A Framework For Fully-Simulatable h-Out-Of-n Oblivious Transfer

Zeng Bing, Tang Xueming, and Chingfang Hsu

Abstract—We present a framework for fully-simulatable h-out-of-n oblivious transfer (OT_h^n) with security against non-adaptive malicious adversaries. The framework costs six communication rounds and costs at most $40n$ public-key operations in computational overhead. Compared with the known protocols for fully-simulatable oblivious transfer that works in the plain mode (where there is no trusted common reference string available) and proven to be secure under standard model (where there is no random oracle available), the instantiation based on the decisional Diffie-Hellman assumption of the framework is the most efficient one, no matter seen from communication rounds or computational overhead.

Our framework uses three abstract tools, i.e., information-theoretically binding commitment, information-theoretically hiding commitment and our new smooth projective hash. This allows a simple and intuitive understanding of its security.

We instantiate the new smooth projective hash under the lattice assumption, the decisional Diffie-Hellman assumption, the decisional N-th residuosity assumption, the decisional quadratic residuosity assumption. This indeed shows that the folklore that it is technically difficult to instantiate the projective hash framework under the lattice assumption is not true. What's more, by using this lattice-based hash and lattice-based commitment scheme, we gain a concrete protocol for OT_h^n which is secure against quantum algorithms.

Index Terms—oblivious transfer (OT) protocols.

1 Introduction

1.1 Oblivious transfer

Oblivious transfer (OT), first introduced by [52] and later defined in another way with equivalent effect [16] by [18], 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, [28], [32], [35], [59] 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.

Considering a variety of attack we have to confront in real environment, a protocol for OT_h^n with security against malicious adversaries (a malicious adversary may act in any arbitrary malicious way to learn as much extra information as possible) is more desirable than the one 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 he). Using Goldreich's compiler [25], [28], we can gain the former version from the corresponding latter version. However, the resulting protocol is prohibitive expensive for practical use, because it is embedded with so many invocations of zero-knowledge for NP. Thus, directly constructing the protocol based on specific intractability assumptions seems more feasible.

The first step in this direction is independently made by [46] and [1] which respectively presents a two-round efficient protocol for OT_1^2 based on the decisional Diffie-Hellman (DDH) assumption. Starting from these works and using the tool smooth projective hashing, [33] abstracts and generalizes the ideas of [1], [46] to a framework for OT_1^2. Besides DDH assumption, the framework can be instantiated under the decisional N-th residuosity (DNR) assumption and decisional quadratic residuosity (DQR) assumption [33].

Unfortunately, these protocols (or frameworks) are only half-simulatable not fully-simulatable. By saying a protocol is fully-simulatable, we means that the protocol can be strictly proven its security under the real/ideal model simulation paradigm. The paradigm requires that for any adversary in the real world, there exists a corresponding adversary simulating him in the ideal world. Thus, the real adversary can not do more harm than the corresponding ideal adversary does. Therefore the security level of the protocol is guaranteed not to be lower than that of the ideal world. Undesirably, a half-simulatable protocol for OT_1^2 only provides a simulator in the case the receiver is corrupted such as [1], [46] or in the case the sender is corrupted such as [33].

The authors are with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan City, Hubei 430074 China (e-mail:zeng.bing.zb@gmail.com, tang.xueming.txm@gmail.com, cherryjingfang@gmail.com).

Considering security, requiring a protocol to be fully-simulatable is necessary. Specifically, a fully-simulatable protocol provides security against all kinds of attacks, especially the future unknown attacks taken by any adversary whose computational resource is fixed when constructing the protocol (generally, it is assumed that the adversaries run arbitrary probabilistic polynomial-time) [7], [25], while a not fully-simulatable protocol doesn't. For example, the protocols proposed by [1], [33], [46] suffer the selective-failure attacks, in which a malicious sender can induce transfer failures that are dependent on the messages that the receiver requests [47].

Constructing fully-simulatable protocols for OT with security against malicious adversaries naturally becomes the focus of the research community. [6] first presents such a fully-simulatable protocol. In detail, the OT is an adaptive h-out-n oblivious transfer (denoted by $OT^n_{h\times 1}$ in related literature) and based on q-Power Decisional Diffie-Hellman and q-Strong Diffie-Hellman assumptions. Unfortunately, these two assumptions are not standard assumptions used in cryptography and seem significantly stronger than DDH, DQR and so on. Motivated by basing OT on weaker complexity assumption, [29] presents a protocol for OT^n_h using a blind identity-based encryption which is based on decisional bilinear Diffie-Hellman (DBDH) assumption. Using cut-choose technique, [37] later presents two efficient protocols for fully-simulatable OT^2_1 respectively based on DDH assumption and DNR assumption, where the DDH-based protocol is the most efficient one among these fully-simulatable works.

The protocols mentioned above are proved their securities in the plain stand-alone model which not necessarily allows concurrent composition with other arbitrary malicious protocols. [51] overcomes this weakness and further the research by presenting a framework under common reference string (CRS) model for fully-simulatable, universally composable OT^2_1 and instantiating the framework respectively under DDH, DQR and worst-case lattice assumption. It is notable that conditioning on a trusted CRS is available, the DDH-based instantiation of the framework is the most efficient protocol for OT^2_1 no matter seen from the number of communication rounds or the computational overhead. Recently, [21], using a novel compiler and somewhat non-committing encryption they present, convert [51]'s instantiations based on DDH, DQR to the corresponding protocols with higher security level. In more detail, the resulting protocols for OT^2_1 are secure against adaptive malicious adversaries, which corrupts the parties dynamically based on his knowledge gathered so far. Note that, the fully-simulatable protocols for OT^2_1 mentioned so far except the one presented by [6] are only secure against non-adaptive malicious adversaries, which only corrupts the parties preset before the running of the protocol.

Though constructing protocols for fully-simulatable OT^2_1 with security against malicious adversaries has been studied well, constructing protocols for such OT^n_h hasn't. We note that there are some works aiming to extend known cryptographic protocols to OT^n_h. [44] shows how to implementation OT^n_h using $\log n$ invocation of OT^2_1 under half-simulation. A similar implementation for adaptive OT^n_h can be seen in [45]. What's more, the same authors of [44], [45] propose a way to transform a singe-server private-information retrieval scheme (PIR) into an oblivious transfer scheme under half-simulation too [47]. With the help of a random oracle, [31] shows how to extend k oblivious transfers (for some security parameter k) into many more, without much additional effort. However, the Random Oracle Model is risky. First, [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". Second, [10] shows efficient implementing the random oracle is impossible. Later, [36] 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, [36] shows that how the hash function defects deadly damages the securities of the cryptographic schemes presented in [4], [5]. Therefore, in this paper, we only consider the schemes which are fully-simulatable and without turning to a random oracle. To our best knowledge, only [6] and [29] respectively present such fully-simulatable protocols for OT^n_h. However, the assumptions the former uses are not standard and the latter uses is too expensive. Therefore, a well-motivated problem is to find a protocol or framework for efficient, fully-simulatable, secure against malicious adversaries OT^n_h under weaker complexity assumptions.

1.2 Our Contribution

In this paper, we present a framework for efficient, fully-simulatable, secure against non-adaptive malicious adversaries OT^n_h whose security is proven under stand model (i.e., without turning to a random oracle). To our best knowledge, this is the first framework for such OT^n_h. The framework have the following features,

1) Fully-simulatable and secure against malicious adversaries without using a CRS. [33]'s framework for OT^2_1 is half-simulatable. Thought [51]'s framework for OT^2_1 is fully-simulatable, it doesn't work without a CRS. What is more, 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 [51], are only conjectures without formal proofs. The same problem remains in its adaptive version presented by [21]. 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 framework are better.

2) Efficient. Compared with the existing protocols for fully-simulatable OT that without resorting to a CRS or a random oracle, i.e., the protocols presented by [6], [29], [37], the DDH-based instantiation of our framework costs the minimum number of communication rounds and costs the minimum computational overhead. Please see Section 4.4 and Section 4.5 for the detailed comparisons.
   We admit that, in the context of a trusted CRS is available and only OT_1^2 is needed, the DDH-based instantiation of [51] is the most efficient one.
3) Abstract and modular. The framework is described using just three high-level cryptographic tools, i.e., information-theoretically binding commitment (IBC), information-theoretically hiding commitment (IHC) and our new smooth projective hash (denoted by $SPHDHC_{t,h}$ for simplicity). This allows a simple and intuitive understanding of its security.
4) Generally realizable. The high-level cryptographic tools PBC, PHC and $SPHDHC_{t,h}$ are realizable from a variety of known specific assumptions, even future assumptions maybe. This makes our framework generally realizable. In particular, we instantiate $SPHDHC_{t,h}$ from the DDH assumption, the DNR assumption, the DQR assumption and the lattice assumption. Instantiating PBC or PHC under specific assumptions is beyond the scope of this paper. Please see [24], [27] for such examples. Generally realizability is vital to make the framework live long, considering the future progress in breaking a specific intractable problem. If this case happen, replacing the instantiation based on the broken problem with that based on a unbroken problem suffices.

What is more, we fix a folklore [51] that it appears technically difficult to instantiate the projective hash under lattice assumption by presenting a lattice-based $SPHDHC_{t,h}$ instantiation. It is notable that we gain an OT_h^n instantiation which is secure against quantum algorithms, using this lattice-based $SPHDHC_{t,h}$ instantiation and appropriate lattice-based commitment schemes. Considering that factoring integers and finding discrete logarithms are efficiently feasible for quantum algorithms [55]–[57], this is an example showing the benefits from the generally realizability of the framework.

As an independent contribution, we present several propositions/lemmas related to the indistinguishability of probability ensembles defined by sampling polynomial instances. Such propositions/lemmas simplify our security proof very much. We believe that they are as useful in security proof somewhere else as in this paper.

1.3 Our Approach

We note that the smooth projective hash is a good abstract tool. Using this tool, [33] in fact presents a framework for half-simulatable OT_1^2, [22] present a framework for password-based authenticated key exchange protocols. We also note that the cut-and-choose is a good technique to make protocol fully-simulatable. Using this tool, [37] present several fully-simulatable protocol for OT_1^2, [38] presents a general fully-simulatable protocol for two-party computation. Indeed, we are inspired by such works. Our basic ideal is to use cut-and-choose technique and smooth projective hash to get a fully-simulatable framework.

Loosely speaking, a smooth projective hash (SPH) is a set of operations defined over two languages $\dot{L}$ and $\ddot{L}$, where $\dot{L} \cap \ddot{L} = \emptyset$. For any projective instance $\dot{x} \in \dot{L}$, there are two ways to obtain its hash value, i.e., the way using its hash key or the way using its projective key and its witness $\dot{w}$. For any smooth instance $\ddot{x} \in \ddot{L}$, there is only one way to obtain its hash value, i.e., the way using its hash key. The version of SPH presented by [33] (denoted by $VSPHH$ for simplicity) holds a property called verifiable smoothness that can judge whether at least one of arbitrary two instances is smooth. Another property $VSPHH$ holds, called hard subset membership, makes sure $\ddot{x}$ and $\dot{x}$ are computationally indistinguishable.

We observe that the $VSPHH$ indeed is easy to be extended to deal with OT_1^n, but seems difficult to be extended to deal with the general OT_h^n. The reason is that, to hold verifiable smoothness, $\dot{x}$s and $\ddot{x}$s have to be generated in a dependent way. This makes the verifiable smoothness for multiple $\dot{x}$s and multiple $\ddot{x}$s (i.e., judge whether at least $n-h$ of arbitrary n instances are smooth) difficult to hold without leaking information which is conductive to distinguish such $\dot{x}$s and $\ddot{x}$s. We also observe that, there is no way to construct a fully-simulatable framework using $VSPHH$, because there is no way to extract the real input of the adversary in the case that the receiver is corrupted.

We define a new smooth projective hash called t-smooth h-projective hash family that holds properties distinguishability, hard subset membership, feasible cheating (denoted by $SPHDHC_{t,h}$ for simplicity). The key solution in $SPHDHC_{t,h}$ to the mentioned problems is that requiring each $\ddot{x}$ to hold a witness too. This solution enables us to generate $\dot{x}$s and $\ddot{x}$s in a independent way. Correspondingly, the verifiable smoothness is not needed any more and replaced by a property called distinguishability, which provides a way to distinguish $\dot{x}$s and $\ddot{x}$s if their witnesses are given.

Since the receiver encodes his input as a permutation of $\dot{x}$s and $\ddot{x}$s, a simulator can the extract the real input of the adversary in the case that the receiver is corrupted if their witnesses are available. Combining the application of the technique cut-and-choose, a simulator can see such witnesses by rewinding the adversary. To extract the real input of the adversary in the case that the sender is corrupted, the property feasible cheating provides way to cheat out of the real input of the adversary. Naturally, all the properties and the correlated algorithm in $SPHDHC_{t,h}$ are extended to deal with n instances rather than only 2 instances. Please see Section 3.2 for a detailed comparison this new hash with previous hash systems.

We show that constructing $SPHDHC_{t,h}$ can be reduced to constructing considerably simpler hash systems. Our lattice-based $SPHDHC_{t,h}$ instantiation is builded on the lattice-based cryptosystem presented by [51]. It is noticeable that it appears difficult to get lattice-based instantiation for SPH [51]. Our solution is to let the instance x ($x \in \dot{L} \cup \ddot{L}$) be available to the algorithm that is responsible for generating pair of the hash key and the projective key. The other three intractability-assumption-based $SPHDHC_{t,h}$ instantiations can be ultimately built from known SPH schemes such as that presented by [33] with necessary modifications.

Using $SPHDHC_{t,h}$ we construct the framework described with high-level as follows .

1) The receiver generates hash parameters and appropriate many instance vectors, then sends them to the sender after disordering each vector.
2) The receiver and the sender cooperate to toss coin to decide which vector to be opened.
3) The receiver opens the chosen instances, encodes his private input by reordering each unchosen vector and sends the resulting code, which in fact is a sequence of permutations, to the sender.
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 values of the instances of the unchosen vectors in the way indicated by the code of receiver's private input, and sends the ciphertexts together with some auxiliary information (i.e., the projective hash keys) that is conductive to decrypt some ciphertexts to the receiver.
5) The receiver decrypts the ciphertexts with the help of the auxiliary information 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 extra new knowledge is negligible.

1.4 Organization

In Section 2, we describe the notations used in this paper, the security definition of OT^n_h, the definition of commitment scheme. In Section 3, we define our new hash system, i.e., $SPHDHC_{t,h}$. In Section 4, we construct our framework. In Section 5, we prove the security of the framework. In Section 6, we reduce constructing $SPHDHC_{t,h}$ to constructing considerably simpler hash systems. In Section 7, we instantiate the simpler hash systems under the DDH, lattice, DNR, DQR assumptions, respectively.

2 PRELIMINARIES

Most notations and concepts mentioned in this section originate from [7], [24], [25] which are basic literature in the filed of secure multi-party computation (SMPC). We tailor them to the need of dealing with OT^n_h.

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 $i \in \mathbb{N}$, $[i] \overset{def}{=} \{1, 2, \ldots, i\}$. We denote the set consists of all prime numbers by $\mathbb{P}$.

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 $X \overset{def}{=} \{X(1^k, a)\}_{k\in\mathbb{N}, a\in\{0,1\}^*}$ 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 $X \overset{def}{=} \{X(1^k, a)\}_{k\in\mathbb{N}, a\in\{0,1\}^*}$ and $Y \overset{def}{=} \{Y(1^k, a)\}_{k\in\mathbb{N}, a\in\{0,1\}^*}$ be two probability ensembles. They are computationally indistinguishable, denoted $X \overset{c}{=} Y$, 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 $X \overset{c}{=} Y$.

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 $\vec{x}$, $\vec{x}\langle I\rangle \overset{def}{=} (\vec{x}\langle i\rangle)_{i\in I, i\leq \#\vec{x}}$.

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 Definition Of A Protocol For OT_h^n

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

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

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 the 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 [25] 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, ideal world, real world were introduced. Note that the security target in this paper is to be secure against non-adaptive malicious adversaries, so only concepts related to this case is referred to in the following.

2.2.2 Non-Adaptive Malicious Adversary

Before running OT_h^n, the adversary A has to corrupt all parties listed in $I \subseteq [2]$. In the case that $U \in \{P_1, P_2\}$ is not corrupted, U will strictly follow the prescribed protocol as an honest party. In the case that party U is corrupted, U will be fully controlled by A as a corrupted party. In this case, U will have to pass all his knowledge to A before the protocol runs and follows A's instructions from then on — so there is a probability that U arbitrarily deviates from prescribed protocol. In fact, after A finishes corrupting, A and all corrupted parties have formed a coalition led by 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 corrupted parties are dummy. A receives messages addressed to the members of the coalition and sends messages on behalf of the members.

Loosely speaking, we say OT_h^n is secure, if and only if, for any malicious adversary A, the knowledge 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 malicious adversary A, what harm A can do in real world is not more than what harm he can do 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. In the real world, there is no TTP, and the computation of $f(.)$ is finished by A and all parties' interactions.

2.2.3 OT_h^n In The Ideal World

In the ideal world, an execution of OT_h^n proceeds as follows.

Initial Inputs. All entities know the public security parameter k. P_1 holds a private input $\vec{m} \in (\{0,1\}^*)^n$. Party P_2 holds a private input $H \in \Psi$. Adversary A holds a name list $I \subseteq [2]$, a randomness $r_A \in \{0,1\}^*$ and an infinite auxiliary input sequence $z = (z_k)_{k \in \mathbb{N}}$, where $z_k \in \{0,1\}^*$. Before proceeds to next stage, A corrupts parties listed in I and learns $\vec{x}\langle I \rangle$, where $\vec{x} \stackrel{def}{=} (\vec{m}, H)$.

Submitting inputs to TTP. Each honest party P_i always submits its private input $\vec{x}\langle i \rangle$ unchanged to TTP. A submits arbitrary string based on his knowledge to TTP for the corrupted parties. The string TTP receives is a two-dimensional vector $\vec{y}$ which is formally described as follows.

$$\vec{y}\langle i \rangle = \begin{cases} \vec{x}\langle i \rangle & \text{if } i \notin I \ , \\ \alpha & \text{if } i \in I \end{cases}$$

where $\alpha \in \{\vec{x}\langle i \rangle\} \cup \{0,1\}^{|\vec{x}\langle i \rangle|} \cup \{abort_i\}$ and $\alpha \leftarrow A(1^k, I, r_A, z_k, \vec{x}\langle I \rangle)$. Obviously, there is a probability that $\vec{x} \neq \vec{y}$.

TTP computing f. TTP checks that $\vec{y}$ is a valid input to f, i.e., no entry of $\vec{y}$ is of the form $abort_i$. If $\vec{y}$ passes the check, then TTP computes f and sets $\vec{w}$ to be $f(1^k, \vec{y})$. Otherwise, TTP sets $\vec{w}$ to be $(abort_i, abort_i)$. Finally, for each $i \in [n]$ TTP hands $\vec{w}\langle i \rangle$ to each P_i respectively and halts.

Outputs. Each honest party P_i always outputs the message $\vec{w}\langle i \rangle$ it obtains from the TTP. Each corrupted party P_i outputs nothing (i.e., λ). The adversary outputs something generated by executing arbitrary function of the information he gathers so far. Without loss of generality, this can be assumed to be $(1^k, I, r_A, z_k, \vec{x}\langle I \rangle, \vec{w}\langle I \rangle)$.

The output of the whole execution in the ideal world, denoted by $Ideal_{f,I,A(z_k)}(1^k, \vec{m}, H)$, is defined by the outputs of all parties and that of the adversary as follows.

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

Obviously, $Ideal_{f,A(z),I}(1^k,\vec{x})$ is a random variable whose randomness is r_A.

2.2.4 OT_h^n In The Real World

In the real world, there is no TTP. A execution of OT_h^n proceeds as follows.

Initial Inputs. Initial input each entity holds in the real world is the same as in the ideal world but there are some difference as follows. A randomness r_i is held by each party P_i. After finishes corrupting, in addition to the knowledge A learns in ideal world, the corrupted parties' randomness $\vec{r}\langle I\rangle$ is also learn by A, where $\vec{r} \overset{def}{=} (r_1, r_2)$.

Computing f. In the real world, computing f is finished by all entities' interaction. Each honest party strictly follows the prescribed protocol (i.e., the concrete protocol, usually denoted π , for OT_h^n). The corrupted parties have to follow A's instructions and may arbitrarily deviate from prescribed protocol.

Outputs. Each honest party P_i always outputs what the prescribed protocol instructs. Each corrupted party P_i outputs nothing. The adversary outputs something generated by executing arbitrary function of the information he gathers so far. Without loss of generality, this can be assumed to be a string consisting of $1^k, I, r_A, \vec{r}\langle I\rangle, z_k, \vec{x}\langle I\rangle$ and messages addressed to the corrupted parties.

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

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

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

2.2.5 Security Definition

The security of a protocol for OT_h^n is formally captured by the following definition.

Definition 1 (The security of a protocol for OT_h^n). *Let f denotes the functionality of OT_h^n and let π be a concrete protocol for OT_h^n. We say π securely computes f, if and only if for any non-uniform probabilistic polynomial-time adversary A with an infinite sequence $z=(z_k)_{k\in\mathbb{N}}$ in the real world, there exists a non-uniform probabilistic expected polynomial-time adversary A' with the same sequence in the ideal world such that, for any $I \subseteq [2]$, it holds that*

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

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

The concept, non-uniform probabilistic expected polynomial-time, mentioned in Definition 1 is formulated in distinct way in distinct literature such as [8], [24]. We prefer to the following definition [34], because it is clearer in formulation and more closely related to our issue.

Definition 2 (M_1 runs in expected polynomial–time with respect to M_2). *Let M_1, M_2 be two interactive Turing machines running a protocol. By $< M_1(x_1,r_1,z_1), M_2(x_2,r_2,z_2) > (1^k)$, we denote a running which starts with M_i holding a private input x_i, a randomness r_i, an auxiliary input z_i, the public security parameter k. By $IDN_{M_1}(< M_1(x_1,r_1,z_1), M_2(x_2,r_2,z_2) > (1^k))$, we denote the number of total direct deduction steps M_1 takes in the whole running. We say M_1 runs in expected polynomial-time with respect to M_2, if and only if there exists a polynomial $poly(.)$ such that for every $k \in \mathbb{N}$, it holds that*

$$\max(\{E_{R_1,R_2}(IDN_{M_1}(< M_1(x_1,R_1,z_1), M_2(x_2,R_2,z_2) > (1^k))) \mid |x_1|=|x_2|=k, z_1,z_2 \in \{0,1\}^*\}) \le poly(k)$$

where R_1, R_2 are random variables with uniform distribution over $\{0,1\}^*$.

For Definition 1, it in fact requires that adversary A's simulator A' should run in expected polynomial-time with respect to TTP who computes OT_h^n's functionality f.

We point out that the security definition presented in [7], [24], [25] requires the simulator A' to run in strictly polynomial-time, but the one presented in [8], [37], [38] allow A' to run in expected polynomial-time. Definition 1 follows the latter. We argue that this is justified, since [3] shows that there is no (non-trivial) constant-round zero-knowledge proof or argument having a strictly polynomial-time black-box simulator, which means allowing simulator to run in expected polynomial-time is essential for achieving constant-round protocols. See [34] for further discussion.

2.3 Commitment Scheme

In this section, we briefly introduce the cryptographic tool commitment scheme which will be used in our framework. For the strict definition and the details, please see [24] or [27].

Definition 3 (commitment scheme, non-strict description, [24], [27]). *A commitment scheme is a two-party protocol involving two phases.*

- *Initial Inputs. At the beginning, all parties know the public security parameter k. The unbounded sender P_1 holds a randomness $r_1 \in \{0,1\}^*$, a value $m \in \{0,1\}^{poly(k)}$ to be committed to, where the polynomial $poly(.)$ is public. The PPT receiver P_2 holds a randomness $r_2 \in \{0,1\}^*$.*
- *Commit Phase. P_1 computes a commitment, denoted α, based on his knowledge, i.e., $\alpha \leftarrow P_1(1^k, m, r_1)$, then P_1 send α to P_2.*
  - *The security for P_1 is implied by the property computationally hiding, which prevents P_2 from knowledge of the value committed by P_1. That is, for any PPT P_2, any $m_1, m_2 \in \{0,1\}^{poly(k)}$, it holds that*

$$\{ViewC_{P_2}(< P_1(m), P_2 > (1^k))\}_{k\in\mathbb{N}} \stackrel{c}{=} \{ViewC_{P_2}(< P_1(m'), P_2 > (1^k))\}_{k\in\mathbb{N}},$$

    where $ViewC_{P_2}(.)$ denotes P_2's view in commit phase.
- *Reveal Phase. P_1 computes and sends a de-commitment, which typically consists of m, r_1, to P_2 to let P_2 know m. Receiving de-commitment, P_2 checks its validity. Typically P_2 checks that $\alpha = P_1(1^k, m, r_1)$ holds. If de-commitment pass the check, P_2 knows and accepts m.*
  - *The security for P_2 is implied by the property information-theoretically binding, which guarantees that for any unbounded P_1, any $m_1, m_2 \in \{0,1\}^{poly(k)}$ such that $m_1 \neq m_2$, the probability that P_2 accepts m_2 while P_1 commits to m_1 is negligible, where the probability is taken only over the randomness used by P_2.*

The above definition defines perfectly binding commitment schemes (denoted by PBC). Relaxing the property binding to allow the probability of successful cheat of unbounded P_1 to be negligible, then the above definition defines statically binding commitment schemes. Correspondingly, in the setting that P_1 is PPT and P_2 is unbounded, there exists perfectly hiding commitment schemes (denoted by PHC) and statically hiding commitment schemes, which provide perfectly hiding and statically hiding to P_1 respectively, and only computationally binding to P_2. If a property is secure against unbounded adversaries, we say this property is information-theoretically secure. We remark that there is no commitment scheme holding both information-theoretically binding and information-theoretically hiding.

3 A New Smooth Projective Hash - $SPHDHC_{t,h}$

3.1 The Definition Of $SPHDHC_{t,h}$

In this section, we define a new smooth projective hash — t-smooth h-projective hash family that holds properties distinguishability, hard subset membership, feasible cheating, denoted $SPHDHC_{t,h}$ for simplicity, which will be used to construct our framework for OT^n_h. In section 7, we instantiate $SPHDHC_{t,h}$ respectively under four distinct intractability assumptions.

Let us recall some related works before defining $SPHDHC_{t,h}$. [12], [58] present the classic notation of "universal hashing". Based on "universal hashing", [15] first introduces the concept of universal projective hashing, smooth projective hashing and hard subset membership problem in terms of languages and sets. In order to construct a framework for password-based authenticated key exchange, [22] modifies such definition to some extent. That is, smoothness is defined over every instance of a language rather than a randomly chosen instance. [33] refines the modified version in terms of the procedures used to implement it. What is more, a new requirement called verifiable smoothness is added to the hashing so as to construct a framework for OT^2_1. The resulting hashing is called verifiablely-smooth projective hash family that has hard subset membership property (denoted by $VSPHH$ for simplicity). Note that, the framework presented by [33] is not fully-simulatable. The difference between $SPHDHC_{t,h}$ and the works mentioned above will be under a detailed discussion after we define $SPHDHC_{t,h}$.

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 4 (t-smooth h-projective hash family that holds properties distinguishability, hard subset membership and feasible cheating). *$\mathcal{H} = (PG, IS, DI, KG, Hash, pHash, Cheat)$ is an t-smooth h-projective hash family that holds properties distinguishability, hard subset membership and feasible cheating ($SPHDHC_{t,h}$), 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 $L_{\ddot{R}_\Lambda}$. 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 make 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., $y \leftarrow pHash(1^k, \Lambda, x, pk, w)$.
- 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
$$Hash(1^k, \Lambda, \dot{x}, hk) = pHash(1^k, \Lambda, \dot{x}, pk, \dot{w})$$
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 for each $j \in [n] - [h]$, $y_j \in_U Range(Hash(1^k, \Lambda, \vec{x}\langle j \rangle, .))$.
   $Sm_i(1^k)$: $(\Lambda, \overrightarrow{xpky}) \leftarrow SmGen_i(1^k)$, $\widetilde{\overrightarrow{xpky}} \leftarrow \pi(\overrightarrow{xpky})$, finally outputs $(\Lambda, \widetilde{\overrightarrow{xpky}})$.
3) Distinguishability. Intuitively speaking, it requires that the DI can distinguish the projective instances and smooth instances with the 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 $(\Lambda, \pi'(\vec{x}^{\vec{a}}))$.

Remark 5 (The Witnesses Of The Instances)**.** *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 x. In contrast, with respect to an instance $\ddot{x} \in L_{\ddot{R}_\Lambda}$, it services as a proof of $\ddot{x} \in L_{\ddot{R}_\Lambda}$. The property distinguishability guarantees that given the needed witness, the projective instances and the smooth instances are distinguishable. For OT^n_h, 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 $\ddot{x}$.*

Remark 6 (Hard Subset Membership)**.** *The property hard subset membership guarantees that for any $\vec{x} \in Range(IS(1^k, \Lambda))$, any $\pi \in \Pi$, any PPT adversary A, the advantage of 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 A, every entry of $\pi(\vec{x})$ seems the same.*

With respect to OT^n_h, this means that the receiver can encode his private input into a permutation of a vector

$\vec{x} \in L^n_{R_\Lambda}$ without leaking any information. For example, if the receiver expects to gain $\vec{m}\langle H\rangle$, then he may generates a $\vec{x}$ and randomly chooses a permutation $\pi \in \Pi$ such that $\pi(\vec{x})\langle i\rangle \in L_{\dot{R}_\Lambda}$ for each $i \in H$. Any PPT adversary knows no new knowledge about H if only given $\pi(\vec{x})$.

However, if the witnesses of the instances of $\vec{x}$ are available (the simulator can gain the witnesses by rewinding the adversary), then the receiver's input is known. Therefore, there is way for the simulator to extract the real input of the adversary controlling the corrupted receiver.

Remark 7 (Feasible Cheating)**.** *In our framework for OT^n_h, the sender uses the hash values of the instances generated by the receiver to encrypt its private inputs. The property feasible cheating makes cheating out of the sender's all private inputs feasible. Note that, this is a key for the simulator to extract the real inputs of the adversary controlling the corrupted sender. Therefore, it is conductive to construct a fully-simulatable protocol for OT^n_h.*

3.2 The Difference Between $SPHDHC_{t,h}$ And Related Hash Systems

Now we discuss the difference between our $SPHDHC_{t,h}$ and related hash systems previous works present or use. For simplicity, we only compare our $SPHDHC_{t,h}$ with the hash system $VSPHH$ which is presented by [33]. We argue that this is justified, on the one hand, the version of [33] is the version holding most properties among previous works. On the other hand, the aim of [33] is the closest to ours. They aim to construct a framework for OT^2_1 which actually is half-simulatable, while we aim to establish a fully-simulatable framework for OT^n_h.

Loosely speaking, our $SPHDHC_{t,h}$ can be viewed as a generalized version of $VSPHH$. Indeed, $VSPHH$ resembles $SPHDHC_{1,1}$ very much and can be converted into $SPHDHC_{1,1}$ though some modification is needed. The essential differences are listed as follows.

1) The key difference is that, besides each projective instance $\dot{x}$ holding a witness $\dot{w}$, $SPHDHC_{t,h}$ also requires each smooth instance $\ddot{x}$ to hold a witness $\ddot{w}$.
2) To deal with OT^n_h, $SPHDHC_{t,h}$ extends the IS algorithm to generate h $\dot{x}$s and t $\ddot{x}$s in a invocation. As a natural result, $SPHDHC_{t,h}$ extends the property smoothness to hold with respect to t $\ddot{x}$s, and extends the property hard subset membership to hold with respect to h $\dot{x}$s and t $\ddot{x}$s.
3) In $VSPHH$ there exists a instance test IT algorithm that takes two instances as input and outputs a bit indicating whether at least one of the two instances is smooth, i.e., $b \leftarrow IT(x_1, x_2)$. $SPHDHC_{t,h}$ discards this verifiability of smoothness and the correlated IT, and instead provides a distinguisher DI algorithm which is conducive to apply the technique cut-and-choose.
4) $SPHDHC_{t,h}$ requires a additional property feasible cheating and the necessary algorithm $Cheat$. This property provides a simulator with a way to extract the real inputs of the adversary in the case that the sender is corrupted.
5) $SPHDHC_{t,h}$ extends KG algorithm such that the information of the instance is available to it. This makes constructing hash system easier. In indeed, this makes lattice-based hash system come true which is thought difficult by [51].

We observe that the $VSPHH$ indeed is easy to be extended to deal with OT^n_1, but seems difficult to be extended to deal with the general OT^n_h. The reason is that, to hold verifiable smoothness, $\dot{x}$s and $\ddot{x}$s have to be generated in a dependent way. This makes designing IT dealing with n instances without leaking information which is conductive to distinguish such $\dot{x}$s and $\ddot{x}$s difficult. Therefore, even constructing a framework for OT^n_h that is half-simulatable as [33] seems impossible. We also observe that, there is no way to construct a fully-simulatable framework using $VSPHH$, because there is no way to extract the real input of the adversary in the case that the receiver is corrupted.

The difficulties mentioned above can be overcame by requiring each $\ddot{x}$ to hold a witness too. Since the receiver encodes his input as a permutation of $\dot{x}$s and $\ddot{x}$s, a simulator can the extract the real input of the adversary in the case that the receiver is corrupted if their witnesses are available. Combining the application of the technique cut-and-choose, a simulator can see such witnesses by rewinding the adversary. What is more, the implementation of DI is easier than that of its predecessor IT. Because the operated object essentially is a pair of the form (x, w) which is simpler than $(x_1, \ldots, x_n)$ which is the general form of the objects operated by IT.

4 Constructing A Framework For Fully-simulatable OT^n_h

In this section, we construct a framework for OT^n_h. 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.

4.1 The Framework For OT^n_h

- Common inputs: All entities know the public security parameter k, an positive polynomial $poly_s(.)$, a $SPHDHC_{t,h}$ (where $n = h + t$) hash system $\mathcal{H}$, a information-theoretically hiding commitment scheme (denoted by IHC), a information-theoretically binding commitment scheme (denoted by IBC).

- 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 (i.e., the receiver) holds a private input $H \in \Psi$ and a randomness $r_2 \in \{0,1\}^*$. The adversary A holds a name list $I \subseteq [2]$ and a randomness $r_A \in \{0,1\}^*$.
- Auxiliary Inputs: The adversary A holds an infinite auxiliary input sequence $z = (z_k)_{k\in\mathbb{N}}, z_k \in \{0,1\}^*$.

The protocol works as follow. For clarity, we omit some trivial error-handlings such as P_1 refusing to send P_2 something which is supposed to be sent. Handling such errors is easy. P_2 halting and outputting $abort_1$ suffices.

- Receiver's step (R1): P_2 generates hash parameters and samples instances.
  1) P_2 samples $poly_s(k)$ instance vectors. Let $K \overset{def}{=} poly_s(k)$. 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 instances and the corresponding hash parameters, i.e., $(\Lambda, \tilde{\vec{x}}_1, \tilde{\vec{x}}_2, \ldots, \tilde{\vec{x}}_K)$, to P_1, where $\tilde{\vec{x}}_i \overset{def}{=} \vec{x}^{\tilde{\vec{a}}_i}$ (correspondingly, $\tilde{\vec{w}}_i \overset{def}{=} \vec{w}^{\tilde{\vec{a}}_i}$).
- Receiver's step (R2-R3)/Sender's step (S1-S2): P_1 and P_2 cooperate to toss coin to choose instance vectors to open.
  1) P_1: $s \in_U \{0,1\}^K$, sends $IHC(s)$ to P_2.
  2) P_2: $s' \in_U \{0,1\}^K$, sends $IBC(s')$ to P_1.
  3) P_1 and P_2 respectively sends each other the de-commitments to $IHC(s)$ or $IBC(s')$, and respectively checks the received de-commitments are valid. If the check fails, P_1 (P_2 respectively) halts and outputs $abort_2$ ($abort_1$ respectively). If no check fails, then they proceed to next step.
  4) P_1 and P_2 share a common randomness $r = s \oplus s'$. The instance vectors whose index fall into $CS \overset{def}{=} \{i | r\langle i \rangle = 1, i \in [K]\}$ (correspondingly, $\overline{CS} \overset{def}{=} [K] - CS$) are chosen to open.
- Receiver's step (R4): P_2 opens the chosen instances to P_1, encodes and sends his private input to P_1.
  1) P_2 opens the chosen instances to prove that the instances he generates 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 $J_i \overset{def}{=} \{j | \tilde{\vec{x}}_i\langle j \rangle \in L_{\ddot{R}_\Lambda}, j \in [n]\}$.
  2) P_2 encodes his private input and sends the resulting code to P_1. Let $G_i \overset{def}{=} \{j | \tilde{\vec{x}}_i\langle j \rangle \in L_{\dot{R}_\Lambda}, i \in \overline{CS}\}$. 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 encode his private input into sequences such as $\pi_i^2(\tilde{\vec{x}}_i)$ where $i \in \overline{CS}$.

  Note 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 (S3): P_1 checks the chosen instances, encrypts and sends his private input to P_2.
  1) P_1 verifies that each chosen instance vectors is legal, i.e., the number of the entries belonging to $L_{\dot{R}_\Lambda}$ is not more than 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 the check fails, P_1 halts and outputs $abort_2$, 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: $(hk_{ij}, pk_{ij}) \leftarrow KG(1^k, \Lambda, \tilde{\tilde{\vec{x}}}_i\langle j \rangle)$, $\beta_{ij} \leftarrow Hash(1^k, \Lambda, \tilde{\tilde{\vec{x}}}_i\langle j \rangle, hk_{ij})$, $\vec{\beta}_i \overset{def}{=} (\beta_{i1}, \beta_{i2}, \ldots, \beta_{in})^T$, $\vec{c} \leftarrow \vec{m} \oplus (\oplus_{i\in\overline{CS}} \vec{\beta}_i)$, $\overrightarrow{pk}_i \overset{def}{=} (pk_{i1}, pk_{i2}, \ldots, pk_{in})^T$, sends $\vec{c}$ and $(\overrightarrow{pk}_i)_{i\in\overline{CS}}$ to P_2.
- Receiver's step (R5): 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: $\beta'_{ij} \leftarrow pHash(1^k, \Lambda, \tilde{\tilde{\vec{x}}}_i\langle j \rangle, \overrightarrow{pk}_i\langle j \rangle, \tilde{\tilde{\vec{w}}}_i\langle j \rangle)$, $m'_j \leftarrow \vec{c}\langle j \rangle \oplus (\oplus_{i\in\overline{CS}} \beta'_{ij})$. Finally, P_2 gains the messages $(m'_j)_{j\in H}$.

4.2 The Correctness Of The Framework

Now let us check the correctness of the framework, i.e., the framework works in the case that 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.

4.3 The Security Of The Framework

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

Theorem 8 (The protocol is secure against the malicious adversaries)**.** *Assume that $\mathcal{H}$ is an t-smooth h-projective hash family that holds properties distinguishability, hard subset membership and feasible cheating, IHC is a information-theoretically hiding commitment, IBC is a information-theoretically binding commitment. Then, the protocol securely computes the oblivious transfer functionality in the presence of non-adaptive malicious adversaries.*

We defer the strick proof of Theorem 8 to section 5 and first give an intuitive analysis here 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 makes sure with some probability that each instance vector contains no more than h projective instance, which leads to P_2 learning extra messages is difficult. The following theorem guarantees that this probability is overwhelming.

Theorem 9. *Assume that the commitment schemes employed in the framework are a perfectly hiding commitment and a perfectly binding commitment. Then, in case that P_1 is honest and P_2 is corrupted, the probability that P_2 cheats to obtain more than h messages is at most $1/2^{poly_s(k)}$.*

Proof: According to the framework, there are two 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 case.
   a) In the case that $\overline{CS} \neq \{l_1, l_2, \ldots, l_d\}$ 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 the case that $\overline{CS} \neq \{l_1, l_2, \ldots, l_d\}$ 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 $\vec{c}$.

Now, let us estimate the probability that the second necessary condition is met. Note that, $IHC(s)$ is a perfectly hiding commitment, $IBC(s')$ is a perfectly binding commitment, and P_1 is honest, so the shared randomness r is uniformly distributed. We have

$$\begin{aligned} Pr(\overline{CS} = \{l_1, l_2, \ldots, l_d\}) &= (1/2)^d (1/2)^{poly_s(k)-d} \\ &= 1/2^{poly_s(k)} \end{aligned}$$

This means that the probability that P_2 cheats to obtain more than h messages is at most $1/2^{poly_s(k)}$. □

From the proof of Theorem 9, it is easy to see that if the commitment schemes employed are the ones with statically properties, the probability that P_2 cheats to obtain more than h messages is negligible too, since the upper-bound of this probability deviates $1/2^{poly_s(k)}$ at most negligible distance.

For the security of P_2, the framework first should prevent P_1 from learning P_2's private input. There is a potential risk in Step R4 where P_2 encodes his private input. From Remark 6, we know that hard subset membership guarantees that for any PPT malicious P_1, without being given π_i^1, the probability that P_1 learns any new knowledge is negligible. Thus P_2's encoding is safe. 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} which $(hk_{ij}, pk_{ij}) \notin Range(KG(1^k, \Lambda, x_{ij}))$, 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.

4.4 The Communication Rounds

Step R1 and Step R2 can be taken in one round. Step R5 is taken without communication. Each of other steps is taken in one round. Therefore, the total number of the communication rounds is six.

Compared with existing fully-simulatable protocols for oblivious transfer that without resorting to a random oracle or a trusted common reference string (CRS), our protocol is the most efficient one. 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 $OT^n_{h\times 1}$ of [6] 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^n_h of [29] 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 [54] and taken as a zero-knowledge proof of knowledge protocol in [29], to our best knowledge, is not known to be zero-knowledge. However, turning to the techniques of Σ-protocol, [14] 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 [29]. Taking all into consideration, this protocol costs at least ten rounds. The protocol for OT^2_1 of [37] costs six rounds.

4.5 The Computational Overhead

We measure the computational overhead of the framework 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. Please see [39] to know which cryptographic operation is public key operation or private key operation.

As to the framework, the public key operations are $Hash(.)$ and $pHash(.)$, and the symmetric key operations are $IHC(.)$ and $IBC(.)$. In Step S3, P_1 takes $n \cdot \#\overline{CS}$ invocations of $Hash(.)$ to encrypt his private input. In Step R5, P_2 takes $h \cdot \#\overline{CS}$ invocations of $pHash(.)$ to decrypt the messages he want. The value of $\#\overline{CS}$ is $poly_s(k)$, $poly_s(k)/2$, 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 $poly_s(k)$. In Section 5 where we strictly prove the security of the framework, we'll see that in the case that only P_2 is corrupted, our simulator doesn't consider a situation in the real world that arises with probability at most $1/2^{poly_s(k)}$. Therefore, setting $poly_s(k) = 40$ is secure enough to use our framework in practice. 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 average case the computational overhead mainly consists of $20n$ invocations of $Hash()$ taken by P_1 and $20h$ invocations of $pHash()$ taken by P_2.

We point out that, our simulator also may fail (with negligible probability) in the case that P_1 is corrupted, but the probability of this event arising depends on the computational hiding of IBC and on the computational binding of IHC rather than the value of $poly_s(k)$ and has no influence on computational overhead. So we don't need to take this case into consideration here.

Compared with existing fully-simulatable protocols for oblivious transfer that without resorting to a random oracle or a trusted CRS, our DDH-based instantiation that will be presented in Section 7.1 is the most efficient one in computational overhead. The operations of the protocol in [6] are based on the non-standard assumptions, i.e., q-Power Decisional Diffie-Hellman and q-Strong Diffie-Hellman (q-SDH) assumptions, which both are associated with bilinear groups. [13] indicates that q-SDH-based operations are more expensive that standard-assumption-based operations. The operations of the protocol in [29] are based on Decisional Bilinear Diffie-Hellman (DBDH) assumption. Since bilinear curves are considerably more expensive than regular Elliptic curves [20] and DDH is obtainable from Elliptic curves, the operations in [6], [29] are considerably more expensive than that DDH-based operations. Therefore, our DDH-based instantiation are more efficient than the protocols presented by [6], [29]. The DDH-based protocol for OT_1^2 presented by [37] also are very efficient. However, it can be viewed as a specific case of our framework, thought some modification of the protocol is needed.

We have to admit that, in the context of a trusted CRS is available and only OT_1^2 is needed, [51]'s DDH-based instantiation, which is two-round efficient and of two public key encryption operations and one public key decryption operation, is the most efficient one, no matter seen from the number of communication rounds or the computational overhead.

5 A Security Proof Of The Framework

We prove Theorem 8 holds in this section. For notational clarity, we denote the entities, the parties and the adversary in the real world by P_1, P_2, A, and denote the corresponding entities in the ideal world by P_1', P_2', A'. In the light of the parties being corrupted, there are four cases to be considered and we prove Theorem 8 holds in each case. For simplicity, we assume that the commitment schemes employed are a perfectly binding commitment scheme and a perfectly hiding commitment scheme. If the statically ones are employed, the proof can be done in the same way with a slight modification.

We don't know how to construct a strictly polynomial-time simulator for the adversary in the real world, in the case that only P_1 or P_2 is corrupted. Instead, expected polynomial-time simulators are constructed (see section 2.2 for the justification), which results in a failure of standard black-box reduction technique. Fortunately, the problem and its derived problems can be solved using the technique given by [27].

5.1 In the case that P_1 Is Corrupted

In the case that P_1 is corrupted, A takes the full control of P_1 in the real world. Correspondingly, A's simulator, A', takes the full control of P_1' in the ideal world, where A' is constructed as follow.

- Initial input: A' holds the same k, $I \stackrel{def}{=} \{1\}$, $z = (z_k)_{k \in \mathbb{N}}$, as A. What is more, A' holds a uniform distributed randomness $r_{A'} \in \{0,1\}^*$. The parties P_1' and P_1, whom A' and A respectively is to corrupt, hold the same $\vec{m}$.
- A' works as follows.
  - Step Sm1: A' corrupts P_1' and learns P_1''s private input $\vec{m}$. Let $\bar{A}$ be a copy of A, i.e., $\bar{A} = A$. A' use $\bar{A}$ as a subroutine. A' fixes the initial inputs of $\bar{A}$ to be identical to his except that fixes the randomness of $\bar{A}$ to be a uniformly distributed value. A' activates $\bar{A}$, and supplies $\bar{A}$ with $\vec{m}$ before $\bar{A}$ engages in the protocol for OT_h^n.
    In the following steps, A' builds an environment for $\bar{A}$ which simulates the real world. That is, A' disguises himself as P_1 and P_2 at the same time to interact with $\bar{A}$.
  - Step Sm2: A' uniformly chooses a randomness $r \in_U \{0,1\}^K$ ($K \stackrel{def}{=} poly_s(k)$) as the shared randomness. Let CS and $\overline{CS}$ be the sets decided by r. For each $i \in CS$, A' honestly generates the hash parameters and instance vectors. For each $i \in \overline{CS}$, A' calls $Cheat(1^k)$ to generate such parameters and vectors. A' sends these hash parameters and instance vectors to $\bar{A}$.
    Remark 10. *From the remark 6, we know that each entry of the instance vector generated by $Cheat(1^k)$ is projective. If such instance vectors are not chosen to be open, then the probability of $\bar{A}$ detecting this fact is negligible, and A' can extract the real input of $\bar{A}$, which is we want.*

- Step Sm3: A' plays the role of P_2 and executes Step R2-R3 of the framework to cooperate with $\bar{A}$ to toss coin. When tossing coin is completed successfully, A' learns and records the value s $\bar{A}$ commits to.

  Remark 11. *The aim of doing this tossing coin is to know the randomness s $\bar{A}$ choses. What A' will do next is to take $IBC(r \oplus s)$ as his commitment to redo tossing coin.*
- Step Sm4: A' repeats the following procedure, denoted Υ, until $\bar{A}$ correctly reveals the recorded value s.

  Υ: A' rewinds $\bar{A}$ to the end of Step S1 of the framework. Then, taking $IBC_\gamma(r \oplus s)$ as his commitment, A' executes Step R2 and R3 of the framework, where γ is a fresh randomness uniformly chosen.
- Step Sm5: Now A' and $\bar{A}$ shares the common randomness r. A' executes Step R4 of the framework as the honest P_2 do. On receiving $\vec{c}$ and $(\overrightarrow{pk}_i)_{i \in \overline{CS}}$, A' correctly decrypts all entries of $\vec{c}$ and gains $\bar{A}$'s full real private input $\vec{m}$. Then A' sends $\vec{m}$ to the TTP.
- Step Sim6: When $\bar{A}$ halts, A' halts with outputting what $\bar{A}$ outputs.

Without considering Step Sim4, A' is polynomial-time. However, taking Step Sim4 into consideration, this is not true any more. Let $q(\alpha)$, $p(\alpha)$ respectively denotes the probability that $\bar{A}$ correctly reveals his commitment in Step Sim3 and in Procedure Υ, where $\alpha \overset{def}{=} (1^k, z_k, I, \vec{m}, r_{\bar{A}})$. Then, the expected times of repeating Υ in Step Sim4 is $q(\alpha)/p(\alpha)$. Since the view $\bar{A}$ holds before revealing his commitment in Step Sim3 is different from that in procedure Υ, $q(\alpha)$, $p(\alpha)$ are distinct. What the computational secrecy of IBC guarantees and only guarantees is $|q(\alpha) - p(\alpha)| = \mu(.)$. However, there is a risk that $q(\alpha)/p(\alpha)$ is not bound by a polynomial. For example, $q(\alpha) = 1/2^k, p(\alpha) = 1/2^{2k}$, which result in $q(\alpha)/p(\alpha) = 2^k$. This is a big problem and gives rise to many other difficulties we will encounter later.

Fortunately, [27] encounters and solves the same problem and its derived problem as ours. In a little more details, [27] presents a protocol, in which P_1, P_2 respectively sends a perfectly hiding commitment, a perfectly binding commitment, and the corresponding decommitments to each other as the situation of tossing coin of our framework. To prove the security in the case that P_1 is corrupted, [27] constructs a simulator in the same way as ours and encounters the same problem as ours.

Using the idea of [27], we can overcome such problem too. Specifically, an expected polynomial-time simulator can be obtained by replacing Step Sim4 with Step $Sim4.1$, $Sim4.2$ given as follow.

- Step $Sim4.1$: A' estimates the value of $q(\alpha)$. A' repeats the following procedure, denoted Φ, until the number of the time of $\bar{A}$ correctly revealing his commitment is up to $poly(k)$, where $poly(.)$ is a big enough polynomial.

  Φ: A' rewinds $\bar{A}$ to the end of Step S1 of the framework and A' honestly executes Step R2 and R3 of the framework to interact with it.

  Denote the number of times that Φ is repeated by d, then $q(\alpha)$ is estimated as $\tilde{q}(\alpha) \overset{def}{=} poly(k)/d$.
- Step $Sim4.2$: A' repeats the procedure Υ. In case $\bar{A}$ correctly reveals the recorded value s, A' proceeds to the next step. In case $\bar{A}$ correctly reveals a value which is different from s, A' outputs $ambiguity_1$ and halts. In case the number of the time of repeating Υ exceeds the value of $poly(k)/\tilde{q}(\alpha)$, A' outputs $timeout$ and halts.

Proposition 12. *The simulator A' is expected polynomial-time.*

Proof: Conditioning on Step $Sim4.1$ is executed, the expected value of d is $poly(k)/q(\alpha)$. Choosing a big enough $poly(.)$, $\tilde{q}(\alpha)$ is within a constant factor of $q(\alpha)$ with probability $1 - 2^{poly(k)}$. Therefore, the expected running time of A',

$$\begin{aligned} ExpTime_{A'} \leq\ & Time_{Sim1} + Time_{Sim2} + Time_{Sim3} \\ & + q(\alpha) \cdot (Time_\Phi \cdot poly(k)/q(\alpha) + \\ & Time_\Upsilon \cdot poly(k)/\tilde{q}(\alpha)) \\ & + Time_{Sim5} + Time_{Sim6} \end{aligned}$$

, is bounded by a polynomial. □

What is more, we have

1) The probability that A' outputs $timeout$ is negligible.
2) The probability that A' outputs $ambiguity_1$ is negligible.
3) The output of A' in the ideal world and the output of A in the real world are computationally indistinguishable, i.e.,

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

Since the propositions above can be proven in the same way as [27], we don't iterate such details here.

Proposition 13. *In the case that P_1 was corrupted, i.e., $I = \{1\}$, the equation* (1) *required by Definition 1 holds.*

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

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

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

$$Real_{\pi,\{1\},A(z_k)}(1^k, \vec{m}, H) \equiv h(Real_{\pi,\{1\},A(z_k)}(1^k, \vec{m}, H)\langle 0\rangle)$$

Similarly, in the ideal world, we have

$$Ideal_{f,\{1\},A'(z_k)}(1^k, \vec{m}, H) \overset{c}{=} h(Ideal_{f,\{1\},A'(z_k)}(1^k, \vec{m}, H)\langle 0\rangle)$$

We use $\overset{c}{=}$ not $\equiv$ here because there is a negligible probability that A' outputs $timeout$ or $ambiguity_1$, which makes $h(.)$ undefined.

Let $X(1^k, \vec{m}, H, z_k, \{1\}) \overset{def}{=} Real_{\pi,\{1\},A(z_k)}(1^k, \vec{m}, H)\langle 0\rangle$, $Y(1^k, \vec{m}, H, z_k, \{1\}) \overset{def}{=} Ideal_{f,\{1\},A'(z_k)}(1^k, \vec{m}, H)\langle 0\rangle$. Following equation (2), $X \overset{c}{=} Y$. Let $F \overset{def}{=} (h)_{k\in\mathbb{N}}$. What is more, assume that A' runs in a strictly polynomial-time. According to Proposition 20 we will present in Section 7, the proposition holds.

In fact, A' doesn't run in strictly polynomial-time, which results in a failure of above standard reduction. Fortunately, this difficulty can be overcome by truncating the rare executions of A' which are too long, then applying standard reduction techniques. Since the details is the same as [27], we don't give them here and please see [27] for them. □

5.2 In the case that P_2 Is Corrupted

In the case that P_2 is corrupted, A takes the full control of P_2 in the real world. Correspondingly, A' takes the full control of P_2' in the ideal world. We construct A' as follows.

- Initial input: A' holds the same k, $I \overset{def}{=} \{2\}$, $z = (z_k)_{k\in\mathbb{N}}$ as A, and holds a uniformly distributed randomness $r_{A'} \in \{0,1\}^*$. The parties P_2' and P_2 hold the same private input H.
- A' works as follows.
  - Step Sim1: A' corrupts P_2' and learns P_2''s private input H. A' takes A's copy $\bar{A}$ as a subroutine, fixes $\bar{A}$'s initial input, activates $\bar{A}$, supplies $\bar{A}$ with H, builds an environment for $\bar{A}$ in the same way as A' does in the case that P_1 is corrupted.
  - Step Sim2: Playing the role of P_1, A' honestly executes the sender's steps until reaches Step S3.3. If Step S3.3 is reached, A' records the shared randomness r and the messages, denoted msg, which he sends to $\bar{A}$. Then A' proceeds to next step. Otherwise, A' sends $abort_2$ to TTP, outputs what $\bar{A}$ outputs and halts.
  - Step Sim3: A' repeats the following procedure, denoted Ξ, until the hash parameters and the instance vectors $\bar{A}$ sends in Step R1 passes the check. A' records the shared randomness $\tilde{r}$, the messages $\bar{A}$ sends to open the chosen instance vectors.

    Ξ: A' rewinds $\bar{A}$ to the beginning of Step R2, and honestly follows sender's steps until reaches Step S3.3 to interact with $\bar{A}$.

    Note that, in each repeating Ξ, the value A' commits to and the randomness used to generate the commitment in Step S1 are fresh and uniformly chosen.
  - Step Sim4:
    1) In case $r = \tilde{r}$, A' outputs $failure$ and halts;
    2) In case $r \neq \tilde{r} \wedge \forall i(r\langle i\rangle \neq \tilde{r}\langle i\rangle \rightarrow r\langle i\rangle = 1 \wedge \tilde{r}\langle i\rangle = 0)$, A' runs from scratch;
    3) Otherwise, i.e., in case $r \neq \tilde{r} \wedge \exists i(r\langle i\rangle = 0 \wedge \tilde{r}\langle i\rangle = 1)$, A' records the first one, denoted e, of these is and proceeds to next step.

    Remark 14. *The aim of Step Sim3 and Sim4 is to prepare to extract the real input of $\bar{A}$. If the third case happens, then A' knows each entry of $\tilde{\vec{x}}_e$ he sees in Step Sim2 belong to $L_{\dot{R}_{\Lambda_e}}$ or $L_{\ddot{R}_{\Lambda_e}}$. 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 A' in Step Sim3. Combing π_e^2 received in Step Sim2, A' knows the real input of $\bar{A}$.*

    Note that, $\bar{A}$'s initial input is fixed by A' 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, A' can reproduce the same scenario as he meets in Step Sim2.
  - Step Sim5: A' 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 14, A' can extract $\bar{A}$'s real input H'. A' does so and sends H' to TTP and receives message $\vec{m}\langle H'\rangle$.
  - Step Sim6: A' constructs $\vec{m}'$ as follows. For each $i \in H'$, $\vec{m}'\langle i\rangle \leftarrow \vec{m}\langle i\rangle$. For each $i \notin H'$, $\vec{m}'\langle i\rangle \in_U \{0,1\}^*$. Playing the role of P_1 and taking $\vec{m}'$ as his real input, A' follows Step S3.3 to complete the interaction with $\bar{A}$.
  - Step Sim6: When $\bar{A}$ halts, A' halts with outputting what $\bar{A}$ outputs..

Note that S doesn't simulate a situation in the real world that A cheats P_1 of more than h message. Fortunately, Theorem 9 guarantees that this situation arises with probability at most $1/2^{poly_s(k)}$ and so can be ignored.

Proposition 15. *The simulator A' is expected polynomial-time.*

Proof: First, let us focus on Step Sim3. In each repetition of Ξ, because of the perfectly hiding of $IHC(.)$, and the uniform distribution of the value A' commits to, the chosen instance vectors are uniformly distributed. This lead to the probability that $\bar{A}$ passes the check in each repetition is the 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 A' 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 the case that A' needs to run from scratch. Note that the initial inputs A' holds is the same in each trial. Thus the probability that A' runs from scratch in each trial is the same. We denote this probability by $1 - q$. Then the expected time that A' 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 there is 1 here is that A' has to run from scratch at least one time in any case.

The expected running time of A' in a whole execution is

$$\begin{aligned} ExpTime_{A'} &\leq ExpTime_{Sim1 \to Sim5} + Time_{Sim5} \\ &+ Time_{Sim6} \\ &= (1 + 1/q) \cdot (Time_{Sim1} + Time_{Sim2} \\ &+ Time_{\Xi} + Time_{Sim4}) \\ &+ Time_{Sim5} + Time_{Sim6} \end{aligned} \quad (3)$$

Third, let us estimate the value of q, which is the probability that A' 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 events happens.

1) Event B happens, where B denotes the even that A' halts before reaching Step Sim3.
2) Event $\bar{B}$ happens and $R = \tilde{R}$, where R and $\tilde{R}$ respectively denotes the random variable which is defined as the shared randomness A' gets 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

$$\begin{aligned} q =& Pr(C) \\ =& Pr(B) + Pr(\bar{B} \cap R = \tilde{R}) \\ &+ Pr(\bar{B} \cap \exists i(R\langle i \rangle = 0 \wedge \tilde{R}\langle i \rangle = 1)) \\ =& Pr(B) + Pr(\bar{B}) \cdot (Pr(R = \tilde{R}|\bar{B}) \\ &+ Pr(\exists i(R\langle i \rangle = 0 \wedge \tilde{R}\langle i \rangle = 1)|\bar{B})) \end{aligned} \quad (4)$$

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 $\#S_1 = 2^K$, $\#S_2 = \#S_3 = (2^K \cdot 2^K - 2^K)/2$.

Because of the perfectly hiding of $IHC(.)$, and the uniform distribution of the value A' commits to, R and $\tilde{R}$ are all uniformly distributed. We have

$$Pr(R = \tilde{R}|\bar{B}) = \#S_1 / \#(\{0,1\}^K)^2 = 1/2^K \quad (5)$$

and

$$\begin{aligned} Pr(\exists i(R\langle i \rangle = 0 \wedge \tilde{R}\langle i \rangle = 1)|\bar{B}) &= \#S_3 / \#(\{0,1\}^K)^2 \\ &= 1/2 - 1/2^{K+1} \end{aligned} \quad (6)$$

Combining equation (4), (5) and (6), we have

$$\begin{aligned} q &= Pr(B) + Pr(\bar{B})(1/2 + 1/2^{K+1}) \\ &= 1/2 + 1/2^{K+1} + (1/2 - 1/2^{K+1})Pr(B) \\ &> 1/2 \end{aligned} \quad (7)$$

Combining equation (3) and (7), we have

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

which means the expected running time of A' is bound by a polynomial. □

Lemma 16. *The probability that A' 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 15, we know two facts. First, $Pr(X = i) = (1 - q)^{i-1} q < 1/2^{i-1}$. Second, in each trial the event A' outputs $failure$ is the combined event of $\bar{B}$ and $R = \tilde{R}$, where the combined event happens with the following 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 (5), this probability is not more than $1/2^K$. Therefore, the probability that A' outputs $failure$ in a whole execution is

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

□

Lemma 17. *The output of the adversary A in the real world and that of the simulator A' in the ideal world are computationally indistinguishable, i.e.,*

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

Proof: First, we claim that the outputs of A' and $\bar{A}$ are computationally indistinguishable. The only point that the output of A' is different from that of $\bar{A}$ is A'

may outputs $failure$. Since the probability that this point arises is negligible, our claim holds.

Second, we claim that the outputs of A and $\bar{A}$ are computationally indistinguishable. The only point that the view of $\bar{A}$ is different from that of A is that the ciphertext $\bar{A}$ receives is generated by encrypting $\vec{m}'$ not $\vec{m}$. Fortunately, $SPHDHC_{t,h}$'s property smoothness guarantees that the ciphertext generated in the two way are computationally indistinguishable. Therefore, our claim holds.

Combining the two claims, the proposition holds. □

Proposition 18. *In the case that 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 A' and A are computationally indistinguishable, which is supported by Lemma 17, directly prove this proposition holds. □

5.3 Other Cases

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

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

6 How To Construct $SPHDHC_{t,h}$ Easily

$SPHDHC_{t,h}$ holds so many properties that constructing it from scratch is not always easy. In this section, we reduce constructing $SPHDHC_{t,h}$ to constructing seemingly simpler hash systems. A idea naturally arising is that generating the instances independently in essence to obtain the required properties. We keep this idea in mind to proceed to construct $SPHDHC_{t,h}$.

6.1 Smoothness

In this section, we describe how to obtain smoothness for a hash family. First, we introduce a lemma from [25].

Lemma 19 ([25]). *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, and $X \stackrel{c}{=} Y$, then*

$$\vec{X} \stackrel{c}{=} \vec{Y}$$

where $\vec{X} \stackrel{def}{=} \{\vec{X}(1^k,a)\}_{\substack{k\in\mathbb{N}\\ a\in\{0,1\}^*}}$, $\vec{X}(1^k,a) \stackrel{def}{=} (X_i(1^k,a))_{i\in[poly(k)]}$, each $X_i(1^k,a) = X(1^k,a)$, $\vec{Y} \stackrel{def}{=} \{\vec{Y}(1^k,a)\}_{\substack{k\in\mathbb{N}\\ a\in\{0,1\}^*}}$, $\vec{Y}(1^k,a) \stackrel{def}{=} (Y_i(1^k,a))_{i\in[poly(k)]}$, each $Y_i(1^k,a) = Y(1^k,a)$, and all $X_i(1^k,a)$, $Y_i(1^k,a)$ are independent.

Proposition 20. *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, $X \stackrel{c}{=} Y$, $F \stackrel{def}{=} (f_k)_{k\in\mathbb{N}}$, $f_k : \{0,1\}^* \to \{0,1\}^*$ is polynomial-time computable, then*

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

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

$$|Pr(D(1^k, z_k, a, f_k(X(1^k,a))) = 1) - Pr(D(1^k, z_k, a, f_k(Y(1^k,a))) = 1)| \geq 1/poly(k)$$

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

$$|Pr(D'(1^k, z_k, a, X(1^k,a)) = 1) - Pr(D'(1^k, z_k, a, Y(1^k,a)) = 1)| \geq 1/poly(k)$$

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

Lemma 21. *Let $\mathcal{H} = (PG, IS, DI, KG, Hash, pHash, Cheat)$ be a Hash Family. $n \stackrel{def}{=} h + t$. For each $i \in [2]$ and $j \in [n]$, $Sm_i^j \stackrel{def}{=} \{Sm_i^j(1^k)\}_{k\in\mathbb{N}} \stackrel{def}{=} \{(SmGen_i(1^k)\langle 1\rangle, SmGen_i(1^k)\langle 2\rangle\langle j\rangle)\}_{k\in\mathbb{N}}$, where $SmGen_i(1^k)$ is defined in Definition 4. If $\mathcal{H}$ meets the following three conditions*

1) *All random variables $SmGen_i(1^k)\langle 2\rangle\langle j\rangle$ are independent, where $i \in [2]$,$j \in [n] - [h]$.*
2) $Sm_1^{h+1} = \ldots = Sm_1^n$, *and* $Sm_2^{h+1} = \ldots = Sm_2^n$.

3) $Sm_1^{h+1} \overset{c}{\equiv} Sm_2^{h+1}$.

then $\mathcal{H}$ has property smoothness.

Proof: Following Lemma 19,

$$\{(Sm_1^{h+1}(1^k), \ldots, Sm_1^n(1^k))\}_{k\in\mathbb{N}} \overset{c}{\equiv} \{(Sm_2^{h+1}(1^k), \ldots, Sm_2^n(1^k))\}_{k\in\mathbb{N}}$$

holds. Let $\vec{X} \overset{def}{=} \{(Sm_1^1(1^k), \ldots, Sm_1^n(1^k))\}_{k\in\mathbb{N}}$, and $\vec{Y} \overset{def}{=} \{(Sm_2^1(1^k), \ldots, Sm_2^n(1^k))\}_{k\in\mathbb{N}}$. From the definition of $SmGen_i(1^k)$, we notice that, for each $j \in [h]$ $Sm_1^j(1^k) = Sm_2^j(1^k)$. So it holds that

$$\vec{X} \overset{c}{\equiv} \vec{Y}$$

Since each $Sm_i^j(1^k)$ is polynomial-time constructible, thus both$\vec{X}$ and $\vec{Y}$ are polynomial-time constructible. Let $F \overset{def}{=} (\pi)_{k\in\mathbb{N}}$, where $\pi \in \Pi$. Following Proposition 20, we have $F(\vec{X}) \overset{c}{\equiv} F(\vec{Y})$, i.e.,

$$\{\pi(Sm_1^1(1^k), \ldots, Sm_1^n(1^k))\}_{k\in\mathbb{N}} \overset{c}{\equiv} \{\pi(Sm_2^1(1^k), \ldots, Sm_2^n(1^k))\}_{k\in\mathbb{N}}$$

Notice that $SmGen_1(1^k)\langle 1\rangle = SmGen_2(1^k)\langle 1\rangle$, we have

$$\{(SmGen_1(1^k)\langle 1\rangle, \pi(SmGen_1(1^k)\langle 2\rangle))\}_{k\in\mathbb{N}} \overset{c}{\equiv} \{(SmGen_2(1^k)\langle 1\rangle, \pi(SmGen_2(1^k)\langle 2\rangle))\}_{k\in\mathbb{N}}$$

That is

$$Sm_1 \overset{c}{\equiv} Sm_2$$

, which meets the requirement of the smoothness. □

Loosely speaking, following Lemma 21, given a hash family $\mathcal{H}$, if each $\ddot{x}$ was sampled in an independent way and its projective key is useless to obtain the value of $Hash(1^k, \Lambda, \ddot{x}, .)$, then $\mathcal{H}$ is smooth.

6.2 Hard Subset Membership

In this section, we deal with how to obtain hard subset membership for a hash family.

Proposition 22. *Let $X \overset{def}{=} \{X(1^k, a)\}_{k\in\mathbb{N}, a\in\{0,1\}^*}$ and $Y \overset{def}{=} \{Y(1^k, a)\}_{k\in\mathbb{N}, a\in\{0,1\}^*}$ be two polynomial-time constructible probability ensembles, and $X \overset{c}{\equiv} Y$. Then*

$$\overrightarrow{XY} \overset{c}{\equiv} \Phi(\overrightarrow{\overrightarrow{XY}})$$

where $\overrightarrow{XY}$ and $\Phi(\overrightarrow{\overrightarrow{XY}})$ are two probability ensembles defined as follows.

- $\overrightarrow{XY} \overset{def}{=} \{\overrightarrow{XY}(1^k, a)\}_{k\in\mathbb{N}, a\in\{0,1\}^*}$, $\overrightarrow{XY}(1^k, a) \overset{def}{=} (X_1(1^k, a), \ldots, X_{poly_1(k)}(1^k, a), Y_{poly_1(k)+1}(1^k, a), \ldots, Y_{poly(k)}(1^k, a))$, *each* $X_i(1^k, a) = X(1^k, a)$, *each* $Y_i(1^k, a) = Y(1^k, a)$, $poly_1(.) \leq poly(.)$, *all* $X_i(1^k, a)$ *and* $Y_i(1^k, a)$ *are independent;*
- $\Phi(\overrightarrow{\overrightarrow{XY}}) \overset{def}{=} \{\Phi_k(\overrightarrow{\overrightarrow{XY}}(1^k, a))\}_{k\in\mathbb{N}, a\in\{0,1\}^*}$, $\overrightarrow{\overrightarrow{XY}}(1^k, a) = \overrightarrow{XY}(1^k, a)$, $\Phi \overset{def}{=} (\Phi_k)_{k\in\mathbb{N}}$, *each* Φ_k *is a permutation over* $[poly(k)]$.

Proof: In case $\Phi_k([poly_1(k)]) \subseteq [poly_1(k)]$, it obviously holds. We proceed to prove it also holds in case $\Phi_k([poly_1(k)]) \not\subseteq [poly_1(k)]$. Assume it does not hold in this case, then there exists a non-uniform PPT distinguisher D with an infinite sequence $z = (z_k)_{k\in\mathbb{N}}$, a polynomial $poly_2(.)$, a infinite positive integer set $G \subseteq \mathbb{N}$ such that, for each $k \in G$,

$$|Pr(D(1^k, z_k, a, \overrightarrow{XY}(1^k, a)) = 1) - Pr(D(1^k, z_k, a, \Phi_k(\overrightarrow{\overrightarrow{XY}}(1^k, a)) = 1)| \geq 1/poly_2(k) \quad (8)$$

$V \overset{def}{=} \{i | i \in [poly_1(k)], \Phi_k(i) \in [poly(k)] - [poly_1(k)]\}$. We list the elements of V in order as $i_1 < ... < i_j ... < i_{\#V}$. Let $V_j \overset{def}{=} \{i_1, \ldots, i_j\}$. We define the following permutations over $[poly(k)]$.

$$\Phi_k^{0'}(i) = i \quad i \in [poly(k)]$$

$$\Phi_k^0(i) = \begin{cases} i & i \in V \cup \Phi_k(V) \\ \Phi_k(i) & i \in [poly(k)] - V - \Phi_k(V) \end{cases}$$

For $j \in [\#V]$,

$$\Phi_k^j(i) = \begin{cases} i & i \in (V - V_j) \cup \Phi_k(V - V_j), \\ \Phi_k(i) & i \in [poly(k)] - (V - V_j) - \Phi_k(V - V_j). \end{cases}$$

It is easy to see that $\overrightarrow{\overrightarrow{XY}}(1^k, a) = \Phi_k^{0'}(\overrightarrow{\overrightarrow{XY}}(1^k, a)) \equiv \Phi_k^0(\overrightarrow{\overrightarrow{XY}}(1^k, a))$, and $\Phi_k = \Phi_k^{\#V}$. Since $\overrightarrow{XY}(1^k, a) = \overrightarrow{\overrightarrow{XY}}(1^k, a)$, then $\overrightarrow{XY}(1^k, a) \overset{c}{\equiv} \Phi_k^0(\overrightarrow{\overrightarrow{XY}}(1^k, a))$. So we have

$$\begin{aligned} &|Pr(D(1^k, z_k, a, \overrightarrow{XY}(1^k, a)) = 1) - \\ &\quad Pr(D(1^k, z_k, a, \Phi_k(\overrightarrow{\overrightarrow{XY}}(1^k, a))) = 1)| \\ &= |Pr(D(1^k, z_k, a, \Phi_k^0(\overrightarrow{\overrightarrow{XY}}(1^k, a))) = 1) - \\ &\quad Pr(D(1^k, z_k, a, \Phi_k^{\#V}(\overrightarrow{\overrightarrow{XY}}(1^k, a))) = 1)| \quad (9) \end{aligned}$$

Following triangle inequality, we have

$$\begin{aligned} &|Pr(D(1^k, z_k, a, \Phi_k^0(\overrightarrow{\overrightarrow{XY}}(1^k, a))) = 1) - \\ &\quad Pr(D(1^k, z_k, a, \Phi_k^{\#V}(\overrightarrow{\overrightarrow{XY}}(1^k, a))) = 1)| \leq \\ &\sum_{j=1}^{\#V} |Pr(D(1^k, z_k, a, \Phi_k^{j-1}(\overrightarrow{\overrightarrow{XY}}(1^k, a))) = 1) - \\ &\quad Pr(D(1^k, z_k, a, \Phi_k^j(\overrightarrow{\overrightarrow{XY}}(1^k, a))) = 1)| \quad (10) \end{aligned}$$

Combining equation (8) (9) (10), we have

$$\begin{aligned} &\sum_{j=1}^{\#V} |Pr(D(1^k, z_k, a, \Phi_k^{j-1}(\overrightarrow{\overrightarrow{XY}}(1^k, a))) = 1) - \\ &\quad Pr(D(1^k, z_k, a, \Phi_k^j(\overrightarrow{\overrightarrow{XY}}(1^k, a))) = 1)| \geq 1/poly_2(k) \end{aligned}$$

So there exists $j \in [\#V]$ such that

$$|Pr(D(1^k, z_k, a, \Phi_k^{j-1}(\overrightarrow{\overrightarrow{XY}}(1^k, a))) = 1) - Pr(D(1^k, z_k, a, \Phi_k^{j}(\overrightarrow{\overrightarrow{XY}}(1^k, a))) = 1)| \geq 1/(\#V \cdot poly_2(k)) \quad (11)$$

According to the definition of Φ_k^{j-1}, Φ_k^j, the differences between them are the values of points $i_j, \Phi_k(i_j)$. Similarly, the only differences between $\Phi_k^{j-1}(\overrightarrow{\overrightarrow{XY}}(1^k, a))$ and $\Phi_k^{j}(\overrightarrow{\overrightarrow{XY}}(1^k, a))$ are the i_j-th and $\Phi_k(i_j)$-th entries, i.e., $\Phi_k^{j-1}(\overrightarrow{\overrightarrow{XY}}(1^k, a))\langle i_j\rangle = X(1^k, a)$, $\Phi_k^{j-1}(\overrightarrow{\overrightarrow{XY}}(1^k, a))\langle \Phi_k(i_j)\rangle = Y(1^k, a)$, $\Phi_k^{j}(\overrightarrow{\overrightarrow{XY}}(1^k, a))\langle i_j\rangle = Y(1^k, a)$, $\Phi_k^{j}(\overrightarrow{\overrightarrow{XY}}(1^k, a))\langle \Phi_k(i_j)\rangle = X(1^k, a)$.

Let $\overrightarrow{MXY} \stackrel{def}{=} \{\overrightarrow{MXY}(1^k, a)\}_{k\in\mathbb{N}, a\in\{0,1\}^*}$, where $\overrightarrow{MXY}(1^k, a)$ is defined as follows. For each $d \in [poly(k)]$,

$$\overrightarrow{MXY}(1^k, a)\langle d\rangle = \begin{cases} \Phi_k^{j-1}(\overrightarrow{\overrightarrow{XY}}(1^k, a))\langle d\rangle & d \neq \Phi_k(i_j) \\ X(1^k, a) & d = \Phi_k(i_j) \end{cases}$$

The difference between $\overrightarrow{MXY}(1^k, a)$ and $\Phi_k^{j-1}(\overrightarrow{\overrightarrow{XY}}(1^k, a))$ is that $\overrightarrow{MXY}(1^k, a)\langle \Phi_k(i_j)\rangle = X(1^k, a)$, $\Phi_k^{j-1}(\overrightarrow{\overrightarrow{XY}}(1^k, a))\langle \Phi_k(i_j)\rangle = Y(1^k, a)$. The difference between $\overrightarrow{MXY}(1^k, a)$ and $\Phi_k^{j}(\overrightarrow{\overrightarrow{XY}}(1^k, a))$ is that $\overrightarrow{MXY}(1^k, a)\langle i_j\rangle = X(1^k, a)$, $\Phi_k^{j}(\overrightarrow{\overrightarrow{XY}}(1^k, a))\langle i_j\rangle = Y(1^k, a)$. Following triangle inequality, we have

$$\begin{aligned} &|Pr(D(1^k, z_k, a, \Phi_k^{j-1}(\overrightarrow{\overrightarrow{XY}}(1^k, a))) = 1) - \\ &\quad Pr(D(1^k, z_k, a, \overrightarrow{MXY}(1^k, a)) = 1)| + \\ &\quad |Pr(D(1^k, z_k, a, \overrightarrow{MXY}(1^k, a)) = 1) - \\ &\quad Pr(D(1^k, z_k, a, \Phi_k^{j}(\overrightarrow{\overrightarrow{XY}}(1^k, a))) = 1)| \\ &\geq |Pr(D(1^k, z_k, a, \Phi_k^{j-1}(\overrightarrow{\overrightarrow{XY}}(1^k, a))) = 1) - \\ &\quad Pr(D(1^k, z_k, a, \Phi_k^{j}(\overrightarrow{\overrightarrow{XY}}(1^k, a))) = 1)| \quad (12) \end{aligned}$$

Combining (11) (12), we know that

$$|Pr(D(1^k, z_k, a, \Phi_k^{j-1}(\overrightarrow{\overrightarrow{XY}}(1^k, a))) = 1) - Pr(D(1^k, z_k, a, \overrightarrow{MXY}(1^k, a)) = 1)| \geq 1/(2\#V \cdot poly_2(k)) \quad (13)$$

or

$$|Pr(D(1^k, z_k, a, \overrightarrow{MXY}(1^k, a)) = 1) - Pr(D(1^k, z_k, a, \Phi_k^{j}(\overrightarrow{\overrightarrow{XY}}(1^k, a))) = 1)| \geq 1/(2\#V \cdot poly_2(k)) \quad (14)$$

holds. Without loss of generality, we assume equation (13) holds (in case equation (14) holds, the proof can be done in similar way). We can construct a distinguisher D' with an infinite sequence $z = (z_k)_{k\in\mathbb{N}}$ for the probability ensembles X and Y as follows.

$D'(1^k, z_k, a, \gamma)$: $\overrightarrow{xy}\langle\Phi_k^{j-1}(i)\rangle \leftarrow S_X(1^k, a)$ $\forall i \in [poly_1(k)]$, $\overrightarrow{xy}\langle\Phi_k^{j-1}(i)\rangle \leftarrow S_Y(1^k, a)$ $\forall i \in [poly(k)] - [poly_1(k)] - \{\Phi_k(i_j)\}$, $\overrightarrow{xy}\langle\Phi_k(i_j)\rangle \leftarrow \gamma$, finally outputs $D(1^k, z_k, a, \overrightarrow{xy})$.

Obviously, if γ is sampled from $Y(1^k, a)$, then $\overrightarrow{xy}$ is an instance of $\Phi_k^{j-1}(\overrightarrow{\overrightarrow{XY}}(1^k, a))$; if γ is sampled from $X(1^k, a)$, then $\overrightarrow{xy}$ is an instance of $\overrightarrow{MXY}(1^k, a)$. So we have

$$\begin{aligned} &|Pr(D'(1^k, z_k, a, X(1^k, a)) = 1) - \\ &\quad Pr(D'(1^k, z_k, a, Y(1^k, a)) = 1)| = \\ &\quad |Pr(D(1^k, z_k, a, \overrightarrow{MXY}(1^k, a)) = 1) - \\ &\quad Pr(D(1^k, z_k, a, \Phi_k^{j-1}(\overrightarrow{\overrightarrow{XY}}(1^k, a))) = 1)| \quad (15) \end{aligned}$$

Combining (13) (15), we have

$$|Pr(D'(1^k, z_k, a, X(1^k, a)) = 1) - Pr(D'(1^k, z_k, a, Y(1^k, a)) = 1)| \geq 1/(2\#V \cdot poly_2(k))$$

This contradicts the fact $X \stackrel{c}{=} Y$. Therefore, the proposition also holds in case $\Phi_k([poly_1(k)]) \nsubseteq [poly_1(k)]$ too. □

Lemma 23. *Let $\mathcal{H} = (PG, IS, DI, KG, Hash, pHash, Cheat)$ be a hash family. Let $n \stackrel{def}{=} h + t$. For each $i \in [n]$, $HSM^i \stackrel{def}{=} \{HSM^i(1^k)\}_{k\in\mathbb{N}}$, $HSM^i(1^k) \stackrel{def}{=} (HSM_1(1^k)\langle 1\rangle, HSM_1(1^k)\langle i+1\rangle)$, where $HSM_1(1^k)$ is defined in Definition 4. If $\mathcal{H}$ meets the following three conditions,*

1) *All variables $HSM_1(1^k)\langle i+1\rangle$ are independent, where $i \in [n]$.*
2) *$HSM^1 = \ldots = HSM^h$, $HSM^{h+1} = \ldots = HSM^n$.*
3) *$HSM^1 \stackrel{c}{=} HSM^{h+1}$.*

then $\mathcal{H}$ has property hard subset membership.

Proof: Let $\pi \in \Pi$, $X \stackrel{def}{=} HSM^1$, $Y \stackrel{def}{=} HSM^{h+1}$, $\Phi = (\pi)_{k\in\mathbb{N}}$, $poly_1(.) \stackrel{def}{=} h$, $poly(.) \stackrel{def}{=} n$. Following Proposition 22, we know

$$\overrightarrow{XY} \stackrel{c}{=} \Phi(\overrightarrow{\overrightarrow{XY}})$$

That is

$$\begin{aligned} &((HSM_1(1^k)\langle 1\rangle, HSM_1(1^k)\langle 2\rangle), \ldots \\ &\quad (HSM_1(1^k)\langle 1\rangle, HSM_1(1^k)\langle n+1\rangle)) \stackrel{c}{=} \\ &\quad (HSM_2(1^k)\langle 1\rangle, HSM_2(1^k)\langle 2\rangle), \ldots \\ &\quad (HSM_2(1^k)\langle 1\rangle, HSM_2(1^k)\langle n+1\rangle)) \end{aligned}$$

where $HSM_1(1^k)$, $HSM_2(1^k)$ are taken from Definition 4. Note that $HSM_1(1^k)\langle 1\rangle = HSM_2(1^k)\langle 1\rangle$, so

$$\begin{aligned} &(HSM_1(1^k)\langle 1\rangle, HSM_1(1^k)\langle 2\rangle, \ldots, HSM_1(1^k)\langle n+1\rangle) \stackrel{c}{=} \\ &(HSM_2(1^k)\langle 1\rangle, HSM_2(1^k)\langle 2\rangle, \ldots, HSM_2(1^k)\langle n+1\rangle) \end{aligned}$$

i.e.,

$$HSM_1 \stackrel{c}{=} HSM_2$$

, which meets the requirement of the property hard subset membership. □

Loosely speaking, Lemma 23 shows that, given a hash family $\mathcal{H}$, if random variables $IS(1^k,\Lambda)\langle 1\rangle,\ldots,IS(1^k,\Lambda)\langle n\rangle$ are independent, $IS(1^k,\Lambda)\langle 1\rangle,\ldots,IS(1^k,\Lambda)\langle h\rangle$ sample $\dot{x}$ from $L_{\dot{R}_\Lambda}$ in the same way , $IS(1^k,\Lambda)\langle h+1\rangle,\ldots,IS(1^k,\Lambda)\langle n\rangle$ sample $\ddot{x}$ from $L_{\ddot{R}_\Lambda}$ in the same way, $L_{\dot{R}_\Lambda}$ and $L_{\ddot{R}_\Lambda}$ are computationally indistinguishable, then $\mathcal{H}$ has hard subset membership.

6.3 Reducing To Constructing Considerably Simpler Hash

In this section, we reduce constructing $SPHDHC_{t,h}$ to constructing considerably simpler hash.

Definition 24 (smooth projective hash family that holds properties distinguishability and hard subset membership)**.** *$\mathcal{H}=(PG,IS,DI,KG,Hash,pHash)$ is a smooth projective hash family that holds properties distinguishability and hard subset membership (SPHDH), if and only if $\mathcal{H}$ is specified as follows*

- *The algorithms PG, DI, KG, $Hash$, and $pHash$ are specified as same as in $SPHDHC_{t,h}$'s definition, i.e., Definition 4.*
- *The instance-sampler IS is a PPT algorithm that takes a security parameter k, a family parameter Λ, a work mode $\delta\in\{0,1\}$ as input and outputs a instance along with its witness (x,w), i.e., $(x,w)\leftarrow IS(1^k,\Lambda,\delta)$.*
  Correspondingly, we define relations $R_\Lambda,\dot{R}_\Lambda,\ddot{R}_\Lambda$ as follows. $\dot{R}_\Lambda \stackrel{def}{=} \cup_{k\in\mathbb{N}}Rang(IS(1^k,\Lambda,0))$, $\ddot{R}_\Lambda \stackrel{def}{=} \cup_{k\in\mathbb{N}}Rang(IS(1^k,\Lambda,1))$, $R_\Lambda \stackrel{def}{=} \dot{R}_\Lambda\cup\ddot{R}_\Lambda$.

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

1) *The properties projection and distinguishability are specified as same as in $SPHDHC_{t,h}$'s definition, i.e., Definition 4.*
2) *Smoothness. Intuitively speaking, it requires that for any instance $\ddot{x}\in L_{\ddot{R}_\Lambda}$, the hash value of $\ddot{x}$ is unobtainable unless its hash key is known. That is, 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., $Sm_1 \stackrel{c}{=} Sm_2$.*
   $Sm_1(1^k)$: $\Lambda\leftarrow PG(1^k)$, $(\ddot{x},\ddot{w})\leftarrow IS(1^k,\Lambda,1)$, $(hk,pk)\leftarrow KG(1^k,\Lambda,\ddot{x})$, $y\leftarrow Hash(1^k,\Lambda,\ddot{x},hk)$. Finally outputs $(\Lambda,\ddot{x},pk,y)$.
   $Sm_2(1^k)$: compared with $Sm_1(1^k)$, the only difference is that $y\in_U Range(Hash(1^k,\Lambda,\ddot{x},.))$.
3) *Hard Subset Membership. Intuitively speaking, it requires that the instances of $L_{\dot{R}_\Lambda}$ and that of $L_{\ddot{R}_\Lambda}$ are computationally indistinguishable. That is, the two probability ensembles $Hm_1 \stackrel{def}{=} \{Hm_1(1^k)\}_{k\in\mathbb{N}}$ and $Hm_2 \stackrel{def}{=} \{Hm_2(1^k)\}_{k\in\mathbb{N}}$ defined as follows, are computationally indistinguishable, i.e., $Hm_1 \stackrel{c}{=} Hm_2$.*
   $Hm_1(1^k)$: $\Lambda\leftarrow PG(1^k)$, $(\dot{x},\dot{w})\leftarrow IS(1^k,\Lambda,0)$, finally outputs $(\Lambda,\dot{x})$.
   $Hm_2(1^k)$: $\Lambda\leftarrow PG(1^k)$, $(\ddot{x},\ddot{w})\leftarrow IS(1^k,\Lambda,1)$, finally outputs $(\Lambda,\ddot{x})$.

It is easy to see that the projection and smoothness are two contradictory properties. That is, for any instance x, it holds at most one of the two. Therefore, $\dot{R}_\Lambda\cap\ddot{R}_\Lambda=\emptyset$.

Theorem 25 (reduce constructing $SPHDHC_{t,h}$ to constructing SPHDH)**.** *Given a SPHDH $\mathcal{H}$, then we can efficiently gain a $SPHDHC_{t,h}$ $\overline{\mathcal{H}}$.*

Proof: Let $\mathcal{H}=(PG,IS,DI,KG,Hash,pHash)$. First, we construct a new hash system $\overline{\mathcal{H}}=(\overline{PG},\overline{IS},\overline{DI},\overline{KG},\overline{Hash},\overline{pHash},\overline{Cheat})$ as follows.

- The procedures $\overline{PG}$, $\overline{DI}$, $\overline{KG}$, $\overline{Hash}$, $\overline{pHash}$ directly take the corresponding procedures from $\mathcal{H}$.
- $\overline{IS}(1^k,\Lambda)$: For each $i\in[h]$, $\vec{a}\langle i\rangle\leftarrow IS(1^k,\Lambda,0)$; for each $i\in[n]-[h]$, $\vec{a}\langle i\rangle\leftarrow IS(1^k,\Lambda,1)$; finally outputs $\vec{a}$.
- $\overline{Cheat}(1^k,\Lambda)$: For each $i\in[n]$, $\vec{a}\langle i\rangle\leftarrow IS(1^k,\Lambda,0)$; finally outputs $\vec{a}$.

Second, we prove $\overline{\mathcal{H}}$ is a $SPHDHC_{t,h}$. From the construction, we know that it remains to prove that $\overline{\mathcal{H}}$ holds properties smoothness, hard subset membership and feasible cheating. However, this fact directly follows Lemma 21, Lemma 23 and Lemma **??**. Therefore, $\overline{\mathcal{H}}$ is a $SPHDHC_{t,h}$. □

Sometimes it is not easy to gain smoothness for a hash family. In this case we have to construct a hash family, defined as follows, as the first step to our goal.

Definition 26 (ϵ-universal projective hash family that holds properties distinguishability and hard subset membership)**.** *$\mathcal{H}=(PG,IS,DI,KG,Hash,pHash)$ is a ϵ-universal projective hash family that holds properties distinguishability and hard subset membership (ϵ-UPHDH), if and only if $\mathcal{H}$ is specified as follows.*

- *All algorithms are specified as same as in $SPHDH$'s definition, i.e., Definition 24.*

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

1) *The properties projection, distinguishability and hard subset membership are specified as same as in Definition 24.*
2) *ϵ-universality. Intuitively speaking, it requires the probability of guessing the hash value of $\ddot{x}$ is at most ϵ. That is, for any sufficiently large k, any $\Lambda\in Range(PG(1^k))$, any $\ddot{x}\in Range(IS(1^k,\Lambda,1))$, any $pk\in Range(KG(1^k,\Lambda,\ddot{x})\langle 2\rangle)$, any $y\in Range(Hash(1^k,\Lambda,\ddot{x},.))$, it holds that*
   $$Pr(Hash(1^k,\Lambda,\ddot{x},HK)=y|PK=pk)\le\epsilon,$$
   where $(HK,PK)\leftarrow KG(1^k,\Lambda,\ddot{x})$, the probability is taken over the randomness of KG.

Compared with SPHDH, ϵ-UPHDH relaxes the upper bound of the probability of guessing the hash value of $\ddot{x}$ to a higher value. Assume $\epsilon<1$, as [15], [33], we can efficiently gain a SPHDH from a ϵ-UPHDH.

Theorem 27. *Given a ϵ-UPHDH $\widetilde{\mathcal{H}}$, where $\epsilon < 1$, then we can efficiently gain a SPHDH $\mathcal{H}$.*

The way to prove this theorem is to construct a required algorithm, which can be gained by a simply application of the Leftover Hash Lemma (please see [40] for this lemma). The detailed construction essentially is the same as [15]. Considering the space, we don't iterate it here.

Combining Theorem 25 and Theorem 27, we have the following corollary.

Corollary 28 (reduce constructing $SPHDHC_{t,h}$ to constructing ϵ-UPHDH)**.** *Given a ϵ-UPHDH $\widetilde{\mathcal{H}}$, then we can efficiently gain a $SPHDHC_{t,h}$ $\overline{\mathcal{H}}$.*

7 CONSTRUCTING $SPHDHC_{t,h}$

In this section, we construct $SPHDHC_{t,h}$ respectively under the lattice assumption, the decisional Diffie-Hellman assumption, the decisional N-th residuosity assumption and the decisional quadratic residuosity assumption. Theorem 25 and Corollary 28 show that, to construct a $SPHDHC_{t,h}$, what we need to do is to construct a SPHDH or construct a ϵ-UPHDH ($\epsilon < 1$).

7.1 A Construction Under The Decisional Diffie-Hellman Assumption

7.1.1 Background

Let $Gen(1^k)$ be an algorithm such that randomly chooses a cyclic group and outputs the group's description $G =< g, q, * >$, where g, q, $*$ respectively is the generator, the order, the operation of the group.

The DDH problem is how to construct an algorithm to distinguish the two probability ensembles $DDH_1 \stackrel{def}{=} \{DDH_1(1^k)\}_{k\in\mathbb{N}}$ and $DDH_2 \stackrel{def}{=} \{DDH_2(1^k)\}_{k\in\mathbb{N}}$ which are formulate as follows.

- $DDH_1(1^k)$: $< g, q, * >\leftarrow Gen(1^k)$, $a \in_U Z_q$, $b \in_U Z_q$, $c \leftarrow ab$, finally outputs $(< g, q, * >, g^a, g^b, g^c)$.
- $DDH_2(1^k)$: Basically operates in the same way as $DDH_1(1^k)$ except that $c \in_U Z_q$.

At present, there is no efficient algorithm solving the problem. Therefore, it is assumed that $DDH_1 \stackrel{c}{=} DDH_2$.

7.1.2 Detailed Construction

We now present our DDH-based instantiation of SPHDH as follows. For simplicity, we assume the groups generated by $Gen(1^k)$ is of prime order.

- $PG(1^k)$: $\Lambda \leftarrow Gen(1^k)$, finally outputs Λ.
- $IS(1^k, \Lambda, \delta)$: $(g, q, *) \leftarrow \Lambda$, $a \in_U Z_q$, $b \in_U Z_q$, $\dot{x} \leftarrow (g^a, g^b, g^{ab})$, $\dot{w} \leftarrow (a, b)$, $c \in_U Z_q$, $\ddot{x} \leftarrow (g^a, g^b, g^c)$, $\ddot{w} \leftarrow (a, b)$, finally outputs $(\dot{x}, \dot{w})$ if $\delta = 0$, $(\ddot{x}, \ddot{w})$ if $\delta = 1$.
- $DI(1^k, \Lambda, x, w)$: $(g, q, *) \leftarrow \Lambda$, $(\alpha, \beta, \gamma) \leftarrow x$, $(a, b) \leftarrow w$, if $(\alpha, \beta, \gamma) = (g^a, g^b, g^{ab})$ holds, then outputs 0; if $(\alpha, \beta) = (g^a, g^b)$ and $\gamma \neq g^{ab}$ holds, then outputs 1.
- $KG(1^k, \Lambda, x)$: $(g, q, *) \leftarrow \Lambda$, $(\alpha, \beta, \gamma) \leftarrow x$, $u \in_U Z_q$, $v \in_U Z_q$, $pk \leftarrow \alpha^u g^v$, $hk \leftarrow \gamma^u \beta^v$, finally outputs (hk, pk).
- $Hash(1^k, \Lambda, x, hk)$: $y \leftarrow hk$, outputs y.
- $pHash(1^k, \Lambda, x, pk, w)$: $(a, b) \leftarrow w$, $y \leftarrow pk^b$, finally outputs y.

Lemma 29. *The hash system holds the property projection.*

Proof: Let $(\dot{x}, \dot{w}) \in Range(IS(1^k, \Lambda, 0))$. Let $(hk, pk) \in Range(KG(1^k, \Lambda, \dot{x}))$. Then,

$$\begin{aligned} Hash(1^k, \Lambda, \dot{x}, hk) &= Hash(1^k, \Lambda, (g^a, g^b, g^{ab}), (g^{abu} g^{bv})) \\ &= g^{abu} g^{bv} \end{aligned}$$

$$\begin{aligned} pHash(1^k, \Lambda, \dot{x}, hk, \dot{w}) &= pHash(1^k, \Lambda, (g^a, g^b, g^{ab}), \\ &\quad (g^{au} g^v), (a, b)) \\ &= g^{abu} g^{bv} \end{aligned}$$

That is,

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

□

Lemma 30. *Assuming DDH is a hard problem, the hash system holds the property smoothness.*

Proof: For this system, the probability ensembles Sm_1, Sm_2 mentioned in the definition of SPHDH can be described as follows.

- $Sm_1(1^k)$: $\Lambda \leftarrow PG(1^k)$, $(g, q, *) \leftarrow \Lambda$, $a \in_U Z_q$, $b \in_U Z_q$, $c \in_U Z_q$, $\ddot{x} \leftarrow (g^a, g^b, g^c)$, $u \in_U Z_q$, $v \in_U Z_q$, $pk \leftarrow g^{au+v}$, $hk \leftarrow g^{cu+bv}$, $y \leftarrow hk$. Finally outputs $(\Lambda, \ddot{x}, pk, y)$.
- $Sm_2(1^k)$: Operates as same as $Sm_1(1^k)$ with an exception that y is generated as follows. $d \in_U Z_q$, $y \leftarrow g^d$.

Because b, c, u, v are chosen uniformly and q is prime, both cu and bv are uniformly distributed over Z_q. Thus $cu + bv$ is uniformly distributed over Z_q too. Therefore, $Sm_1 \equiv Sm_2$. □

Lemma 31. *The hash system holds the property distinguishability.*

The proof of this lemma is trivial, so we omit it.

Lemma 32. *Assuming DDH is a hard problem, the hash system holds the property hard subset membership.*

Proof: For this system, the probability ensembles Hm_1, Hm_2 mentioned in the definition of SPHDH can be described as follows.

- $Hm_1(1^k)$: $\Lambda \leftarrow PG(1^k)$, $(g, q, *) \leftarrow \Lambda$, $a \in_U Z_q$, $b \in_U Z_q$, $\dot{x} \leftarrow (g^a, g^b, g^{ab})$. Finally outputs $(\Lambda, \dot{x})$.
- $Hm_2(1^k)$: $\Lambda \leftarrow PG(1^k)$, $(g, q, *) \leftarrow \Lambda$, $a \in_U Z_q$, $b \in_U Z_q$, $c \in_U Z_q$, $\ddot{x} \leftarrow (g^a, g^b, g^c)$. Finally outputs $(\Lambda, \ddot{x})$.

Obviously, $Hm_1 \stackrel{c}{=} Hm_2$. □

Combining all lemmas above, we have the following theorem.

Theorem 33. *Assuming DDH is a hard problem, the hash system is a SPHDH.*

7.1.3 A Concrete Protocol For OT_h^n Based On DDH

It's known that the encryption scheme presented by [17] can be used as a perfectly binding commitment scheme. The encryption scheme is directly based on the problem of discrete log. Since the task of solving the problem DDH can be reduced to that of solving the problem discrete log, the encryption scheme is based on DDH essentially. The DDH-based commitment scheme presented by [50] is a perfectly hiding one. Therefore, using those two commitment schemes and our DDH-based $SPHDHC_{t,h}$, we gain a concrete protocol for OT_h^n based only on DDH. To reach the best efficiency, we should use the DDH of the group which is on elliptic curves. See Section 4.5 for further discussion.

7.2 A Construction Under Lattice

7.2.1 Background

Learning with errors (LWE) is an average-case problem. [53] shows that its hardness is implied by the worst-case hardness of standard lattice problem for quantum algorithms.

In lattice, the modulo operation is defined as $x \mod y \stackrel{def}{=} x - \llcorner x/y \lrcorner y$. Then we know $x \mod 1 \stackrel{def}{=} x - \llcorner x \lrcorner$. Let β be an arbitrary positive real number. Let Ψ_β be a probability density function whose distribution is over $[0,1)$ and obtained by sampling from a normal variable with mean 0 and standard deviation $\beta/\sqrt{2\pi}$ and reducing the result modulo 1, more specifically

$$\Psi_\beta : [0,1) \to R^+$$
$$\Psi_\beta(r) \stackrel{def}{=} \sum_{k=-\infty}^{\infty} \frac{1}{\beta} \exp(-\pi(\frac{r-k}{\beta})^2)$$

Given an arbitrary integer $q \geq 2$, an arbitrary probability destiny function $\phi : [0,1) \to R^+$, the discretization of ϕ over Z_q is defined as

$$\bar{\phi} : Z_q \to R^+$$
$$\bar{\phi}(i) \stackrel{def}{=} \int_{(i-1/2)/q}^{(i+1/2)/q} \phi(x)dx$$

LWE can be formulated as follows.

Definition 34 (Learning With Errors)**.** *Learning with errors problem ($LWE_{q,\chi}$) is how to construct an efficient algorithm that receiving $q, g, m, \chi, (\vec{a}_i, b_i)_{i\in[m]}$, outputs $\vec{s}$ with nonnegligible probability. The input and the output is specified in the following way.*

$q \leftarrow q(1^k)$, $g \leftarrow g(1^k)$, $m \leftarrow poly(1^k)$, $\chi \leftarrow \chi(1^k)$, $\vec{s} \in_U (Z_q)^k$. For each $i \in [m]$, $\vec{a}_i \in_U (Z_q)^k$, $e_i \in_\chi Z_q$, $b_i \leftarrow \vec{s}^T \cdot \vec{a}_i + e_i \mod q$.

where q, g are positive integers, $\chi : Z_q \to R^+$ is a probability density function.

With respect to the hardness of LWE, [53] proves that setting appropriate parameters, we can reduce two worst-case standard lattice problems to LWE, which means LWE is a very hard problem.

Lemma 35 ([53])**.** *Setting security parameter k to be a value such that q is a prime, $\beta \leftarrow \beta(1^k)$, $\beta \in (0,1)$, and $\beta \cdot q > 2\sqrt{k}$. Then the lattice problems $SIVP$ and $GapSVP$ can be reduced to $LWE_{q,\bar{\Psi}_\beta}$. More specifically, if there exists an efficient (possibly quantum) algorithm that solves $LWE_{q,\bar{\Psi}_\beta}$, then there exists an efficient quantum algorithm solving the following worst-case lattice problems in the l_2 norm.*

- *SIVP: In any lattice Λ of dimension k, find a set of k linearly independent lattice vectors of length within at most $\tilde{O}(k/\beta)$ of optimal.*
- *GapSVP: In any lattice Λ of dimension m, approximate the length of a shortest nonzero lattice vector to within a $\tilde{O}(k/\beta)$ factor.*

We emphasize the fact that the reduction of Lemma 35 is quantum, which implies that any algorithm breaking any cryptographic schemes which only based on LWE is an algorithm solving at least one of the problems SIVP and GapSVP.

How to precisely set the parameters as values to gain a concrete LWE, which is as hard as required in Lemma 35 is beyond the scope of this paper. To see such examples and more details, we recommend [53] and [51].

The instantiation of *SPHDH*, which we will present soon, needs to use a LWE-based public key cryptosystem presented by [23], which is a slight variant of [53]'s cryptosystem. This cryptosystem is described as follow.

- Message space: $\{0,1\}$.
- $Setup(1^k)$: $q \leftarrow q(1^k) \wedge q \in \mathbb{P} \wedge q \in [k^2, 2k^2]$, $m \leftarrow (1+\varepsilon)(k+1)\log q$ (where $\varepsilon > 0$ is an arbitrary constant), $\chi \leftarrow \bar{\Psi}_{\alpha(k)} \wedge \alpha(k) = o(1/(\sqrt{k}\log k))$ (e.g., $\alpha(k) = \frac{1}{\sqrt{k}(\log k)^2}$). $para \leftarrow (q, m, \chi)$, finally outputs $para$.
- $KeyGen(1^k, para)$: $A \in_U (Z_q)^{m\times k}$, $\vec{s} \in_U (Z_q)^k$, $\vec{e} \in_\chi (Z_q)^m$ (which means each entry of $\vec{e}$ is independently drawn from Z_q according to χ), $\vec{b} \leftarrow A\vec{s} + \vec{e} \mod q$, $pubk \leftarrow (A, \vec{b})$, $sk \leftarrow \vec{s}$, finally outputs a public-private key pair $(pubk, sk)$.
- $Enc(.)$, $Dec(.)$: Since $Enc(.)$, $Dec(.)$ are immaterial to understand this paper, we omit their detailed procedure here.

[23] shows that if $LWE_{q,\bar{\Psi}_\alpha}$ is hard, choosing appropriate parameters, this cryptosystem holds the following properties.

1) It provides security against chosen plaintext attack, though we only need semantic security here.
2) For each $A \in (Z_q)^{m\times k}$, we have

$$Pr(\vec{b} \text{ is messy} | \vec{b} \in_U (Z_q)^m) \geq 1 - 2/q^k,$$

where $\vec{b}$ is said to be messy if and only if, $\forall m_0, m_1 \in \{0,1\}$, the statistical distance between the distribution of $Enc_{A,\vec{b}}(m_0)$ and that of $Enc_{A,\vec{b}}(m_1)$ is negligible. In other word, $\vec{b}$ is said to be messy if and only if, $Enc_{A,\vec{b}}(.)$ loses messages and so its ciphertext can't be decrypted using any private key $\vec{s} \in (Z_q)^k$.

3) Given $A \in (Z_q)^{m\times k}$ and its trapdoor T, then there exists an efficient decision algorithm $IsMessy$ holds the following two property. First, $Pr(IsMessy(A,T,\vec{b})=0|\vec{b}\in_U (Z_q)^m)$ is negligible. Second, $\vec{b}$ is indeed messy if $IsMessy(A,T,\vec{b})=1$.

7.2.2 Detailed Construction

We now present our LWE-based instantiation of *SPHDH* as follows.

- $PG(1^k)$: $\Lambda \leftarrow Setup(1^k)$, finally outputs Λ.
- $IS(1^k,\Lambda,b)$: $(q,m,\chi)\leftarrow\Lambda$, $A\in_U (Z_q)^{m\times k}$ along with its trapdoor T, $\vec{s}\in_U (Z_q)^k$, $\vec{e}\in_\chi (Z_q)^m$, $\dot{x}\leftarrow (A, A\vec{s}+\vec{e} \mod q)$, $\dot{w}\leftarrow (0,\vec{s})$, uniformly chooses $\vec{b}\in (Z_q)^m$ such that $IsMessy(A,T,\vec{b})=1$ (recall that only negligible fraction of $\vec{b}$ are not messy, therefore such $\vec{b}$ can be efficiently chosen) , $\ddot{x}\leftarrow (A,\vec{b})$, $\ddot{w}\leftarrow (1,T)$, finally outputs $(\dot{x},\dot{w})$ if $b=0$, $(\ddot{x},\ddot{w})$ if $b=1$.
- $DI(1^k,\Lambda,x,w)$: $(q,m,\chi)\leftarrow\Lambda$, $(A,\vec{b})\leftarrow x$, $(i,\varrho)\leftarrow w$, if $i=1$ and $IsMessy(A,\varrho,\vec{b})=1$ holds, then outputs 1; otherwise outputs 0.
- $KG(1^k,\Lambda,x)$: $(q,m,\chi)\leftarrow\Lambda$, $(A,\vec{b})\leftarrow x$, $a\in_U\{0,1\}$, $\alpha\leftarrow Enc_{A,\vec{b}}(a)$, $hk\leftarrow a$, $pk\leftarrow\alpha$, finally outputs (hk,pk).
- $Hash(1^k,\Lambda,x,hk)$: $(q,m,\chi)\leftarrow\Lambda$, $a\leftarrow hk$, finally outputs a.
- $pHash(1^k,\Lambda,x,pk,w)$: $(m,q,\chi)\leftarrow\Lambda$, $\alpha\leftarrow pk$, $(i,\varrho)\leftarrow w$, $a\leftarrow Dec_\varrho(\alpha)$, finally outputs a.

In the above construction of SPHDH, each instance holds a matrix A, which seems expensive. However, in the corresponding construction of $SPHDHC_{t,h}$, this overhead can be reduced by each instance vector sharing a matrix A. We point out that it's not secure that all instance vectors share a matrix A. The reason is that in this case, seeing matrix A's trapdoor T in Step S2 of the framework, P_1 can distinguish smooth instances and projective instances of the unchosen instance vectors, which leads to P_1 deducing P_2's private input.

Lemma 36. *Assuming LWE is a hard problem, the hash system holds the property projection.*

Proof: Let $\dot{x}=(A,\vec{b})\in Range(IS(1^k,\Lambda,0))$, $\dot{w}=(0,\vec{s})$. Obviously, $((A,\vec{b}),\vec{s})$ is a correct public-private key pair. Then, we have

$$\begin{aligned}Hash(1^k,\Lambda,\dot{x},hk)&=a,\\ pHash(1^k,\Lambda,\dot{x},pk,\dot{w})&=Dec_{\vec{s}}(\alpha)\\ &=Dec_{\vec{s}}(Enc_{A,\vec{b}}(a))\\ &=a,\end{aligned}$$

This means that for any $(\dot{x},\dot{w},\Lambda)$ generated by the hash system, it holds that

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

□

Lemma 37. *The hash system holds the property smoothness.*

Proof: For this system, the probability ensembles Sm_1, Sm_2 mentioned in the definition of SPHDH can be described as follows.

- $Sm_1(1^k)$: $\Lambda\leftarrow PG(1^k)$, $(q,m,\chi)\leftarrow\Lambda$, $A\in_U (Z_q)^{m\times k}$ along with its trapdoor T, uniformly chooses $\vec{b}\in (Z_q)^m$ such that $IsMessy(A,T,\vec{b})=1$, $\ddot{x}\leftarrow (A,\vec{b})$, $\ddot{w}\leftarrow (1,T)$, $a\in_U\{0,1\}$, $\alpha\leftarrow Enc_{A,\vec{b}}(a)$, $pk\leftarrow\alpha$, $y\leftarrow a$, finally outputs $(\Lambda,\ddot{x},pk,y)$.
- $Sm_2(1^k)$: Operates as same as $Sm_1(1^k)$ with an exception that $y\in_U\{0,1\}$.

Obviously, $Sm_1(1^k)\equiv Sm_2(1^k)$. □

Lemma 38. *Assuming LWE is a hard problem, the hash system holds the property distinguishability.*

Proof: Recalling the property of $IsMessy$, we know if $(A,\vec{b})$ isn't messy, $IsMessy(A,T,\vec{b})=0$; if $(A,\vec{b})$ is messy, $IsMessy(A,T,\vec{b})=1$ with a probability close to 1. Thus, if $(x,w)\in\ddot{R}_\Lambda$, DI outputs 1; if $(x,w)\in\dot{R}_\Lambda$, DI outputs 0. DI correctly computes ζ. □

Lemma 39. *Assuming LWE is a hard problem, the hash system holds the property hard subset membership.*

Proof: For this system, the probability ensembles Hm_1, Hm_2 mentioned in the definition of SPHDH can be described as follows.

- $Hm_1(1^k)$: $\Lambda\leftarrow PG(1^k)$, $(q,m,\chi)\leftarrow\Lambda$, $A\in_U (Z_q)^{m\times k}$ along with its trapdoor T, $\vec{s}\in_U (Z_q)^k$, $\vec{e}\in_\chi (Z_q)^m$, $\dot{x}\leftarrow (A,A\vec{s}+\vec{e} \mod q)$, finally outputs $(\Lambda,\dot{x})$.
- $Hm_2(1^k)$: $\Lambda\leftarrow PG(1^k)$, $(q,m,\chi)\leftarrow\Lambda$, $A\in_U (Z_q)^{m\times k}$ along with its trapdoor T, uniformly chooses $\vec{b}\in (Z_q)^m$ such that $IsMessy(A,T,\vec{b})=1$, $\ddot{x}\leftarrow (A,\vec{b})$, finally outputs $(\Lambda,\ddot{x})$.

Obviously, $Hm_1\stackrel{c}{=}Hm_2$. □

Combining Lemma 35 and above lemmas, we have the following theorem.

Theorem 40. *If $SIVP$ or $GapSVP$ is a hard problem, the hash system is a SPHDH.*

7.2.3 A Concrete Protocol For OT^n_h With Security Against Quantum Algorithms

The security proof of the framework guarantees that, any adversary breaking the framework is an algorithm breaking at least one of cryptographic tools used in the framework. Moreover, it's generally believed that lattice-based cryptography resists quantum attacks [43]. Therefore, to gain an instantiation of our framework with security against quantum algorithms, it suffices to adopt lattice-based instantiations of the cryptographic tools. Thus, it remains to find a IHC and a IBC of such security level.

First, we can take the LWE-based perfectly binding commitment scheme presented by [60] as an instantiation of IBC. In fact, there are some other good instantiations. For instance, we can obtain a statistically binding commitment scheme by combining the work of [19], which proposes a simple and efficient construction of a pseudo-randomness generator based on the intractability of an $\mathcal{NP}$-complete problem from the area of error-correcting codes, and the work of [48], which proposes a general way to obtain statistically binding commitment schemes from any pseudo-randomness generator. We also can use McEliece's public-key cryptosystem [41] as a statistically binding commitment scheme. The reason why we take [60]'s scheme is because it's based on lattice problems too.

Second, combining the works of [2], [26], [30], [42], we can get a relatively efficient statically hiding commitment. [30] presents a efficient way to construct statically hiding commitments from any collision-free hash. Assuming one of the lattice problems $SIVP$ and SVP is hard, [26] shows that [2]'s lattice-based hash of suitably chosen parameters is collision-free. Under the assumption that $GapSVP^2_{\tilde{O}(n)}$ is hard in the worst case, [42] later also shows that the lattice-based hash is collision-free. Therefore, applying [30]'s method to [2]'s hash, we get a lattice-based statically hiding commitment.

Now we can gain a concrete protocol for OT^n_h with security against quantum algorithms, this is summarized by the following theorem.

Theorem 41. *Assuming that one of the lattice problems $SIVP$ and $GapSVP$ is hard for quantum algorithms, instantiating the OT^n_h framework with the lattice-based $SPHDHC_{t,h}$, and the lattice-based commitment schemes we suggests above, the resulting concrete protocol for OT^n_h is secure against quantum algorithms.*

7.3 A Construction Under The Decisional N-th Residuosity Assumption

7.3.1 Verifiable-ϵ-universal Projective Hash Family

In this section, we will build a instantiation of ϵ-UPHDH ($\epsilon < 1$) from a instantiation of a hash system called verifiable-ϵ-universal projective hash family by [33]. Therefore, it is necessary to introduce the definition of this hash system.

Definition 42 (verifiable-ϵ-universal projective hash family, [33])**.** *$\mathcal{H} = (PG, IS, IT, KG, Hash, pHash)$ is a ϵ-universal projective hash family (ϵ-VUPH), if and only if $\mathcal{H}$ is specified as follows.*

- *The algorithms PG, IS, KG, $Hash$, $pHash$ are specified as same as in ϵ-UPHDH's definition, i.e., Definition 26.*
- *IS is a PPT algorithm that takes a security parameter k, a family parameter Λ as input and outputs a tuple, i.e., $(\dot{w}, \dot{x}, \ddot{x}) \leftarrow IS(1^k, \Lambda)$.*
- *IT is a PPT algorithm that takes a security parameter k, a family parameter Λ, two instances as input and outputs a bit , i.e., $b \leftarrow IT(1^k, \Lambda, x_1, x_2)$.*

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

1) *The properties projection, ϵ-universality are specified as same as that in ϵ-UPHDH's definition, i.e., Definition 26.*
2) *Verifiability. First, for any sufficiently large k, any $\Lambda \in Range(PG(1^k))$, any $(\dot{w}, \dot{x}, \ddot{x}) \in Range(IS(1^k, \Lambda))$, it holds that $IT(1^k, \Lambda, \dot{x}, \ddot{x}) = IT(1^k, \Lambda, \ddot{x}, \dot{x}) = 1$. Second, for any sufficiently large k, any (Λ, x_1, x_2) such that $IT(1^k, \Lambda, x_1, x_2) = 1$, at least one of x_1, x_2 is ϵ-universal.*

It is easy to see that verifiability guarantees any instance x holds at most one of the properties projection and universality. Therefore, we have the following lemma.

Lemma 43. *Let $\mathcal{H} = (PG, IS, IT, KG, Hash, pHash)$ be a ϵ-universal projective hash family, then*

$$\dot{L} \cap \ddot{L} = \emptyset$$

where $\dot{L} \stackrel{def}{=} \{\dot{x} | \Lambda \leftarrow PG(1^k), (\dot{w}, \dot{x}, \ddot{x}) \leftarrow IS(1^k, \Lambda)\}$ and $\ddot{L} \stackrel{def}{=} \{\ddot{x} | \Lambda \leftarrow PG(1^k), (\dot{w}, \dot{x}, \ddot{x}) \leftarrow IS(1^k, \Lambda)\}$.

7.3.2 Background

Let $Gen(1^k)$ be an algorithm that operates as follows.

- $Gen(1^k)$: $(p, q) \in_U \{(p, q) | (p, q) \in (\mathbb{P}, \mathbb{P}), p, q > 2, |p| = |q| = k, \gcd(pq, (p-1)(q-1)) = 1\}$, $N \leftarrow pq$, finally outputs N.

The problem decisional N-th residuosity (DNR), first presented by [49], is how to construct an algorithm to distinguish two probability ensembles $DNR_1 \stackrel{def}{=} \{DNR_1(1^k)\}_{k \in \mathbb{N}}$ and $DNR_2 \stackrel{def}{=} \{DNR_2(1^k)\}_{k \in \mathbb{N}}$ which are formulate as follows.

- $DNR_1(1^k)$: $N \leftarrow Gen(1^k)$, $a \in_U Z^*_{N^2}$, $b \leftarrow a^N \mod N^2$, finally outputs (N, b).
- $DNR_2(1^k)$: $N \leftarrow Gen(1^k)$, $b \in_U Z^*_{N^2}$, finally outputs (N, b).

The DNR assumption is that there is no efficient algorithm solving the problem. In other words, it is assumed that $DNR_1 \stackrel{c}{=} DNR_2$.

Our instantiation of ϵ-UPHDH is build from a DNR-based instantiation of ε-VUPH ($\varepsilon < 1$) presented by [33]. The instantiation of ε-VUPH is stated as follows.

- $PG(1^k)$: $N \leftarrow Gen(1^k)$, $a \in_U Z^*_{N^2}$, $T \leftarrow N^{\ulcorner 2 \log N \urcorner}$, $g \leftarrow a^{N \cdot T} \mod N^2$, $\Lambda \leftarrow (N, g)$, finally outputs Λ.
- $IS(1^k, \Lambda)$: $(N, g) \leftarrow \Lambda$, $r, v \in_U Z^*_N$, $w \leftarrow r$, $\dot{x} \leftarrow g^r \mod N^2$, $\ddot{x} \leftarrow \dot{x}(1 + vN) \mod N^2$, finally outputs $(w, \dot{x}, \ddot{x})$.
- $IT(1^k, \Lambda, \dot{x}, \ddot{x})$: $(N, g) \leftarrow \Lambda$. Checks that $N > 2^{2k}$, $g, \dot{x} \in Z^*_{N^2}$. $d \leftarrow \ddot{x}/\dot{x} \mod N^2$ and checks $N | (d-1)$. $v \leftarrow (d-1)/N$ and checks $\gcd(v, N) = 1$. Outputs 1 if all the test pass and 0 otherwise.
- $KG(1^k, \Lambda)$: $(N, g) \leftarrow \Lambda$, $hk \in_U Z_{N^2}$, $pk \leftarrow g^{hk} \mod N^2$, finally outputs (hk, pk).
- $Hash(1^k, \Lambda, x, hk)$: $(N, g) \leftarrow \Lambda$, $y \leftarrow x^{hk} \mod N^2$, finally outputs y.
- $pHash(1^k, \Lambda, x, pk, w)$: $(N, g) \leftarrow \Lambda$, $y \leftarrow pk^w \mod N^2$, finally outputs y.

7.3.3 Detailed Construction

We now present our DNR-based instantiation of ϵ-UPHDH ($\epsilon < 1$) as follows.

- $PG(1^k)$: $N \leftarrow Gen(1^k)$, $a \in_U Z^*_{N^2}$, $T \leftarrow N^{\ulcorner 2\log N \urcorner}$, $g \leftarrow a^{N\cdot T} \mod N^2$, $\Lambda \leftarrow (N, g)$, finally outputs Λ.
- $IS(1^k, \Lambda, \delta)$: $(N, g) \leftarrow \Lambda$, $r \in_U Z^*_N$, $\dot{x} \leftarrow g^r \mod N^2$, $\dot{w} \leftarrow (r, 0)$, $v \in_U Z^*_N$, $\ddot{x} \leftarrow g^r(1 + vN) \mod N^2$, $\ddot{w} \leftarrow (r, v)$, finally outputs $(\dot{x}, \dot{w})$ if $\delta = 0$, $(\ddot{x}, \ddot{w})$ if $\delta = 1$.
- $DI(1^k, \Lambda, x, w)$: $(N, g) \leftarrow \Lambda$, $(r, v) \leftarrow w$,
  1) if $v = 0 \mod N$, operates as follows: checks that $N > 2^{2k}$, $g, x \in Z^*_{N^2}$, $r \in Z^*_N$, $x = g^r \mod N^2$. Outputs 0 if all the test pass.
  2) if $v \neq 0 \mod N$, operates as follows: checks that $N > 2^{2k}$, $g, x \in Z^*_{N^2}$, $r \in Z^*_N$, $x = g^r(1+vn) \mod N^2$. Outputs 1 if all the test pass.
- $KG(1^k, \Lambda, x)$: $(N, g) \leftarrow \Lambda$, $hk \in_U Z_{N^2}$, $pk \leftarrow g^{hk} \mod N^2$, finally outputs (hk, pk).
- $Hash(1^k, \Lambda, x, hk)$: $(N, g) \leftarrow \Lambda$, $y \leftarrow x^{hk} \mod N^2$, finally outputs y.
- $pHash(1^k, \Lambda, x, pk, w)$: $(N, g) \leftarrow \Lambda$, $y \leftarrow pk^w \mod N^2$, finally outputs y.

Theorem 44. *Assuming DNR is a hard problem, the hash system is a ϵ-UPHDH ($\epsilon < 1$).*

Proof: It is easy to see that the hash system directly inherits properties ε-universality and projection from the instantiation of ϵ-VUPH. Following Lemma 43, the hash system holds property distinguishability. It remains to prove that the hash system holds the property hard subset membership.

For this system, the probability ensembles Hm_1, Hm_2 mentioned in the definition of ϵ-UPHDH can be described as follows.

- $Hm_1(1^k)$: $\Lambda \leftarrow PG(1^k)$, $(N, g) \leftarrow \Lambda$, $r \in_U Z^*_N$, $\dot{x} \leftarrow g^r \mod N^2$. Finally outputs $(\Lambda, \dot{x})$.
- $Hm_2(1^k)$: $\Lambda \leftarrow PG(1^k)$, $(N, g) \leftarrow \Lambda$, $r, v \in_U Z^*_N$, $\ddot{x} \leftarrow g^r(1 + vN) \mod N^2$. Finally outputs $(\Lambda, \ddot{x})$.

It is clear that $Hm_1 \stackrel{c}{=} Hm_2$. Therefore, the hash system holds the property hard subset membership. □

7.4 A Construction Under The Decisional Quadratic Residuosity Assumption

We reuse $Gen(1^k)$ defined in section 7.3.2. Let J_N be the subgroup of Z^*_N of elements with Jacobi symbol 1. The problem decisional quadratic residuosity (DQR) is how to construct an algorithm to distinguish the two probability ensembles $DQR_1 \stackrel{def}{=} \{DQR_1(1^k)\}_{k\in\mathbb{N}}$ and $DQR_2 \stackrel{def}{=} \{DQR_2(1^k)\}_{k\in\mathbb{N}}$ which are formulated as follows.

- $DQR_1(1^k)$: $N \leftarrow Gen(1^k)$, $x \in_U J_N$, finally outputs (N, x).
- $DQR_2(1^k)$: $N \leftarrow Gen(1^k)$, $r \in_U Z^*_N$, $x \leftarrow r^2 \mod N$, finally outputs (N, x).

The DQR assumption is that there is no efficient algorithm solving the problem. That is, it is assumed that $DQR_1 \stackrel{c}{=} DQR_2$.

As in section 7.3, the hash system we aim to achieve is an instantiation of ϵ-UPHDH. We will build it on an instantiation of ε-VUPH presented by [33] which is constructed under DQR assumption. Considering the space, we do not iterate the instantiation of ε-VUPH here, and directly present our instantiation of ϵ-UPHDH as follows.

- $PG(1^k)$: $(p, q) \in_U (\mathbb{P}, \mathbb{P})$, where $|p| = |q| = k$, $p < q < 2p-1$, $p = q = 3 \mod 4$, $a \in_U Z^*_N$, $T \leftarrow 2^{\ulcorner \log N \urcorner}$, $g \leftarrow a^{2\cdot T} \mod N$, $\Lambda \leftarrow (N, g)$, finally outputs Λ.
- $IS(1^k, \Lambda, \delta)$: $(N, g) \leftarrow \Lambda$, $r \in_U Z_N$, $\dot{x} \leftarrow g^r \mod N$, $\ddot{x} \leftarrow N - g^r \mod N$, $\ddot{w} \leftarrow r$, finally outputs $(\dot{x}, \dot{w})$ if $\delta = 0$, $(\ddot{x}, \ddot{w})$ if $\delta = 1$.
- $DI(1^k, \Lambda, x, w)$: $(N, g) \leftarrow \Lambda$, $r \leftarrow w$; checks that $N > 2^{2k}$, $g, x \in Z^*_N$. Outputs 0, if $x = g^r \mod N$ and all the test pass. Outputs 1, if $x = N - g^r \mod N$ and all the test pass.
- $KG(1^k, \Lambda, x)$: $(N, g) \leftarrow \Lambda$, $hk \in_U Z_N$, $pk \leftarrow g^{hk} \mod N$, finally outputs (hk, pk).
- $Hash(1^k, \Lambda, x, hk)$: $(N, g) \leftarrow \Lambda$, $y \leftarrow x^{hk} \mod N$, finally outputs y.
- $pHash(1^k, \Lambda, x, pk, w)$: $(N, g) \leftarrow \Lambda$, $y \leftarrow pk^w \mod N$, finally outputs y.

Theorem 45. *Assuming DQR is a hard problem, the hash system is a ϵ-UPHDH, where $\epsilon < 1$.*

This theorem can be proven in a similar way in which Theorem 44 is proven.

References

[1] 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.

[2] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 99–108. ACM, 1996.

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

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

[5] 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.

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

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

[8] R. Canetti, I. Damgard, S. Dziembowski, Y. Ishai, and T. Malkin. Adaptive versus non-adaptive security of multi-party protocols. *Journal of Cryptology*, 17(3):153–207, 2004.

[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] J.L. Carter and M.N. Wegman. Universal classes of hash functions. *Journal of computer and system sciences*, 18(2):143–154, 1979.

[13] J. Cheon. Security analysis of the strong Diffie-Hellman problem. *Advances in Cryptology-EUROCRYPT 2006*, pages 1–11, 2006.

[14] 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.

[15] 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.

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

[17] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions On Information Theory*, 31(4):469–472, 1985.

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

[19] Jean-Bernard Fischer and Jacques Stern. *An Efficient Pseudo-Random Generator Provably as Secure as Syndrome Decoding*, volume 1070 of *Lecture Notes in Computer Science*, pages 245–255. Springer Berlin / Heidelberg, 1996.

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

[21] 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.

[22] 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.

[23] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Stoc'08: Proceedings of the 2008 Acm International Symposium on Theory of Computing*, pages 197–206 798, 2008. full paper available on http://eprint.iacr.org/2007/432.

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

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

[26] O. Goldreich, S. Goldwasser, and S. Halevi. Collision-free hashing from lattice problems. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 3, 1996.

[27] O. Goldreich and A. Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(3):167–189, 1996.

[28] 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.

[29] 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.

[30] Shai Halevi and Silvio Micali. Practical and provably-secure commitment schemes from collision-free hashing. In Neal Koblitz, editor, *Advances in Cryptology ł CRYPTO 96*, volume 1109 of *Lecture Notes in Computer Science*, pages 201–215. Springer Berlin / Heidelberg, 1996.

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

[32] 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.

[33] 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. .

[34] J. Katz and Y. Lindell. Handling expected polynomial-time strategies in simulation-based security proofs. *Journal of Cryptology*, 21(3):303–349, 2008.

[35] 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.

[36] 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.

[37] 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.

[38] Y. Lindell and B. Pinkas. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In *Advances in Cryptology-Eurocrypt'2007*, pages 52–78. Springer-Verlag, 2007.

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

[40] M.G. Luby and M. Luby. *Pseudorandomness and cryptographic applications*. Princeton University Press, 1996.

[41] R.J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN progress report*, 42(44):114–116, 1978.

[42] D. Micciancio and O. Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM Journal on Computing*, 37(1):267–302, 2008.

[43] Daniele Micciancio and Oded Regev. *Lattice-based Cryptography*, pages 147–191. Springer Berlin Heidelberg, 2009.

[44] 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.

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

[46] 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.

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

[48] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.

[49] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *Advances in Cryptology-Eurocrypt'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, 1999.

[50] T.P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology-Crypto'1991*, volume 91, pages 129–140. Springer, 1991.

[51] 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.

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

[53] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

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

[55] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[56] P.W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Annual Symposium On Foundations Of Computer Science*, volume 35, pages 124–124. Citeseer, 1994.

[57] P.W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[58] M.N. Wegman and L. Carter. New hash functions and their use in authentication and set equality. *Journal of computer and system sciences*, 22(3):265–279, 1981.

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

[60] Bing Zeng, Liang Chen, and Xueming Tang. A perfectly binding commitment scheme against quantum attacks. Cryptology ePrint Archive, Report 2011/223, 2011. http://eprint.iacr.org/.