Practical Frameworks For h-Out-Of-n Oblivious Transfer With Security Against Covert and Malicious Adversaries

Zeng Bing, Tang Xueming, Xu Peng, and Jing Jiandu

Abstract—We present two practical frameworks for h-out-of-n oblivious transfer (OT_h^n). The first one is secure against covert adversaries who are not always willing to cheat at any price. The security is proven under the ideal/real simulation paradigm (call such security fully simulatable security). The second one is secure against malicious adversaries who are always willing to cheat. It provides fully simulatable security and privacy respectively for the sender and the receiver (call such security one-sided simulatable security). The two frameworks can be implemented from the decisional Diffie-Hellman (DDH) assumption, the decisional N-th residuosity assumption, the decisional quadratic residuosity assumption and so on.

The DDH-based instantiation of our first framework costs the minimum communication rounds and the minimum computational overhead, compared with existing practical protocols for oblivious transfer with fully simulatable security against covert adversaries or malicious adversaries.

Though our second framework is not efficient, compared with existing practical protocols with one-sided simulatable security against malicious adversaries. However, it first provides a way to deal with general OT_h^n on this security level. What is more, its DDH-based instantiation is more efficient than the existing practical protocols for oblivious transfer with fully simulatable security against malicious adversaries.

Index Terms—oblivious transfer (OT) protocols, secure two-party computation.

1 Introduction

1.1 Oblivious transfer

Oblivious transfer (OT), first introduced by [39] and later defined in another way with equivalent effect [14] by [15], is a fundamental primitive in cryptography and a concrete problem in the filed of secure multi-party computation. Considerable cryptographic protocols can be built from it. Most remarkable, [23], [27], [29], [41] proves that any secure multi-party computation can be based on a secure oblivious transfer protocol. In this paper, we concern a variant of OT, h-out-of-n oblivious transfer (OT_h^n). OT_h^n deals with the following scenario. A sender holds n private messages, $m_1, m_2, \ldots, m_n$. A receiver holds h private positive integers $i_1, i_2, \ldots, i_h$, where $i_1 < i_2 < \ldots < i_h \leqslant n$. The receiver expects to get the messages $m_{i_1}, m_{i_2}, \ldots, m_{i_h}$ without leaking any information about his private input, i.e., the h positive integers he holds. The sender expects all new knowledge learned by the receiver from their interaction is at most h messages. Obviously, the OT most literature refer to is OT_1^2 and can be viewed as a special case of OT_h^n.

It is known that, by using Goldreich's compiler [21], [23], we can gain a protocol for OT_h^n with security against malicious adversaries (a malicious adversary act in any arbitrary malicious way to learn as much extra information as possible) from a protocol for OT_h^n with security against semi-honest adversaries (a semi-honest adversary, on one side, honestly does everything told by a prescribed protocol; on one side, records the messages he sees to deduce extra information which is not supposed to be known to him), which can be built from any collection of enhanced trapdoor permutations [21]. The security of the resulting protocol can be proven under the real/ideal simulation paradigm. The paradigm requires that for any adversary in the real world, there exists a corresponding adversary in the ideal world who can simulate him, where the ideal world holds desirable security level. That is, for any malicious adversary, he can not do more harm in the real world than in the ideal world, which implies that the proven protocol is secure. However, the mentioned resulting protocol for OT_h^n is prohibitive expensive for practical use, because it is embedded with so many invocations of zero-knowledge proof for NP and invocations of commitment scheme. Thus, directly constructing the protocol based on specific intractable assumptions seems more feasible.

This work is supported by China Postdoctoral Science Foundation (No. 20100480900).

Zeng Bing is with the College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan City, Hubei 430074 China (e-mail: zeng.bing@smail.hust.edu.cn, zeng.bing.zb@gmail.com).
Tang Xueming is with the College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan City, Hubei 430074 China (e-mail:tang.xueming.txm@gmail.com).
Xu Peng is with the College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan City, Hubei 430074 China (e-mail:xupeng@mail.hust.edu.cn).
Jing Jiandu is with the College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan City, Hubei 430074 China (e-mail: jingjd@smail.hust.edu.cn).

1.2 Our Contribution

In this paper, we aim to construct practical and efficient frameworks for OT_h^n which can be instantiated from the decisional Diffie-Hellman (DDH) assumption, the decisional N-th residuosity (DNR) assumption, the decisional quadratic residuosity (DQR) assumption and so on. To this end, compared with the security mentioned above (for simplicity, we call this security standard security), the security we guarantee is relaxed to some extent. Specifically, we construct two frameworks respectively with the following two security level.

- Security against covert adversaries. This security, presented by [3], guarantees that in case a adversary commits an effective cheating, then the adversary can be caught with probability at least ϵ, where ϵ is called the deterrence factor. By effective cheating, we refer to a cheating in which a adversary learning new knowledge is possible. The security definition follows the real/ideal simulation paradigm. Therefore, compared with the standard security, the only relaxed point in this security is that it does not require to rule out any possibility that the adversaries learns new knowledge in an execution of a protocol. In any setting that the malicious adversaries are not always willing to cheat at any price, this security makes sense.
  In our first framework, in case a adversary is caught in committing an effective cheating, he will learn no new knowledge. What is more, the deterrence factor can be set to be very close to 1, e.g., $\epsilon = 1 - 7.25 \times 10^{-12}$.
- Security against malicious adversaries based on one-sided simulation. This security, called the security based on half-simulation by [25] and also called the security based on half-simulation by [7], guarantees that, in case one party is corrupted, it provides privacy to the honest party; in case another party is corrupted, it provides standard security to the honest party. Specifically, in case one party is corrupted, the security only requires to prove that the adversary controlling the corrupted party can't distinguish his views obtained in the different interactions with the honest party who uses different privates inputs; in case another party is corrupted, the security proof requires as same as the standard security does. Therefore, compared with the standard security, the properties correctness and the independence of private inputs are not guaranteed any more in this security. However, this security indeed is useful in the computation in which only one party receives a output. For example, [16], [25], [28], [35] take this security definition to design efficient protocols. In these works, the party receiving a output is provided with the privacy, and the other party is provided with the full security. In our second framework, this is also the case.

To our best knowledge, our first framework is the first known framework for OT_h^n with security against covert adversaries. Though a framework for OT_1^2 with security against malicious against malicious adversaries based on one-sided simulation is by [28], to our best knowledge, our second framework is the first known framework for general OT_h^n with this security level.

Our two frameworks have the following features.

First, practical. On one side, the securities of our frameworks are obtained without restoring to a random oracle. Though the schemes presented by [26], [37] are efficient, their security is proven under the Random Oracle Model. However, the Random Oracle Model seems risky. [10] shows that a scheme is secure in the Random Oracle Model does not necessarily imply that a particular implementation of it (in the real world) is secure, or even that this scheme does not have any ”structural flaws”. [10] also shows efficient implementing the random oracle is impossible. Later, [30] finds that the random oracle instantiations proposed by Bellare and Rogaway from 1993 and 1996, and the ones implicit in IEEE P1363 and PKCS standards are weaker than a random oracle. What is worse, [30] shows that how the defects of the random oracle instantiations deadly damage the securities of the cryptographic schemes presented in [5], [6]. Therefore, considering practical use, our frameworks are better.

On one side, the securities of our frameworks are obtained without restoring to a trusted common reference string (CRS). Though the framework for OT_1^2 presented by [38] and its adaptive instantiations presented by [18] hold higher level that is secure against malicious adversaries even under universal composition, they don't work without a trusted CRS. How to provide a trusted CRS before the protocol run still is a unsolved problem. The existing possible solutions, such as natural process suggested by [38], are only conjectures without formal proofs. What is worse, [9], [11] show that even given a authenticated communication channel, implementing a universal composable protocol providing useful trusted CRS in the presence of malicious adversaries is impossible. Therefore, considering practical use, our frameworks are better.

Second, efficient. Our first framework costs four communication rounds. Fixing the deterrence factor to be $\frac{1}{2}$, this framework costs n public key encryption operations, h public key decryption operations. Compared with the existing practical protocols for OT with standard security or with security against covert adversaries, i.e. the protocols presented by [3], [7], [24], [31], [42], the DDH-based instantiation of our first framework is the most efficient one in the sense that it costs the minimum number of communication rounds and costs the minimum computational overhead. Please see Section 3.4 and Section 3.5 for the detailed comparisons.

Our second framework costs four communication rounds. To be securely used in practice, our framework, in the worst case, costs the sender $40 \cdot n$ public key encryption operations and costs the receiver $40 \cdot h$ public key decryption operations. Compared with the existing practical protocols for OT with the security based on one-sided simulation, i.e. the protocols presented by [2], [28], [33]–[36], our second framework is not efficient. However, it still makes sense since it does not require k is far less than n and is the first framework/protocol dealing with general OT_k^n on this security level. Compared with the existing practical protocols for OT with standard security, i.e. the protocols presented by [7], [24], [31], [42], the DDH-based instantiation of our second framework is the most efficient one in the sense mentioned above. Please see Section 4.3 and Section 4.4 for the details.

Third, abstract and modular. The framework is described using only one high-level cryptographic tool, i.e., a variant of smooth projective hash presented by [42] (for simplicity, refer to it as $SPHDHC_{t,h}$. This allows a simple and intuitive understanding of its security.

Fourth, generally realizable. [42] shows that $SPHDHC_{t,h}$ are realizable from a variety of known specific assumption, such as DDH, DNR, DQR. We remark that $SPHDHC_{t,h}$ also is possible to be realized from future assumptions. This makes our framework generally realizable. Considering the future progress in breaking a specific intractable problem, generally realizability is vital to make framework live long. If this considered case happen, replacing the instantiation based on the broken problem with that based on a unbroken problem suffices.

1.3 Construction Overview

Our first framework is described with high-level as follows.

1) Let K, g be two predetermined positive integers. The receiver generates hash parameters and K instance vectors, then sends them to the sender after disordering each vector.
2) The receiver chooses g instance vectors at random to open.
3) The receiver opens the chosen instances and encodes his private input by reordering each unchosen vector.
4) The sender checks that the chosen vectors are generated in the legal way which guarantees that the receiver learns at most h message. If the check pass, the sender encrypts his private input (i.e., the n messages he holds) using the hash system, and sends the ciphertext together with some information to the receiver which is conductive to decrypt some ciphertext.
5) The receiver decrypts the ciphertext and gains the messages he expects.

Intuitively speaking, the receiver's security is implied by the property hard subset membership of $SPHDHC_{t,h}$. This property guarantees that the receiver can securely encode his private input by reordering each unchosen instance vector. The sender's security is implied by the cut-and-choose technique, which guarantees that the probability that the adversaries controlling a corrupted receiver learns new knowledge is small enough.

Our second framework is obtained by modifying the first framework by letting K be a polynomial in security parameter and granting the sender to uniformly choose arbitrary many instance vectors to open.

In some sense, our frameworks are based on the framework presented in [28] for OT_1^2 with security based on one-sided simulation. However, the technique and tool we use are very different. In our frameworks, we use cut-and-choose to guarantee the sender's security rather than use the property verifiability of a tool. What is more, the tool we uses is different from the smooth projective hash [28] uses. These differences are keys to make our first framework fully simulatable. With the help of cut-and-choose, the property feasible cheating and distinguishability of $SPHDHC_{t,h}$ respectively enable the simulator to extract the adversary's real input in case the sender is corrupted and in case the receiver is corrupted. The differences also are keys to make our second framework able to deal with OT_h^n. Since $SPHDHC_{t,h}$ can directly manipulate n instances and provide a novel way to guarantee the sender's security.

1.4 Related Work

The first step to construct practical and efficient protocol for OT is independently made by [35] and [2]. The protocols are for OT_1^2, based on the DDH assumption, with security based on one-sided simulation. Later, using the tool smooth projective hash, [28] generalizes the ideas of the two previous works and presents a framework for OT_1^2. [28] shows the framework can be instantiated from not only the DDH assumption but also the DNR assumption and the DQR assumption.

There are some work aiming to build protocols for OT_h^n from known protocols for OT_1^2 with security based on one-sided simulation. [33] shows how to implementation OT_h^n using $\log n$ invocation of OT_1^2 with security based on one-sided simulation. A similar implementation for adaptive OT_h^n can be seen in [34]. These mentioned works later are described in [36]. With the help of a random oracle, [26] shows how to extend k oblivious transfers (for some security parameter k) into many more, without much additional effort. [37] proposes an efficient adaptive h-out-n oblivious transfer (denoted by $OT_{h\times 1}^n$ in related literature) schemes under Random Oracle Model.

The protocols for oblivious transfer with standard security is first presented by [7]. This protocol is for $OT_{k\times 1}^n$ and based on q-Power Decisional Diffie-Hellman and q-Strong Diffie-Hellman assumptions which are not standard assumptions. Later, using a blind identity-based encryption, [24] first presents a protocol for OT_h^n based on a standard assumption with standard security. [31] presents a DDH-based protocol for OT_1^2 with standard security and with efficiency over the two mentioned works. Recently, [42] presents a framework for OT^n_h with standard security, which also can be instantiated from all the assumptions used by [28], [31]. The DDH-based instantiation of this framework is the most efficient one of the protocols that works in the setting that there are no set-up assumption such as a trusted CRS available.

Under CRS model, [38] presents a framework for OT^2_1 with higher security level that the security is preserved even other arbitrary malicious protocols concurrently run with it. [38] shows that the framework be instantiated from the DDH, DQR and worst-case lattice assumption. We remark that if a trusted CRS is available, then the DDH-based instantiation of the framework is the most efficient protocol for OT^2_1. Later [18] strengthen the security of the instantiations based on DDH, DQR to against adaptive malicious adversaries.

Using homomorphic encryption, [3] first presents a protocol for OT^2_1 with security against covert adversaries.

1.5 Organization

In Section 2, we describe the notations used in this paper, the security definitions of OT^n_h and a variant of smooth projective hash which is the basic tool we use. In Section 3, we construct the first framework for OT^n_h, prove its security and analyze its performance. In Section 4, we construct the second framework for OT^n_h, prove its security and analyze its performance. In Section 5, we discuss the technique cut-and-choose in the frameworks.

2 Preliminaries

2.1 Basic Notations

We denote an unspecified positive polynomial by $poly(.)$. We denote the set consists of all natural numbers by $\mathbb{N}$. For any $i \in \mathbb{N}$, $[i] \stackrel{def}{=} \{1, 2, \ldots, i\}$.

We denote security parameter used to measure security and complexity by k. A function $\mu(.)$ is negligible in k, if there exists a positive constant integer n_0, for any $poly(.)$ and any k which is greater than n_0 (for simplicity, we later call such k sufficiently large k), it holds that $\mu(k) < 1/poly(k)$. 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 $X(1^k, a)$. Probability ensemble X is polynomial-time constructible, if there exists a probabilistic polynomial-time (PPT) sample algorithm $S_X(.)$ such that for any a, any k, the random variables $S_X(1^k, a)$ and $X(1^k, a)$ are identically distributed. We denote sampling an instance according to $X(1^k, a)$ by $\alpha \leftarrow S_X(1^k, a)$.

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

$$|Pr(D(1^k, X(1^k, a), a, z_k) = 1) - Pr(D(1^k, Y(1^k, a), a, z_k) = 1)| \leqslant \mu(k)$$

They are same, denoted $X = Y$, if for any sufficiently large k, any a, $X(1^k, a)$ and $Y(1^k, a)$ are defined in the same way. They are equal, denoted $X \equiv Y$, if for any sufficiently large k, any a, the distributions of $X(1^k, a)$ and $Y(1^k, a)$ are identical. Obviously, if $X = Y$ then $X \equiv Y$; If $X \equiv Y$ then $X \stackrel{c}{=} Y$.

Let $\vec{x}$ be a vector (note that arbitrary binary string can be viewed as a vector). We denote its i-th element by $\vec{x}\langle i\rangle$, denote its dimensionality by $\#\vec{x}$, denote its length in bits by $|\vec{x}|$. For any positive integers set I, any vector $\vec{x}$, $\vec{x}\langle I\rangle \stackrel{def}{=} (\vec{x}\langle i\rangle)_{i\in I, i\leq \#\vec{x}}$.

Let M be a probabilistic (interactive) Turing machine. By $M_r(.)$ we denote M's output generated at the end of an execution using randomness r.

Let $f : D \rightarrow R$. Let $D' \subseteq \{0,1\}^*$. Then $f(D') \stackrel{def}{=} \{f(x)|x \in D' \cap D\}$, $Range(f) \stackrel{def}{=} f(D)$.

Let $x \in_\chi Y$ denotes sampling an instance x from domain Y according to the distribution law (or probability density function) χ. Specifically, let $x \in_U Y$ denotes uniformly sampling an instance x from domain Y.

2.2 Security Against Covert Adversaries

Security against covert adversaries is presented by [3]. The key observation of this security is that in many real-world setting, such as business, political settings and playing remote games, the assumption that the adversaries are always ready to do cheating at any price and under any circumstance, implicitly used in traditional model of secure multiparty computation [8], [21], is overly pessimistic and unnecessary. Therefore, using great amount of resource to preclude such covert adversaries from commenting cheating seems unnecessary. Instead, this security aims at catching the cheating of the covert adversaries with some probability, called deterrence factor and denoted ε, rather than aims at eliminating any successful cheating of covert adversaries. [3] presents three versions of covert security. In this paper, we deals with the strongest one, which implies the other two versions of covert security. For clarity and simplicity, we tailor it to the need of dealing with OT^n_h.

[3] shows that if $1 - \varepsilon$ is negligible, then this security turns to be the standard security against malicious adversaries; if $\varepsilon \geq 1/poly(k)$, then this security implies the security against semi-honest adversaries. It is easy to deduce that, if $\varepsilon \in (0, 1)$ is a constant value, then this security also implies the security against semi-honest adversaries.

2.2.1 Functionality Of OT_h^n

OT_h^n involves two parties, party P_1 (i.e., the sender) and party P_2 (i.e., the receiver). OT_h^n's functionality is formally defined as follows

$$\begin{aligned} f:\mathbb{N}\times\{0,1\}^*\times\{0,1\}^* &\rightarrow \{0,1\}^*\times\{0,1\}^* \\ f(1^k,\vec{m},H) &= (\lambda,\vec{m}\langle H\rangle) \end{aligned}$$

and also be denoted by

$$(\vec{m},H)\mapsto(\lambda,\vec{m}\langle H\rangle)$$

for simplicity, where

- k is the public security parameter.
- $\vec{m}\in(\{0,1\}^*)^n$ is P_1's private input, and each $|\vec{m}\langle i\rangle|$ is same.
- $H\in\Psi\stackrel{def}{=}\{B|B\subseteq[n],\#B=h\}$ is P_2's private input.
- λ denotes a empty string and is supposed to be got by P_1. That is, P_1 is supposed to get nothing.
- $\vec{m}\langle H\rangle$ is supposed to be got by P_2.

Note that, the length of all parties' private input have to be identical in SMPC (please see [21] for the reason and related discussion). This means that $|\vec{m}|=|H|$ is required. Without loss of generality, in this paper, we assume $|\vec{m}|=|H|$ always holds, because padding can be easily used to meet such requirement.

Intuitively speaking, the security of OT_h^n requires that P_1 can't learn any new knowledge — typically, P_2's private input — from the interaction at all, and P_2 can't learn more than h messages held by P_1. To capture the security in a formal way, the concepts such as adversary, trusted third party, the ideal world, real world were introduced. Since the security target here is to be secure against non-adaptive covert adversaries, we only referred to concepts related to this case in the following.

2.2.2 Non-Adaptive Covert Adversary

Before running OT_h^n, the adversary $\mathcal{A}$ has to corrupt all parties listed in $I\subseteq[2]$. In case $P_i\in\{P_1,P_2\}$ is not corrupted, P_i will strictly follow the prescribed protocol as an honest party. In case party P_i is corrupted, P_i will be fully controlled by $\mathcal{A}$ as a corrupted party. In this case, P_i will have to pass all his knowledge to $\mathcal{A}$ before the protocol runs and follows $\mathcal{A}$'s instructions from then on — so there is a probability that P_i arbitrarily deviates from prescribed protocol. In fact, after $\mathcal{A}$ finishes corrupting, $\mathcal{A}$ and all cheating parties have formed a coalition led by $\mathcal{A}$ to learn as much extra knowledge, e.g. the honest parties' private inputs, as possible. From then on, they share knowledge with each other and coordinate their behavior. Without loss of generality, we can view this coalition as follows. All cheating parties are dummy. $\mathcal{A}$ receives messages addressed to the members of the coalition and sends messages on behalf of the members.

2.2.3 OT_h^n In the ideal world

In the ideal world, there is an incorruptible trusted third party (TTP). All parties hand their private inputs to TTP. TTP computes f and sends back $f(.)\langle i\rangle$ to P_i. An execution of OT_h^n with deterrence factor $\epsilon\in(0,1]$ proceeds as follows.

1) Initial Inputs. All entities know the public security parameter k. Party P_1 holds a private input $\vec{m}\in(\{0,1\}^*)^n$. Party P_2 holds a private input $H\in\Psi$. Adversary $\mathcal{A}$ holds a name list $I\subseteq[2]$, a randomness $r_{\mathcal{A}}\in\{0,1\}^*$ and an infinite auxiliary input sequence $z=(z_k)_{k\in\mathbb{N}}$, where $z_k\in\{0,1\}^*$. Before proceeding to the next stage, $\mathcal{A}$ corrupts parties listed in I and learns $\vec{x}\langle I\rangle$, where $\vec{x}\stackrel{def}{=}(\vec{m},H)$.
2) The parties submitting inputs to TTP. Each honest party P_i submits $\vec{x}\langle i\rangle$ to TTP. $\mathcal{A}$ submits arbitrary string based on his knowledge to TTP for cheating parties. The inputs TTP receives, denoted by $\vec{y}$, can be described as follows.
$$\vec{y}\langle i\rangle=\begin{cases}\vec{x}\langle i\rangle & \text{if } i\notin I\ ,\\ \alpha_i & \text{if } i\in I.\end{cases}$$
where
$$\alpha_i\leftarrow\mathcal{A}(1^k,I,r_{\mathcal{A}},z_k,\vec{x}\langle I\rangle,i)$$
$$\alpha_i\in\{\vec{x}\langle i\rangle\}\cup\{0,1\}^{|\vec{x}\langle i\rangle|}\cup\{abort_i\}\cup\{corrupted_i\}\\ \cup\{cheat_i\}$$
Obviously, there is a probability that $\vec{x}\neq\vec{y}$.
3) TTP computing f. TTP checks $\vec{y}$ and takes the following actions in order.
   a) In case $\exists i(\vec{y}\langle i\rangle=abort_i)$. For $i\in I$, $\vec{w}\langle i\rangle\leftarrow\lambda$; for $i\in[n]-I$, $\vec{w}\langle i\rangle\leftarrow abort_i$. If there are multiple $abort_i$s, TTP chooses the one whose i is the smallest. TTP deals with multiple $corrupted_i$s and $cheat_i$s, which will meet later, in a similar way.
   b) In case $\exists i(\vec{y}\langle i\rangle=corrupted_i)$. For $i\in I$, $\vec{w}\langle i\rangle\leftarrow\lambda$; for $i\in[n]-I$, $\vec{w}\langle i\rangle\leftarrow corrupted_i$.
   c) In case $\exists i(\vec{y}\langle i\rangle=cheat_i)$. TTP does nothing.
   d) In other cases, $\vec{w}\leftarrow f(1^k,\vec{y})$.
4) TTP delivering results to the parties.
   a) In case $\exists i(\vec{y}\langle i\rangle=abort_i)$ or in case $\exists i(\vec{y}\langle i\rangle=corrupted_i)$. TTP sends each $\vec{w}\langle i\rangle$ to each honest party $P_i(i\in[n]-I)$. Finally, TTP halts.
   b) In case $\exists i(\vec{y}\langle i\rangle=cheat_i)$.
      - With probability ϵ, TTP sends $corrupted_i$ to $\mathcal{A}$ and honest parties. That is, TTP sets $\vec{w}\langle i\rangle\leftarrow(corrupted_i,corrupted_i)$. TTP sends each $\vec{w}\langle i\rangle$ to each honest party $P_i(i\in[n]-I)$, sends $\vec{w}\langle I\rangle$ to $\mathcal{A}$. Finally, TTP halts.
      - With probability $1-\epsilon$, TTP sets $\vec{w}\langle I\rangle\leftarrow(\vec{x}\langle[n]-I\rangle,undetected)$ and sends $\vec{w}\langle I\rangle$ to $\mathcal{A}$. On receiving $\vec{w}\langle I\rangle$, $\mathcal{A}$ determines the results the honest parties will receive as follows.
      $$c_j\leftarrow A(1^k,I,r_{\mathcal{A}},z_k,\vec{x}\langle I\rangle,\vec{w}\langle I\rangle,j)$$

$\mathcal{A}$ sends them to TTP.

For each $j \in [n]-I$, TTP sets $\vec{w}\langle j\rangle \leftarrow c_j$ and sends $\vec{w}\langle j\rangle$ to P_j. Finally, TTP halts.

We remark that the inputs of honest parties are leaked to $\mathcal{A}$ only if $\mathcal{A}$ commits the cheating without being caught.

c) In other cases, TTP first sends back the corrupted parties' results, i.e., sends $\vec{w}\langle I\rangle$ to $\mathcal{A}$. $\mathcal{A}$ computes

$$\beta \leftarrow A(1^k, I, r_{\mathcal{A}}, z_k, \vec{x}\langle I\rangle, \vec{w}\langle I\rangle)$$
$$\beta \in \{abor_i | i \in I\} \cup \{continue\}$$

and sends β to TTP.

TTP checks β. If $\beta \in \{abor_i | i \in I\}$, for each $i \in [n]-I$, TTP sets $\vec{w}\langle i\rangle \leftarrow \beta$. Then, TTP sends each $\vec{w}\langle i\rangle (i \in [n]-I)$ to each honest party P_i. Finally, TTP halts.

5) Outputs. Each honest party P_i outputs $\vec{w}\langle i\rangle$. Each corrupted party outputs nothing (i.e., λ). The adversary outputs something generated by executing an arbitrary function of the information he gathers during the execution. Without loss of generality, this can be assumed to be $(1^k, I, r_{\mathcal{A}}, z_k, \vec{x}\langle I\rangle, \vec{w}\langle I\rangle)$.

The whole execution is denoted by $Ideal^{\epsilon}_{f,\mathcal{A}(z),I}(1^k, \vec{x}, r_{\mathcal{A}})$. The output of the execution is defined by the outputs of all parties and the adversary as follows.

$$Ideal^{\epsilon}_{f,\mathcal{A}(z),I}(1^k, \vec{x}, r_{\mathcal{A}})\langle i\rangle \stackrel{def}{=} \begin{cases} \mathcal{A}\text{'s output, i.e., } (1^k, I, r_{\mathcal{A}}, \\ \quad z_k, \vec{x}\langle I\rangle, \vec{w}\langle I\rangle), & i = 0; \\ P_i\text{'s output, i.e., } \lambda, & i \in I; \\ P_i\text{'s output, i.e., } \vec{w}\langle i\rangle, & i \in [n]-I. \end{cases}$$

Obviously, $Ideal^{\epsilon}_{f,\mathcal{A}(z),I}(1^k, \vec{x})$ is a random variable whose randomness is $r_{\mathcal{A}}$.

2.2.4 OT^n_h In Real World

In real world, there is no TTP. Let π be a protocol for OT^n_h. A execution of OT^n_h proceeds as follows.

1) Initial Inputs. Initial input each entity holds in real world is the same as in the ideal world with following exceptions. A randomness r_i is held by each party P_i. After finishing the corrupting, in addition to the knowledge $\mathcal{A}$ learns in the ideal world, the corrupted parties' randomness $\vec{r}\langle I\rangle$ is also learn by $\mathcal{A}$, where $\vec{r} \stackrel{def}{=} (r_1, r_2)$.
2) Computing f. In the real world, computing f is finished by all entities' interaction. Each honest party strictly follows the prescribed protocol π. The corrupted parties have to follow $\mathcal{A}$'s instructions and may arbitrarily deviate from π.
3) Outputs. Each honest party P_i always outputs what π instructs. Each corrupted party P_i outputs nothing. The adversary outputs something generated by executing an arbitrary function of the information he gathers during the execution. Without loss of generality, this can be assumed to be $(1^k, I, r_{\mathcal{A}}, \vec{r}\langle I\rangle, z_k, \vec{x}\langle I\rangle)$ and the messages addressed to the corrupted parties (We denote these messages by msg_I).

The whole execution is denoted by $Real_{\pi,I,\mathcal{A}(z_k)}(1^k, \vec{m}, H, r_{\mathcal{A}}, \vec{r})$. The output of the execution is defined by the outputs of all parties and the adversary as follows.

$$Real_{\pi,I,\mathcal{A}(z_k)}(1^k, \vec{m}, H, r_{\mathcal{A}}, \vec{r})\langle i\rangle \stackrel{def}{=} \begin{cases} \mathcal{A}'s \text{ output, i.e., } (1^k, I, r_{\mathcal{A}}, \\ \quad \vec{r}\langle I\rangle, z_k, \vec{x}\langle I\rangle, msg_I), & i = 0; \\ P_i's \text{ output, i.e., } \lambda, & i \in I; \\ P_i's \text{ output, i.e., what} \\ \quad \text{instructed by } \pi, & i \in [n]-I. \end{cases}$$

Obviously, $Real_{\pi,I,\mathcal{A}(z_k)}(1^k, \vec{m}, H)$ is a random variable whose randomnesses are $r_{\mathcal{A}}$ and $\vec{r}$.

2.2.5 Security definition

Loosely speaking, we say that protocol π securely computes OT^n_h in the presence of covert adversaries, if and only if, for any covert adversary $\mathcal{A}$, the knowledge $\mathcal{A}$ learns in the real world is not more than that he learns in the ideal world. In other words, if and only if, for any covert adversary $\mathcal{A}$, what harm $\mathcal{A}$ can do in the real world is not more than what harm he can do in the ideal world.

Definition 1 (security for OT^n_h against covert adversaries)**.** *Let f denote the functionality of OT^n_h. Let π be a concrete protocol for OT^n_h. Let $\epsilon \in (0, 1]$. We say π securely computes f in the presence of covert adversaries with deterrence factor of ϵ, if and only if for any non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ with an infinite sequence $z = (z_k)_{k\in\mathbb{N}}$ in the real world, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ with the same sequence in the ideal world such that, for any $I \subseteq [2]$, the following equation holds.*

$$\{Real_{\pi,I,\mathcal{A}(z_k)}(1^k, \vec{m}, H)\}_{\substack{k\in\mathbb{N}, \vec{m}\in(\{0,1\}^*)^n \\ H\in\Psi, z_k\in\{0,1\}^*}} \stackrel{c}{=} \{Ideal^{\epsilon}_{f,I,\mathcal{S}(z_k)}(1^k, \vec{m}, H)\}_{\substack{k\in\mathbb{N}, \vec{m}\in(\{0,1\}^*)^n \\ H\in\Psi, z_k\in\{0,1\}^*}} \quad (1)$$

where the parameters input to the two probability ensembles are same, $|\vec{m}| = |H|$, and each $\vec{m}\langle i\rangle$ is of the same length. The adversary $\mathcal{S}$ in the ideal world is called a simulator of the adversary $\mathcal{A}$ in the real world.

2.3 Security Against Malicious Adversaries Based On One-Sided Simulation

The standard security against malicious adversaries in the field of multi-party computation is first presented by [8] and also can be seen in [21]. Compared with the security against covert adversaries, the essential difference is that this security rules outs any successful cheating of the adversaries. In other worlds, there are no choices of

$cheat_i$ and $corrupted_i$ available for the adversary in the ideal world. Therefore, removing such choice and related processes, we gain the execution of OT_h^n in the ideal world with the security against malicious adversaries. Similarly modifying Definition 1, we gain the definition of this standard security.

One-sided simulation security, named by [25], is a slightly weaker security against malicious adversaries than the standard security. Specifically, if P_2 is corrupted, the security is proven in the same way as that under the standard security; if P_1 is corrupted, the security is proven in a way that only guarantees that the adversaries know nothing about P_2's input. With respect to OT_h^n, the definition is specified as follows.

Definition 2 (security for OT_h^n against malicious adversaries under one-sided simulation)**.** *Let f denote the functionality of OT_h^n. Let π be a concrete protocol for OT_h^n. We say π securely computes f in the presence of malicious adversaries under one-sided simulation, if and only if the following holds.*

1) *In case P_2 (i.e., the receiver) is corrupted, for any non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ with an infinite sequence $z = (z_k)_{k\in\mathbb{N}}$ in the real world, there exists a expected non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ with the same sequence in the ideal world such that, the following equation holds.*

$$\{Real_{\pi,\{2\},\mathcal{A}(z_k)}(1^k,\vec{m},H)\}_{\substack{k\in\mathbb{N},\vec{m}\in(\{0,1\}^*)^n\\ H\in\Psi,z_k\in\{0,1\}^*}} \stackrel{c}{=} \{Ideal_{f,\{2\},\mathcal{S}(z_k)}(1^k,\vec{m},H)\}_{\substack{k\in\mathbb{N},\vec{m}\in(\{0,1\}^*)^n\\ H\in\Psi,z_k\in\{0,1\}^*}} \quad (2)$$

where the parameters input to the two probability ensembles are same, $|\vec{m}| = |H|$, and each $\vec{m}\langle i\rangle$ is of the same length.

2) *In case P_1 (i.e., the sender) is corrupted, for any non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ with an infinite sequence $z = (z_k)_{k\in\mathbb{N}}$ in the real world, the following equation holds.*

$$\{View^{\mathcal{A}}_{\pi,\{1\},\mathcal{A}(z_k)}(1^k,\vec{m},H)\}_{\substack{k\in\mathbb{N},\vec{m}\in(\{0,1\}^*)^n\\ H\in\Psi,z_k\in\{0,1\}^*}} \stackrel{c}{=} \{View^{\mathcal{A}}_{\pi,\{1\},\mathcal{A}(z_k)}(1^k,\vec{m},\tilde{H})\}_{\substack{k\in\mathbb{N},\vec{m}\in(\{0,1\}^*)^n\\ \tilde{H}\in\Psi,z_k\in\{0,1\}^*}} \quad (3)$$

One-sided simulation security is first considered by [35] and later considered by [25], [28]. Both [35] and [28] provide OT_1^2 with such security. [25] provides this security to the problem set intersection. Compared with standard security, some security properties, such as independence of inputs and correctness, are not guaranteed. However, the standard security against semi-honest adversaries is indeed guaranteed. What is more, one-sided simulation security enable us to construct efficient protocol against malicious adversaries.

We point out that here we allow the simulator to run in expected polynomial-time, rather than require the simulator to run in strictly polynomial-time as the one-sided simulation security considered in [25], [28], [35]. We argue that this is justified, first, [4] shows that constant-round proof system for sets outside $\mathcal{BPP}$ do not have strict polynomial-time black-box simulators; second, in many cases (also when strict polynomial-time simulators exist), the expected running time of the simulator provides a better bound than the worst-case running time of the simulator [22].

2.4 A Variant Of Smooth Projective Hash

Basic smooth projective hash is first presented by [13]. Though it is originally used to design chosen-ciphertext secure encryption schemes, now it and its variants are used as tools to solve other problems, such as password-based authenticated key exchange [19], oblivious transfer [28], [42], extractable commitment [1]. In this paper, we use the variant of smooth projective hash presented by [42] to construct frameworks for h-out-of-n oblivious transfer. In this section, we introduce this tool.

For clarity in presentation, we assume $n = h + t$ always holds and introduce additional notations. Let $R = \{(x,w)|x,w \in \{0,1\}^*\}$ be a relation, then $L_R \stackrel{def}{=} \{x|x \in \{0,1\}^*, \exists w((x,w) \in R)\}$, $R(x) \stackrel{def}{=} \{w|(x,w) \in R\}$. $\Pi \stackrel{def}{=} \{\pi|\pi : [n] \to [n], \pi \text{ is a permutation}\}$. Let $\pi \in \Pi$ (to comply with other literature, we also use π somewhere to denote a protocol without bringing any confusion). Let $\vec{x}$ be an arbitrary vector. By $\pi(\vec{x})$, we denote a vector resulted from applying π to $\vec{x}$. That is, $\vec{y} = \pi(\vec{x})$, if and only if $\forall i(i \in [d] \to \vec{x}\langle i\rangle = \vec{y}\langle\pi(i)\rangle) \wedge \forall i(i \notin [d] \to \vec{x}\langle i\rangle = \vec{y}\langle i\rangle)$ holds, where $d \stackrel{def}{=} \min(\#\vec{x}, n)$.

Definition 3 (t-smooth h-projective hash family that holds properties distinguishability, hard subset membership, feasible cheating, $SPHDHC_{t,h}$, [42])**.** *$\mathcal{H} = (PG, IS, DI, KG, Hash, pHash, Cheat)$ is an t-smooth h-projective hash family with witnesses and hard subset membership ($SPHDHC_{t,h}$), if and only if $\mathcal{H}$ is specified as follows*

- *The parameter-generator PG is a PPT algorithm that takes a security parameter k as input and outputs a family parameter Λ, i.e., $\Lambda \leftarrow PG(1^k)$. Λ will be used as a parameter to define three relations $R_\Lambda, \dot{R}_\Lambda$ and $\ddot{R}_\Lambda$, where $R_\Lambda = \dot{R}_\Lambda \cup \ddot{R}_\Lambda$. Moreover, $\dot{R}_\Lambda \cap \ddot{R}_\Lambda = \emptyset$ are supposed to hold.*
- *The instance-sampler IS is a PPT algorithm that takes a security parameter k, a family parameter Λ as input and outputs a vector $\vec{a}$, i.e., $\vec{a} \leftarrow IS(1^k, \Lambda)$.*
  Let $\vec{a} = ((\dot{x}_1,\dot{w}_1),\ldots,(\dot{x}_h,\dot{w}_h),(\ddot{x}_{h+1},\ddot{w}_{h+1}),\ldots,(\ddot{x}_n,\ddot{w}_n))^T$ be a vector generated by IS. We call each $\dot{x}_i$ or $\ddot{x}_i$ an instance of L_{R_Λ}. For each pair $(\dot{x}_i,\dot{w}_i)$ (resp., $(\ddot{x}_i,\ddot{w}_i)$), $\dot{w}_i$ (resp., $\ddot{w}_i$) is called a witness of $\dot{x}_i \in L_{\dot{R}_\Lambda}$ (resp., $\ddot{x}_i \in L_{\ddot{R}_\Lambda}$). Note that, by this way we indeed have defined the relationship $R_\Lambda, \dot{R}_\Lambda$ and $\ddot{R}_\Lambda$ here. The properties smoothness and projection we will mention later makes sure $\dot{R}_\Lambda \cap \ddot{R}_\Lambda = \emptyset$ holds.
  For simplicity in formulation later, we introduce some additional notations here. For $\vec{a}$ mentioned above, $\vec{x}^{\vec{a}} \stackrel{def}{=} (\dot{x}_1,\ldots,\dot{x}_h,\ddot{x}_{h+1},\ldots,\ddot{x}_n)^T$,

$\vec{w}^{\vec{a}} \stackrel{def}{=} (\dot{w}_1, \ldots, \dot{w}_h, \ddot{w}_{h+1}, \ldots, \ddot{w}_n)^T$. What is more, we abuse notation $\in$ to some extent. We write $\vec{x} \in Range(IS(1^k, \Lambda))$ if and only if there exists a vector $\vec{x}^{\vec{a}}$ such that $\vec{x}^{\vec{a}} = \vec{x}$ and $\vec{a} \in Range(IS(1^k, \Lambda))$. We write $x \in Range(IS(1^k, \Lambda))$ if and only if there exists a vector $\vec{x}$ such that $\vec{x} \in Range(IS(1^k, \Lambda))$ and x is an entry of $\vec{x}$.

- *The distinguisher DI is a PPT algorithm that takes a security parameter k, a family parameter Λ and a pair strings (x, w) as input and outputs an indicator bit b, i.e., $b \leftarrow DI(1^k, \Lambda, x, w)$.*
- *The key generator KG is a PPT algorithm that takes a security parameter k, a family parameter Λ and an instance x as input and outputs a hash key and a projection key, i.e., $(hk, pk) \leftarrow KG(1^k, \Lambda, x)$.*
- *The hash $Hash$ is a PPT algorithm that takes a security parameter k, a family parameter Λ, an instance x and a hash key hk as input and outputs a value y, i.e., $y \leftarrow Hash(1^k, \Lambda, x, hk)$.*
- *The projection $pHash$ is a PPT algorithm that takes a security parameter k, a family parameter Λ, an instance x, a witness w of x and a projection key pk as input and outputs a value y, i.e., $y \leftarrow pHash(1^k, \Lambda, x, w, pk)$.*
- *The cheat $Cheat$ is a PPT algorithm that takes a security parameter k, a family parameter Λ as input and outputs n elements of $\dot{R}_\Lambda$, i.e., $((\dot{x}_1, \dot{w}_1), \ldots (\dot{x}_n, \dot{w}_n)) \leftarrow Cheat(1^k, \Lambda)$.*

and $\mathcal{H}$ has the following properties

1) *Projection. Intuitively speaking, it requires that for any instance $\dot{x} \in L_{\dot{R}_\Lambda}$, the hash value of $\dot{x}$ is obtainable with the help of its witness $\dot{w}$. That is, for any sufficiently large k, any $\Lambda \in Range(PG(1^k))$, any $(\dot{x}, \dot{w})$ generated by $IS(1^k, \Lambda)$, any $(hk, pk) \in Range(KG(1^k, \Lambda, \dot{x}))$, it holds that*
$$Hash(1^k, \Lambda, \dot{x}) = pHash(1^k, \Lambda, \dot{x}, \dot{w})$$
2) *Smoothness. Intuitively speaking, it requires that for any instance vector $\vec{\ddot{x}} \in L^t_{\ddot{R}_\Lambda}$, the hash values of $\vec{\ddot{x}}$ are random and unobtainable unless their hash keys are known. That is, for any $\pi \in \Pi$, the two probability ensembles $Sm_1 \stackrel{def}{=} \{Sm_1(1^k)\}_{k\in\mathbb{N}}$ and $Sm_2 \stackrel{def}{=} \{Sm_2(1^k)\}_{k\in\mathbb{N}}$ defined as follows, are computationally indistinguishable, i.e., $Sm_1 \stackrel{c}{=} Sm_2$.*
$SmGen_1(1^k)$: $\Lambda \leftarrow PG(1^k)$, $\vec{a} \leftarrow IS(1^k, \Lambda)$, $\vec{x} \leftarrow \vec{x}^{\vec{a}}$, for each $j \in [n]$ operates as follows: $(hk_j, pk_j) \leftarrow KG(1^k, \Lambda, \vec{x}\langle j\rangle)$, $y_j \leftarrow Hash(1^k, \Lambda, \vec{x}\langle j\rangle, hk_j)$, $\overrightarrow{xpky}\langle j\rangle \leftarrow (\vec{x}\langle j\rangle, pk_j, y_j)$. Finally outputs $(\Lambda, \overrightarrow{xpky})$.
$SmGen_2(1^k)$: compared with $SmGen_1(1^k)$, the only difference is that $y_j \in_U Range(Hash(1^k, \Lambda, \vec{x}\langle j\rangle, .))$ for each $j \in [n] - [h]$.
$Sm_i(1^k)$: $(\Lambda, \overrightarrow{xpky}) \leftarrow SmGen_i(1^k)$, $\widetilde{\overrightarrow{xpky}} \leftarrow \pi(\overrightarrow{xpky})$, finally outputs $(\Lambda, \widetilde{\overrightarrow{xpky}})$.
3) *Distinguishability. Intuitively speaking, it requires that the DI can distinguish the projective instances and smooth instances with help of their witnesses. That is, it requires that the DI correctly computes the following function.*
$$\zeta : \mathbb{N} \times (\{0,1\}^*)^3 \rightarrow \{0,1\}$$
$$\zeta(1^k, \Lambda, x, w) = \begin{cases} 0 & \text{if } (x, w) \in \dot{R}_\Lambda, \\ 1 & \text{if } (x, w) \in \ddot{R}_\Lambda, \\ \text{undefined} & \text{otherwise .} \end{cases}$$
4) *Hard Subset Membership. Intuitively speaking, it requires that for any $\vec{x} \in Range(IS(1^k, \Lambda))$, $\vec{x}$ can be disordered without being detected. That is, for any $\pi \in \Pi$, the two probability ensembles $HSM_1 \stackrel{def}{=} \{HSM_1(1^k)\}_{k\in\mathbb{N}}$ and $HSM_2 \stackrel{def}{=} \{HSM_2(1^k)\}_{k\in\mathbb{N}}$, specified as follows, are computationally indistinguishable, i.e., $HSM_1 \stackrel{c}{=} HSM_2$.*
$HSM_1(1^k)$: $\Lambda \leftarrow PG(1^k)$, $\vec{a} \leftarrow IS(1^k, \Lambda)$, finally outputs $(\Lambda, \vec{x}^{\vec{a}})$.
$HSM_2(1^k)$: Operates as same as $HSM_1(1^k)$ with an exception that finally outputs $(\Lambda, \pi(\vec{x}^{\vec{a}}))$.
5) *Feasible Cheating. Intuitively speaking, it requires that there is a way to cheat to generate a $\vec{x}$ which is supposed to fall into $L^h_{\dot{R}_\Lambda} \times L^t_{\ddot{R}_\Lambda}$ but actually falls into $L^n_{\dot{R}_\Lambda}$ without being caught. That is, for any $\pi \in \Pi$, for any $\pi' \in \Pi$, the two probability ensembles HSM_2 and $HSM_3 \stackrel{def}{=} \{HSM_3(1^k)\}_{k\in\mathbb{N}}$ are computationally indistinguishable, i.e., $HSM_2 \stackrel{c}{=} HSM_3$, where HSM_2 is defined above and HSM_3 is defined as follows.*
$HSM_3(1^k)$:$\Lambda \leftarrow PG(1^k)$, $\vec{a} \leftarrow Cheat(1^k)$, finally outputs $(\Lambda, \pi'(\vec{x}^{\vec{a}}))$.

[42] shows that $SPHDHC_{t,h}$ can be instantiated under various hardness assumptions, such as the decisional Diffie-Hellman assumption, the decisional N-th residuosity assumption, the decisional quadratic residuosity assumption. Please see [42] for such instantiations.

3 Constructing A Framework For OT^n_h Against Covert Adversaries

In this section, we construct a framework for OT^n_h against covert adversaries. In the framework, we will use a PPT algorithm, denoted Γ , that receiving $B_1, B_2 \in \Psi$, outputs a uniformly chosen permutation $\pi \in_U \Pi$ such that $\pi(B_1) = B_2$, i.e., $\pi \leftarrow \Gamma(B_1, B_2)$. We give an example implementation of Γ as follows.

$\Gamma(B_1, B_2)$: First, $E \leftarrow \emptyset$, $C \leftarrow [n] - B_1$. Second, for each $j \in B_2$, then $i \in_U B_1$, $B_1 \leftarrow B_1 - \{i\}$, $E \leftarrow E \cup \{j \rightleftharpoons i\}$. Third, $D \leftarrow [n] - B_2$, for each $j \in D$, then $i \in_U C$, $C \leftarrow C - \{i\}$, $E \leftarrow E \cup \{j \rightleftharpoons i\}$. Fourth, define π as $\pi(i) = j$ if and only if $j \rightleftharpoons i \in E$. Finally, outputs π.

3.1 The Detailed Framework For OT^n_h Against Covert Adversaries

- Common inputs: the public security parameter k, a $SPHDHC_{t,h}$ (where $n = h + t$) hash system $\mathcal{H}$, the number of instance vector P_2 (i.e., the receiver) should generate K, the number of instance vector P_2 should open g, where K, g are positive integer, $g < K$, and $K \leq poly(k)$.

- Private Inputs: Party P_1 (i.e., the sender) holds a private input $\vec{m} \in (\{0,1\}^*)^n$ and a randomness $r_1 \in \{0,1\}^*$. Party P_2 holds a private input $H \in \Psi$ and a randomness $r_2 \in \{0,1\}^*$. The adversary $\mathcal{A}$ holds a name list $I \subseteq [2]$ and a randomness $r_{\mathcal{A}} \in \{0,1\}^*$.
- Auxiliary Inputs: The adversary $\mathcal{A}$ holds an infinite auxiliary input sequence $z = (z_k)_{k \in \mathbb{N}}, z_k \in \{0,1\}^*$.

The protocol are described as follow.

- Convention: For clarity in description, we make conventions here about some trivial error-handlings such as P_1 refusing to send P_2 some message which is supposed to be sent, P_1 sending a invalid message that P_2 can not process. Handling such errors is easy. P_2 halting and outputting $abort_1$ suffices. P_1 can handle such errors in a similar way. In the following description, we will not explicitly iterate such details any more.
- Receiver's step (R1): P_2 generates hash parameters and samples instances.
  1) P_2 samples K instance vectors.
     P_2 does: $\Lambda \leftarrow PG(1^k)$; for each $i \in [K]$, $\vec{a}_i \leftarrow IS(1^k, \Lambda)$. Without loss of generality, we assume $\vec{a}_i = ((\dot{x}_1, \dot{w}_1), \ldots, (\dot{x}_h, \dot{w}_h), (\ddot{x}_{h+1}, \ddot{w}_{h+1}), \ldots, (\ddot{x}_n, \ddot{w}_n))^T$.
  2) P_2 disorders each instance vector.
     For each $i \in [K]$, P_2 uniformly chooses a permutation $\pi_i^1 \in_U \Pi$, then $\tilde{\vec{a}}_i \leftarrow \pi_i^1(\vec{a}_i)$.
  3) P_2 sends the instance vectors and the corresponding hash parameters to P_1.
     P_2 sends $(\Lambda, \tilde{\vec{x}}_1, \tilde{\vec{x}}_2, \ldots, \tilde{\vec{x}}_K)$, where $\tilde{\vec{x}}_i \stackrel{def}{=} \vec{x}^{\tilde{\vec{a}}_i}$ (correspondingly, $\tilde{\vec{w}}_i \stackrel{def}{=} \vec{w}^{\tilde{\vec{a}}_i}$), to P_1.
- Sender's step (S1): P_1 choose g instance vectors at random to open.
  P_1 does: $r \in_U \{0,1\}^K$ such that $\#\{i | r\langle i \rangle = 1, i \in [K]\} = g$, and sends r to P_2.
  r indicates that the instance vectors whose indexes fall into $CS \stackrel{def}{=} \{i | r\langle i \rangle = 1, i \in [K]\}$ (correspondingly, $\overline{CS} \stackrel{def}{=} [K] - CS$) are chosen by P_1 to be open. We call such r choose indicator from now on.
- Receiver's step (R2): P_2 opens the chosen instances to P_1, encodes and sends his private input to P_1.
  1) P_2 verifies that the choice indicator is legal, i.e. the number of 1-bits contained in r just is g. If r is not legal, P_2 halts and outputs $abort_1$; otherwise, P_2 proceeds to the next step.
  2) P_2 sends the witnesses of the chosen instances to prove that the instances generated by him are legal.
     P_2 sends $((i, j, \tilde{\vec{w}}_i\langle j \rangle))_{i \in CS, j \in J_i}$ to P_1, where $J_i \stackrel{def}{=} \{j | \tilde{\vec{x}}_i\langle j \rangle \in L_{\ddot{R}_\Lambda}, j \in [n]\}$.
  3) P_2 encodes his private input and sends the resulting code to P_1.
     Let $G_i \stackrel{def}{=} \{j | \tilde{\vec{x}}_i\langle j \rangle \in L_{\dot{R}_\Lambda}, i \in \overline{CS}\}$. For each $i \in \overline{CS}$, P_2 does $\pi_i^2 \leftarrow \Gamma(G_i, H)$, sends $(\pi_i^2)_{i \in \overline{CS}}$ to P_1. That is, P_2 encodes his private input into sequences such as $\pi_i^2(\tilde{\vec{x}}_i)$ where $i \in \overline{CS}$.

  We comment that, to prove instances' legality, there is no need for P_2 to send the witnesses of the projective instances of the chosen instance vectors to P_1, since P_1 only care whether each chosen instance vector contains enough smooth instances, which we will see later. We also comment that P_2 can send $((i, j, \tilde{\vec{w}}_i\langle j \rangle))_{i \in CS, j \in J_i}$ and $(\pi_i^2)_{i \in \overline{CS}}$ in one step.
- Sender's step (S2): P_1 checks the chosen instances, sends his private input after encrypting them to P_2.
  1) P_1 verifies that each chosen instance vector is legal, i.e., for each chosen instance vector the number of the entries belonging to $L_{\ddot{R}_\Lambda}$ is at least $n - h$.
     P_1 checks that, for each $i \in CS$, $\#J_i \geq n - h$, and for each $j \in J_i$, $DI(1^k, \Lambda, \tilde{\vec{x}}_i\langle j \rangle, \tilde{\vec{w}}_i\langle j \rangle)$ is 1. If P_2 does not send the opening or the check fails, P_1 halts and outputs $corrupted_2$, otherwise P_1 proceeds to next step.
  2) P_1 reorders the entries of each unchosen instance vector in the way told by P_2.
     For each $i \in \overline{CS}$, P_1 does $\tilde{\tilde{\vec{x}}}_i \leftarrow \pi_i^2(\tilde{\vec{x}}_i)$.
  3) P_1 encrypts and sends his private input to P_2 together with some auxiliary messages.
     For each $i \in \overline{CS}$, $j \in [n]$, P_1 does: $(hk_{ij}, pk_{ij}) \leftarrow KG(1^k, \Lambda, \tilde{\tilde{\vec{x}}}_i\langle j \rangle)$, $\beta_{ij} \leftarrow Hash(1^k, \Lambda, hk_{ij}, \tilde{\tilde{\vec{x}}}_i\langle j \rangle)$, $\vec{\beta}_i \stackrel{def}{=} (\beta_{i1}, \beta_{i2}, \ldots, \beta_{in})^T$, $\vec{c} \leftarrow \vec{m} \oplus (\oplus_{i \in \overline{CS}} \vec{\beta}_i)$, $\overrightarrow{pk}_i \stackrel{def}{=} (pk_{i1}, pk_{i2}, \ldots, pk_{in})^T$, sends $\vec{c}$ and $(\overrightarrow{pk}_i)_{i \in \overline{CS}}$ to P_2.
- Receiver's step (R3): P_2 decrypts the ciphertext $\vec{c}$ and gains the message he want.
  For each $i \in \overline{CS}$, $j \in H$, P_2 operates: $\beta'_{ij} \leftarrow pHash(1^k, \Lambda, \tilde{\tilde{\vec{x}}}_i\langle j \rangle, \tilde{\tilde{\vec{w}}}_i\langle j \rangle, \overrightarrow{pk}_i\langle j \rangle)$, $m'_j \leftarrow \vec{c}\langle j \rangle \oplus (\oplus_{i \in \overline{CS}} \beta'_{ij})$. Finally, P_2 gains the messages $(m'_j)_{j \in H}$.

3.2 The Correctness Of The Framework

In our framework, the main use of the witnesses of an instance $\dot{x} \in L_{\dot{R}_\Lambda}$ is to project and gain the hash value of $\dot{x}$. In contrast, with respect to an instance $\ddot{x} \in L_{\ddot{R}_\Lambda}$, the witness services as a proof of $\ddot{x} \in L_{\ddot{R}_\Lambda}$. This means that a receiver can use the witnesses of $\ddot{x}$ to persuade a sender to believe that the receiver is unable to gain the hash value of $\ddot{x}$.

Now let us check the correctness of the framework, i.e., the framework works in case P_1 and P_2 are honest. For each $i \in \overline{CS}$, $j \in H$, we know

$$\vec{c}\langle j \rangle = \vec{m}\langle j \rangle \oplus (\oplus_{i \in \overline{CS}} \vec{\beta}_i\langle j \rangle)$$
$$m'_j = \vec{c}\langle j \rangle \oplus (\oplus_{i \in \overline{CS}} \beta'_{ij})$$

Because of the projection of $\mathcal{H}$, we know

$$\vec{\beta}_i\langle j \rangle = \beta'_{ij}$$

So we have

$$\vec{m}\langle j \rangle = m'_j$$

This means what P_2 gets is $\vec{m}\langle H \rangle$ that indeed is P_2 wants.

3.3 The Security Of The Framework

With respect to the security of the framework, we have the following theorem.

Theorem 4 (The framework is secure against covert adversaries)**.** *Assume that $\mathcal{H}$ is t-smooth h-projective hash family that holds properties distinguishability, hard subset membership, feasible cheating ($SPHDHC_{t,h}$). Then, the framework securely computes h-out-of-n oblivious transfer functionality in the presence of non-adaptive covert adversaries with deterrence factor $1 - 1/C_K^{K-g}$.*

Before prove Theorem 4, we first give an intuitive analysis as a warm-up. For the security of P_1, the framework should prevent P_2 from gaining more than h messages. Using cut and choose technique, P_1 can detect with probability $1 - \frac{g}{K}$ the cheating that P_2 generates a instance vector containing more than h projective instance. Unexpectedly, we find that the probability that P_2 cheats to obtain more than h messages may be far less than $\frac{g}{K}$.

Theorem 5. *In case P_1 is honest and P_2 is corrupted, the probability that P_2 cheats to obtain more than h messages is at most $1/C_K^{K-g}$.*

Proof: According to the framework, there are following necessary conditions for P_2's success in the cheating.

1) P_2 has to generate at least one illegal $\vec{x}_i$ which contains more than h entries belonging to $L_{\dot{R}_\Lambda}$. If not, P_2 cann't correctly decrypt more than h entries of $\vec{c}$, because of the smoothness of $\mathcal{H}$. Without loss of generality, we assume the illegal instance vectors are $\vec{x}_{l_1}, \vec{x}_{l_2}, \ldots, \vec{x}_{l_d}$.
2) All illegal instance vectors are lucky not to be chosen and all the instance vectors unchosen just are the illegal instance vectors, i.e., $\overline{CS} = \{l_1, l_2, \ldots, l_d\}$. We prove this claim in two cases.
   a) In case $\overline{CS} \neq \{l_1, l_2, \ldots, l_d\}$ and $\overline{CS} - \{l_1, l_2, \ldots, l_d\} = \emptyset$, there exists $j(j \in [d] \wedge l_j \in CS)$. So P_1 can detect P_2's cheating and P_2 will gain nothing.
   b) In case $\overline{CS} \neq \{l_1, l_2, \ldots, l_d\}$ and $\overline{CS} - \{l_1, l_2, \ldots, l_d\} \neq \emptyset$, there exists $j(j \in \overline{CS} \wedge \vec{x}_j$ is legal). Because of the smoothness of $\mathcal{H}$, P_2 cannot correctly decrypt more than h entries of $\vec{c}$.
3) The number of illegal instance vectors just are $K-g$, i.e., $d = K-g$. On one side, in the framework, P_1 choose g instance vectors to open, therefore the number of illegal instance vectors has to be no more than $K-g$ to avoid being caught. On another side, following the analysis of the secondary condition, to learn extra messages for P_2, the number of illegal instance vectors has to be no less than $K-g$ to make possible that all d unchosen instance vectors are illegal.

Let us estimate the probability that both the second and the third necessary conditions are met. We have

$$\begin{aligned} Pr(\overline{CS} = \{l_1, l_2, \ldots, l_d\} \wedge d = K-g) &= 1/C_K^d \\ &= 1/C_K^{K-g} \end{aligned}$$

This means that the probability that P_2 cheats to obtain more than h messages is at most $1/C_K^{K-g}$. □

From the proof of Theorem 5, we know there is a possibility that P_2 commits a not effective cheating in the sense that even if such cheating is not detected by P_1, P_2 still learns no new knowledge. What is more, such not effective cheating even diminish the amount of knowledge P_2 deserves. Therefore, the deterrence factor is at least $1 - 1/C_K^{K-g}$. Following the properties of binomial factor, the maximum lower-bound of the deterrence factor can be achieved by setting g to be $K - \llcorner \frac{K+1}{2} \lrcorner$. This is in contrast to our intuition that the more instance vectors chosen to be open, the higher the deterrence factor is. The essential reason is that our intuition is apt to neglect the confusion induced by $SPHDHC_{t,h}$'s property smoothness. Since K, g are predetermined constant value, the distance between the deterrence factor and 1 can not be negligibly small. However, a very small distance is achievable. For example, setting K, g to be $40, 20$ respectively, we gain a deterrence factor $\epsilon = 1 - 7.25 \times 10^{-12}$.

For the security of P_2, the framework first should prevent P_1 from learning P_2's private input. There is a possible leakage of P_2's private input in Step R2 where P_2 encodes his private input. However, $SPHDHC_{t,h}$'s property hard subset membership guarantees that for any $\vec{x} \in Range(IS(1^k, \Lambda))$, any $\pi \in \Pi$, any PPT adversary $\mathcal{A}$, without being given π, the advantage of $\mathcal{A}$ identifying an entry of $\pi(\vec{x})$ falling into $L_{\dot{R}_\Lambda}$ (resp., $L_{\ddot{R}_\Lambda}$) with probability over prior knowledge h/n (resp., t/n) is negligible. That is, seen from $\mathcal{A}$, every entry of $\pi(\vec{x})$ seems the same. This implies that the receiver encodes his private inputs without leaking any information in our framework.

Besides cheating P_2 of private input, it seems there is another obvious attack that malicious P_1 sends invalid messages, e.g. pk_{ij} that $(hk_{ij}, pk_{ij}) \notin Range(KG(1^k, \Lambda, x_{ij}))$, to P_2. This attack in fact doesn't matter. Its effect is equal to that of P_1's altering his real input, which is allowed in the ideal world too. Seen from intuition, it seems that there is no way for P_1 to cheat out the inputs of P_2. Indeed, our security proof later shows this intuition is correct.

We now proceeds to prove Theorem 4 holds. For notational clarity, we denote the parties and the adversary in the real world by P_1, P_2, $\mathcal{A}$, and denote the corresponding entities in the ideal world by P_1', P_2', $\mathcal{S}$. In the light of the parties corrupted by adversaries, there are four cases to be considered and we separately prove Theorem 4 holds in each case.

3.3.1 *In Case P_1 Is Corrupted*

In case P_1 is corrupted, $\mathcal{A}$ takes the full control of P_1 in the real world. Correspondingly, $\mathcal{A}$'s simulator, $\mathcal{S}$, takes the full control of P_1' in the ideal world, where $\mathcal{S}$ is constructed as follow.

- Initial input: $\mathcal{S}$ holds the same k, $I \stackrel{def}{=} \{1\}$, $z = (z_k)_{k\in\mathbb{N}}$, as $\mathcal{A}$. What is more, $\mathcal{S}$ holds a uniform distributed randomness $r_{\mathcal{S}} \in \{0,1\}^*$. The parties P_1' and P_1, whom $\mathcal{S}$ and $\mathcal{A}$ respectively are to corrupt, hold the same $\vec{m}$.
- $\mathcal{S}$ works as follows.
  - Step Sim1: $\mathcal{S}$ performs initialization operations in this step. Specifically, first, $\mathcal{S}$ corrupts P_1' and learns P_1''s private input $\vec{m}$. Second, let $\bar{\mathcal{A}}$ be a copy of $\mathcal{A}$, i.e., $\bar{\mathcal{A}} = \mathcal{A}$. $\mathcal{S}$ use $\bar{\mathcal{A}}$ as a subroutine. $\mathcal{S}$ fixes the initial inputs of $\bar{\mathcal{A}}$ to be his own initial input, i.e., k, I, z, with exception that fixes the randomness of $\bar{\mathcal{A}}$ to be a uniformly distributed value. Third, $\mathcal{S}$ activates $\bar{\mathcal{A}}$.
    We comments that, during what follows until $\bar{\mathcal{A}}$ halts, $\mathcal{S}$ builds an environment for $\bar{\mathcal{A}}$ which simulates the real world. That is, $\mathcal{S}$ simultaneously disguises himself as P_1 and P_2 to interact with $\bar{\mathcal{A}}$.
    Fourth, before engaging in the framework for OT_h^n, $\bar{\mathcal{A}}$ sends a message as in the real world indicating to corrupt P_1. $\mathcal{S}$ plays the role of P_1 to supplies $\bar{\mathcal{A}}$ with $\vec{m}$. Now, $\bar{\mathcal{A}}$ is engaging the framework.
  - Convention: In the following interactions between $\bar{\mathcal{A}}$ and $\mathcal{S}$, in case $\bar{\mathcal{A}}$ refuses to send some message which is supposed to be sent or $\bar{\mathcal{A}}$ sends a invalid message that $\mathcal{S}$ can not process, $\mathcal{S}$ sends $abort_1$ to the TTP, and halts with outputting whatever $\bar{\mathcal{A}}$ outputs. We make this convention here, and will not explicitly iterate such details any more.
  - Step Sim2: $\mathcal{S}$ repeats the following procedure, denoted Υ, until the output of Υ is λ. In each repeating, all randomness used by $\mathcal{S}$ is fresh. On finishing the repeating, $\mathcal{S}$ proceeds to the next step.
    * Procedure Υ:
      1) $\mathcal{S}$ rewinds $\bar{\mathcal{A}}$ to the beginning of Step S1 of the framework.
      2) $\mathcal{S}$ uniformly chooses a choice indicator as a honest party P_1 does in Step S1 of the framework. $\mathcal{S}$ executes receiver's Step R1 of the framework with following exception, for each $\bar{r}\langle i\rangle = 1$, $\mathcal{S}$ honestly generates instance vector $\vec{x}_i$; for each $\bar{r}\langle i\rangle = 0$, $\mathcal{S}$ call $Cheat(1^k, \Lambda)$ to generate instance vector $\vec{x}_i$.
      3) $\mathcal{S}$ receives a choice indicator r from $\bar{\mathcal{A}}$.
      4) If $r = \bar{r}$, $\alpha \leftarrow \lambda$; otherwise, $\alpha \leftarrow \perp$. Finally outputs α.

    Remark 6. *Step Sim2 is a key step to extract $\bar{\mathcal{A}}$'s real input. Note that, in case event $r = \bar{r}$ happens, then each entry of each unchosen instance vector is projective. $SPHDHC_{t,h}$'s property feasible cheating guarantees that $\bar{\mathcal{A}}$ can't detect this cheating. Recalling the proof of Theorem 5, we know $\mathcal{S}$ can succeed in extracting $\bar{\mathcal{A}}$'s real input.*
  - Step Sim3: Playing the role of P_2 who takes an arbitrary element of Ψ as private input, $\mathcal{S}$ honestly execute receiver's Step R2 to interact with $\bar{\mathcal{A}}$. On receiving $\vec{c}$, $(\overrightarrow{pk}_i)_{i\in\overline{CS}}$ sent by $\bar{\mathcal{A}}$, $\mathcal{S}$ decrypts every entry of $\vec{c}$, and gains $\bar{\mathcal{A}}$'s real input $\vec{m}'$.
  - Step Sim4: $\mathcal{S}$ sends $\vec{m}'$ to TTP and receives back a message $\alpha \in \{abort_1, corrupt_1, \lambda\}$.
  - Step Sim5: When $\bar{\mathcal{A}}$ halts, $\mathcal{S}$ halts with outputting what $\bar{\mathcal{A}}$ outputs.

Lemma 7. *The simulator $\mathcal{S}$ is expected polynomial-time.*

Proof: First, let us focus on Step Sim2. Note that the number of legal choice indicators is C_K^g. So we have $Pr(r = \bar{r}) = 1/C_K^g$. Therefore, the expected number of repeating Υ is C_K^g which is a predetermined constant value. Then, Step Sim2 runs expected polynomial-time.

Second, let us focus on other steps of $\mathcal{S}$. Obviously, they all run strict polynomial-time. Following the above analysis, $\mathcal{S}$ is expected polynomial-time. □

However, the security definition requires a strictly polynomial-time simulator. Observer that, the reason why the above $\mathcal{S}$ doesn't run strictly polynomial-time is that the probability that Υ outputs $\perp$ is relatively too high. The following lemma guarantees that we can get a new version of Υ that run strictly polynomial-time and outputs $\perp$ negligible probability.

Lemma 8. *Given a PPT machine M that outputs $\perp$ with probability p, where $p \in (0,1)$ is a constant value, we can have a PPT machine $\widetilde{M}$ such that*

$$\begin{aligned}\forall x \to & Pr(\widetilde{M}(x) = \perp) = \mu(k) \\ \forall x \forall \alpha \to & Pr(\widetilde{M}(x) = \alpha | \widetilde{M}(x) \neq \perp) \\ & = Pr(M(x) = \alpha | M(x) \neq \perp)\end{aligned}$$

holds.

Proof: Let v be some polynomial that $v(k) > 1/p$. We construct $\widetilde{M}$ as follows.

$\widetilde{M}$: on receiving input x, repeats $M(x)$ at most v times. If $M(x)$ outputs something different from $\perp$ in a repetition, then sets α to be this output, and stops repeating; otherwise, set α to be $\perp$. Finally outputs α.

Let $M_i(x)$ be the output of $M(x)$ in i-th repetition. Let X_i be $0/1$ random variable defined as follows.

$$X_i \stackrel{def}{=} \begin{cases} 1, & M_i(x) = \perp; \\ 0, & M_i(x) \neq \perp. \end{cases}$$

Then, we know $E(X_i) = p$, $D(X_i) = p(1-p)$.

$$\begin{aligned} Pr(\widetilde{M}(x) = \perp) &= Pr(\sum_i^v X_i < 1) \\ &= Pr(p - \frac{\sum_i^v X_i}{v} > p - \frac{1}{v}) \\ &\leq Pr(|p - \frac{\sum_i^v X_i}{v}| \geq p - \frac{1}{v}) \end{aligned}$$

Following Chernoff Bound (can be seen in [20]), we have

$$Pr(\widetilde{M}(x) = \perp) \leq \frac{2}{e^{\frac{v}{2\sigma^2}(p-\frac{1}{v})^2}}$$

Therefore, $Pr(\widetilde{M}(x) = \perp)$ is negligible. □

Modifying Step Sim2 as follows

- Step Sim2: $\mathcal{S}$ executes the new version of Υ guaranteed by Lemma 8. If Υ outputs $\perp$, then $\mathcal{S}$ outputs $\perp$ and halts; otherwise, $\mathcal{S}$ proceeds to the next step.

, we gain a new version of simulator. Combining the proof of Lemma 7, we have

Proposition 9. *The new simulator $\mathcal{S}$ is polynomial-time.*

Note that $\mathcal{S}$ never sends $cheat_1$ to TTP. Thus, we actually construct a standard simulator for $\mathcal{A}$, and so provide a standard security against malicious adversaries to P_2. This means we need to prove for any non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ with an infinite sequence $z = (z_k)_{k\in\mathbb{N}}$ in the real world, the following equation holds.

$$\begin{aligned} &\{Ideal_{f,\{1\},\mathcal{S}(z_k)}(1^k, \vec{m}, H)\}_{\substack{k\in\mathbb{N},\vec{m}\in(\{0,1\}^*)^n \\ H\in\Psi, z_k\in\{0,1\}^*}} \stackrel{c}{\equiv} \\ &\qquad \{Real_{\pi,\{1\},\mathcal{A}(z_k)}(1^k, \vec{m}, H)\}_{\substack{k\in\mathbb{N},\vec{m}\in(\{0,1\}^*)^n \\ H\in\Psi, z_k\in\{0,1\}^*}} \end{aligned} \tag{4}$$

Proposition 10. *In case P_1 was corrupted, i.e., $I = \{1\}$, the equation* (4) *holds.*

Before proving Proposition 10, we first prove the some lemmas.

Lemma 11. *The output of $\mathcal{S}$ in the ideal world and the output of $\mathcal{A}$ in the real world are computationally indistinguishable, i.e. the following equation holds.*

$$\begin{aligned} &\{Ideal_{f,\{1\},\mathcal{S}(z_k)}(1^k, \vec{m}, H)\langle 0\rangle\}_{\substack{k\in\mathbb{N},\vec{m}\in(\{0,1\}^*)^n \\ H\in\Psi, z_k\in\{0,1\}^*}} \stackrel{c}{\equiv} \\ &\qquad \{Real_{\pi,\{1\},\mathcal{A}(z_k)}(1^k, \vec{m}, H)\langle 0\rangle\}_{\substack{k\in\mathbb{N},\vec{m}\in(\{0,1\}^*)^n \\ H\in\Psi, z_k\in\{0,1\}^*}} \end{aligned}$$

Proof: First, we claim that the outputs of $\mathcal{S}$ and $\bar{\mathcal{A}}$ are computationally indistinguishable. From the construction of $\mathcal{S}$, we know that the event that $\mathcal{S}$ doesn't take $\bar{\mathcal{A}}$'s output as its output arises only if the event that Υ outputs $\perp$ arises. Since the latter arises with negligible probability, this claim holds.

Second, we claim that the views of $\bar{\mathcal{A}}$ and $\mathcal{A}$ are computationally indistinguishable. The only point $\bar{\mathcal{A}}$'s view is different from $\mathcal{A}$'s is that the unchosen instance vectors $\bar{\mathcal{A}}$ sees are generated by calling $Cheat(1^k, \Lambda)$. $SPHDHC_{t,h}$'s property feasible cheating guarantees such instance vectors are indistinguishable from the that generated honestly. therefore, this claim holds.

Combining the above two claims, this lemma holds. □

Lemma 12. *Let $X \stackrel{def}{=} \{X(1^k, a)\}_{k\in\mathbb{N},a\in\{0,1\}^*}$ and $Y \stackrel{def}{=} \{Y(1^k, a)\}_{k\in\mathbb{N},a\in\{0,1\}^*}$ be two polynomial-time constructible probability ensembles, $X \stackrel{c}{\equiv} Y$, $F \stackrel{def}{=} (f_k)_{k\in\mathbb{N}}$, $f_k : \{0,1\}^* \to \{0,1\}^*$ is polynomial-time computable, then*

$$F(X) \stackrel{c}{\equiv} F(Y)$$

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

Proof: Assume the proposition is false, then there exists a non-uniform PPT distinguisher D with an infinite sequence $z = (z_k)_{k\in\mathbb{N}}$, a polynomial $poly(.)$, an infinite positive integer set $G \subseteq \mathbb{N}$ such that, for each $k \in G$, it holds that

$$\begin{aligned} &|Pr(D(1^k, z_k, a, f_k(X(1^k, a))) = 1) - \\ &\qquad Pr(D(1^k, z_k, a, f_k(Y(1^k, a))) = 1)| \geq 1/poly(k) \end{aligned}$$

We construct a distinguisher D' with an infinite sequence $z = (z_k)_{k\in\mathbb{N}}$ for the ensembles X and Y as follows.

$D'(1^k, z_k, a, \gamma)$: $\delta \leftarrow f_k(\gamma)$, finally outputs $D(1^k, z_k, a, \delta)$.

Obviously, $D'(1^k, z_k, a, X(1^k, a)) = D(1^k, z_k, a, f_k(X(1^k, a)))$, $D'(1^k, z_k, a, Y(1^k, a)) = D(1^k, z_k, a, f_k(Y(1^k, a)))$. So we have

$$\begin{aligned} &|Pr(D'(1^k, z_k, a, X(1^k, a)) = 1) - \\ &\qquad Pr(D'(1^k, z_k, a, Y(1^k, a)) = 1)| \geq 1/poly(k) \end{aligned}$$

This contradicts the fact $X \stackrel{c}{\equiv} Y$. □

Now we proceed to prove Proposition 10.

Proof: First let us focus on the real world. $\mathcal{A}$'s real input can be formulated as $\gamma \leftarrow A(1^k, \vec{m}, z_k, r_{\mathcal{A}}, r_1)$. Note that in this case, P_2's output is a determinate function of $\mathcal{A}$'s real input. Since $\mathcal{A}$'s real input is in its view, without loss of generality, we assume $\mathcal{A}$'s output, denoted α, constains its real input. Therefore, P_2's output is a determinate function of $\mathcal{A}$'s output, where the function is

$$g(\alpha) = \begin{cases} abort_1 & \text{if } \gamma = abort_1, \\ corrupted_1 & \text{if } \gamma = corrupted_1, \\ \gamma\langle H\rangle & otherwise. \end{cases}$$

Let $h(\alpha) \stackrel{def}{=} (\alpha, \lambda, g(\alpha))$. Then we have

$$\begin{aligned} &Real_{\pi,\{1\},\mathcal{A}(z_k)}(1^k, \vec{m}, H) \equiv \\ &\qquad h(Real_{\pi,\{1\},\mathcal{A}(z_k)}(1^k, \vec{m}, H)\langle 0\rangle) \end{aligned}$$

Similarly, in the ideal world, we have

$$\begin{aligned} &Ideal_{f,\{1\},\mathcal{S}(z_k)}(1^k, \vec{m}, H) \stackrel{c}{\equiv} \\ &\qquad h(Ideal_{f,\{1\},\mathcal{S}(z_k)}(1^k, \vec{m}, H)\langle 0\rangle) \end{aligned}$$

We use $\stackrel{c}{=}$ not $\equiv$ here because there is a negligible probability that $\mathcal{S}$ outputs $\perp$, which makes $h(.)$ undefined.

Let $X(1^k, \vec{m}, H, z_k, \{1\}) \stackrel{def}{=} Real_{\pi,\{1\},\mathcal{A}(z_k)}(1^k, \vec{m}, H)\langle 0\rangle$, $Y(1^k, \vec{m}, H, z_k, \{1\}) \stackrel{def}{=} Ideal_{f,\{1\},\mathcal{S}(z_k)}(1^k, \vec{m}, H)\langle 0\rangle$. Following Lemma 11, $X \stackrel{c}{=} Y$. Let $F \stackrel{def}{=} (h)_{k\in\mathbb{N}}$. According to Lemma 12, the proposition holds. □

3.3.2 In Case P_2 Is Corrupted

In case P_2 is corrupted, $\mathcal{A}$ takes the full control of P_2 in the real world. Correspondingly, $\mathcal{S}$ takes the full control of P'_2 in the ideal world. We construct $\mathcal{S}$ as follows.

- Initial input: $\mathcal{S}$ holds the same k, $I \stackrel{def}{=} \{2\}$, $z = (z_k)_{k\in\mathbb{N}}$ as $\mathcal{A}$, and holds a uniformly distributed randomness $r_{\mathcal{S}} \in \{0,1\}^*$. The parties P'_2 and P_2 hold the same private input H.
- $\mathcal{S}$ works as follows.
  - Step Sim1: In this step, $\mathcal{S}$ performs initialization operations in a similar way that the simulator does in case P_1 is corrupted. That is, $\mathcal{S}$ corrupts P'_2 and learns P'_2's private input H. $\mathcal{S}$ takes $\mathcal{A}$'s copy $\bar{\mathcal{A}}$ as a subroutine, fixes $\bar{\mathcal{A}}$'s initial input, activates $\bar{\mathcal{A}}$, supplies $\bar{\mathcal{A}}$ with H. Now, $\bar{\mathcal{A}}$ is engaging the framework.
  - Convention: In the following interactions between $\bar{\mathcal{A}}$ and $\mathcal{S}$, in case $\bar{\mathcal{A}}$ refuses to send some message which is supposed to be sent or $\bar{\mathcal{A}}$ sends a invalid message that $\mathcal{S}$ can not process, $\mathcal{S}$ sends $abort_2$ to the TTP, and halts with outputting whatever $\bar{\mathcal{A}}$ outputs. We make this convention here, and will not explicitly iterate such details any more.
  - Step Sim2: On receiving $(\Lambda, \tilde{\vec{x}}_1, \tilde{\vec{x}}_2, \ldots, \tilde{\vec{x}}_K)$ sent from $\bar{\mathcal{A}}$, $\mathcal{S}$ rewinds $\bar{\mathcal{A}}$ C^g_K times to knows $\bar{\mathcal{A}}$'s distinct responses to distinct choice indicators. Specifically, at the i-th time, $\mathcal{S}$ works as follows.
    1) $\mathcal{S}$ choose a choice indicator r never used. Note that there are just C^g_K choice indicators.
    2) Playing the role of P_1, $\mathcal{S}$ sends r to $\bar{\mathcal{A}}$, records $\bar{\mathcal{A}}$'s response (i.e., a rejection or a corresponding opening $((i, j, \tilde{\vec{w}}_i\langle j\rangle))_{i\in CS, j\in J_i}$).
    3) $\mathcal{S}$ rewinds $\bar{\mathcal{A}}$ to the beginning of Step R2 of the framework.
  - Step Sim3: Now $\mathcal{S}$ knows $\bar{\mathcal{A}}$'s all responses. $\mathcal{S}$ proceeds as follows basing on such knowledge.
    - Case 1, no $\bar{\mathcal{A}}$'s response will cause honest party P_1 outputting $corrupted_2$. $\mathcal{S}$ proceeds to Step Sim4.
    - Case 2, $\bar{\mathcal{A}}$'s all responses will cause honest party P_1 outputting $corrupted_2$. $\mathcal{S}$ sends $corrupted_2$ to TTP and receives back $corrupted_2$ from TTP. Then, playing the role of P_1, $\mathcal{S}$ honestly follows the sender's steps of the framework which begins from Step S1 to interact with $\bar{\mathcal{A}}$. When $\bar{\mathcal{A}}$ halts, $\mathcal{S}$ halts with outputting what $\bar{\mathcal{A}}$ outputs.
    - Case 3, the number of $\bar{\mathcal{A}}$'s responses that will cause honest party P_1 outputting $corrupted_2$ is in the domain $[1, C^g_K - 1]$. $\mathcal{S}$ send $cheat_2$ to TTP. Since $\mathcal{S}$ knows $\bar{\mathcal{A}}$'s all responses, $\mathcal{S}$ knows the the probability that $\bar{\mathcal{A}}$ is being caught in cheating by honest party P_1 in the real world. We denote this probability By ϵ. It is easy to see that $0 < \epsilon < 1$.
      - Case 3.1, with probability ϵ, TTP replies $\mathcal{S}$ with $corrupted_2$. $\mathcal{S}$ uniformly chooses one of the choice indicator that will cause $\bar{\mathcal{A}}$'s cheating to be caught and sends it to $\bar{\mathcal{A}}$. Then, playing the role of P_1, $\mathcal{S}$ honestly follows the sender's steps which begins from Step S2 to interact with $\bar{\mathcal{A}}$.
      - Case 3.2, with probability $1-\epsilon$, TTP replies $\mathcal{S}$ with $undetected$ and honest party P_1's private input $\vec{m}$. Playing the role of P_1 with private input $\vec{m}$, $\mathcal{S}$ honestly follows the sender's steps which begins from Step S1 to interact with $\bar{\mathcal{A}}$.

      When $\bar{\mathcal{A}}$ halts, $\mathcal{S}$ halts with outputting what $\bar{\mathcal{A}}$ outputs.
  - Step Sim4: Playing the role of P_1, $\mathcal{S}$ honestly follows the sender's steps Step S1, Step S2.1, Step S2.2 to interact with $\bar{\mathcal{A}}$. Basing on the instances' witnesses $\mathcal{S}$ records in Step Sim2 and the codes $(\pi^2_i)_{i\in\overline{CS}}$ sent by $\bar{\mathcal{A}}$, $\mathcal{S}$ extracts $\bar{\mathcal{A}}$'s real input H'. Specifically, $\mathcal{S}$ operates as follows: for each $i \in \overline{CS}$, $H'_i \leftarrow \emptyset$; for each $i \in \overline{CS}$ and $j \in [n]$, if $\tilde{\tilde{\vec{x}}}_i\langle j\rangle$ is projective, then $H'_i \leftarrow H'_i \cup \{j\}$; finally, $H' \leftarrow \cap_{i\in\overline{CS}} H'_i$.
  - Step Sim5: $\mathcal{S}$ sends H' to TTP and receives back $\vec{m}\langle H'\rangle$. To proceed to the interaction with $\bar{\mathcal{A}}$, $\mathcal{S}$ fabricates $\vec{m}'$ as follows. For each $i \in H'$, $\vec{m}'\langle i\rangle \leftarrow \vec{m}\langle i\rangle$. For each $i \notin H'$, set $\vec{m}'\langle i\rangle$ to be an arbitrary message of appropriate length.
  - Step Sim6: Playing the role of P_1 with private input $\vec{m}'$, $\mathcal{S}$ follows Step S2.3 to complete the interaction with $\bar{\mathcal{A}}$. When $\bar{\mathcal{A}}$ halts, $\mathcal{S}$ halts with outputting what $\bar{\mathcal{A}}$ outputs.

Proposition 13. *The simulator $\mathcal{S}$ is polynomial-time.*

Looking at the construction of $\mathcal{S}$, each step is polynomial-time. Obviously, Proposition 13 holds.

Lemma 14. *If Case 1 of Step Sim3 happens, then the output of $\mathcal{A}$ in the real world and that of $\bar{\mathcal{A}}$ in the ideal world are indistinguishable.*

Proof: Note that the only point makes $\bar{\mathcal{A}}$'s view different from $\mathcal{A}$'s view is that the ciphertext $\vec{c}'$ $\bar{\mathcal{A}}$ receives is generated by encrypting fabricated $\vec{m}'$ rather than party P_1's private input $\vec{m}$.

In case $\#H' = h$, $SPWH_{h,t}$'s property smoothness directly guarantees that $\bar{\mathcal{A}}$'s view and $\mathcal{A}$'s view are indistinguishable.

In case $\#H' < h$, $\bar{\mathcal{A}}$'s view and $\mathcal{A}$'s are indistinguishable too. This is because $SPWH_{h,t}$'s property smoothness guarantees that for each $j \in [n] - [H']$, what $\bar{\mathcal{A}}$, $\mathcal{A}$ can respectively get from $\vec{c}'\langle j\rangle$, $\vec{c}\langle j\rangle$ are values uniformly distributed over appropriate domains.

There is no case $\#H' > h$, since all instance vectors are legal.

Combining above analysis, we know that $\mathcal{A}$'s output and $\bar{\mathcal{A}}$'s output are indistinguishable. □

We remark that the proof of Lemma 14 also shows that both $\mathcal{A}$'s effective private input and $\bar{\mathcal{A}}$'s are H'.

Lemma 15. *If Case 2 or Case 3 of Step Sim3 happens, then the outputs of $\mathcal{A}$ in the real world and $\bar{\mathcal{A}}$ in the ideal world are identical.*

Proof: Looking at the construction, we know that, in case $\mathcal{S}$ receives back $corruptted_2$ from TTP, during the following interaction with $\bar{\mathcal{A}}$, $\mathcal{S}$ plays the role of honest party P_1 without the need to know $\vec{m}$. In case $\mathcal{S}$ receives back $undetected$ from TTP, $\mathcal{S}$ knows $\vec{m}$ and perfectly plays the role of honest party P_1. Therefore, $\mathcal{A}$'s view and $\bar{\mathcal{A}}$'s view are identical and their outputs are identical. □

Lemma 16. *The output of the adversary $\mathcal{A}$ in the real world and that of the simulator $\mathcal{S}$ in the ideal world are computationally indistinguishable, i.e.,*

$$\{Real_{\pi,\{2\},\mathcal{A}(z_k)}(1^k,\vec{m},H)\langle 0\rangle\}_{\substack{k\in\mathbb{N},\vec{m}\in(\{0,1\}^*)^n\\ H\in\Psi,z_k\in\{0,1\}^*}} \stackrel{c}{=}$$
$$\{Ideal_{f,\{2\},\mathcal{S}(z_k)}(1^k,\vec{m},H)\langle 0\rangle\}_{\substack{k\in\mathbb{N},\vec{m}\in(\{0,1\}^*)^n\\ H\in\Psi,z_k\in\{0,1\}^*}}$$

Proof: First, we claim that the outputs of $\mathcal{S}$ and $\bar{\mathcal{A}}$ are identical. This follows from the fact that $\mathcal{S}$ always takes $\bar{\mathcal{A}}$'s output as his own output.

Second, we claim that the outputs of $\mathcal{A}$ and $\bar{\mathcal{A}}$ are computationally indistinguishable. Looking at Step Sim3 and Step Sim4, the probability that each case happens is identical to that in the real world. Following Lemma 14 and Lemma 15, this claim holds.

Combining the two claims, the proposition holds. □

Proposition 17. *In case P_2 was corrupted, i.e., $I = \{2\}$, the equation* (1) *required by Definition 1 holds.*

Proof: Note that the honest parties P_1 and P_1' end up with outputting nothing. Thus, the fact that the outputs of $\mathcal{S}$ and $\mathcal{A}$ are computationally indistinguishable, which is supported by Proposition 24, directly prove this proposition holds. □

3.3.3 In Other Cases

In case both P_1 and P_2 are corrupted, $\mathcal{A}$ takes the full control of the two corrupted parties. In the ideal world, a similar situation also holds with respect to $\mathcal{S}$, P_1' and P_2'. Liking in previous cases, $\mathcal{S}$ uses $\mathcal{A}$'s copy, $\bar{\mathcal{A}}$, as a subroutine and builds a simulated environment for $\bar{\mathcal{A}}$. $\mathcal{S}$ provids $\bar{\mathcal{A}}$ with P_1' and P_2''s initial inputs before $\bar{\mathcal{A}}$ engages in the protocol. When $\bar{\mathcal{A}}$ halts, $\mathcal{S}$ halts with outputting what $\bar{\mathcal{A}}$ outputs. Obviously, $\mathcal{S}$ runs in strictly polynomial-time and the equation (1) required by Definition 1 holds in this case.

In case none of P_1 and P_2 is corrupted. The simulator $\mathcal{S}$ is constructed as follows. $\mathcal{S}$ uses $\bar{\mathcal{A}}$, $\bar{P}_1$, $\bar{P}_2$ as subroutines, where $\bar{\mathcal{A}}$, $\bar{P}_1$, $\bar{P}_2$, respectively, is the copy of $\mathcal{A}$, P_1 and P_2. $\mathcal{S}$ fixes $\bar{\mathcal{A}}$'s initial inputs in the same way as in previous cases. $\mathcal{S}$ chooses an arbitrary $\bar{\vec{m}} \in (\{0,1\}^*)^n$ and a uniformly distributed randomness $\bar{r}_1$ as $\bar{P}_1$'s initial inputs. $\mathcal{S}$ chooses an arbitrary $\bar{H} \in \Psi$ and a uniformly distributed randomness $\bar{r}_2$ as $\bar{P}_2$'s initial inputs. $\mathcal{S}$ actives these subroutines and make the communication between $\bar{P}_1$ and $\bar{P}_2$ available to $\bar{\mathcal{A}}$. Note that, in case none of P_1 and P_2 is corrupted, what adversaries can see in real life only is the communication between honest parties. When $\bar{\mathcal{A}}$ halts, $\mathcal{S}$ halts with outputting what $\bar{\mathcal{A}}$ outputs. Obviously, $\mathcal{S}$ runs in strictly polynomial-time and the equation (1) required by Definition 1 holds in this case.

3.4 The Communication Rounds

Step R3 is performed without communication. Each of the rest steps is performed in one round. Thus, the number of the total communication rounds is four.

Seen from practical use, there is no sense to compare our work to the works whose security are not guaranteed in the implementation or the works that can not be implemented at present. Therefore, we do not consider any protocols presented by [18], [26], [37], [38] in the rest of this paper.

Compared with existing fully-simulatable protocols for OT_h^n, our framework is the most efficient one in communications rounds. On counting the total communication rounds of a protocol, we count that of the modified version. In the modified version, the consecutive communications of the same direction are combined into one round. The protocol for $OT_{h\times 1}^n$ of [7] costs one, two zero-knowledge proofs of knowledge respectively in initialization and in transfer a message, where each zero-knowledge proofs of knowledge is performed in four rounds. The whole protocol costs at least ten rounds. The protocol for OT_h^n of [24] costs one zero-knowledge proof of knowledge in initialization which is performed in three rounds at least, one protocol to extract a secret key corresponding to the identity of a message which is performed in four rounds, one zero-knowledge proof of knowledge in transfer a message which is performed in three rounds at least. We point out that the interactive proof of knowledge of a discrete logarithm modulo a prime, presented by [40] and taken as a zero-knowledge proof of knowledge protocol in [24], to our best knowledge, is not known to be zero-knowledge. However, turning to the techniques of Σ-protocol, [12] make it zero-knowledge at cost of increment of three rounds in communication, which in turn induces the increment in communication rounds of the protocol of [24]. Taking all into consideration, this protocol costs at least ten rounds. The framework for OT_h^n of [42] costs six rounds.

Compared with existing fully-simulatable protocols for OT_1^2 that works without a trusted common reference string (CRS), our framework is also the most efficient one in communications rounds. The protocol presented by [31] can be viewed as a special DDH-based instantiation of the framework in [42], though modifying the protocol to some extent is needed. Therefore, there is no need to considered this protocol separately. In the rest of this paper, we will not repeat this point again. The protocol presented by [3] is secure against covert adversaries with deterrence factor $\frac{1}{2}$ and costs four rounds.

We point that the protocols presented in [7], [24], [31], [42] are secure against malicious adversaries, which leads to the security level of the protocols are higher than that of ours. However, as pointed out by [3], the adversaries are not always malicious and ready to cheat at any price. Therefore, in such setting, the mentioned protocol seems too pessimistic and our framework indeed makes sense.

3.5 The Computational Overhead

We measure the computational overhead of a protocol mainly in terms of the number of public key operations (i.e. operations based on trapdoor functions, or similar operations), because the overhead of public key operations, which depends on the length of their inputs, is greater than that of symmetric key operations (i.e. operations based on one-way functions) by orders of magnitude. However, in comparison of two protocols, the number of private key operations is also taken into consideration, if their overhead of public key operations are same. Please see [32] to know which cryptographic operation is public key operation or private key operation.

As to our framework, the public key operations are $Hash(.)$ and $pHash(.)$. In Step S2, P_1 takes $n \cdot (K - g)$ invocations of $Hash(.)$ to encrypt his private input. In Step R3, P_2 takes $h \cdot (K - g)$ invocations of $pHash(.)$ to decrypt the messages he want. To compare with existing fully-simulatable protocols in computational overhead, we instantiate our framework with the DDH-based $SPHDHC_{t,h}$ presented by [42] and set K, g to be $2, 1$ respectively. In the resulting DDH-based instantiation, P_1 costs n public key encryption operations, P_2 costs h public key decryption operations, and the deterrence factor is $\frac{1}{2}$.

Compared with existing fully-simulatable protocols for OT_h^n, our DDH-based instantiation is the most efficient one in computational overhead. The operations of the protocol in [7] are based on the non-standard assumptions, i.e., q-Power Decisional Diffie-Hellman and q-Strong Diffie-Hellman assumptions. Such operations are more expensive than DDH-based operations. The operations of the protocol in [24] are based on Decisional Bilinear Diffie-Hellman (DBDH) assumption. Since bilinear curves are considerably more expensive than regular Elliptic curves [17] and DDH is obtainable from Elliptic curves, the DBDH-based operations are also considerably more expensive than that based on DDH. The most efficient instantiation of the framework presented by [42] are DDH-based too. However, used in practice, the instantiation costs P_1 $40n$ public key encryption operations and costs P_2 $40h$ public key decryption operations in the worst case. The computational overhead in the average case is half of that in the worst case. Therefore, our DDH-based instantiation are the most efficient protocol for OT_h^n.

Compared with existing fully-simulatable protocols for OT_1^2 that works without a trusted CRS, our DDH-based instantiation for OT_1^2 is also the most efficient one in computational overhead. The operations of the protocol for OT_1^2 in [3] are based on homomorphic encryption. Fixing the same functionality OT_1^2, the number of public operations of the protocol and our instantiation are equivalent. However, to our best knowledge, there no homomorphic encryption whose operations are more efficient than or are as efficient as that based on DDH.

3.6 Extensions

In our original framework, we require K, g to be predetermined constant values. However, K, g can be relaxed to be any functions in k with a limitation that C_K^g is polynomial in k. The reason why we place such limitation on K, g is to make sure that the simulators runs in strictly polynomial time.

4 Constructing A Framework For OT_h^n Against Malicious Adversaries

4.1 The Detailed Framework For OT_h^n Against Malicious Adversaries

In this section, we present a framework for OT_h^n against malicious adversaries. This framework is obtained by modifying the framework against covert adversaries presented in Section 3.1 as follows.

- Common inputs: Remove the limitation that K is a predetermined constant. Instead, K is relaxed to be a function $K = poly(k)$. Remove the parameter g from the common inputs.
- Sender's step (S1): Remove the limitation placed on way of P_1 choosing the choice indicator. Instead, P_1 is granted to uniformly choose choice indicator over $\{0, 1\}^K$.
- Receiver's step (R2): Remove the step that P_2 checks the legality of the choice indicator.

It is easy to verify the correctness of the framework. So, we omit such details.

4.2 The Security Of The Framework

Theorem 18 (The framework is secure against malicious adversaries)**.** *Assume that $\mathcal{H}$ is a t-smooth h-projective hash family that holds properties distinguishability, hard subset membership, feasible cheating ($SPHDHC_{t,h}$). Then, the protocol securely computes h-out-of-n oblivious transfer functionality in the presence of malicious adversaries under one-sided simulation.*

As previous, before proving Theorem 18, we first give an intuitive analysis as a warm-up. With respect to the security of P_1, we have Theorem 19.

Theorem 19. *In case P_1 is honest and P_2 is corrupted, the probability that P_2 cheats to obtain more than h messages is at most $1/2^K$.*

Proof: Compared with previous framework, the necessary conditions for malicious P_2's succeeding in the cheating here are the same ones mentioned in the proof of Theorem 5 except the third necessary condition. Let us estimate the probability that the second necessary condition is met. We have

$$\begin{aligned} Pr(\overline{CS} = \{l_1, l_2, \dots, l_d\}) &= \frac{1}{2^d} \cdot \frac{1}{2^{K-d}} \\ &= 1/2^K \end{aligned}$$

This means that the probability that P_2 cheats to obtain more than h messages is at most $1/2^K$. □

With respect to the security of P_2, the intuitive analysis given to the framework against covert adversary also holds in this case. Therefore, seen from intuition, P_2's security is guaranteed by this new framework. Now, we proceeds to prove Theorem 18.

4.2.1 In Case P_1 Is Corrupted

In case P_1 is corrupted, we don't need to provide a simulator $\mathcal{S}$ for the real adversary $\mathcal{A}$ which is required by the ideal/real simulation paradigm. Instead, we prove that honest party P_2's privacy is guaranteed.

Proposition 20. *In case P_1 was corrupted, i.e., $I = \{1\}$, the equation* (3) *required by Definition 2 holds.*

Proof: Looking at the framework, we know the only difference between $View^{\mathcal{A}}_{\pi,\{1\},\mathcal{A}(z_k)}(1^k, \vec{m}, H)$ and $View^{\mathcal{A}}_{\pi,\{1\},\mathcal{A}(z_k)}(1^k, \vec{m}, \tilde{H})$ are the codes of honest party P_2's private input, i.e., π_i^2s and $\pi_i^2(\tilde{\vec{x}}_i)$s, where $i \in \overline{CS}$.

First, we claim that each π_i^2 is uniformly distributed over Π. Observe that π_i^1 is uniformly distributed. This leads to that G_i is uniformly distributed. Therefore, for any image $j \in [n]$, its pre-image under π_i^2 is is uniformly distributed. That is, π_i^2 is uniformly distributed.

Second, we claim that $\vec{x}_i \overset{c}{=} \pi_i^2(\tilde{\vec{x}}_i)$. Following $SPHDHC_{t,h}$'s property hard membership, we have $\vec{x}_i \overset{c}{=} \pi_i^1(\vec{x}_i) = \tilde{\vec{x}}_i$, and $\tilde{\vec{x}}_i \overset{c}{=} \pi_i^2(\tilde{\vec{x}}_i)$. Therefore, $\vec{x}_i \overset{c}{=} \pi_i^2(\tilde{\vec{x}}_i)$.

Combining the above two claims, we know the codes in $View^{\mathcal{A}}_{\pi,\{1\},\mathcal{A}(z_k)}(1^k, \vec{m}, H)$ and $View^{\mathcal{A}}_{\pi,\{1\},\mathcal{A}(z_k)}(1^k, \vec{m}, \tilde{H})$ are indistinguishable. Thus the proposition holds. □

4.2.2 In Case P_2 Is Corrupted

In case P_2 is corrupted, $\mathcal{A}$ takes the full control of P_2 in the real world. Correspondingly, $\mathcal{S}$ takes the full control of P_2' in the ideal world. We construct $\mathcal{S}$ as follows.

- Initial input: $\mathcal{S}$ holds the same k, $I \overset{def}{=} \{2\}$, $z = (z_k)_{k\in\mathbb{N}}$ as $\mathcal{A}$, and holds a uniformly distributed randomness $r_{\mathcal{S}} \in \{0,1\}^*$. The parties P_2' and P_2 hold the same private input H.
- $\mathcal{S}$ works as follows.
  - Step Sim1: In this step, $\mathcal{S}$ performs initialization operations in a similar way that the simulator for the first framework does in case P_1 is corrupted. That is, $\mathcal{S}$ corrupts P_2' and learns P_2''s private input H. $\mathcal{S}$ takes $\mathcal{A}$'s copy $\bar{\mathcal{A}}$ as a subroutine, fixes $\bar{\mathcal{A}}$'s initial input, activates $\bar{\mathcal{A}}$, supplies $\bar{\mathcal{A}}$ with H. Now, $\bar{\mathcal{A}}$ is engaging the framework.
  - Convention: In the following interactions between $\bar{\mathcal{A}}$ and $\mathcal{S}$, in case $\bar{\mathcal{A}}$ refuses to send some message which is supposed to be sent or $\bar{\mathcal{A}}$ sends a invalid message that $\mathcal{S}$ can not process, $\mathcal{S}$ sends $abort_2$ to the TTP, and halts with outputting whatever $\bar{\mathcal{A}}$ outputs. We make this convention here, and will not explicitly iterate such details any more.
  - Step Sim2: Playing the role of P_1, $\mathcal{S}$ honestly executes the sender's steps until reaching the beginning of Step S2.3. If Step S2.3 is reached, $\mathcal{S}$ records the choice indicator r and the messages, denoted msg, which he sends to $\bar{\mathcal{A}}$. Then $\mathcal{S}$ proceeds to next step. Otherwise, $\mathcal{S}$ halts with outputting what $\bar{\mathcal{A}}$ outputs.
  - Step Sim3: $\mathcal{S}$ repeats the following procedure, denoted Ξ, until the hash parameters and the instance vectors $\bar{\mathcal{A}}$ sends in Step R1 of the framework passes the check. In each repeating, the randomness $\mathcal{S}$ uses is fresh. After the repeating, $\mathcal{S}$ records the choice indicator $\tilde{r}$ and the messages $\bar{\mathcal{A}}$ sent to open the chosen instance vectors in the last repeating.
    Ξ: $\mathcal{S}$ rewinds $\bar{\mathcal{A}}$ to the beginning of Step R2, and honestly follows sender's steps to interact with $\bar{\mathcal{A}}$ which from Step S1 to the beginning of Step S2.3.
  - Step Sim4:
    * Case 1, $r = \tilde{r}$, $\mathcal{S}$ outputs $failure$ and halts;
    * Case 2, $r \neq \tilde{r}$ and $\forall i(r\langle i\rangle \neq \tilde{r}\langle i\rangle \rightarrow r\langle i\rangle = 1 \wedge \tilde{r}\langle i\rangle = 0)$, $\mathcal{S}$ runs from scratch;
    * Case 3, $r \neq \tilde{r}$ and $\exists i(r\langle i\rangle = 0 \wedge \tilde{r}\langle i\rangle = 1)$, $\mathcal{S}$ records arbitrary one of these is, denotes it by e, and proceeds to next step.

  Remark 21. *The aim of Step Sim3 and Sim4 is to extract one of $\bar{\mathcal{A}}$'s possible private inputs. If Case 3 of Step Sim4 happens, then $\mathcal{S}$ knows which instances in $\tilde{\vec{x}}_e$ are smooth. What is more, $\tilde{\vec{x}}_e$ is indeed a legal instance vector. This is because $\tilde{\vec{x}}_e$ passes the check executed by $\mathcal{S}$ in Step Sim3. Combing π_e^2 received in Step Sim2, $\mathcal{S}$ knows the private input of $\bar{\mathcal{A}}$ which corresponds to $\pi_e^2(\tilde{\vec{x}}_e)$.*
  We stress the fact that this is one of $\bar{\mathcal{A}}$'s possible private inputs. Because the elements in $\{\pi_i^2(\tilde{\vec{x}}_i) | i \in [K], r\langle i\rangle = 0\}$ may correspond to distinct private inputs. However, as we will see in the proof of

Proportion 24, this inconsistence does not bring any trouble to the correctness of our simulation because of $SPHDHC_{t,h}$'s property smoothness.
Note that, $\bar{A}$'s initial input is fixed by $\mathcal{S}$ in Step Sim1. So receiving the same messages, $\bar{A}$ responds in the same way. Therefore, rewinding $\bar{A}$ to the beginning of Step R2, sending the message sent in Step Sim2, $\mathcal{S}$ can reproduce the same scenario as he meets in Step Sim2.

– Step Sim5: $\mathcal{S}$ rewinds $\bar{A}$ to the beginning of Step R2 of the framework, and sends msg previously recorded to $\bar{A}$ in order. According to the analysis of Remark 21, $\mathcal{S}$ can extract one of $\bar{A}$'s possible private inputs. We denote this one by H'_e. $\mathcal{S}$ does so and gets H'_e.
– Step Sim6: $\mathcal{S}$ sends H'_e to TTP and receives back $\vec{m}\langle H_e\rangle$. To proceed to the interaction with $\bar{A}$, $\mathcal{S}$ fabricates $\vec{m}'$ as follows. For each $i \in H'_e$, $\vec{m}'\langle i\rangle \leftarrow \vec{m}\langle i\rangle$. For each $i \notin H'_e$, set $\vec{m}'\langle i\rangle$ to be an arbitrary message of appropriate length.
– Step Sim7: Playing the role of P_1 with private input $\vec{m}'$, $\mathcal{S}$ follows Step S2.3 to complete the interaction with $\bar{A}$. When $\bar{A}$ halts, $\mathcal{S}$ halts with outputting what $\bar{A}$ outputs.

Lemma 22. *The simulator $\mathcal{S}$ is expected polynomial-time.*

Proof: First, let us focus on Step Sim3. In each repetition of Ξ, the chosen instance vectors are uniformly distributed. This leads to the probability that $\bar{A}$ passes the check in each repetition is same. Denote this probability by p. The expected time of Step Sim3 is

$$ExpTime_{Sim3} = (1/p) \cdot Time_{\Xi}$$

Under the same analysis, the probability that $\bar{A}$ passes the check in Step Sim2 is p too. Then, the expected time that $\mathcal{S}$ runs once from Step Sim1 to the beginning of Step Sim4 is

$$\begin{aligned} OncExpTime_{Sim1\to Sim4} \leq\ & Time_{Sim1} + Time_{Sim2} \\ & + p \cdot ExpTime_{Sim3} \\ =\ & Time_{Sim1} + Time_{Sim2} \\ & + Time_{\Xi} \end{aligned}$$

Second, let us focus on Step Sim4, especially Case 2. Note that the initial inputs $\mathcal{S}$ holds is same in each trial. Thus the probability that $\mathcal{S}$ runs from scratch in each trial is same. We denote this probability by $1-q$. Then the expected time that $\mathcal{S}$ runs from Step Sim1 to the beginning of Step Sim5 is

$$\begin{aligned} ExpTime_{Sim1\to Sim5} \leq\ & (1 + 1/q) \\ & \cdot (OncExpTime_{Sim1\to Sim4} \\ & + Time_{Sim4}) \\ =\ & (1+1/q) \cdot (Time_{Sim1} + \\ & Time_{Sim2} + Time_{\Xi} + Time_{Sim4}) \end{aligned}$$

The reason why there is 1 in $(1+1/q)$ is that $\mathcal{S}$ has to run from scratch at least one time in any case.

The expected running time of $\mathcal{S}$ in a whole execution is

$$\begin{aligned} ExpTime_{\mathcal{S}} \leq\ & ExpTime_{Sim1\to Sim5} + Time_{Sim5} \\ & + Time_{Sim6} + Time_{Sim7} \\ =\ & (1+1/q) \cdot (Time_{Sim1} + Time_{Sim2} \\ & + Time_{\Xi} + Time_{Sim4}) \\ & + Time_{Sim5} + Time_{Sim6} + Time_{Sim7} \end{aligned} \tag{5}$$

Third, let us estimate the value of q, which is the probability that $\mathcal{S}$ does not run from scratch in a trial. We denote this event by C. It's easy to see that event C happens, if and only if one of the following mutually disjoint events happens.

1) Event B happens, where B denotes the even that $\mathcal{S}$ halts before reaching Step Sim3.
2) Event $\bar{B}$ happens and $R = \tilde{R}$, where R and $\tilde{R}$ respectively denote the random variables which are defined as the choice indicators $\mathcal{S}$ records in Step Sim2 and Step Sim3.
3) Event $\bar{B}$ happens and there exists i such that $R\langle i\rangle = 0 \wedge \tilde{R}\langle i\rangle = 1$.

So

$$\begin{aligned} q =& Pr(C) \\ =& Pr(B) + Pr(\bar{B} \cap R = \tilde{R}) \\ & + Pr(\bar{B} \cap \exists i(R\langle i\rangle = 0 \wedge \tilde{R}\langle i\rangle = 1)) \\ =& Pr(B) + Pr(\bar{B}) \cdot (Pr(R = \tilde{R}|\bar{B}) \\ & + Pr(\exists i(R\langle i\rangle = 0 \wedge \tilde{R}\langle i\rangle = 1)|\bar{B})) \end{aligned} \tag{6}$$

Let $S_1 \stackrel{def}{=} \{(r,\tilde{r})|(r,\tilde{r}) \in (\{0,1\}^K)^2, r = \tilde{r}\}$, $S_2 \stackrel{def}{=} \{(r,\tilde{r})|(r,\tilde{r}) \in (\{0,1\}^K)^2, r \neq \tilde{r}, \forall i(r\langle i\rangle \neq \tilde{r}\langle i\rangle \to r\langle i\rangle = 1 \wedge \tilde{r}\langle i\rangle = 0)\}$, $S_3 \stackrel{def}{=} \{(r,\tilde{r})|(r,\tilde{r}) \in (\{0,1\}^K)^2, r \neq \tilde{r}, \exists i(i \in [K] \wedge r\langle i\rangle = 0 \wedge \tilde{r}\langle i\rangle = 1)\}$. It is easy to see that S_1, S_2, S_3 constitute a complete partition of $(\{0,1\}^K)^2$ and $\#S_1 = 2^K$, $\#S_2 = \#S_3 = (2^K \cdot 2^K - 2^K)/2$.

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

$$Pr(R = \tilde{R}|\bar{B}) = \#S_1/\#(\{0,1\}^K)^2 = 1/2^K \tag{7}$$

and

$$\begin{aligned} Pr(\exists i(R\langle i\rangle = 0 \wedge \tilde{R}\langle i\rangle = 1)|\bar{B}) &= \#S_3/\#(\{0,1\}^K)^2 \\ &= 1/2 - 1/2^{K+1} \end{aligned} \tag{8}$$

Combining equation (6), (7) and (8), we have

$$\begin{aligned} q &= Pr(B) + Pr(\bar{B})(1/2 + 1/2^{K+1}) \\ &= 1/2 + 1/2^{K+1} + Pr(B)/2 + Pr(\bar{B})/2^{K+1} \\ &> 1/2 \end{aligned} \tag{9}$$

Combining equation (5) and (9), we have

$$\begin{aligned} ExpTime_{A'} < 3(& Time_{Sim1} + Time_{Sim2} \\ & + Time_{\Xi} + Time_{Sim4}) \\ & + Time_{Sim5} + Time_{Sim6} + Time_{Sim7} \end{aligned}$$

which means the expected running time of $\mathcal{S}$ is bound by a polynomial. □

Lemma 23. *The probability that $\mathcal{S}$ outputs $failure$ is less than $1/2^{K-1}$.*

Proof: Let X be a random variable defined as the number of the trials in a whole execution. From the proof of Proposition 22, we know two facts. First, $Pr(X = i) = (1-q)^{i-1}q \leq 1/2^{i-1}$. Second, in each trial the event $\mathcal{S}$ outputs $failure$ is the combined event of $\bar{B}$ and $R = \tilde{R}$, and this event happens with probability

$$Pr(\bar{B} \cap R = \tilde{R}) = Pr(\bar{B})Pr(R = \tilde{R}|\bar{B}) \leq Pr(R = \tilde{R}|\bar{B})$$

Combining equation (7), this probability is less than $1/2^K$. Therefore, the probability that $\mathcal{S}$ outputs $failure$ in a whole execution is

$$\sum_{i=1}^{\infty} Pr(X = i)Pr(\bar{B} \cap R = \tilde{R}) < (1/2^K) \cdot \sum_{i=1}^{\infty} 1/2^{i-1} = 1/2^{K-1}$$

□

Lemma 24. *The output of the adversary $\mathcal{A}$ in the real world and that of the simulator $\mathcal{S}$ in the ideal world are computationally indistinguishable, i.e.,*

$$\{Real_{\pi,\{2\},\mathcal{A}(z_k)}(1^k, \vec{m}, H)\langle 0\rangle\}_{\substack{k\in\mathbb{N},\vec{m}\in(\{0,1\}^*)^n \\ H\in\Psi, z_k\in\{0,1\}^*}} \stackrel{c}{=} \{Ideal_{f,\{2\},\mathcal{S}(z_k)}(1^k, \vec{m}, H)\langle 0\rangle\}_{\substack{k\in\mathbb{N},\vec{m}\in(\{0,1\}^*)^n \\ H\in\Psi, z_k\in\{0,1\}^*}}$$

Proof: First, we claim that the outputs of $\mathcal{S}$ and $\bar{\mathcal{A}}$ are computationally indistinguishable. The only point that the output of $\mathcal{S}$ is different from that of $\bar{\mathcal{A}}$ is $\mathcal{S}$ may outputs $failure$. Since Lemma 23 shows that this point arises with negligible probability, our claim holds.

Second, we claim that the outputs of $\mathcal{A}$ and $\bar{\mathcal{A}}$ are computationally indistinguishable. The only point makes $\bar{\mathcal{A}}$'s view different from $\mathcal{A}$'s view is that the ciphertext $\vec{c}'$ $\bar{\mathcal{A}}$ receives is generated by encrypting fabricated $\vec{m}'$ rather than party P_1's private input $\vec{m}$. This is the situation we meet in the proof of Lemma 14. Therefore, this proof can be done in a similar way with little modification.

Let $H' \stackrel{def}{=} \cap_{i\in\overline{CS}} H'_i$. The proof of the Lemma 14 shows H' is the effective private input of $\bar{\mathcal{A}}$ and $\mathcal{A}$. If $H'_e = H'$, then the proof of the second claim can be done in the same way as the proof of the Lemma 14 is done. If $H'_e \supset H'$, for each $j \in H'_e - H'$, there exists $\pi_i^2(\tilde{\vec{x}}_i)$ whose j-th entry is a smooth instance, which leads to both $\vec{c}\langle j\rangle$ $\mathcal{A}$ sees and $\vec{c}'\langle j\rangle$ $\bar{\mathcal{A}}$ sees are values distributed over appropriate domains. Therefore, $\mathcal{A}$'s view and $\bar{\mathcal{A}}$'s also are indistinguishable in this case.

Combining the above two claims, this proposition holds. □

Proposition 25. *In case P_2 was corrupted, i.e., $I = \{2\}$, the equation* (2) *required by Definition 2 holds.*

Proof: Note that the honest parties P_1 and P'_1 end up with outputting nothing. Thus, the fact that the outputs of $\mathcal{S}$ and $\mathcal{A}$ are computationally indistinguishable, which is supported by Proposition 24, directly prove this proposition holds. □

4.3 The Communication Rounds

It is easy to see that this framework costs four communication rounds. Since this framework is secure against malicious adversaries under one-sided simulation, we only compare it with protocols that are secure with this level, and protocols that are secure against malicious adversaries under full simulation. Compared with the protocols for OT_1^2 with security based on one-sided simulation, i.e. the protocols presented by [2], [28], [35], our framework is not efficient in communications, since these protocols only cost two rounds. However, we argue that our framework still makes sense. Since the known protocols for OT_h^n on this security level [33], [34], [36] are not constant-round and only work when h is far less that n, e.g., $n = 10^6$, $h \leq 6$. To our best knowledge, our framework is the first protocol for general OT_h^n on this security level.

Compared with the protocols that are secure against malicious adversaries under full simulation, the conclusions for the framework against covert adversaries still holds for this framework. Because the two framework cost the same number of communication rounds.

4.4 The Computational Overhead

In Step S2, P_1 takes $n \cdot \#\overline{CS}$ invocations of $Hash(.)$ to encrypt his private input. In Step R3, P_2 takes $h \cdot \#\overline{CS}$ invocations of $pHash(.)$ to decrypt the messages he want. The value of $\#\overline{CS}$ is K, $K/2$, respectively, in the worst case and in the average case. Thus, fixing the problem we tackle (i.e. fixing the values of n and h), the efficiency only depends on the value of K. Lemma 23 shows that the simulator may fails with probability at most $1/2^{K-1}$ in case P_2 is corrupted. Thus, conditioning on the cryptographic primitives without being broken, the real world and the ideal world can be distinguished at most $1/2^{K-2}$. Setting K to be 40, we obtains such a probability 3.6×10^{-12}, which is secure enough to be used in practice. As a result, in the worst case, the computational overhead mainly consists of $40n$ invocations of $Hash()$ taken by P_1 and $40h$ invocations of $pHash()$ taken by P_2; in the worst case, the computational overhead mainly consists of $20n$ invocations of $Hash()$ taken by P_1 and $20h$ invocations of $pHash()$ taken by P_2.

Compared with the protocols with the same security level, i.e., the protocols in [2], [28], [33]–[36], our framework is not efficient. However, as pointed in Section 4.3, our framework still makes sense.

Compared with the protocols for OT_h^n that are secure against malicious adversaries under full simulation, our DDH-based instantiation is the most efficient one. Following the analysis in Section 3.5, the operations of the protocol in [7], [24] are far more expensive than DDH-based operations. Though the DDH-based instantiation in [42] costs the same amount of pubic-key operations, it costs two additional private-key operations, i.e., two commitment operations.

5 Discussion On The Technique Cut-And-Choose

Looking at the application of the technique cut-and-choose in our two frameworks, it contradicts our intuition that the more the instance vectors chosen to be open, the lower the probability that P_2 succeeds in cheating. Theorem 5 and Theorem 19 show that the minimum probability can be much less than our intuitive value $1/K$. The essential reason is that our intuition is apt to neglect the $SPHDHC_{t,h}$'s property smoothness. This indeed inspire us that, to make the best of this technique, something seems beyond the technique self should be taken into account.

Comparing the two frameworks, the difference between them is the way of applying the technique cut-and-choose, i.e., the way P_1 chooses instance vectors to open. Though this difference seems trivial, its influence is not trivial at all. Specifically, for the framework against covert adversary, we cannot reduce the probability of P_2's successful cheating to be negligibly small. The reason is that, on one side, the possible minimum value of this probability always is a constant value; on one side, to the make sure the simulator runs in strictly polynomial time, this probability has to be noticeable. For the framework against malicious adversary, we indeed solve the mentioned problem. However, we can not provide a security proof under the ideal/real simulation paradigm in case P_1 is corrupted. Because a simulator extracting $\bar{\mathcal{A}}$'s real input by cheating, which is the idea used for the framework against covert adversary, needs expected 2^K times of rewinding $\bar{\mathcal{A}}$. This problem seems unsolvable if the instance vectors to be open are only chosen by P_1. Therefore, P_2's participating in choosing such instance vectors seems essential. In fact, [42] uses this idea to solve this problem. However, [42] also needs two additional tools, a perfectly hiding commitment scheme and a perfectly binding commitment schemes. The resulting framework costs two more communication round and two more commitment operations.

References

[1] M. Abdalla, C. Chevalier, and D. Pointcheval. Smooth projective hashing for conditionally extractable commitments. *Advances in Cryptology - Crypto 2009*, 5677:671–689 689, 2009. Blc86 Times Cited:0 Cited References Count:31 Lecture Notes in Computer Science.

[2] B. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In *Advances in Cryptology-Eurocrypt'2001*, pages 119–135. Springer, 2001.

[3] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology*, 23(2):281–343, 2010.

[4] B. Barak and Y. Lindell. Strict Polynomial-time in Simulation and Extraction. *SIAM Journal on Computing*, 33(4):783–818, 2004.

[5] D. Bernstein. Proving tight security for Rabin-Williams signatures. pages 70–87. Springer, 2008.

[6] D. Boneh, C. Gentry, and M. Hamburg. Space-efficient identity based encryptionwithout pairings. In *Foundations of Computer Science, 2007. FOCS'07. 48th Annual IEEE Symposium on*, pages 647–657. IEEE, 2007.

[7] J. Camenisch, G. Neven, and A. Shelat. Simulatable adaptive oblivious transfer. In *Advances in Cryptology-Eurocrypt'2007*, page 590. Springer-Verlag, 2007.

[8] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[9] R. Canetti and M. Fischlin. Universally composable commitments. In *Advances in CryptologyłCRYPTO 2001*, pages 19–40. Springer, 2001.

[10] R Canetti, O Goldreich, and S Halevi. The random oracle methodology, revisited. *Journal of the ACM (JACM)*, 51(4):557–594, 2004.

[11] R. Canetti, E. Kushilevitz, and Y. Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *Journal of Cryptology*, 19(2):135–167, 2006.

[12] R. Cramer, I. Damgård, and P. MacKenzie. Efficient zero-knowledge proofs of knowledge without intractability assumptions. In *Public Key Cryptography*, pages 354–373. Springer, 2000.

[13] R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In L. Knudsen, editor, *Advances in Cryptology - Eurocrypt'2002*, pages 45–64, Amsterdam, NETHERLANDS, 2002. Springer-Verlag Berlin.

[14] C. Crépeau. Equivalence Between Two Flavours of Oblivious Transfers. In *Advances in Cryptology-Crypto'87*, page 354. Springer-Verlag, 1987.

[15] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):647, 1985.

[16] M.J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. *Theory of Cryptography*, pages 303–324, 2005.

[17] S.D. Galbraith, K.G. Paterson, and N.P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[18] J.A. Garay, D. Wichs, and H.S. Zhou. Somewhat non-committing encryption and efficient adaptively secure oblivious transfer. In *Advances in Cryptology-Crypto'2009*, page 523. Springer, 2009.

[19] R. Gennaro and Y. Lindell. A framework for password-based authenticated key exchange. *ACM Transactions on Information and System Security (TISSEC)*, 9(2):234, 2006.

[20] O. Goldreich. *Foundations of cryptography,volume 1*. Cambridge university press, 2001.

[21] O. Goldreich. *Foundations of cryptography, volume 2*. Cambridge university press, 2004.

[22] O. Goldreich. On expected probabilistic polynomial-time adversaries: A suggestion for restricted definitions and their benefits. *Journal of Cryptology*, 23(1):1–36, 2010.

[23] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.

[24] M. Green and S. Hohenberger. Blind identity-based encryption and simulatable oblivious transfer. In K. Kurosawa, editor, *Advances in Cryptology-Asiacrypt'2007*, pages 265–282, Kuching, MALAYSIA, 2007. Springer-Verlag Berlin.

[25] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. *Journal of Cryptology*, 23(3):422–456, 2010.

[26] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology-Crypto'03*, pages 145–161. Springer.

[27] Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In D. Wagner, editor, *Advances in Cryptology-Crypto'2008*, pages 572–591, Santa Barbara, CA, 2008. Springer-Verlag Berlin.

[28] Yael Tauman Kalai. Smooth projective hashing and two-message oblivious transfer. In *Advances in Cryptology C EUROCRYPT 2005*, volume 3494, pages 78–95. Springer, 2005. .

[29] J Kilian. Founding crytpography on oblivious transfer. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 20–31, Inc, One Astor Plaza, 1515 Broadway, New York, NY,10036-5701, USA, 1988. ACM New York, NY, USA. STOC.

[30] Gatan Leurent and Phong Nguyen. How risky is the random-oracle model? In Shai Halevi, editor, *Advances in Cryptology - Crypto 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 445–464. Springer Berlin / Heidelberg, 2009.

[31] A.Y. Lindell. Efficient Fully-Simulatable Oblivious Transfer. In *Topics in cryptology: CT-RSA 2008: the cryptographers' track at the RSA conference 2008, San Francisco, CA, USA, April 8-11, 2008: proceedings*, page 52. Springer-Verlag New York Inc, 2008.

[32] Chi-Jen Lu. On the security loss in cryptographic reductions. In *Advances in Cryptology-Eurocrypt'2009*, volume 5479, pages 72–87. Springer, 2009.

[33] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 245–254. ACM New York, NY, USA, 1999.

[34] M. Naor and B. Pinkas. Oblivious transfer with adaptive queries. In *Advances in Cryptology-Crypto'99*, pages 573–590. Springer, 1999.

[35] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, page 457. Society for Industrial and Applied Mathematics, 2001.

[36] M. Naor and B. Pinkas. Computationally secure oblivious transfer. *Journal of Cryptology*, 18(1):1–35, 2005.

[37] W. Ogata and K. Kurosawa. Oblivious keyword search. *Journal of Complexity*, 20(2-3):356–371, 2004.

[38] C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In D. Wagner, editor, *Advances in Cryptology-CRYPTO'2008*, pages 554–571, Santa Barbara, CA, 2008. Springer-Verlag Berlin.

[39] M. Rabin. How to exchange secrets by oblivious transfer. Technical report, Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981, 1981.

[40] CP Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[41] A.C.C. Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1985., 27th Annual Symposium on*, pages 162–167, 1986.

[42] B. Zeng, X. Tang, and C. Hsu. A framework for fully-simulatable h-out-of-n oblivious transfer. Cryptology ePrint Archive, Report 2010/199, 2010. http://eprint.iacr.org/.