a VSPH-HM generates tuples consisting of a smooth instance, a projective instance and its witness. To deal with OT_t^n, in a SPH-DHM, the sampling algorithm returns vectors containing t projective instance-witness pairs and $n - t$ smooth instance-witness pairs. As a natural result, the properties of smoothness and hard subset membership are extended to consider instance vectors of n entries.
5. A SPH-DHM does not need the verifiable smoothness property of a VSPH-HM (implemented by algorithm IT in [16]). This property was used by Halevi and Tauman Kalai to verify whether at least one of two instances is smooth. Instead, a SPH-DHM exhibits the distinguishability property (implemented by algorithm DI).
6. A SPH-DHM additionally provides an algorithm pIS which plays a key role in simulation-based security proof. In the case that the sender is corrupted, our simulator invokes pIS to cheat the adversary and extract its real input. However, Halevi-Tauman can not offer a simulation-based proof in this case.

5. A Fully-Simulatable Framework for OT_1^2

5.1. Description of the Protocol

For clarity, we make the following convention. If $\mathcal{S}$ refuses to send $\mathcal{R}$ a message which is supposed to be sent, or $\mathcal{S}$ sends an invalid message that $\mathcal{R}$ cannot process then $\mathcal{R}$ halts the protocol and outputs Abort_1. Likewise, if $\mathcal{R}$ deviates in a similar way, $\mathcal{S}$ outputs Abort_2. The inputs are described as follows:

- Public Inputs: The security parameter k, the statistical security parameter K, and descriptions of three cryptographic tools: a $(2, 1)$-SPH-DHM $\mathcal{H}$, a perfectly binding commitment BC, and a perfectly hiding commitment HC.
- Private Inputs: The sender $\mathcal{S}$ holds two values $(m_1, m_2) \in \{0,1\}^*$ and holds a random tape $r_1 \in \{0,1\}^*$. The receiver $\mathcal{R}$ holds an index $o \in \{1, 2\}$ and holds a random tape $r_2 \in \{0,1\}^*$. The adversary $\mathcal{A}$ holds a name list $I \subseteq \{\mathcal{S}, \mathcal{R}\} = \{1, 2\}$ and a random tape $r_{\mathcal{A}} \in \{0,1\}^*$.
- Auxiliary Inputs: The adversary $\mathcal{A}$ holds an auxiliary input $z \in \{0,1\}^*$.

The interactions between $\mathcal{R}$ and $\mathcal{S}$ as described in Protocol 1.

Remark 28. *Note that $\mathcal{S}$ only cares about whether each chosen instance vector contains at least one smooth instances. Thus, in Step R2, to prove the legalities of the chosen instance vectors, $\mathcal{R}$ only needs to send the witnesses of the smooth instances. Formally speaking, $\mathcal{R}$ only needs to send $((i, j, \tilde{\vec{w}}_i\langle j\rangle))_{i\in C, j\in J_i}$ to $\mathcal{S}$, where $J_i \stackrel{def}{=} \{j : \tilde{\vec{x}}_i\langle j\rangle$ is smooth$\}$.*

5.2. Analysis of Π

5.2.1. Correctness of the Protocol

We now check that, when both the sender $\mathcal{S}$ and the receiver $\mathcal{R}$ are honest, Π is sound. For each $i \notin C$, we know:

$$\vec{c}\langle o\rangle = \vec{m}\langle o\rangle \oplus (\oplus_{i\notin C}\vec{\beta}_i\langle o\rangle) \quad \text{and} \quad m'_o = \vec{c}\langle o\rangle \oplus (\oplus_{i\notin C}\beta'_{io}).$$

$\tilde{\tilde{\vec{x}}}_i\langle o\rangle$ is a projective instance. $\mathcal{H}$'s property projection guarantees that its projection value equals its hash value. That is to say: $\vec{\beta}_i\langle o\rangle = \beta'_{io}$. Thus, we get: $\vec{m}\langle o\rangle = m'_o$ which proves that $\mathcal{R}$ gets $\vec{m}\langle o\rangle$ which is the sought data.

For $j \neq o$, $\tilde{\tilde{\vec{x}}}_i\langle j\rangle$ is a smooth instance. $\mathcal{H}$'s smoothness property guarantees that its projection value reveals no knowledge of its hash value. Thus, $\mathcal{R}$ cannot get any information about $\vec{m}\langle j\rangle$.

5.2.2. Security of the Scheme

The security of Π can be stated as follows.

Theorem 29. *Assume that in Π (i.e., Protocol 1) the smooth projective hash family $\mathcal{H}$ holds properties distinguishability and hard subset membership, the commitment scheme BC is perfectly binding and the commitment scheme HC is perfectly hiding. Then, Π securely computes functionality OT_1^2 in the presence of non-adaptive malicious adversaries where the sender's security is statistical and the receiver's security is computational.*

Protocol 1 Π

R1 (Receiver's step): $\mathcal{R}$ chooses a hash parameter and samples its instance-witness vectors as follows.

1. $\mathcal{R}$ chooses a hash parameter $\Lambda \leftarrow \mathsf{PG}(1^k)$.
2. $\mathcal{R}$ samples K instance-witness vectors: $\forall i \in [K] \quad \vec{a}_i = ((\dot{x}_{i1}, \dot{w}_{i1}), (\ddot{x}_{i2}, \ddot{w}_{i2})) \leftarrow \mathsf{IS}(1^k, \Lambda)$.
3. $\mathcal{R}$ shuffles each of these K vectors. That is, for each $i \in [K]$, $\mathcal{R}$ uniformly chooses a bit $d_i \in_U \{0,1\}$. If $d_i = 1$, then $\mathcal{R}$ does nothing, $\tilde{\vec{a}}_i \leftarrow \vec{a}_i$; otherwise exchange its two entries, $\tilde{\vec{a}}_i \leftarrow ((\ddot{x}_{i2}, \ddot{w}_{i2}), (\dot{x}_{i1}, \dot{w}_{i1}))$.
4. $\mathcal{R}$ sends the hash parameter Λ and the instance vectors $\tilde{\vec{x}}_1, \tilde{\vec{x}}_2, \ldots, \tilde{\vec{x}}_K$ to $\mathcal{S}$, where $\tilde{\vec{x}}_i$ is the instance vector of $\tilde{\vec{a}}_i$ (i.e., $\tilde{\vec{x}}_i = x(\tilde{\vec{a}}_i)$).

S1-S3 (Sender's step)/R2-R4 (Receiver's step): $\mathcal{S}$ and $\mathcal{R}$ cooperate to toss a bit-string $r = r_1 \oplus r_2$ called *choice indicator*. r indicates that the instance vectors $\{\tilde{\vec{x}}_i : r\langle i\rangle = 1\}$ are chosen.

S1 $\mathcal{S}$ uniformly chooses a string $r_1 \in_U \{0,1\}^K$, commits to it $\gamma_1 \leftarrow \mathsf{HC}(r_1)$ and sends γ_1.
R2 $\mathcal{R}$ computes similarly, $r_2 \in_U \{0,1\}^K$, $\gamma_2 \leftarrow \mathsf{BC}(r_2)$ and sends γ_2.
S2/R3 $\mathcal{S}$ ($\mathcal{R}$, respectively) sends the de-commitment to γ_1 (γ_2, respectively).
S3/R4 $\mathcal{S}$ and $\mathcal{R}$ compute $r \leftarrow r_1 \oplus r_2$ respectively and share this bit-string.

Error Handling. In above steps, $\mathcal{R}$ and $\mathcal{S}$ respectively checks that the received commitments and de-commitments are legal. $\mathcal{S}$ additionally verifies that the hash parameter is legal by calling $\mathsf{Check}(1^k, \Lambda)$. If one check fails, $\mathcal{S}$ ($\mathcal{R}$, respectively) halts and outputs Abort_2 (Abort_1, respectively).

R5 (Receiver's step): According to choice indicator, $\mathcal{R}$ sends the witnesses of the chosen vectors and instructs $\mathcal{S}$ to rearrange each unchosen vector. For convenience later, we let $C = \{i : r\langle i\rangle = 1\}$ be the set of indices of chosen instance vectors and call C *the choice set defined by* r.

1. For each unchosen vector $\tilde{\vec{x}}_i (i \notin C)$, $\mathcal{R}$ prepares a *rearrangement indicator* b_i, where $b_i \leftarrow 1$ if $\tilde{\vec{x}}_i\langle o\rangle$ is projective; $b_i \leftarrow 0$ otherwise.
2. For each chosen vector $\tilde{\vec{x}}_i (i \in C)$, $\mathcal{R}$ prepares the witness $\tilde{\vec{w}}_i$.
3. $\mathcal{R}$ sends the witnesses $\{\tilde{\vec{w}}_i : i \in C\}$ and rearrangement indicators $\{b_i : i \notin C\}$.

Comment: The value of b_i tells whether $\mathcal{S}$ should exchange the two entries of $\tilde{\vec{x}}_i$.
S4 (Sender's step): $\mathcal{S}$ checks the legality of the chosen instance vectors and sends the encryption of $\vec{m}$ to $\mathcal{R}$ via the following instructions.

1. $\mathcal{S}$ verifies that each chosen instance vector is legal by checking if it contains at least one smooth instance. Knowing the witness vectors $\{\tilde{\vec{w}}_i\}_{i\in C}$, $\mathcal{R}$ invokes algorithm DI to check the validity of instance vectors $\{\tilde{\vec{x}}_i\}_{i\in C}$. If $\mathcal{R}$ did not send the witness vectors or if the check fails, then $\mathcal{S}$ halts and outputs Abort_2; otherwise $\mathcal{S}$ proceeds to the next step.
2. $\mathcal{S}$ rearranges each non-chosen instance vector: $\forall i \notin C$, exchanges the two entries of $\tilde{\vec{x}}_i$ if $b_i = 0$. Let $\tilde{\tilde{\vec{x}}}_i$ denote each resultant vector.
3. $\mathcal{S}$ encrypts the value $\vec{m} = (m_1, m_2)$. That is, for each $i \notin C$ and for each $j \in \{1,2\}$, $(hk_{ij}, pk_{ij}) \leftarrow \mathsf{KG}(1^k, \Lambda, \tilde{\tilde{\vec{x}}}_i\langle j\rangle), \beta_{ij} \leftarrow \mathsf{Hash}(1^k, \Lambda, \tilde{\tilde{\vec{x}}}_i\langle j\rangle, hk_{ij}), \vec{\beta}_i \stackrel{def}{=} (\beta_{i1}, \beta_{i2}), \vec{c} \leftarrow \vec{m} \oplus (\oplus_{i\notin C}\vec{\beta}_i), \overrightarrow{pk}_i \stackrel{def}{=} (pk_{i1}, pk_{i2})$.
4. $\mathcal{S}$ sends the encryption of $\vec{m}$ and the projection keys, $(\vec{c}, (\overrightarrow{pk}_i)_{i\notin C})$, to $\mathcal{R}$.

R6 (Receiver's step) $\mathcal{R}$ decrypts $\vec{c}$. That is, for each $i \notin C$, the receiver computes $\beta'_{io} \leftarrow \mathsf{pHash}(1^k, \Lambda, \tilde{\tilde{\vec{x}}}_i\langle o\rangle, \overrightarrow{pk}_i\langle o\rangle, \tilde{\tilde{\vec{w}}}_i\langle o\rangle), m'_o \leftarrow \vec{c}\langle o\rangle \oplus (\oplus_{i\notin C}\beta'_{io})$. Finally, $\mathcal{R}$ recovers the values m'_o.

We defer the strict proof of Theorem 29 to Section 6 and simply give an intuitive analysis here. First, let's focus on the sender's security. The framework should prevent any malicious adversary controlling the corrupted receiver from learning more than 1 value. This is achieved by using a cut and choose technique. The following lemma shows that the probability that the malicious adversaries learn extra values is negligible.

Lemma 30. *In the case where the sender is honest and the receiver is corrupted by a malicious adversary, the probability that the adversary learns more than one value is at most* $1/2^K$.

Proof. Let $\mathcal{A}$ be a malicious adversary. According to the design of Π, the following conditions are necessary for $\mathcal{A}$ to learn any extra value.

1. $\mathcal{A}$ has to generate at least one illegal instance vector containing more than one projective instance. If not, $\mathcal{A}$ cannot correctly decrypt more than one encryptions due to the smoothness of $\mathcal{H}$. Without loss of generality, we assume the illegal instance vectors are $\vec{x}_{\ell_1}, \vec{x}_{\ell_2}, \ldots, \vec{x}_{\ell_d}$.
2. All illegal instance vectors are lucky not to be chosen by the sender and all the unchosen instance vectors are just the illegal instance vectors: i.e., $\overline{C} = \{\ell_1, \ell_2, \ldots, \ell_d\}$. We prove this claim in the following two cases.
   (a) If $\overline{C} \subset \{\ell_1, \ell_2, \ldots, \ell_d\}$, then there exists an illegal instance vector chosen by the sender who detects the cheating and $\mathcal{A}$ gains nothing.
   (b) If $\overline{C} \supset \{\ell_1, \ell_2, \ldots, \ell_d\}$, then there exists a legal instance vector not chosen by the sender. Following the instructions of Π, this instance vector is used to encrypt the values (m_1, m_2). Looking at Step S4, the final encryptions are gained by XOR-ing the two values and the hash values of all unchosen instance vectors. Because a legal instance vector holds at least one smooth instances, at least one encryptions statistically hide their encrypted values. As a result, $\mathcal{A}$ knows almost nothing about at least one encrypted values.

Now, let us estimate the probability that the second necessary condition is met. Since $\mathcal{S}$ is honest, r_1 is uniformly distributed. Since HC is perfectly hiding, adversary $\mathcal{A}$ learns nothing about r_1 before the reveal phase. Perfectly binding of BC guarantees that $\mathcal{A}$ can not open its commitment to be a value different from r_2. So the choice indicator r is uniformly distributed. We have:

$$\text{Prob}(\overline{C} = \{\ell_1, \ell_2, \ldots, \ell_d\}) = (1/2)^d (1/2)^{K-d} = 1/2^K.$$

This means that the probability that $\mathcal{A}$ cheats to learn more than one value is at most $1/2^K$. □

Second, consider the receiver's security. It is necessary for Π to prevent any malicious adversary controlling the corrupted sender from learning which values the receiver chose. Intuitively, there might be a possible information leakage from rearrangement indicators b_i of $\mathcal{R}$. Since the o-th entry of $\tilde{\tilde{\vec{x}}}_i$ is projective for each $i \notin C$, learning the receiver's private input o means identifying a projective entry of some $\tilde{\tilde{\vec{x}}}_i$. Since this is OT_1^2, $\mathcal{A}$ indeed can guess o with probability $1/2$. However, $\mathcal{H}$'s hard subset membership property guarantees that $\tilde{\tilde{\vec{x}}}_i$'s projective instances and smooth instances are computationally indistinguishable. As a result, the probability of $\mathcal{A}$ identifying a projective entry of $\tilde{\tilde{\vec{x}}}_i$ is negligibly greater than $1/2$. This guarantees that $\mathcal{A}$ does not have any better algorithms than random guessing to learn o.

Another cheating strategy that $\mathcal{A}$ can follow is to send invalid messages. If $\mathcal{R}$ cannot process these messages (e.g., the messages are malformed), then $\mathcal{R}$ detects a cheating tentative and aborts. If $\mathcal{R}$ can process them nonetheless, this can be viewed as $\mathcal{A}$ altering its private values. This has no significance whatsoever since in the ideal world $\mathcal{A}$ is also allowed to alter its input before sending it to the TTP. In a word, this cheating approach is not effective and $\mathcal{A}$ cannot gain any extra knowledge following this strategy.

5.2.3. Efficiency of the Framework

It is clear that the receiver's Step R3-S5 can be executed in one round while the sender's Step S3-S4 can be executed in one round too. As Step R6 can be performed without communication, Π needs 6 communication rounds .

Abstractly, we can see an invocation of Hash as a call to an encryption algorithm and an invocation of pHash as a request to the corresponding decryption algorithm. Such a consideration is justified since, in Π, Hash plays an encrypting role to hide $\mathcal{S}$'s values while pHash can be considered as decryption machine recovering the values that

$\mathcal{R}$ wants. This parallel to a cryptosystem is not fortuitous as the first usage of a SPH was to construct public-key encryption schemes [36].

Looking at the generation of choice indicator, it is easy to see that the expected number of chosen instance vector is $K/2$. Therefore, the main expected computational overhead is K encryptions at the sender (Step S4) and $K/2$ decryptions at the receiver (Step R6).

Since K determines the computational overhead of Π, a natural question is how to set it in practice. In the proof of Theorem 29, the simulator $\mathfrak{S}$ does not simulate the case where the adversary learns more than one values. This is justified, because Lemma 30 shows this case arises with negligible probability $1/2^K$ (see Remark 40 for the details). Thus, we must set K to be a value so that $1/2^K$ is small enough to be negligible in the security parameter.

In [20], it is said that an error of $1/2^{40} \approx 9.09 \times 10^{-13}$ would be negligible in practice. So, we set $K = 40$ for our purpose. It follows that, in a practical point of view, the main expected computational overhead is 40 encryptions and 20 decryptions.

Remark 31. *How to set the security parameter k or the statistical security parameter K in practice? This is essentially the question of bridging theoretical cryptography and practice in order to translate a guarantee of asymptotic security into a concrete security one. The task of determining the value of the (statistical) security parameter to use, generally speaking, is complex and depends on the scheme in question as well as other considerations. This problem is discussed in [29, 37] for instance but no concrete standard solution is suggested.*

6. Formal Security Proof

This section is dedicated to the demonstration of Theorem 29. To distinguish the entities in two worlds, we denote the sender, the receiver and the adversary in the real world by $\mathcal{S}$, $\mathcal{R}$, $\mathcal{A}$ and their corresponding entities in the ideal world by $\mathcal{S}'$, $\mathcal{R}'$, $\mathfrak{S}$.

Based on the parties corrupted by $\mathcal{A}$, there are four cases to be considered: only $\mathcal{S}$ is corrupted, only $\mathcal{R}$ is corrupted, both parties are corrupted, no party is corrupted. Because the security proofs of the last two cases are trivial, we omit them to save space.

6.1. Only $\mathcal{S}$ is Corrupted

In this case, $\mathcal{A}$ takes the full control of $\mathcal{S}$ in the real world. Correspondingly, $\mathcal{A}$'s simulator $\mathfrak{S}$ takes the full control of $\mathcal{S}'$ in the ideal world, where $\mathfrak{S}$ is constructed as described in Algorithm 2.

Without considering Step Sim3, $\mathfrak{S}$ is polynomial-time. However, taking Step Sim3 into consideration, this is not true any more. Let p_1, p_2 respectively denotes the probability that $\overline{\mathcal{A}}$ correctly reveals its commitment first time (in Step Sim2) and second time (in Step Sim3). Since $\mathfrak{S}$ commits to a random value $r_2 \in_U \{0,1\}^K$ in Step Sim2 while commits to value $r_2 \leftarrow r_1 \oplus r$ in Step Sim3, the views $\overline{\mathcal{A}}$ holds before revealing his commitment in the two times are distinct. Thus, p_1, p_2 are distinct too.

The expected number of times of rewinding in Step Sim3 is

$$p_1/p_2.$$

The computational hiding of BC only guarantees that $|p_1 - p_2 \leq \mu(k)|$ rather than $p_1/p_2 \leq \mu(k)$. What is worse, there is a risk that p_1/p_2 is not bound by a polynomial. For example, $p_1 = 2^k$, $p_2 = 2^{2k}$, which result in $p_1/p_2 = 2^k$. This is a big problem and gives rise to many other difficulties we will encounter later.

Fortunately, [35] encounters and solves a problem similar to ours. Using the idea of [35], we can overcome our problem too. See Algorithm 3 for the *modified simulator*.

Proposition 32. *The modified simulator $\mathfrak{S}$ runs in expected polynomial-time.*

Proof. It is easy to see that Step $\mathsf{Sim3.1}$ and $\mathsf{Sim3.2}$ expectedly cost at most $g(k)/p_1$ and $s(k)/\tilde{p}_1$ rewinding. Let T_i denote the running time of the i-th Step of the simulator and T_* the the running time of one rewinding in subroutine Getr. The expected running time $\tau(k)$ of modified simulator is

$$\begin{aligned} \tau(k) &\leq (1-p_1)(\mathsf{T}_1 + \mathsf{T}_2) + p_1[\mathsf{T}_1 + \mathsf{T}_2 + \mathsf{T}_* \cdot g(k)/p_1 + \mathsf{T}_* \cdot s(k)/\tilde{p}_1] \\ &= \mathsf{T}_1 + \mathsf{T}_2 + \mathsf{T}_* g(k) + \mathsf{T}_* s(k) \cdot \frac{p_1}{\tilde{p}_1} \end{aligned} \tag{2}$$

Algorithm 2 Simulator $\mathfrak{S}$ when $\mathcal{S}$ is the only Corrupted Player

Input: The security parameter k, the statistical security parameter K, the auxiliary input z_k, the name list $I = \{1\}$ (recall that the value 1 is associated to the sender) and a uniformly distributed randomness tape $r_{\mathfrak{S}} \in \{0,1\}^*$.

Sim1 (Initialization step):

1. $\mathfrak{S}$ corrupts $\mathcal{S}'$ and learns $\mathcal{S}'$'s private input $\vec{m} = (m_1, m_2)$.
2. $\mathfrak{S}$ takes $\overline{\mathcal{A}}$, a copy of $\mathcal{A}$, as a subroutine. $\mathfrak{S}$ sets the initial inputs of $\overline{\mathcal{A}}$ as k, I, z_k and $r_{\overline{\mathcal{A}}}$ with $r_{\overline{\mathcal{A}}} \in_U \{0,1\}^*$.
3. $\mathfrak{S}$ activates $\overline{\mathcal{A}}$. As in the real world, $\overline{\mathcal{A}}$ sends a message to corrupt the sender. $\mathfrak{S}$ plays the role of $\mathcal{S}$ and supplies $\overline{\mathcal{A}}$ with $\vec{m}$.

Comment: To simulate the environment that $\overline{\mathcal{A}}$ can see in the real world, $\mathfrak{S}$ simultaneously plays the roles who $\overline{\mathcal{A}}$ can interact with in the real world.

Sim2: Playing the role of $\mathcal{R}$, $\mathfrak{S}$ follows the receiver's Step R1 of Π except it samples instances in this different way:

1. $\mathfrak{S}$ uniformly chooses a choice indicator $r \in_U \{0,1\}^K$.
2. Let C be the set defined by r. For each $i \in C$, $\mathfrak{S}$ honestly samples vector $\vec{a}_i$. For each $i \notin C$, $\mathfrak{S}$ samples a projective vector $\vec{a}_i \leftarrow \mathsf{pIS}(1^k, \Lambda)$.

Sim2: $\mathfrak{S}$ honestly follows the receiver's Step R2-R4 and learns the value r_1 committed by $\overline{\mathcal{A}}$.

Sim3: $\mathfrak{S}$ repeats the following procedure until $\overline{\mathcal{A}}$ correctly open its committed value to be r_1 again.

- $\mathfrak{S}$ rewinds $\overline{\mathcal{A}}$ to the state that $\overline{\mathcal{A}}$ has sent its commitment γ_1 and is waiting for a commitment γ_2 from a receiver, i.e., the end of Step S1 of the framework.
- $\mathfrak{S}$ follows receiver's Step R2-R4 except that it sets $r_2 \leftarrow r_1 \oplus r$ and uses fresh randomness to compute $\mathsf{BC}(r_2)$ in each repetition.

Comment: If Step Sim3 succeeds, then choice indicator r chosen by $\mathfrak{S}$ is agreed by $\overline{\mathcal{A}}$ too. If this is the case, $\mathfrak{S}$ will pass legality checks by $\overline{\mathcal{A}}$. Further, $\mathfrak{S}$ can decrypt all values encrypted by $\overline{\mathcal{A}}$.

Sim4: $\mathfrak{S}$ honestly follows receiver's Step R5. On receiving $(\vec{c}, (\overrightarrow{pk}_i)_{i \notin C})$ from $\overline{\mathcal{A}}$, $\mathfrak{S}$ correctly decrypts all entries of $\vec{c}$ and gains $\overline{\mathcal{A}}$'s full real private input $\tilde{\vec{m}}$.

Sim5 (Output step): $\mathfrak{S}$ sends $\tilde{\vec{m}}$ to the TTP and receives λ in return. Then, $\mathfrak{S}$ sends Continue to the TTP. When $\overline{\mathcal{A}}$ halts, $\mathfrak{S}$ outputs what $\overline{\mathcal{A}}$ returned.

Error Handling. Throughout this algorithm, if $\overline{\mathcal{A}}$ refuses to send some message it is supposed to send or $\overline{\mathcal{A}}$ sends an invalid message that $\mathfrak{S}$ cannot process, unless specified, the simulator $\mathfrak{S}$ sends Abort_1 to the TTP and halts with outputting whatever $\overline{\mathcal{A}}$ returns.

Algorithm 3 Modified Simulator $\mathfrak{S}$ when $\mathcal{S}$ is the only Corrupted Player

Comment: We modify Algorithm 2 by replacing Step Sim3 with two new sub-steps: Step Sim3.1 and Step Sim3.2, which invoke a common subroutine Getr(x, l).
Subroutine Getr(y, l): Simulator $\mathfrak{S}$ rewinds $\overline{\mathcal{A}}$ until it de-commits to r_1, where parameter y tells the procedure to repeat Step Sim2 or original Step Sim3, parameter l tells the procedure the upperbound number of repeating original Step Sim3.

1. If y = "Step Sim3" and the number of rewinding exceeds l, then returns 1.
2. Rewinds $\overline{\mathcal{A}}$ to the state that $\overline{\mathcal{A}}$ has sent its commitment γ_1 and is waiting for a commitment γ_2 from a receiver, i.e., the end of Step S1 of the framework.
3. $\mathfrak{S}$ executes receiver's Step R2-R4 except that
   - Case y = "Step Sim2": sets $r_2 \in_U \{0,1\}^K$.
   - Case y = "Step Sim3": sets $r_2 \leftarrow r_1 \oplus r$.

   What is more, $\mathfrak{S}$ uses *fresh randomness* to computes commitment BC(r_2) in each repetition.
4. According to $\overline{\mathcal{A}}$'s distinct responses, $\mathfrak{S}$ executes in distinct ways:
   - If $\overline{\mathcal{A}}$ does not open its commitment correctly, then $\mathfrak{S}$ goes to next repetition, i.e., go to Step 1 of this procedure.
   - If $\overline{\mathcal{A}}$ correctly reveals a value $r_1' \neq r_1$ different from r_1, $\mathfrak{S}$ outputs Ambiguity and halts.
   - If $\overline{\mathcal{A}}$ correctly reveals the value r_1, this procedure returns 0.

Sim3.1: $\mathfrak{S}$ estimates the value of p_1. Let $g(k)$ be a sufficiently large polynomial. $\mathfrak{S}$ invokes Procedure Getr("Step Sim2", 0) $g(k)$ times, i.e., rewinds $\overline{\mathcal{A}}$ until it de-commits to r_1 $g(k)$ times. $\mathcal{S}$ estimates the value of p_1 as

$$\tilde{p_1} = \frac{g(k)}{h},$$

where h is the number of rewinding $\overline{\mathcal{A}}$.
Sim3.2: Let $s(k)$ be another a sufficiently large polynomial. $\mathcal{S}$ invokes Procedure Getr("Step Sim3", $s(k)/\tilde{p_1}$) to get a de-commitment to r_1, i.e., rewinds $\overline{\mathcal{A}}$ at most $s(k)/\tilde{p_1}$ times to get a de-commitment to r_1. If $\overline{\mathcal{A}}$ does not correctly de-commits to r_1, $\mathcal{S}$ halts with outputting Timeout.

Since $\mathsf{T}_1, \mathsf{T}_2, \mathsf{T}_*, g(k), s(k)$ are polynomials, it remains to consider the ratio $p_1/\tilde{p}_1$. Let's focus on subroutine Getr and denote by Y_i a binary random variable that equals 1 if $\overline{\mathcal{A}}$ correctly reveals the value r_1 in the i-th repetition and equals 0 otherwise. Let $0 \leq \delta \leq 1$. We have:

$$\begin{aligned}
\text{Prob}\left(\frac{p_1}{\tilde{p}_1} > \frac{1}{1-\delta}\right) &= \text{Prob}\left(\frac{p_1}{\sum_{i=1}^{h} Y_i/h} > \frac{1}{1-\delta}\right) \\
&= \text{Prob}\left(\sum_{i=1}^{h} Y_i < (1-\delta)hp_1\right) \\
&\leq \frac{1}{e^{\delta^2 hp_1/2}}
\end{aligned}$$

where the right hand side inequality follows the Chernoff bound [38]. Setting $\delta = 1/2$, we get:

$$\text{Prob}\left(\frac{p_1}{\tilde{p}_1} > 2\right) \leq \frac{1}{e^{hp_1/8}} \leq \frac{1}{e^{g(k)\,p_1/8}}, \tag{3}$$

where the rightmost inequality follows the fact $g(k) \leq h$. This shows that $\frac{p_1}{\tilde{p}_1} > 2$ holds with at most negligible probability in k.

Combining Inequality (2) and Inequality (3), it is easy to see that the modified simulator runs in expected polynomial-time. □

Lemma 33. *The modified simulator* $\mathfrak{S}$ *outputs* Ambiguity *with (at most) negligible probability.*

Proof. Intuitively, this lemma guaranteed by computational binding of HC. The subtlety is that $\mathfrak{S}$ runs in expected polynomial-time, whereas computational binding is secure against strictly polynomial-time attacker.

We employ truncation technique to solve this subtlety. Specifically, assume by contraction that Ambiguity is output with probability $> 1/\mathsf{Poly}(k)$. We truncate the executions of $\mathfrak{S}$ when its running time exceeds $2\mathsf{Poly}(k) \cdot \tau(k)$, where $\tau(k)$ is the expected running time of $\mathfrak{S}$.

Following Markov's inequality, we know the statistical distance between non-truncated $\mathfrak{S}$ and truncated $\mathfrak{S}$ is at most $1/2\mathsf{Poly}(k)$. Thus the truncated $\mathfrak{S}$ outputs Ambiguity with probability $> 1/2\mathsf{Poly}(k)$. Since truncated $\mathfrak{S}$ is strictly polynomial-time, this contradicts computational binding of HC. □

Lemma 34. *The modified simulator* $\mathfrak{S}$ *outputs* Timeout *with (at most) negligible probability.*

Proof. We introduce the following probabilistic event notations:

- $\mathsf{TO} \stackrel{def}{=} \{\mathfrak{S} \text{ outputs Timeout}\}$,

Case 1: p_1 is negligible. In this case, $\mathfrak{S}$ reaches Step Sim3.2 at most negligible probability. Thus, the lemma holds.

Case 2: p_1 is not negligible. We have:

$$\text{Prob}(\mathsf{TO}) = p_1(1-p_2)^{s(k)/\tilde{p}_1}.$$

Case 2.1: $p_2 \geq p_1/2$. We have:

$$\text{Prob}(\mathsf{TO}) \leq (1-p_2)^{s(k)/\tilde{p}_1} \leq (1-p_1/2)^{s(k)/\tilde{p}_1} = [(1-\frac{p_1}{2})^{\frac{2}{p_1}}]^{\frac{p_1}{2}\frac{s(k)}{\tilde{p}_1}} \leq e^{\frac{p_1}{\tilde{p}_1}\frac{s(k)}{2}}.$$

$s(k)$ is a polynomial and it is shown in the proof of Lemma 32 that $\frac{p_1}{\tilde{p}_1} > 2$ is negligible. It follows that Prob(TO) drops exponentially in k and the lemma holds.

Case 2.2: $p_2 < p_1/2$. We have

$$|p_1 - p_2| > p_1/2, \tag{4}$$

Assume by contraction that $\text{Prob}(\mathsf{TO}) > 1/\mathsf{Poly}(k)$. Since $(1-p_2)^{s(k)/\tilde{p}_1} < 1$, we have

$$p_1 > 1/\mathsf{Poly}(k). \tag{5}$$

Combining Inequality (4) (5), we have

$$|p_1 - p_2| > 1/2\mathsf{Poly}(k).$$

Intuitively, this contradicts computational hiding of BC. Since $\mathfrak{S}$ is not strictly polynomial-time, as in the proof of Lemma 33 we should employ truncation technique again and formally prove this intuition. We omit the details. □

Proposition 35. *In the case where only the sender is corrupted, Equation* (1) *in Definition 9 holds.*

Proof. Let's focus on the real world. Because $\mathcal{R}$ is honest, its output is a deterministic function of $\mathcal{A}$'s real private input. Without loss of generality, we assume that $\mathcal{A}$'s output, denoted by $\alpha = \mathsf{Real}_{\Pi,\{1\},\mathcal{A}(z_k)}(1^k, \vec{m}, T)\langle 0\rangle$, contains its real private input, denoted by γ. Therefore, $\mathcal{R}$'s output is a deterministic function of α, where the function is:

$$g(\alpha) = \begin{cases} \mathsf{Abort}_1 & \text{if } \gamma \text{ causes } \mathcal{R} \text{ to output } \mathsf{Abort}_1, \\ \gamma\langle o\rangle & \text{if } \gamma \text{ is a message of form } (m_1, m_2). \end{cases}$$

It is easy to see that the output of the real world is a deterministic function of α. Specifically,

$$\mathsf{Real}_{\Pi,\{1\},\mathcal{A}(z_k)}(1^k, \vec{m}, T) \equiv (\alpha, \lambda, g(\alpha)). \tag{6}$$

Now, we concentrate our attention to the ideal world. We claim that $\mathfrak{S}$'s output, denoted by $\beta = \mathsf{Ideal}_{f,\{1\},\mathfrak{S}(z_k)}(1^k, \vec{m}, T)\langle 0\rangle$, and $\mathcal{A}$'s output are computationally indistinguishable:

$$\alpha \stackrel{c}{=} \beta. \tag{7}$$

Note that $\mathfrak{S}$ almost always takes $\overline{\mathcal{A}}$'s output as its own and, with at most negligible probability, it takes $\mathsf{Timeout}$ or $\mathsf{Ambiguity}$ as its own output (See Lemma34 and 33). Therefore, the claim holds.

Similarly to the real world, we argue as follows that $\mathcal{R}'$'s output is a deterministic function g of $\mathfrak{S}$'s output β.

1. Case 1: $\overline{\mathcal{A}}$ does not correctly reveal r_1 in Step $\mathsf{Sim2}$. $\mathfrak{S}$ outputs Abort_1 and $\mathcal{R}'$ outputs Abort_1 (i.e., $g(\beta)$).
2. Case 2: $\overline{\mathcal{A}}$ correctly reveals r_1 in Step $\mathsf{Sim2}$. Following Lemma34 and 33, we do not need to consider the case $\mathfrak{S}$ outputs $\mathsf{Timeout}$ or or $\mathsf{Ambiguity}$. It remains to consider the case $\mathfrak{S}$ successfully extracts the real input of $\overline{\mathcal{A}}$. In this case, β contains the real input of $\overline{\mathcal{A}}$ and $\mathcal{R}'$ outputs the message it desires (i.e., $g(\beta)$).

Therefore, similarly to the real word, the following equation holds.

$$\mathsf{Ideal}_{f,\{1\},\mathfrak{S}(z_k)}(1^k, \vec{m}, T) \stackrel{c}{=} (\beta, \lambda, g(\beta)). \tag{8}$$

Combining Equations (6), (7), (8), we deduce that Equation (1) holds which proves our proposition. □

6.2. Only $\mathcal{R}$ is Corrupted

In this case, $\mathcal{A}$ takes the full control of $\mathcal{R}$ in the real world. Correspondingly, the simulator $\mathfrak{S}$ takes the full control of $\mathcal{R}'$ in the ideal world. It works as represented in Algorithm 4 where we adopted the same policy as before concerning the incorrect message transmission behavior of $\overline{\mathcal{A}}$.

Remark 36. *We stress the fact that o' is one of $\overline{\mathcal{A}}$'s possible private inputs as distinct rearrangement indicators may correspond to distinct private inputs. However, as we will see in the proof of Lemma 39, this inconsistence does not affect the correctness of our simulation.*

Proposition 37. *The simulator $\mathfrak{S}$ runs in expected polynomial-time.*

Algorithm 4 Simulator $\mathfrak{S}$ when $\mathcal{R}$ is the only Corrupted Player

Input: The security parameter k, the statistical security parameter K, the auxiliary input z_k, the name list $I = \{2\}$ (recall that the value 2 is associated to the receiver) and a uniformly distributed randomness tape $r_{\mathfrak{S}} \in \{0,1\}^*$.

Sim1 (Initialization step):

1. $\mathfrak{S}$ corrupts $\mathcal{R}'$ and learns $\mathcal{R}'$'s private input o.
2. $\mathfrak{S}$ takes $\overline{\mathcal{A}}$, a copy of $\mathcal{A}$, as a subroutine and sets the initial inputs of $\overline{\mathcal{A}}$ as k, I, z_k and $r_{\overline{\mathcal{A}}}$ with $r_{\overline{\mathcal{A}}} \in_U \{0,1\}^*$.
3. $\mathfrak{S}$ activates $\overline{\mathcal{A}}$.
4. As in the real world, $\overline{\mathcal{A}}$ sends a message to corrupt the receiver. $\mathfrak{S}$ plays the role of $\mathcal{R}$ and supplies $\overline{\mathcal{A}}$ with o.

Sim2: Playing the role of $\mathcal{S}$, the simulator $\mathfrak{S}$ honestly executes sender's Step S1-S4.2 of Π. If $\overline{\mathcal{A}}$ cause $\mathfrak{S}$ to output Abort_2, then $\mathfrak{S}$ halts outputting what $\overline{\mathcal{A}}$ returned. Otherwise, $\mathfrak{S}$ stores the current state ζ of $\overline{\mathcal{A}}$ and received messages.

Comment: Now $\mathfrak{S}$ and $\overline{\mathcal{A}}$ have negotiate a choice indicator r. Let C be the set determined by it.

Sim3: The simulator repeats the following procedure Ξ until $\overline{\mathcal{A}}$ does not cause $\mathfrak{S}$ to output Abort_2. $\mathfrak{S}$ records received messages in the last repetition.

1. $\mathfrak{S}$ rewinds $\overline{\mathcal{A}}$ to state that it has sent hash parameter and instance vectors and is waiting for a commitment from a sender, i.e., the beginning of Step R2 of Π.
2. Playing the role of $\mathcal{S}$, the simulator $\mathfrak{S}$ honestly follows Step S1-S4.2 to interact with $\overline{\mathcal{A}}$. In each iteration, $\mathfrak{S}$ uses fresh randomness.

Comment: Now $\mathfrak{S}$ and $\overline{\mathcal{A}}$ have negotiated a new choice indicator $\tilde{r}$. Let $\tilde{C}$ be the set determined by $\tilde{r}$.

Sim4:

- Case 1: $C = \tilde{C}$. The simulator $\mathfrak{S}$ outputs Failure and halts.
- Case 2: $C \neq \tilde{C}$.
  - Case 2.1: $\tilde{C} - C = \emptyset$. The simulator is reset to the beginning of Step Sim1 with a new randomness $r_{\mathfrak{S}} \in_U \{0,1\}^*$.
  - Case 2.2: $\tilde{C} - C \neq \emptyset$. The simulator $\mathfrak{S}$ arbitrarily chooses one element $e \in \tilde{C} - C$. Then, it proceeds to the next step.

Comment: Let us focus on Case 2.2. Since $e \in \tilde{C}$, we know that $\mathfrak{S}$ receives the witness vector $\tilde{\vec{w}}_e$ and $\tilde{\vec{x}}_e$ passes the legality check in Step Sim3. Since $e \notin C$, we know that $\mathfrak{S}$ has received instance vector $\tilde{\vec{x}}_e$ and rearrangement indicator b_e in Step Sim2. Based on $(\tilde{\vec{x}}_e, \tilde{\vec{w}}_e, b_e)$, $\mathfrak{S}$ can easily deduce $\overline{\mathcal{A}}$'s private input.

Sim5: The simulator restores $\overline{\mathcal{A}}$ to the state ζ. Based on $(\tilde{\vec{x}}_e, \tilde{\vec{w}}_e, b_e)$, $\mathfrak{S}$ deduces private input o' of $\overline{\mathcal{A}}$ of state ζ.

Sim6: The simulator sends o' to the TTP and receives $\vec{m}\langle o' \rangle$ in return. Then, $\mathfrak{S}$ sends Continue to the TTP. $\mathfrak{S}$ builds a value vector $\vec{m}'$ as follows: sets $\vec{m}'\langle o' \rangle \leftarrow \vec{m}\langle o' \rangle$ and set $\vec{m}'\langle 3 - o' \rangle$ to be a value uniformly chosen from G_Λ, where G_Λ is a set of all possible hash values (see Definition 25).

Sim7 (Output step): Playing the role of $\mathcal{S}$ with private input $\vec{m}'$, $\mathfrak{S}$ follows Step S4.3-S4.4 of Π to complete the interaction with $\overline{\mathcal{A}}$. When $\overline{\mathcal{A}}$ halts, $\mathfrak{S}$ outputs what $\overline{\mathcal{A}}$ returned.

Proof. First, let's focus on Step Sim3. In each repetition of Ξ, the views of $\overline{\mathcal{A}}$ are identically distributed because of perfectly hiding of HC. This leads to the probability to be the same that $\overline{\mathcal{A}}$ does not cause $\mathfrak{S}$ to output Abort_2. We denote this value by p. Let τ_3 and T_Ξ denote expected the running time of Step Sim3 and the running time of Ξ respectively. We have:

$$\tau_3 = (1/p)\,\mathsf{T}_\Xi.$$

For the same reason, the probability that $\overline{\mathcal{A}}$ does not cause $\mathfrak{S}$ to output Abort_2 in Step Sim2 is p as well. Let τ_{1-3} denote the expected running time of one run of the sequence Step Sim1 to Step Sim3. Let T_j denote the running time of $\mathfrak{S}$'s j-th step. We have:

$$\tau_{1-3} \leq \mathsf{T}_1 + \mathsf{T}_2 + p\,\tau_3 \leq \mathsf{T}_1 + \mathsf{T}_2 + \mathsf{T}_\Xi$$

Second, consider Step Sim4 and especially Case 2.1. Note that the initial inputs of $\mathfrak{S}$ (apart from the refreshed randomness) remain the same in each iteration. Thus, the probability that $\mathfrak{S}$ runs from scratch in each iteration is the same. We denote this probability by $1 - q$. As a consequence, the expected running time of the sequence Step Sim1 to Step Sim4, denoted by τ_{1-4}, is:

$$\begin{aligned}\tau_{1-4} &\leq (1 + 1/q)\,(\tau_{1-3} + \mathsf{T}_4)\\ &\leq (1 + 1/q)\,(\mathsf{T}_1 + \mathsf{T}_2 + \mathsf{T}_\Xi + \mathsf{T}_4).\end{aligned}$$

The reason why there is 1 in the term $(1 + 1/q)$ is that $\mathfrak{S}$ has to run from scratch at least one time in any case. Thus, the simulator's expected running time, denoted by $\tau_{\mathfrak{S}}$, can be upper bounded as:

$$\begin{aligned}\tau_{\mathfrak{S}} &\leq \tau_{1-4} + \mathsf{T}_5 + \mathsf{T}_6 + \mathsf{T}_7\\ &\leq (1 + 1/q)\,(\mathsf{T}_1 + \mathsf{T}_2 + \mathsf{T}_\Xi + \mathsf{T}_4) + \mathsf{T}_5 + \mathsf{T}_6 + \mathsf{T}_7.\end{aligned} \tag{9}$$

Third, let's estimate the value of q being the probability of the event {$\mathfrak{S}$ does not run from scratch in an iteration}. It is easy to see that this event happens if and only if one of the following mutually disjoint events happens.

1. Event {$\mathfrak{S}$ halts before reaching Step Sim4} happens. We denote this event by $\mathcal{E}$.
2. Event $\overline{\mathcal{E}}$ happens and $R = \tilde{R}$ (i.e., $\mathfrak{S}$ reaches Case 1 of Step Sim4), where R (resp., $\tilde{R}$) is the random variable defined as the choice indicator recorded by $\mathfrak{S}$ in Step Sim2 (resp., Step Sim3).
3. Event $\overline{\mathcal{E}}$ happens and $\exists i\ :\ R\langle i\rangle = 0$ and $\tilde{R}\langle i\rangle = 1$, i.e., $\mathfrak{S}$ reaches Case 2.2 of Step Sim4.

Based on this partition of {$\mathfrak{S}$ does not run from scratch in an iteration}, we obtain:

$$\begin{aligned}q =&\mathrm{Prob}(\mathcal{E}) + \mathrm{Prob}(\overline{\mathcal{E}} \cap \{R = \tilde{R}\})\\ &+ \mathrm{Prob}(\overline{\mathcal{E}} \cap \{\exists i\ :\ R\langle i\rangle = 0 \text{ and } \tilde{R}\langle i\rangle = 1\})\\ =&\mathrm{Prob}(\mathcal{E}) + \mathrm{Prob}(\overline{\mathcal{E}})\,[\mathrm{Prob}(\{R = \tilde{R}\}|\overline{\mathcal{E}})\\ &+ \mathrm{Prob}(\{\exists i\ :\ R\langle i\rangle = 0 \text{ and } \tilde{R}\langle i\rangle = 1\}|\overline{\mathcal{E}})].\end{aligned} \tag{10}$$

We introduce the following notations to denote probabilistic events:

- U_1 represents $\{(r, \tilde{r}) : (r, \tilde{r}) \in (\{0,1\}^K)^2 \text{ and } r = \tilde{r}\}$,
- U_2 represents $\{(r, \tilde{r}) : (r, \tilde{r}) \in (\{0,1\}^K)^2 \text{ and } r \neq \tilde{r} \text{ and } \neg\exists i\,(r\langle i\rangle = 0 \text{ and } \tilde{r}\langle i\rangle = 1)\}$,
- U_3 represents $\{(r, \tilde{r}) : (r, \tilde{r}) \in (\{0,1\}^K)^2 \text{ and } r \neq \tilde{r} \text{ and } \exists i\,(r\langle i\rangle = 0 \text{ and } \tilde{r}\langle i\rangle = 1)\}$.

It is easy to see that U_1, U_2, U_3 are mutually disjoint, $(\{0,1\}^K)^2 = U_1 \cup U_2 \cup U_3$, $|U_1| = 2^K$ and $|U_2| = |U_3| = (2^{2k} - 2^K)/2$.

Since both R and $\tilde{R}$ are uniformly distributed, we have:

$$\mathrm{Prob}(\{R = \tilde{R}\}|\overline{\mathcal{E}}) = |U_1|/|(\{0,1\}^K)^2| = 1/2^K, \tag{11}$$

and

$$\begin{aligned}\text{Prob}(\{\exists i\,(R\langle i\rangle = 0 \text{ and } \tilde{R}\langle i\rangle = 1)\}|\overline{\mathcal{E}}) &= |U_3|/|(\{0,1\}^K)^2| \\ &= 1/2 - 1/2^{K+1}.\end{aligned} \tag{12}$$

Combining Equation (10) to Equation (12), we obtain:

$$\begin{aligned}q =& \text{Prob}(\mathcal{E}) + \text{Prob}(\overline{\mathcal{E}})\,(1/2 + 1/2^{K+1}) \\ =& 1/2 + 1/2^{K+1} + (1/2 - 1/2^{K+1})\,\text{Prob}(\mathcal{E})\end{aligned} \tag{13}$$

In particular, we have $q > 1/2$. Combining Equation (9) and Equation (13), we get:

$$\tau_{\mathfrak{S}} < 3\,(\mathsf{T}_1 + \mathsf{T}_2 + \mathsf{T}_\Xi + \mathsf{T}_4) + \mathsf{T}_5 + \mathsf{T}_6 + \mathsf{T}_7,$$

which means that the simulator's expected running time is bounded by a polynomial in the security parameter k. □

Lemma 38. *The simulator* $\mathfrak{S}$ *outputs* Failure *with probability at most* $1/2^{K-1}$.

Proof. Let X be the random variable defined as the number of the trials of $\mathfrak{S}$ in a whole execution. From the proof of Lemma37, we know two facts. First, we have: $\text{Prob}(X = i) = (1-q)^{i-1}q < 1/2^{i-1}$. Second, in each trial the event $\{\mathfrak{S}$ outputs Failure$\}$ is the combined event of $\overline{\mathcal{E}}$ and $R = \tilde{R}$. This combined event happens with probability:

$$\begin{aligned}\text{Prob}(\overline{\mathcal{E}} \cap \{R = \tilde{R}\}) &\leq \text{Prob}(\overline{\mathcal{E}})\,\text{Prob}(\{R = \tilde{R}\}|\overline{\mathcal{E}}) \\ &\leq \text{Prob}(\{R = \tilde{R}\}|\overline{\mathcal{E}}).\end{aligned}$$

Combining the above inequality to Equation (11), we obtain:

$$\text{Prob}(\overline{\mathcal{E}} \cap \{R = \tilde{R}\}) \leq 1/2^K.$$

Therefore, the probability that $\mathfrak{S}$ outputs Failure in a whole execution is

$$\begin{aligned}\sum_{i=1}^{\infty} \text{Prob}(X = i)\,\text{Prob}(\overline{\mathcal{E}} \cap \{R = \tilde{R}\}) &\leq (1/2^K) \sum_{i=1}^{\infty} 1/2^{i-1} \\ &\leq 1/2^{K-1}.\end{aligned}$$

□

Lemma 39. *The output of the adversary* $\mathcal{A}$ *in the real world and that of the simulator* $\mathfrak{S}$ *in the ideal world are statistically indistinguishable, i.e.:*

$$\{\mathsf{Real}_{\Pi,\{2\},\mathcal{A}(z_k)}(1^k, \vec{m}, T)\langle 0\rangle\}_{k\in\mathbb{N},\vec{m}\in(\{0,1\}^*)^n,T\in\Psi,z_k\in\{0,1\}^*} \stackrel{s}{=} \{\mathsf{Ideal}_{f,\{2\},\mathfrak{S}(z_k)}(1^k, \vec{m}, T)\langle 0\rangle\}_{k\in\mathbb{N},\vec{m}\in(\{0,1\}^*)^n,T\in\Psi,z_k\in\{0,1\}^*}.$$

Proof. First, we claim that $\mathfrak{S}$'s output and $\overline{\mathcal{A}}$'s output are statistically indistinguishable. Indeed, $\mathfrak{S}$ always returns $\overline{\mathcal{A}}$'s output with the exception of returning Failure when Case 1 of Step Sim4 happens. Since Lemma 38 shows that this situation arises with negligible probability, our claim holds.

Second, we claim that the outputs of $\mathcal{A}$ and $\overline{\mathcal{A}}$ are statistically indistinguishable. From the proof of Lemma 30, the probability that the adversary learns more than one value is negligible. As a result, we ignore this case. The only difference between the views of real adversary $\mathcal{A}$ and its copy $\overline{\mathcal{A}}$ is that the encryption received by $\overline{\mathcal{A}}$ is the encryption of the constructed $\vec{m}'$ rather than $\vec{m}$.

Looking at Step S4.3 of Π (i.e., Protocol 1), we know that the encryption is generated by XOR-ing the private values of the sender with the hash values of the unchosen instance vectors.

The o'-th private value is hidden by hash values of projective instances. The projection property of $\mathcal{H}$ guarantees that $\overline{\mathcal{A}}$ (resp., $\mathcal{A}$) learns $\vec{m}'\langle o'\rangle$ (resp., $\vec{m}\langle o'\rangle$). Seeing Step Sim6 of $\mathfrak{S}$, we know:

$$\vec{m}'\langle o'\rangle = \vec{m}\langle o'\rangle.$$

The $(3-o')$-th private value is hidden by hash values of smooth instances. The smoothness property of $\mathcal{H}$ guarantees that $\overline{\mathcal{A}}$ (resp., $\mathcal{A}$) learns nothing about $\vec{m}'\langle 3-o'\rangle$ (resp., $\vec{m}\langle 3-o'\rangle$) and

$$\vec{c}'\langle 3-o'\rangle \stackrel{s}{=} \vec{c}\langle 3-o'\rangle.$$

Therefore, $\overline{\mathcal{A}}$'s view and $\mathcal{A}$'s view are statistically indistinguishable. Thus, our second claim holds which achieves our lemma demonstration. □

Remark 40. *Because $\mathfrak{S}$ filters out the case that all chosen instance vectors are legal and all unchosen ones are illegal by repeating procedure Ξ in Step* Sim3*, $\mathfrak{S}$ does not simulate the real world case where $\mathcal{A}$ learns more than* 1 *values. Correspondingly, in the proof of Lemma 39, we do not consider this case too. This is justified by the fact that Lemma 30 shows this situation arises with probability at most* $1/2^K$.

Proposition 41. *In the case where only the receiver is corrupted, Equation* (1) *in Definition 9 holds.*

Proof. Note that the senders $\mathcal{S}$ and $\mathcal{S}'$ end up with outputting nothing as they honestly follow Π. On the other hand, the receivers $\mathcal{R}$ and $\mathcal{R}'$ end up with no output either bcause they are corrupted and so are dummy players. As a consequence, the fact that the outputs of $\mathfrak{S}$ and $\mathcal{A}$ are statistically indistinguishable (Lemma 39) immediately proves this proposition. □

7. A Fully-Simulatable Framework for OT_t^n

Protocol 2 extends our OT_1^2 to OT_t^n, where n, t are arbitrary positive integers and $n > t$.

Theorem 42. *Assume that in Protocol 2 the smooth projective hash family $\mathcal{H}$ holds properties distinguishability and hard subset membership, the commitment scheme* BC *is perfectly binding and the commitment scheme* HC *is perfectly hiding. Then, Protocol 2 securely computes functionality OT_t^n in the presence of non-adaptive malicious adversaries where the sender's security is statistical and the receiver's security is computational.*

Note that hard subset membership of $\mathcal{H}$ guarantees that the adversary can not learn any extra knowledge from rearrangement indicators σ_i^2. The proof of Theorem 42 is almost the same as that of Theorem 29, thus we omit the details.

Protocol 2 A Framework for OT_t^n

This Protocol is constructed via modifying Protocol 1. We list the modifications as follows.

1. Public Inputs: The employed hash family $\mathcal{H}$ is a (n, t)-SPH-DHM rather than a $(2, 1)$-SPH-DHM.
2. Private Inputs: The sender $\mathcal{S}$ holds a vector $\vec{m} \in (\{0,1\}^*)^n$ of size n rather than a vector of size 2. The receiver $\mathcal{R}$ holds an index set $T \in \Psi$ of size t rather than a single index $o \in \{1, 2\}$.
3. Step R1.3. For each $i \in [K]$, the receiver $\mathcal{R}$ uniformly chooses a permutation $\sigma_i^1 \in_U S_n$ and shuffles the vector $\tilde{\vec{a}}_i \leftarrow \sigma_i^1(\vec{a}_i)$.
4. Step R5.1. The rearrangement indicators are permutations σ_i^2 over $[n]$ rather than bits b_i. For each non-chosen vector $\tilde{\vec{x}}_i (i \notin C)$, $\mathcal{R}$ reorders it by applying a random permutation σ_i^2, so that each j-th ($j \in T$) entry of the resultant vector $\sigma_i^2(\tilde{\vec{x}}_i)$ is projective.
5. Step S4. Correspondingly, the sender $\mathcal{S}$ rearranges each unchosen vector according to rearrangement indicator σ_i^2.

References

[1] M. O. Rabin, How to exchange secrets by oblivious transfer, Tech. Rep. Technical Report TR-81, Aiken Computation Lab, Harvard University (1981).

[2] J. Kilian, Founding cryptography on oblivious transfer, in: 20th Annual ACM Symposium on Theory of Computing (STOC'88), ACM Press, Chicago, USA, 1988, pp. 20 – 31.
[3] G. Aggarwal, N. Mishra, B. Pinkas, Secure computation of the median (and other elements of specified ranks), Journal of Cryptology 23 (3) (2010) 373–401.
[4] N. Mohammed, D. Alhadidi, B. C. M. Fung, M. Debbabi, Secure two-party differentially private data release for vertically partitioned data, Dependable and Secure Computing, IEEE Transactions on 11 (1) (2014) 59–71.
[5] C. Hazay, K. Nissim, Efficient set operations in the presence of malicious adversaries, Journal of Cryptology 25 (3) (2012) 383–433.
[6] B. Aiello, Y. Ishai, O. Reingold, Priced oblivious transfer: How to sell digital goods, in: B. Pfitzmann (Ed.), Advances in Cryptology - Eurocrypt'01, Vol. 2045 of Lecture Notes in Computer Science, Springer - Verlag, Innsbruck, Austria, 2001, pp. 119 – 135.
[7] S. Jarecki, X. M. Liu, Private mutual authentication and conditional oblivious transfer, in: S. Halevi (Ed.), Advances in Cryptology - Crypto 2009, Vol. 5677 of Lecture Notes in Computer Science, 2009, pp. 90–107.
[8] Y. Lindell, B. Pinkas, Privacy preserving data mining, Journal of Cryptology 15 (2002) 177–206.
[9] M. Naor, B. Pinkas, Computationally secure oblivious transfer, Journal of Cryptology 18 (1) (2005) 1–35.
[10] M. Green, S. Hohenberger, Practical adaptive oblivious transfer from simple assumptions, in: Y. Ishai (Ed.), 8th Theory of Cryptography Conference (TCC'11), Vol. 6597, Springer - Verlag, Providence, USA, 2011, pp. 347 – 363.
[11] W. Ogata, K. Kurosawa, Oblivious keyword search, Journal of complexity 20 (2) (2004) 356–371.
[12] M. Naor, B. Pinkas, Oblivious polynomial evaluation, SIAM Journal on Computing 35 (2006) 1254 – 1281.
[13] S. Even, O. Goldreich, A. Lempel, A randomized protocol for signing contracts, Communications of the ACM 28 (6) (1985) 637–647.
[14] D. Catalano, R. Cramer, I. Damgård, G. Di Crescenzo, D. Poincheval, T. Takagi, Contemporary Cryptology, Advanced Courses in Mathematics - CRM Barcelona, Birkhäuser, 2005.
[15] M. Naor, B. Pinkas, Efficient oblivious transfer protocols, in: 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01), Society for Industrial and Applied Mathematics, Washington, USA, 2001, pp. 448 – 457.
[16] S. Halevi, Y. Tauman Kalai, Smooth projective hashing and two-message oblivious transfer, Journal of Cryptology 25 (1) (2012) 158–193.
[17] M. Green, S. Hohenberger, Blind identity-based encryption and simulatable oblivious transfer, in: K. Kurosawa (Ed.), Advances in Cryptology - Asiacrypt'07, Vol. 4833 of Lecture Notes in Computer Science, Springer - Verlag, Kuching, Malaysia, 2007, pp. 265 – 282.
[18] J. Camenisch, G. Neven, abhi shelat, Simulatable adaptive oblivious transfer, in: M. Naor (Ed.), Advances in Cryptology - Eurocrypt'07, Vol. 4515 of Lecture Notes in Computer Science, Springer - Verlag, Barcelona, Spain, 2007, pp. 573 – 590.
[19] A. Y. Lindell, Efficient fully-simulatable oblivious transfer, in: T. Malkin (Ed.), Topics in Cryptology - CT-RSA 2008, Vol. 4964 of Lecture Notes in Computer Science, Springer - Verlag, San Francisco, USA, 2008, pp. 52 – 70.
[20] Y. Lindell, B. Pinkas, Secure two-party computation via cut-and-choose oblivious transfer, Journal of Cryptology 25 (4) (2012) 680–722.
[21] C. Peikert, V. Vaikuntanathan, B. Waters, A framework for efficient and composable oblivious transfer, in: D. Wagner (Ed.), Advances in Cryptology - Crypto'08, Vol. 5157 of Lecture Notes in Computer Science, Springer - Verlag, Santa Barbara, USA, 2008, pp. 554 – 571.
[22] J. Han, W. Susilo, Y. Mu, M. H. Au, J. Cao, Aac-ot: Accountable oblivious transfer with access control, IEEE Transactions on Information Forensics and Security 10 (12) (2015) 2502–2514. doi:10.1109/TIFS.2015.2464781.
[23] O. Goldreich, S. Micali, A. Wigderson, How to play any mental game, in: 19th Annual ACM Symposium on Theory of Computing (STOC'87), ACM Press, New York, USA, 1987, pp. 218 – 229.
[24] B. Zeng, New smooth projective hashing for oblivious transfer, Cryptology ePrint Archive, Report 2018, http://eprint.iacr.org/ (2018).
[25] Y. Lindell, B. Pinkas, An efficient protocol for secure two-party computation in the presence of malicious adversaries, Journal of Cryptology 28 (2) (2015) 312–350. doi:10.1007/s00145-014-9177-x.
URL http://dx.doi.org/10.1007/s00145-014-9177-x
[26] B. Zeng, C. Tartary, P. Xu, J. Jing, X. Tang, A practical framework for t-out-of-n oblivious transfer with security against covert adversaries, IEEE Transactions on Information Forensics and Security 7 (2) (2012) 465–479.
[27] Y. Aumann, Y. Lindell, Security against covert adversaries: Efficient protocols for realistic adversaries, Journal of Cryptology 23 (2) (2010) 281–343.
[28] M. Blum., Coin flipping by phone, IEEE Spring COMPCOM, 1982, pp. 133–137.
[29] O. Goldreich, Foundations of Cryptography: Volume I - Basic Tools, Cambridge University Press, 2001.
[30] O. Goldreich, Foundations of Cryptography: Volume II - Basic Applications, Cambridge University Press, 2004.
[31] R. Canetti, Security and composition of multiparty cryptographic protocols, Journal of Cryptology 13 (1) (2000) 143 – 202.
[32] R. Canetti, I. Damgård, S. Dziembowski, Y. Ishai, T. Malkin, Adaptive versus non-adaptive security of multi-party protocols, Journal of Cryptology 17 (3) (2004) 153 – 207.
[33] B. Barak, Y. Lindell, Strict polynomial-time in simulation and extraction, SIAM Journal on Computing 33 (4) (2004) 783 – 818.
[34] O. Goldreich, On expected probabilistic polynomial-time adversaries: A suggestion for restricted definitions and their benefits, Journal of Cryptology 23 (1) (2010) 1 – 36.
[35] O. Goldreich, A. Kahan, How to construct constant-round zero-knowledge proof systems for $\mathcal{NP}$, Journal of Cryptology 9 (3) (1996) 167 – 189.
[36] R. Cramer, V. Shoup, Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption, in: L. R. Knudsen (Ed.), Advances in Cryptology - Eurocrypt'02, Vol. 2332 of Lecture Notes in Computer Science, Springer - Verlag, Amsterdam, The Netherlands, 2002, pp. 45 – 64.
[37] J. P. Degabriele, K. Paterson, G. Watson, Provable security in the real world, Security & Privacy, IEEE 9 (3) (2011) 33–41.
[38] A. Blum, Lecture notes on randomized algorithms, www.cs.cmu.edu/~avrim/Randalgs11/lectures/lect0124.pdf (January 2011).