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

Zeng Bing [*]
College of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan City, Hubei, China
Email: zeng. bing. zb@gmail. com

April 17, 2010

[*]This work was supported by the National Natural Science Foundation of China (Grants No.60703048 and 60903175) and the Natural Science Foundation of Hubei Province (Grant No.2008CDB352).

Abstract

In this paper, we present a framework for efficient, fully-simulatable h-out-of-n oblivious transfer (OT_h^n) with security against nonadaptive malicious adversary. The number of communication round of the framework is six. Compared with existing fully-simulatable OT_h^n, our framework is round-efficient. Conditioning on no trusted common string is available, our DDH-based instantiation is the most efficient protocol for OT_h^n.

Our framework uses three abstract tools, i.e. perfectly binding commitment, perfectly 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, decisional Diffie-Hellman, decisional N-th residuosity, decisional quadratic residuosity assumptions. 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 instantiation and Brassard's commitment scheme, we gain a OT_h^n instantiation which is secure against any quantum algorithm.

Contents

1 Introduction **4**
1.1 Oblivious transfer 4
1.2 Our contribution 6
1.3 Our Approach 8
1.4 Organization 10

2 Preliminaries **10**
2.1 Basic Notations 11
2.2 Security Definition 12
2.3 Commitment Scheme 16

3 A New Smooth Projective Hash — $SPWH_{h,t}$ **17**

4 Constructing A Framework For Fully-simulatable OT_h^n **22**

5 A Security Proof Of The Framework **27**
5.1 In Case P_1 Is Corrupted 27
5.2 In Case P_2 Is Corrupted 31
5.3 Other Cases 35

6 Constructing $SPWH_{h,t}$ **36**
6.1 How To Make Constructing $SPWH_{h,t}$ Easier 36
6.1.1 Smoothness 36
6.1.2 Hard Subset Membership 38
6.2 A Construction Under Lattice 45
6.2.1 Background 45
6.2.2 Detailed Construction 47
6.2.3 An Instantiation of OT_h^n Against Any Quantum Algorithm 49
6.3 A Construction Under The Decisional Diffie-Hellman Assumption 50
6.3.1 Background 50
6.3.2 Detailed Construction 50
6.4 A Construction Under The Decisional N-th Residuosity Assumption 51
6.4.1 Background 51
6.4.2 Detailed Construction 53
6.5 A Construction Under The Decisional Quadratic Residuosity Assumption 54

1 Introduction

1.1 Oblivious transfer

Oblivious transfer (OT), first introduced by [35] and later defined in another way with equivalent effect [9] by [11], 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, [17, 20, 23, 41] proves that any secure multi-party computation can be based on a secure oblivious transfer protocol. In this paper, we concern a variant of OT, h-out-of-n oblivious transfer (OT_h^n). OT_h^n deals with the following scenario. A sender holds n private messages $m_1, m_2, \ldots, m_n$. A receiver holds t private positive integers $i_1, i_2, \ldots, i_t$, where $i_1 < i_2 < \ldots < i_t$ and $i_t \leqslant n$. The receiver expects to get the messages $m_{i_1}, m_{i_2}, \ldots, m_{i_t}$ 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 only and at most h messages. From this respect, the OT most literature referred is OT_1^2 and can be view as a special case of OT_h^n.

Considering a variety of attack we have to confront in real environment, a OT_h^n with security against against malicious adversary (who 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 adversary (who on one side honestly does everything told by a prescribed protocol, on one side records the message he see to deduce extra information which is not supposed to known by him). Using Goldreich's compiler [15, 17], we can gain the latter from the corresponding version of the former. However, the resulting protocols are prohibitive expensive for practical use, because they are embedded with so much invocation of zero-knowledge for NP. Thus, directly constructing the protocols based on specific intractable assumptions seems more feasible.

The first step in this direction is independently made by [30] and [1] which respectively presents an two-round OT_1^2 protocol based on the decisional Diffie-Hellman (DDH) assumption. Starting from the works of [30] and [1], [21] abstracts and generalizes the ideas to a framework for OT_1^2 by smooth projective hashing. Besides DDH assumption, the framework can be instantiated under the decisional N-th residuosity (DNR) assumption and decisional quadratic residuosity (DQR) assumption [21].

Unfortunately, these protocols (or framework) are only half-simulatable not fully-simulatable. So called fully-simulatable, we means that a protocol can be strictly proven its security under the real/ideal model simulation paradigm and without turning to random oracle. The paradigm requires that for any adversary in real life, there exists a corresponding adversary in ideal life which can simulate him. Thus, the real adversary can not do more harm than the corresponding ideal adversary do. Therefore the security level of the protocol is guaranteed to be no lower than that of ideal life. "half-simulation" only provides the simulator for malicious sender.

Considering security, requiring a protocol to be fully-simulatable is necessary. Specifically, a fully-simulatable protocol is secure against all kinds of, especially future unknown attack taken by the adversary whose the power is fixed when constructing the protocol (generally, the adversary's power is probabilistic polynomial-time) [5, 15], while a not fully-simulatable protocol isn't. For example, the protocols proposed by [1, 21, 30] suffer the selective-failure attacks, in which a malicious sender can induce transfer failures that are dependent on the message that the receiver requests [31].

Constructing fully-simulatable protocol against malicious adversary naturally becomes the focus of the research community. [4] first presents such a fully-simulatable OT. 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, [18] presents a OT^n_h using a blind identity-based encryption which is based on decisional bilinear Diffie-Hellman (DBDH) assumption. Using cut-choose technique, [24] later presents two efficient fully-simulatable protocol OT^2_1 respectively based on DDH assumption and DNR assumption, which are weaker than DBDH. It is remarkable that, [24]'s DDH-based protocol is the most efficient one among such fully-simulatable works.

The protocols mentioned above are proved secure in the plain stand-alone model which not necessarily allows concurrent composition with other arbitrary malicious protocols. [34] further the research in this filed by presenting a framework for two round efficient, fully-simulatable, universally composable OT^2_1 and three instantiations respectively based on DDH, DQR and worst-case lattice assumption. Recently, [12], using their novel compiler and somewhat non-committing encryption, converts the instantiations based on DDH, DQR to OT^2_1s with higher security level. In more detail, the resulting OT^2_1s are secure against adaptive malicious adversary, which corrupts the parties dynamically based on his knowledge gathered by far. Note that, the previous fully-simulatable protocol are only secure against non-adaptive malicious adversary, which only corrupts the parties preset before the running of the protocol.

Though constructing fully-simulatable, secure against malicious adversary OT_1^2 has been studied well, constructing such OT_h^n hasn't. We note that there are some works aiming to extend known protocols to OT_h^n. [28] shows how to implementation OT_h^n using $\log n$ invocation of OT_1^2 under "half-simulation". A similar implementation for adaptive OT_h^n can be seen in [29]. What's more, the same authors of [28,29] propose a way to transform a singe-server private-information retrieval scheme (PIR) into an oblivious transfer scheme under "half-simulation" too [31]. Under the help of random oracle, [19] shows how to extend k oblivious transfers (for some security parameter k) into many more, without much additional effort. However, the "half-simulation" and random oracle are undesirable. To our best knowledge, only [4] and [18] respectively present a fully-simulatable OT_k^n. However, as pointed out above, the assumptions the former use are not standard assumptions and the latter use is too expensive. Therefore, a well-motivated problem is to find efficient, fully-simulatable, secure against malicious adversary OT_k^n schemes under weaker complexity assumptions.

1.2 Our contribution

In this paper, we present a framework for efficient, fully-simulatable, secure against non-adaptive malicious adversary OT_h^n which is proven secure under stand model. To our best knowledge, this is the first such framework for OT_h^n. The framework have the following advantages,

1. Fully-simulatable and secure without using the common reference string. [21]'s framework for OT_1^2 is half-simulatable. Thought [34]'s framework for OT_1^2 is fully-simulatable, it doesn't work without a common reference string (CRS). What is more, how to install a trusted CRS before the protocol run is a problem to be solved. The existing possible solutions, such as natural process suggested by [34] , are only conjectures without formal proofs. The same problem remains in its adaptive instantiation presented by [12]. Therefore, considering practical use, our framework are better.

2. Efficient. The number of communication round of our framework is six. Compared with the existing fully-simulatable OT_h^n protocols, i.e. the protocols respectively presented by [4], [18] with round number $4 + 4h$, $a+h\cdot b$ ($a, b \geq 2$ respectively is the round number of two zero-knowledge proof used in the protocol), our framework is round-efficient.

   The computational overhead of our framework consists of $K \cdot n$ public key encryption operations and $K \cdot h$ public key decryption operations, where K is a value such that the possibility of our simulator failing is at most $1/2^{K-1}$. Setting K to be 40 is secure enough to be used in practice. Our framework covers [24]'s DDH-based protocol (with some straightforward modification) as a specific case. Since the price of DDH-based operations are lower than that of the operations [4] and [18] use, our DDH-based instantiation are the most efficient protocol for OT_h^T, in the sense of defending non-adaptive malicious adversary without using the CRS.

We also admit that, in the context of a trusted CRS is available and only OT_1^2 is needed, [34]'s framework is most efficient not only in round number but also in computational overhead.

3. Abstract and modular. The framework is described using just three high-level cryptographic tools, i.e. perfectly binding commitment (PBC), perfectly hiding commitment (PHC) and our new smooth projective hash denoted by $SPWH_{h,t}$. This allows a simple and intuitive understanding of its security.

4. Generally realizable. The high-level cryptographic tools PBC, PHC and $SPWH_{h,t}$ can be realizable from a variety of known specific assumption, even future assumption maybe. This makes our framework generally realizable. In particular, we instantiate $SPWH_{h,t}$ from DDH, DNR, DQR and lattice assumption. Instantiating PBC or PHC under specific assumptions is beyond the scope of this paper. Please see [14, 16] for such examples. Generally realizability is vital to make framework live long, considering the future progress in breaking a specific intractable problem. In this case, replacing the instantiation based on broken problem with that based on unbroken problem suffices.

What is more, we fix a folklore [24] that it appears technically difficult to instantiate the projective hash under lattice assumption by presenting a lattice-based $SPWH_{h,t}$ instantiation. It is notable that we gain a OT_k^n instantiation which is secure against any quantum algorithm, using this $SPWH_{h,t}$ instantiation and [3]'s commitment scheme. Considering that factoring integers and finding discrete logarithms are easy for quantum algorithms [37–39], this is a example showing the benefits from generally realizability.

As a independent contribution, we present several lemmas related to the indistinguishability of possibility ensembles generated by sampling polynomial instances. Such lemmas simplify our security proof very much. We believe that they are as the same useful in other security proof as in this paper.

1.3 Our Approach

We note that the smooth projective hash is a good abstract tool. Using this tool, [21] in fact presents a framework for half-simulatable OT_1^2, [13] 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, [24] present several fully-simulatable protocol for OT_1^2, [25] 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 fully-simulatable framework.

We define a new smooth projective hash called h-smooth t-projective hash family with witnesses and hard subset membership ($SPWH_{h,t}$). Loosely speaking, SPH is a set of operations defined over on two languages $\dot{L}$ and $\ddot{L}$, where $\dot{L} \cap \ddot{L} = \emptyset$. For any projective instance $\dot{x} \in \dot{L}$, its hash value is obtainable under the help of its witness $\dot{w}$. For any smooth instance $\ddot{x} \in \ddot{R}$, its hash value seems random.

For simplicity, we only compare our $SPWH_{h,t}$ with the version of SPH presented by [21] and denoted by $VSPH$, since on the one hand, $VSPH$ is the most complete version among previous works. On the other hand, the aim, constructing a framework for half-simulatable OT_1^2, of [21] is the closest to ours, constructing a framework for fully-simulatable OT_h^n.

$SPWH_{h,t}$'s can be viewed as a generalized version of $VSPH$ to deal with OT_h^n. $VSPH$ mostly resembles $SPWH_{1,1}$ and can be converted into $SPWH_{1,1}$ through some straightforward modifications. $SPWH_{h,t}$ extends $VSPH$ in the following way.

1. Extends the instance-sampler algorithm to generate h $\dot{x}$s and t $\ddot{x}$s in a invocation. What is more, we not only require each $\dot{x}$ to hold a witness $\dot{w}$ as previous work do, but also require each $\ddot{x}$ to hold a witness $\ddot{w}$.

2. Discards the instance test (IT) algorithm and provide a new verification (VF) algorithm which is more useful for applying cut-and-choose technique.

   We observe that the $VSPH$ indeed is easy to be extended to deal with OT_1^n, but seems difficult to be extended to deal with the more general OT_k^n. The reason is that, on one hand, the $\ddot{x}$ lacks a direct witness, which result in $\dot{x}$ and $\ddot{x}$ being generated in a dependent way. This makes designing IT for OT_h^n difficult without leaking information which is conductive to distinguish such $\dot{x}$s and $\ddot{x}$s. Thus, even constructing a framework for OT_h^n which is half-simulatable as [21] seems difficult. On the other hand, to use the technique cut-and-choose, a direct witness for $\ddot{x}$ indeed is needed. Because the simulator have to use such witness to extract the receiver's real input which is encoded as a permutation of $\dot{x}$s and $\ddot{x}$s. The difficulties mentioned above can be overcome by requiring each $\ddot{x}$ to hold a direct witness. What is more, the implementation of VF became easier than that of its predecessor IT. Because the operated object is pair of the form (x, w) which is simpler than $(x_1, \ldots, x_{h+t}, w_1, \ldots, w_h)$ which is the form of operated object of IT.

3. Extends key generation (KG) algorithm such that there is more information available for it. In more details, it can generate hash key and projective key based on the instance (i.e. $\dot{x}$ or $\ddot{x}$). This makes constructing hash system easier. In indeed, this makes lattice-based hash system come true which is thought difficult by [24].

4. Extends "smoothness" to guarantee that, loosely speaking, for any $\vec{\ddot{x}} \overset{def}{=} (\ddot{x}_1, \ldots, \ddot{x}_t)$ generated by invocations of the operations of hash system in some given way, given the corresponding projective keys, $(c_1, c_2, \ldots, c_t)$ and $(r_1, r_2, \ldots, r_t)$ are computationally indistinguishable, where c_i is the hash value of $\ddot{x}_i$ and r_i is uniformly chosen from all possible hash values.

5. Extends "hard subset membership" to guarantee that, loosely speaking, $\overrightarrow{\dot{x}\ddot{y}}$, $\pi(\overrightarrow{\dot{x}\ddot{y}})$ and $\vec{\dot{x}}$ are computationally indistinguishable , where $\overrightarrow{\dot{x}\ddot{y}} \overset{def}{=} (\dot{x}_1, \ldots, \dot{x}_h, \ddot{x}_{h+1}, \ldots, \ddot{x}_{h+t})$, $\vec{\dot{x}} \overset{def}{=} (\dot{x}_1, \ldots, \dot{x}_{h+t})$ are generated by invocations of the operations of hash system in some given way, π is a permutation.

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

1. The receiver generates hash parameters and instance vectors, then sends them to the sender after disorders 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 and encodes his private input by reordering the unchosen vectors.

4. The sender checks that the chosen vectors are generated in legal way which guarantees that the receiver learns at most h message, encrypts

his private input (i.e. the n messages he holds) using the hash system, sends the ciphertext together with some information (i.e. the projective hash keys) to the receiver which is conductive to decrypt some ciphertext.

5. The receiver decrypts the ciphertext and gains the message he expects.

Intuitively speaking, the receiver's security is implied by the hard subset membership, which guarantee that it is difficult for a malicious sender to distinguish $\dot{x}$s and $\ddot{x}$s. The sender's security is implied by the cut-and-choose, which guarantees that a malicious receiver's cheating is catched with possibility nearly 1. Formally speaking, in case the sender is corrupted, a simulator can extract the malicious sender's real input by sending cheatingly generated sample instance vectors while avoid to be catched by rewinding the malicious sender to get a appropriate result of tossing coin. In case the receiver is corrupted, a simulator can extract the malicious receiver's real input by rewinding the malicious receiver to open the instance vectors two times.

Motivated by making instantiating $SPWH_{h,t}$ easier and making use of the existing works as much as possible, we proves some lemmas which essentially guarantees that the existing SPH instantiations can be converted into $SPWH_{h,t}$ instantiations with some modification. The key idea is generating $\dot{x}$s and $\ddot{x}$s in independent way.

Our lattice-based $SPWH_{h,t}$ instantiation is builded on the lattice-based cryptosystem presented by [24]. It is noticeable that it appears difficult to get lattice-based instantiation for SPH [24]. Our solution is to let instance x ($x \in \dot{L} \cup \ddot{L}$) available for KG.

1.4 Organization

In Section 2, we describe the notations used in this paper, the security definition of OT_h^n, the definition of commitment scheme. In Section 3, we define our new hash system, i.e. $SPWH_{h,t}$. In Section 4, we construct our framework. In Section 5, we prove the security of the framework. In Section 6, we instantiate $SPWH_{h,t}$ under the lattice, DDH, DNR, DQR assumptions, respectively.

2 Preliminaries

Most notations and concepts mentioned in this section originate from [5,14, 15] which are basic literature in the filed of secure multi-party computation

(SMPC). We tailor them to the need of dealing with OT_h^n.

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 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 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 random variable $X(1^k, a)$. Probability ensemble X is polynomial-time-constructible, if there exists a probabilistic polynomial-time 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 probabilistic polynomial-time algorithm D with an infinite auxiliary information sequence $z = (z_k)_{k \in \mathbb{N}}$, 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 the 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 $\vec{x}$, $\vec{x}\langle C \rangle \overset{def}{=} (\vec{x}\langle i \rangle)_{i \in I, i \leq \#\vec{x}}$.

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

2.2 Security Definition

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 a function

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

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.
- $H \in \Psi \overset{def}{=} \{B | B \subseteq [n], \#B = h\}$ 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 [15] for the reason and related discussion). This means that $|\vec{m}| = |S|$ is required. Without loss of generality, in this paper, we assume $|\vec{m}| = |S|$ 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 of messages held by P_1. To capture the security in formal way, the concepts such as adversary, trusted third party (TTP), ideal world, real world were introduced. Note that the security target in this paper is to be secure against static malicious adversary, so only concepts related to this case is referred to in the following.

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

to learn as much extra knowledge, e.g. honest parties' private inputs, as possible. From then on, they share knowledge with each other and coordinate their behavior. Without loss of generality, we can view this coalition as follows. All cheating parties are dummy. A receives messages addressed to members of the coalition and sends messages supposed to be sent by 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's interaction.

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 $\vec{x} \overset{def}{=} (\vec{m}, H)$.

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 cheating parties. The string TTP receives is a two-dimensional vector $\vec{y}$ which is formally defined in the following way.

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

where $\alpha \in \{\vec{x}\langle i\rangle, abort_i, \{0,1\}^{|\vec{x}\langle i\rangle|}\}$ and α is generated in the way of $\alpha \leftarrow A(1^k, I, r_A, z_k, \vec{x}\langle I\rangle)$. 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 $abort_i$. 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, TTP hands $\vec{w}\langle i\rangle$ to 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 cheating party P_i outputs nothing (i.e. λ). The adversary outputs something generated by executing arbitrary function of the information he gathers by far. Without loss of generality, this can be assumed to be string consisting of $1^k, I, r_A, z_k, \vec{x}\langle I\rangle, \vec{w}\langle I\rangle$.

The output, denoted $Ideal_{f,I,A(z_k)}(1^k, \vec{m}, S)$, of the protocol OT_h^n in the ideal world is a three-dimensional vector orderly consisting of the outputs of A, P_1, P_2. Obviously, $Ideal_{f,I,A(z_k)}(1^k, \vec{m}, S)$ is a random variable whose randomness is r_A.

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, cheating 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 cheating 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 cheating party P_i outputs nothing (i.e. λ). The adversary outputs something generated by executing arbitrary function of the information he gathers by far. Without loss of generality, this can be assumed to be 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 cheating parties.

The output, denoted $Real_{\pi,I,A(z_k)}(1^k, \vec{m}, S)$, of OT_h^n in the real world is a three-dimensional vector orderly consisting of the outputs of A, P_1, P_2. Obviously, $Real_{f,I,A(z_k)}(1^k, \vec{m}, S)$ is a random variable whose randomness are r_A and $\vec{r}$.

Security definition The security of OT_h^n is formally captured by the following definition.

Definition 1 (Security of 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 evaluates 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 the same and each $\vec{m}\langle i\rangle$ is of the same length.

The concept, non-uniform probabilistic expected polynomial-time, mentioned in Definition 1 is formulated in distinct way in distinct literature such as [6, 14]. We prefer to the following definition [22], 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, a auxiliary input z_i, the common 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*

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

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

As to 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 security definition presented in [5, 14, 15] requires simulator A' to run in strict polynomial-time, but the one presented in [24,25] allows simulator to run in expected polynomial-time. Definition 1 is taken from the latter. We argue that this is justified, since [2] shows that there is no (non-trivial) constant-round zero-knowledge proof or argument having a strict polynomial-time black-box simulator, which means allowing simulator to run in expected polynomial-time is essential for achieving constant-round protocols. See [22] for further discussion.

2.3 Commitment Scheme

In this section, we briefly introduce commitment scheme [14, 16] which will be used in our framework. Loosely speaking, commitment scheme is a two-party protocol involving two phases. In the first phase, a sender U_1 sends a commitment, which hides his private input (i.e. the value he wants to commit to), to a receiver U_2. In the second phase, U_1 releases its commitment to U_2, and U_2 knows the value U_1 commits to.

Definition 3 (Commitment Scheme)**.** *A commitment scheme is defined as follows.*

- *Initial Inputs. At the beginning, all parties know the public security parameter k. The sender U_1 holds a randomness $r_1 \in \{0,1\}^*$, a value $m \in \{0,1\}^{poly(k)}$ to be committed to, where the polynomial $poly(k)$ is public. The receiver U_2 a holds randomness $r_2 \in \{0,1\}^*$.*

- *Commit Phase. U_1 computes a commitment, denoted α, based on his knowledge, ie $\alpha \leftarrow U_1(1^k, m, r_1)$, then U_1 send α to U_2.*

  The security for U_1 is implied by the commitment scheme's hiding, which is supposed to guarantee that for any probabilistic polynomial time (PPT) malicious $\tilde{U}_2$, the probability that he deduces the knowledge of m from information he have gathered by far is negligible. More formally, for any PPT $\tilde{U}_2$, for any string $m' \in \{0,1\}^{poly(k)}$, it holds that,

  $$\{ViewCP_{\tilde{U}_2}(< U_1(m), \tilde{U}_2 > (1^k))\} \overset{c}{=} \{ViewCP_{\tilde{U}_2}(< U_1(m'), \tilde{U}_2 > (1^k))\}$$

  where $ViewCP_{\tilde{U}_2}(.)$ denotes $\tilde{U}_2$'s view at the end of commit phase.

- *Reveal Phase. U_1 computes and sends a de-commitment, which Typically consists of m, r_1, to U_2 to let U_2 know m. Receiving de-commitment, U_2 checks its validity. Typically U_2 checks that $\alpha = U_1(1^k, m, r_1)$ holds. If de-commitment pass the check, U_2 knows and accepts m.*

The security for U_2 is implied by the commitment scheme's binding, which is supposed to guarantee that for any PPT malicious $\tilde{U}_1$, the probability that $\tilde{U}_1$ interprets α as a commitment to a value which is different from m is negligible. More formally, for any TTP $\tilde{U}_1$, any m, $\tilde{U}_1$ do the following experiment,

experiment $\alpha \leftarrow \tilde{U}_1(1^k, m), r_1 \leftarrow \tilde{U}_1(1^k, m), (m^{'}, r_1^{'}) \leftarrow \tilde{U}_1(1^k, m)$
it holds that

$$Pr(ViewCP_{U_2}(<\tilde{U}_1(m), U_2>(1^k)) = ViewCP_{U_2}(<\tilde{U}_1(m^{'}), U_2>(1^k))\wedge$$
$$\alpha = U_1(1^k, m, r_1)\wedge$$
$$\alpha = U_1(1^k, m^{'}, r_1^{'})) = \mu(k)$$

We are to use two stronger version of commitment schemes to construct the framework. One, called perfectly hiding commitment scheme (PHC), provides security for sender against computationally unbound malicious receivers. The other, called perfectly binding commitment scheme (PBC), provides security for receiver against computationally unbound malicious sender. For notational simplicity, we let $PHC_r(m)$ ($PBC_r(m)$) denote a commitment to m which generated by using PHC (PBC) scheme and randomness r.

3 A New Smooth Projective Hash — $SPWH_{h,t}$

In this section, we define a new smooth projective hash — h-smooth t-projective hash family with witnesses and hard subset membership, denoted $SPWH_{h,t}$, which will be used to construct our framework. In section 6, we instantiate $SPWH_{h,t}$ respectively under four distinct intractable problems.

Let us recall some related works before defining $SPWH_{h,t}$. [7,40] present the classic notation of "universal hashing". Based on "universal hashing", [8] 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, [13] modifys such definition to some extent. That is, smoothness is defined over every instance of a language rather than a randomly chosen instance. [21] 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_1^2. The resulting hashing is called verifiablely-smooth projective hash family that has hard subset membership property. A corresponding universal version also is presented by [21]. Note that, the framework presented by [21] is not fully-simulatable. Our $SPWH_{h,t}$ is an ameliorated version of [8, 13, 21]. The difference between $SPWH_{h,t}$ and the works mentioned above will be under a detailed discussion after we define $SPWH_{h,t}$. Usually constructing a hashing with property universality is the first step to gain a hashing with property smoothness. The definition of this property will be given in section 6.1.1 where we deal with how to gain smoothness for a hash.

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 $L_R \overset{def}{=} \{x|x \in \{0, 1\}^*, \exists w((x, w) \in R)\}$, $R(x) \overset{def}{=} \{y|(x, w) \in R\}$. $\Pi \overset{def}{=} \{\pi|\pi : [n] \to [n], \pi \text{ is a permutation}\}$. 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 $\forall i(i \in [d] \to \vec{x}\langle i\rangle = \vec{y}\langle\pi(i)\rangle) \wedge \forall i(i \notin [d] \to \vec{x}\langle i\rangle = \vec{y}\langle i\rangle)$ hold, where $d \overset{def}{=} \min(\#\vec{x}, n)$. Let I be an arbitrary positive integer set. $\pi(I) \overset{def}{=} \{\pi(i)|i \in I \cap [n]\}$.

Definition 4 (h-Smooth t-Projective Hash Family With Witnesses And Hard Subset Membership). *$\mathcal{H} = (PG, IS, VF, KG, Hash, pHash)$ is an h-smooth t-projective hash family with witnesses and hard subset membership ($SPWH_{h,t}$), 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, ie $\Lambda \leftarrow PG(1^k)$. Λ will be used as a parameter to define three relations $R_\Lambda, \dot{R}_\Lambda$ and $\ddot{R}_\Lambda$, where $R_\Lambda = \dot{R}_\Lambda \cup \ddot{R}_\Lambda$ and $\dot{R}_\Lambda \cap \ddot{R}_\Lambda = \emptyset$ 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 $\vec{a} \in (\dot{R}_\Lambda)^h * (\ddot{R}_\Lambda)^t$, ie $\vec{a} \leftarrow IS(1^k, \Lambda)$.*

  Let $\vec{a} \overset{def}{=} ((\dot{x}_1, \dot{w}_1), \dots, (\dot{x}_h, \dot{w}_h), (\ddot{x}_{h+1}, \ddot{w}_{h+1}), \dots, (\ddot{x}_n, \ddot{w}_n))^T$ be a vector generated by IS. We call each $\dot{x}_i$ or $\ddot{x}_i$ an instance of L_{R_Λ}. For each pair $(\dot{x}_i, \dot{w}_i)$ $((\ddot{x}_i, \ddot{w}_i))$, $\dot{w}_i$ $(\ddot{w}_i)$ is called a witness of $\dot{x}_i \in L_{\dot{R}_\Lambda}$ ($\ddot{x}_i \in L_{\ddot{R}_\Lambda}$).

  For simplicity in formulation later, we introduce some additional notations here. $\vec{x}^{\vec{a}} \overset{def}{=} (\dot{x}_1, \dots, \dot{x}_h, \ddot{x}_{h+1}, \dots, \ddot{x}_n)^T$, where $\vec{x}^{\vec{a}}\langle i\rangle = \vec{a}\langle i\rangle\langle 1\rangle$ for each i. $\vec{w}^{\vec{a}} \overset{def}{=} (\dot{w}_1, \dots, \dot{w}_h, \ddot{w}_{h+1}, \dots, \ddot{w}_n)^T$, where $\vec{w}^{\vec{a}}\langle i\rangle = \vec{a}\langle i\rangle\langle 2\rangle$ for each i. 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 verifier VF is a PPT algorithm that computes the following function*

$$\begin{aligned} \zeta : \mathbb{N} \times \quad (\{0,1\}^*)^3 \rightarrow & \quad \{0,1\} \\ \zeta(1^k, \Lambda, x, w) \qquad = & \begin{cases} 0 & \text{if } (x,w) \in \dot{R}_\Lambda, \\ 1 & \text{if } (x,w) \in \ddot{R}_\Lambda, \\ \text{undefined} & \text{otherwise .} \end{cases} \end{aligned}$$

  VF takes a security parameter k, a family parameter Λ and a pair strings (x, w) as input and outputs an indicator bit b, ie $b \leftarrow VF(1^k, \Lambda, x, w)$.

- *The hash-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, ie $(hk, pk) \leftarrow KG(1^k, \Lambda, x)$.*

- *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, ie $y \leftarrow Hash(1^k, \Lambda, x, hk)$.*

- *The projection pHash is a PPT algorithm that takes a security parameter k, a family parameter Λ, an instance x and a projection key pk as input and outputs a value y, ie $y \leftarrow Hash(1^k, \Lambda, x, pk)$.*

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

1. *Projection. $\mathcal{H}$ is projective on every string pair $(\Lambda, \dot{x})$, where $\dot{x} \in L_{\dot{R}_\Lambda}$. That is, for any sufficiently large k, any $\Lambda \in Range(PG(1^k))$, any $(\dot{x}, \dot{w}) \in \dot{R}_\Lambda$, any $(hk, pk) \in Range(KG(1^k))$, it holds that*

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

2. *Smoothness. Loosely speaking, it requires that for any string pair $(\Lambda, \vec{\ddot{x}})$, where $\vec{\ddot{x}} \in L^t_{\ddot{R}_\Lambda}$, the hash values of $\vec{\ddot{x}}$ are random. That is, for any $\pi \in \Pi$, the two probability ensembles $Sm_1 \overset{def}{=} \{Sm_1(1^k)\}_{k \in \mathbb{N}}$ and $Sm_2 \overset{def}{=} \{Sm_2(1^k)\}_{k \in \mathbb{N}}$ defined as follows, are computationally indistinguishable, ie $Sm_1 \overset{c}{=} Sm_2$.*

   $Sm_i(1^k)$: $(\Lambda, \overrightarrow{xpky}) \leftarrow SmGen_i(1^k)$, $\overrightarrow{\widetilde{xpky}} \leftarrow \pi(\overrightarrow{xpky})$, finally outputs $(\Lambda, \overrightarrow{\widetilde{xpky}})$.

$SmGen_1(1^k)$: $\Lambda \leftarrow PG(1^k)$, $\vec{a} \leftarrow IS(1^k,\Lambda)$, $\vec{x} \leftarrow \vec{x}^{\vec{a}}$, for each $j \in [n]$, $(hk_j, pk_j) \leftarrow HG(1^k, \Lambda, \vec{x}\langle j\rangle)$, $y_j \leftarrow Hash(1^k, \Lambda, \vec{x}\langle j\rangle, hk_j)$, $\overrightarrow{xpky}\langle j\rangle \leftarrow (\vec{x}\langle j\rangle, pk_j, y_j)$. Finally outputs $(\Lambda, \overrightarrow{xpky})$.

$SmGen_2(1^k)$: compared with $SmGen_1(1^k)$, the only difference is that $y_j \in_U Range(Hash(1^k, \Lambda, \vec{x}\langle j\rangle, .))$ for each $j \in [n] - [h]$.

3. *Hard Subset Membership. Loosely speaking, it requires that the instances of $L_{\dot{R}_\Lambda}$ and that of $L_{\ddot{R}_\Lambda}$ are computational indistinguishable. To be more precise, it requires $\mathcal{H}$ to meet the following conditions.*

   (a) *For any $\pi \in \Pi$, the two possibility ensembles $HSM_1 \overset{def}{=} \{HSM_1(1^k)\}_{k\in\mathbb{N}}$ and $HSM_2 \overset{def}{=} \{HSM_2(1^k)\}_{k\in\mathbb{N}}$ are computationally indistinguishable, i.e. $HSM_1 \overset{c}{=} HSM_2$.*
   $HSM_1(1^k)$: $\Lambda \leftarrow PG(1^k)$, $\vec{a} \leftarrow IS(1^k,\Lambda)$, finally outputs $(\Lambda, \vec{x}^{\vec{a}})$.
   $HSM_2(1^k)$: Operates in the same way as $HSM_1(1^k)$, but finally outputs $(\Lambda, \pi(\vec{x}^{\vec{a}}))$.

   (b) *For any $\pi \in \Pi$, for any $\pi' \in \Pi$, the two possibility ensembles HSM_2 and $HSM_3 \overset{def}{=} \{HSM_3(1^k)\}_{k\in\mathbb{N}}$ are computationally indistinguishable, i.e. $HSM_2 \overset{c}{=} HSM_3$, where HSM_2 is defined above and HSM_3 is defined as follows.*
   $HSM_3(1^k)$:$(\Lambda, \vec{b}) \leftarrow Cheat(1^k)$, finally outputs $(\Lambda, \pi'(\vec{b}))$.
   $Cheat(1^k)$: Generates n instances of $L_{\dot{R}_\Lambda}$ in the following way. $\Lambda \leftarrow PG(1^k)$, $e \leftarrow \llcorner n/h \lrcorner$, $r \leftarrow n \mod h$, $\vec{a}_i \leftarrow IS(1^k,\Lambda)$ for each $i \in [e+1]$, $\vec{b}\langle (i-1)h + j\rangle \leftarrow \vec{a}_i\langle j\rangle$ for each $i \in [e]$ and $j \in [h]$, $\vec{b}\langle eh + j\rangle \leftarrow \vec{a}_{e+1}\langle j\rangle$ for each $j \in [r]$, finally outputs $(\Lambda, \vec{b})$.

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 rather than to service as a proof of $x \in L_{\dot{R}_\Lambda}$. However, with respect to an instance $\ddot{x} \in L_{\ddot{R}_\Lambda}$, it is on the contrary. For OT^n_h, this means that the receiver use the witnesses of $\ddot{x}$ to persuade the sender to believe that he gain nothing but random (or nearly random) values by using $\ddot{x}$.*

Remark 6 (Hard Subset Membership). *The property 3a 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 $L_{\dot{R}_\Lambda}$ ($L_{\ddot{R}_\Lambda}$) with probability over h/n (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 $\pi \in \Psi$ such that $\pi(\vec{x})\langle i\rangle \in L_{\ddot{R}_\Lambda}$ for each $i \in H$. Any PPT adversary can know nothing about H if only given $\pi(\vec{x})$.*

The property 3b guarantees that one can cheat to generate a $\vec{x}$ which is supposed to fall into $L^h_{\dot{R}_\Lambda} * L^t_{\ddot{R}_\Lambda}$ but actually $L^n_{\dot{R}_\Lambda}$, while the probability of being catched is negligible. Note that, for OT^n_h, this property is a key for the simulator to extract the real input of the corrupted sender and conductive to construct a fully-simulatable OT^n_h.

The difference between our $SPWH_{h,t}$ and previous works For simplicity, we only compare our $SPWH_{h,t}$ with the hash presented by [21] and denoted by $VSPH$. We argue that this is justified, on the one hand, the version of [21] is the most complete version among previous works. On the other hand, the aim of [21] is the closest to ours. The former aims to construct a framework for OT^2_1 which actually is half-simulatable, but we aim to establish framework for fully-simulatable OT^n_h.

Loosely speaking, our $SPWH_{h,t}$ can be viewed as a generalized version of $VSPH$. Indeed, $VSPH$ mostly resembles $SPWH_{1,1}$ and can be converted into $SPWH_{1,1}$ through some straightforward modifications. The essential differences are listed as follows.

1. To deal with OT^n_h, $SPWH_{h,t}$ extends the IS algorithm to generate h $\dot{x}$s and t $\ddot{x}$s in a invocation. What is more, besides each $\dot{x}$ should hold a witness $\dot{w}$, $SPWH_{h,t}$ also require each $\ddot{x}$ to hold a witness $\ddot{w}$.

2. $SPWH_{h,t}$ discards $VSPH$'s the instance test algorithm and provide a new verification (VF) algorithm which is more useful for applying cut-and-choose technique.

   We observe that the $VSPH$ indeed is easy to be extended to deal with OT^n_1, but seems difficult to be extended to deal with the more general OT^n_k. The reason is that, on one hand, the $\ddot{x}$ lacks a direct witness, which result in $\dot{x}$ and $\ddot{x}$ being generated in a dependent way. This makes designing IT for OT^n_h difficult without leaking information which is conductive to distinguish such $\dot{x}$s and $\ddot{x}$s. Thus, even constructing a framework for OT^n_h which is half-simulatable as [21] seems difficult. On the other hand, to use the technique cut-and-choose, a direct witness for $\ddot{x}$ indeed is needed. Because the simulator have to use such witness to extract the receiver's real input which is encoded as a permutation of $\dot{x}$s and $\ddot{x}$s. The difficulties mentioned above can be overcome by requiring each $\ddot{x}$ to hold a direct witness. What is more, the implementation of VF became easier than that of its predecessor IT. Because the operated object is pair of the form (x, w) which is simpler than $(x_1, \ldots, x_{h+t}, w_1, \ldots, w_h)$ which is the form of operated object of IT.

3. $SPWH_{h,t}$ extends KG algorithm such that the information of the instance available for it. This makes constructing hash system easier. In indeed, this makes lattice-based hash system come true which is thought difficult by [24].

4 Constructing A Framework For Fully-simulatable OT_h^n

In this section, we construct a framework for OT_h^n. In the framework, we will use a PPT algorithm, denoted Γ , that receiving $B_1, B_2 \in \Psi \overset{def}{=} \{B|B \subseteq [n], \#B = h\}$, outputs a uniformly chosen a permutation $\pi \in_U \Pi$ s.t. $\pi(B_1) = B_2$, i.e. $\pi \leftarrow \Gamma(B_1, B_2)$. We give an example implementation of Γ as follows.

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

The framework for OT_h^n

- Common inputs: All entities know a public security parameter k, an positive polynomial $poly_s(.)$, a $SPWH_{h,t}$ (where $n = h + t$) hash system $\mathcal{H}$, a perfectly hiding commitment scheme, a perfectly binding commitment scheme.

- 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 $r_2 \in \{0,1\}^*$, where $\#\vec{m} = \#H$. 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: $K \overset{def}{=} poly_s(k)$, for each $i \in [K]$, $\Lambda_i \leftarrow PG(1^k)$, $\vec{a}_i \leftarrow IS(1^k, \Lambda_i)$. Without loss of generality, we assume $\vec{a}_i = ((\dot{x}_1, \dot{w}_1), \ldots, (\dot{x}_h, \dot{w}_1), (\ddot{x}_{h+1}, \ddot{w}_1), \ldots, (\ddot{x}_n, \dot{w}_n))^T$.
  2. P_2 disorders each instance vector. For each $i \in [K]$, P_2 uniformly chooses a permutation $\pi_i^1 \in_R \Pi$, 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. $((\Lambda_1, \tilde{\vec{x}}_1), \ldots, (\Lambda_n, \tilde{\vec{x}}_n))$, 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 $PHC(s)$ to P_2.
  2. P_2: $s' \in_U \{0,1\}^K$, sends $PBC(s')$ to P_1.
  3. P_1 and P_2 respectively sends each other the decommitments to $PHC(s)$ and $PBC(s')$, and respectively checks the received decommitments are valid. If check fails, P_1 (P_2 respectively) halts and outputs $abort_2$ ($abort_1$ respectively). If no check fails, then they continue.
  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 be opened.

- 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 $J_i \overset{def}{=} \{j | \tilde{\vec{w}}_i\langle j \rangle \in L_{\ddot{R}_{\Lambda_i}}, j \in [n]\}$.
  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_2. 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 time.

- 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 $L_{\dot{R}_{\Lambda_i}}$ is not more than h. P_1 checks that, for each $i \in CS$, $\#J_i \geq n - h$, and for each $j \in J_i$, $VF(1^k, \Lambda_i, \tilde{\vec{x}}_i\langle j\rangle, \tilde{\vec{w}}_i\langle j\rangle)$ is 1. If check fails, P_1 halts and outputs $cheat_2$, otherwise P_1 proceeds.
  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: $(hk_{ij}, pk_{ij}) \leftarrow KG(1^k, \Lambda_i, \tilde{\tilde{\vec{x}}}_i\langle j\rangle)$, $\beta_{ij} \leftarrow Hash(1^k, \Lambda_i, hk_{ij}, \tilde{\tilde{\vec{x}}}_i\langle j\rangle)$, $\vec{\beta}_i \overset{def}{=} (\beta_{i1}, \beta_{i2}, \ldots, \beta_{in})^T$,$\vec{c} \leftarrow \vec{m} \oplus (\oplus_{i\in\overline{CS}}\vec{\beta}_i)$, $\overrightarrow{pk}_i \overset{def}{=} (pk_{i1}, pk_{i2}, \ldots, pk_{in})^T$, 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: $\beta'_{ij} \leftarrow pHash(1^k, \Lambda_i, \tilde{\tilde{\vec{x}}}_i\langle j\rangle, \tilde{\tilde{\vec{w}}}_i\langle j\rangle, \overrightarrow{pk}_i\langle j\rangle)$, $m'_j \leftarrow \vec{c}\langle j\rangle \oplus (\oplus_{i\in\overline{CS}}\beta'_{ij})$. Finally, P_2 gains the messages $(m'_j)_{j\in H}$.

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

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

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

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

So we have

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

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

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

Theorem 7 (The protocol is secure against the malicious adversary)**.** *Assume that $\mathcal{H}$ is an h-smooth t-projective hash family with witnesses and hard subset membership, PHC is a perfectly hiding commitment, PBC is a perfectly binding commitment. Then, the protocol securely computes the oblivious transfer functionality in the presence of the malicious adversary.*

We defer the strick proof of Theorem 7 to section 5 and first give an intuitive analysis here as a warm-up. For the security of P_1, the protocol 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. The following theorem guarantees that this probability is overwhelming. In other words, the probability P_2 cheats to learn extra messages is negligible.

Theorem 8. *In case P_1 is honest and P_2 is corrupted, the probability that P_2 cheats to obtain more than h messages is at most* $1/2^{polys_s(k)}$.

Proof. According to the protocol, 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 $L_{\dot{R}_{\Lambda_i}}$. If not, P_2 cannot 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. The illegal instance vectors are lucky not to be chosen, i.e. $\overline{CS} = \{l_1, l_2, \ldots, l_d\}$. We prove this claim in two case.
   (a) In case $\overline{CS} \neq \{l_1, l_2, \ldots, l_d\}$ and $\overline{CS} - \{l_1, l_2, \ldots, l_d\} = \emptyset$, there exists $j(j \in [d] \wedge l_j \in CS)$. so P_1 can detect P_2's cheating and P_2 will gain nothing.
   (b) In case $\overline{CS} \neq \{l_1, l_2, \ldots, l_d\}$ and $\overline{CS} - \{l_1, l_2, \ldots, l_d\} \neq \emptyset$, there exists $j(j \in [n] \wedge j \in \overline{CS} \wedge \vec{x}_j$ is illegal). Because of the smoothness of $\mathcal{H}$, P_2 cannot correctly decrypt more than h entries of $\vec{c}$.

Note that, $PHC(s)$ is a perfectly hiding commitment and P_2 is honest, so the shared randomness is uniformly distributed. We have

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

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

For the security of P_2, the protocol first should prevent P_1 from learning P_2's private input. There is potential risk 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 $(hk_{ij}, pk_{ij}) \notin Range(KG(1^k, \Lambda_i, x_{ij}))$, 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.

The communication complexity of the protocol Step R1 and Step R2 are taken in one round. Step R5 is taken without communication. Each of other steps is taken in one round. The total number of the communication rounds is six.

Compared with existing protocols for OT_h^n which are fully-simulatable without restore to random oracle, our protocol is round-efficient. The round number of [4]'s $OT_{h\times 1}^n$ is $4+4h$. The round number of [18]'s OT_h^n is $a+h\cdot b$, where $a, b \geq 2$ respectively is the round number of two zero-knowledge proof of knowledge protocol used in their protocol.

The computation overhead of the protocol We measure the computation overhead of the protocol 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 [26] to know which cryptographic operation is public key operation or private key operation. As to the protocol, the public key operations are $Hash(.)$ and $pHash(.)$, and the symmetric key operations are $PHC(.)$ and $PBC(.)$. In Step S3, P_1 takes $n \cdot poly_s(k)$ $Hash(.)$s to encrypt his private input. In Step R5, P_2 takes $h \cdot poly_s(k)$ $pHash(.)$s to decrypt the messages he want. Thus, fixing the problem we tackle, the efficiency only depends on the value of $poly_s(k)$. In Section where we strictly prove the security of the protocol, we'll see the probability that the simulator fails is at most $1/2^{poly_s(k)-1}$ in case P_2 is corrupted. Thus, conditioning on the cryptographic primitives without being broken, the real world and the ideal word can be distinguished at most $1/2^{poly_s(k)-2}$. Setting $poly_s(k)$ to be 40, we obtains such a probability 3.6×10^{-12}, which is secure enough to be used in practice. By the way, our simulator also may fail (with negligible probability), but the probability of this event arising depends on the computational hiding of PBC and on the computational binding of PHC not $poly_s(k)$. So we don't need to take this case into consideration here.

Note that the operations, based on the non-standard assumptions and used by [4], ie q-Power Decisional Diffie-Hellman and q-Strong Diffie-Hellman assumptions, are very expensive. What is more, the operations, based on decisional bilinear Diffie-Hellman and used by [18], is also more expensive than that based on DDH. Thus DDH-based instantiation of our framework is the most efficient protocol for OT_h^n.

We have to admit that, in the context of a trusted CRS is available and only OT_1^2 is needed, [34]'s DDH-based instantiation, which is two-round efficient and of 2 public key encryption operations and 1 public key decryption operations, is most efficient not only in round number but also in computation overhead.

5 A Security Proof Of The Framework

We prove theorem 7 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 $P_1^{'}$, $P_2^{'}$, $A^{'}$, TTP. In the light of the parties being corrupted, there are four cases to be considered and we prove theorem 7 holds in each case.

We don't know how to construct a strictly polynomial-time simulator for the adversary in the real world, in case only P_1 or P_2 is corrupted. However, a expected polynomial-time simulator instead, which results in a failure of standard black-box reduction technique. Fortunately, the problem and its derived problems can be solved in the way of [16].

5.1 In Case P_1 Is Corrupted

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

- Initial input: Without considering the randomness they holds, the initial input $A^{'}$ holds is the same as that A holds. That is, $A^{'}$ holds the same k, $I \overset{def}{=} \{1\}$, $z = (z_k)_{k \in \mathbb{N}}$, as A, and holds a uniform distributed randomness $r_{A^{'}} \in \{0,1\}^*$. What is more, 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 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 $r \in_U \{0,1\}^K$ ($K \overset{def}{=} poly_s(k)$) as the shared randomness, and defines the sets CS and $\overline{CS}$ 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 $Cheat(1^k)$ which is defined in Definition 4 to generate these thing. $A^{'}$ sends these hash parameters and instance vectors to $\bar{A}$.

  Remark 9. *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 protocol 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 10. *The aim of doing this tossing coin is to know the randomness s $\bar{A}$ choses. What $A^{'}$ will do next is to take $PBC(r \oplus s)$ 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 protocol. Then taking $PBC_\gamma(r \oplus s)$ as his commitment, where γ is a fresh randomness uniformly chosen, $A^{'}$ executes Step R2 and R2 of the protocol.
- Step Sm5: Now $A^{'}$ and $\bar{A}$ shares the common randomness r. $A^{'}$ executes Step R4 of the protocol as the honest P_2 do. On receiving $\vec{c}$ and $(\overrightarrow{pk_i})_{i \in \overline{CS}}$, $A^{'}$ correctly decrypts the all entries of $\vec{c}$ and gains full $\bar{A}$'s real private input $\vec{m}^{'}$. Then $A^{'}$ sends $\vec{m}^{'}$ to the TTP.
- Step Sim6: When $\bar{A}$ halts, $A^{'}$ outputs what $\bar{A}$ outputs and halts.

Without considering Step Sim4, the running time of $A^{'}$ 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 $\alpha \overset{def}{=} (1^k, z_k, I, \vec{m}, r_{A'})$. 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 PBC 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, [16] encounters and solves the same problem and its derived problem as ours. In a little more details, [16] presents a protocol, in which P_1, P_2 respectively sends a perfectly hiding commitment, a perfectly binding commitment, and the corresponding de-commitments to each other as the situation of tossing coin of our protocol. To prove the security in case P_1 is corrupted, [16] constructs a simulator in the same way as ours and encounters the same problem as ours.

Using the idea of [16], we can overcome such problem too. Specifically, a 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 protocol and A' honestly executes Step R2 and R2 of the protocol 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 a big polynomial $poly(.)$, A' outputs *timeout* and halts.

Proposition 11. *The simulator A' is expected polynomial-time.*

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

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

, 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, ie

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

Since the propositions above can be proven in the same way as [16], we don't iterate such details here.

Proposition 12. *In case P_1 was corrupted, i.e. $I = \{1\}$, the equation* (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, where the function is

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

Let $h(x) \overset{def}{=} (x, g(x))$, then we have

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

Similarly, in the ideal world, we have

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

We use $\overset{c}{=}$ not $\equiv$ here because of considering the events of $A^{'}$ outputting *timeout* and outputting $ambiguity_1$.

Let $X(1^k, \vec{m}, H, z_k, \{1\}) \overset{def}{=} Real_{\pi,\{1\},A(z_k)}(1^k, \vec{m}, H)\langle 1\rangle$, $Y(1^k, \vec{m}, H, z_k, \{1\}) \overset{def}{=} Ideal_{f,\{1\},A^{'}(z_k)}(1^k, \vec{m}, H)\langle 1\rangle$, $F \overset{def}{=} (h)_{k\in\mathbb{N}}$. What is more, assume that $A^{'}$ runs in a strict polynomial time. According to proposition 19 presented in section 6, the proposition holds.

In fact, $A^{'}$ doesn't runs in a strict polynomial time, which results in a failure of the standard reduction above. Fortunately, this difficulty can be overcame by truncating the rare executions of $A^{'}$ which are too long. Since the details is the same as [16], we don't give them here and please see [16] for them. □

5.2 In Case P_2 Is Corrupted

In case 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 uniform 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 case 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, he sends to $\bar{A}$. Then $A^{'}$ proceeds to the next. 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 information $\bar{A}$ 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, 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 the following.

Remark 13. *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 protocol, and sends msg previously recorded to $\bar{A}$ in order. According to the analysis of remark 13, $A^{'}$ extracts $\bar{A}$'s real input $H^{'}$. $A^{'}$ sends $H^{'}$ to TTP and receives message $\vec{m}\langle H^{'}\rangle$.

– Step Sim6: $A^{'}$ constructs $\vec{m}^{'}$ as follow. For each $i \in H^{'}$, $\vec{m}^{'}\langle i\rangle \leftarrow \vec{m}\langle i\rangle$. For each $i \notin H^{'}$, $\vec{m}^{'} \in_U \{0,1\}^*$. 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: $A^{'}$ outputs what $\bar{A}$ outputs and halts.

Proposition 14. *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 $PHC(.)$, 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' from Step Sim1 to the beginning of Step Sim4 once 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 such 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' 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 whatever happens.

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

$$\begin{aligned}ExpTime_{A'} &\leq ExpTime_{Sim1\to Sim5} + Time_{Sim5} + Time_{Sim6}\\ &= (1+1/q)\cdot(Time_{Sim1} + Time_{Sim2} + Time_{\Xi} + Time_{Sim4})\\ &\quad + Time_{Sim5} + Time_{Sim6}\end{aligned} \tag{2}$$

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

$$\begin{aligned}q &= Pr(C)\\ &= Pr(B) + Pr(\bar{B}\cap R=\tilde{R}) + Pr(\bar{B}\cap\exists i(R\langle i\rangle = 0\wedge\tilde{R}\langle i\rangle = 1)) \qquad (3)\\ &= Pr(B) + Pr(\bar{B})(Pr(R=\tilde{R}|\bar{B}) + Pr(\exists i(R\langle i\rangle = 0\wedge\tilde{R}\langle i\rangle = 1)|\bar{B}))\end{aligned}$$

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

Recalling that the values of R and $\tilde{R}$ are all uniformly distributed, we have

$$Pr(R = \tilde{R}|\bar{B}) = \#S_1/\#(\{0,1\}^K)^2 = 1/2^K \tag{4}$$

and

$$Pr(\exists i(R\langle i\rangle = 0 \wedge \tilde{R}\langle i\rangle = 1)|\bar{B}) = \#H_3/\#(\{0,1\}^K)^2 = 1/2 - 1/2^{K+1} \tag{5}$$

Combining equation (3), (4) and (5), we have

$$\begin{aligned} q &= Pr(B) + Pr(\bar{B})(1/2 + 1/2^{K+1}) \\ &= 1/2 + 1/2^{K+1} + (1/2)Pr(B) + (1/2^{K+1})Pr(\bar{B}) \\ &> 1/2 \end{aligned} \tag{6}$$

Combining equation (2) and (6), 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. □

Proposition 15. *The probability that $A^{'}$ outputs $failure$ is less than $1/2^{K-1}$.*

Proof. From the proof of proposition 14, we know two fact. First, $Pr(X = i) \leq 1/2^{i-1}$, where X is a random variable defined as the number of the trials in a whole execution. Second, in each trial the event $A^{'}$ outputs $failure$ is the combined event of $\bar{B}$ and $R = \tilde{R}$, and this event happens with probability

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

Combining equation (4), this probability is less than $1/2^K$. Therefore, the probability that $A^{'}$ outputs $failure$ in a whole execution is

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

□

Proposition 16. *The output of the adversary A in the real world and that of the simulator $A^{'}$ in the ideal world are computationally indistinguishable, ie*

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

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, $SPWH_{h,t}$'s property smoothness guarantees that the ciphertext generated in the two way are computational indistinguishable. Therefore, our claim holds.

Combining the two claims, the proposition holds. □

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

Proof. Note that the honest parties P_1 and $P_1^{'}$ end up with nothing. Thus, the fact that the outputs of $A^{'}$ and A are computational indistinguishable, which is supported by Proposition 16, suffices to prove this proposition. □

5.3 Other Cases

In case both P_1 and P_2 are corrupted, A takes the full control over the two corrupted parties. In the ideal world, a similar situation also holds with respect to A', P_1' and P_2'. Liking in the previous cases, A' uses A's copy, $\bar{A}$, as a subroutine and builds a simulated environment for $\bar{A}$. In the end, A' outputs what $\bar{A}$ output. Since there is no need for A' to extract the real input of $\bar{A}$, the detailed construction of the simulator and the proof of the equation (1) holding in this case are very trivial. So we omit them here.

In case none of P_1 and P_2 is corrupted, we can construct the simulator of A in similar way. We omit the detailed construction and the proof for similar reason.

6 Constructing $SPWH_{h,t}$

6.1 How To Make Constructing $SPWH_{h,t}$ Easier

It is not always easy to construct $SPWH_{h,t}$ from scratch. In this section, we reduce the task of constructing $SPWH_{h,t}$ to that of constructing a hashing which seems easier.

6.1.1 Smoothness

In this section, we deal with how to obtain smoothness for a hash family. First, we introduce a lemma from [15].

Lemma 18 ([15]). *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* $X \overset{c}{=} Y$*, then*

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

where

$$\vec{X} \overset{def}{=} \{\vec{X}(1^k, a)\}_{\substack{k\in\mathbb{N}\\ a\in\{0,1\}^*}} \overset{def}{=} \{(X_1(1^k, a), X_2(1^k, a), \ldots, X_{poly(k)}(1^k, a))\}_{\substack{k\in\mathbb{N}\\ a\in\{0,1\}^*}}$$
$$X_1(1^k, a) = \ldots = X_{poly(k)}(1^k, a) = X(1^k, a)$$
$$\vec{Y} \overset{def}{=} \{\vec{Y}(1^k, a)\}_{\substack{k\in\mathbb{N}\\ a\in\{0,1\}^*}} \overset{def}{=} \{(Y_1(1^k, a), Y_2(1^k, a), \ldots, Y_{poly(k)}(1^k, a))\}_{\substack{k\in\mathbb{N}\\ a\in\{0,1\}^*}}$$
$$Y_1(1^k, a) = ldots = Y_{poly(k)}(1^k, a) = Y(1^k, a)$$

and $X_1(1^k, a), \ldots, X_{poly(k)}(1^k, a), Y_1(1^k, a), \ldots, Y_{poly(k)}(1^k, a)$ *are independent random variables.*

Proposition 19. *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,* $X \overset{c}{=} Y$*,* $F \overset{def}{=} (f_k)_{k\in\mathbb{N}}$*,* $f_k : \{0,1\}^* \to \{0,1\}^*$ *is polynomial time computable, then*

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

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

Proof. Assume the proposition is false, then there exists a non-uniform probabilistic polynomial-time 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

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

We constructs 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 $X \overset{c}{=} Y$. □

Theorem 20. *Let $\mathcal{H} = (PG, IS, VF, HG, Hash, pHash)$ be a Hash Family. $n \overset{def}{=} h + t$. For each $i \in [2]$ and $j \in [n]$, $Sm_i^j \overset{def}{=} \{Sm_i^j(1^k)\}_{k\in\mathbb{N}} \overset{def}{=} \{(SmGen_i(1^k)\langle 1\rangle, SmGen_i(1^k)\langle 2\rangle\langle j\rangle)\}_{k\in\mathbb{N}}$, 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. *$Sm_1^{h+1} \overset{c}{=} Sm_2^{h+1}$.*

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

Proof. Following Lemma 18, $\{(Sm_1^{h+1}(1^k), \ldots, Sm_1^n(1^k))\}_{k\in\mathbb{N}} \overset{c}{=} \{(Sm_2^{h+1}(1^k), \ldots, Sm_2^n(1^k))\}_{k\in\mathbb{N}}$ holds. Let $\vec{X} \overset{def}{=} \{(Sm_1^1(1^k), \ldots, Sm_1^n(1^k))\}_{k\in\mathbb{N}}$, and $\vec{Y} \overset{def}{=} \{(Sm_2^1(1^k), \ldots, Sm_2^n(1^k))\}_{k\in\mathbb{N}}$. Notice that for each $j \in [h]$ $Sm_1^j(1^k) = Sm_2^j(1^k)$, so it holds that

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

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 19, we have $F(\vec{X}) \overset{c}{=} F(\vec{Y})$, i.e.

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

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

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

That is

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

□

Loosely speaking, following Theorem 20, given a hash family $\mathcal{H}$, if each $\ddot{x}$ was sampled in an independent way and the value of $hash(1^k, \Lambda, \ddot{x}, pk)$ is (or nearly) random, then $\mathcal{H}$ is smooth.

Sometimes it is not easy that constructing the hash family $\mathcal{H}$ in such a way to gain smoothness. In this case we have to begin with constructing the hash family defined as follows, which seems easier to be implemented.

Definition 21 (ϵ-h-Universal t-Projective Hash Family With Witnesses And Hard Subset Membership, ϵ-$UPWH_{h,t}$)**.** *The definition of ϵ-$UPWH_{h,t}$ is obtained by relaxing the definition of $SPWH_{h,t}$ (i.e. Definition 4) in the way that replacing the smoothness with a new property, called ϵ-universality. A hash family is ϵ-universal, if for any sufficiently large k, any $\Lambda \in Range(PG(1^k))$, any $\ddot{x} \in L_{\ddot{R}_\Lambda}$, any $pk \in Range(KG(1^k, \Lambda, \ddot{x})\langle 2\rangle)$, any $y \in \{0,1\}^*$, it holds that*

$$Pr(Hash(1^k, \Lambda, \ddot{x}, HK) = y | PK = pk) \leq \epsilon$$

where (HK, PK) is uniformly chosen from $Range(KG(1^k, \Lambda, \ddot{x}))$, i.e. $(HK, PK) \leftarrow KG_R(1^k, \Lambda, \ddot{x})$, where R is random variable defined over $\{0,1\}^*$ with a uniform distribution.

Definition 21 relaxes the requirement of the randomness of the hash value of $\ddot{x}$. Instead, ϵ-$UPH_{h,t}$ requires that, given $\ddot{x}, pk$, the probability of guessing the value of $Hash(1^k, \ddot{x}, hk)$ is at most ϵ. Assume $\epsilon < 1$, as [8, 21], we can efficiently gain a $SPWH_{h,t}$ from a ϵ-$UPWH_{h,t}$.

Theorem 22. *There exists an efficient algorithm that receiving a ϵ-$UPWH_{h,t}$ $\mathcal{H}$, where $\epsilon < 1$, it output a $SPWH_{h,t}$ $\mathcal{H}'$.*

The way to prove this theorem is to construct such an algorithm, which can be done by a simply application of the Leftover Hash Lemma (please see [27] for this lemma). The detailed construction is the same as [8] except some straightforward modification. Considering the space, we don't iterate it here.

Theorem 22 implies that to construct a $SPWH_{h,t}$, what we need to do is only to construct a ϵ-$UPWH_{h,t}$, where $\epsilon < 1$.

6.1.2 Hard Subset Membership

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

Proposition 23. *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 $X \overset{c}{=} Y$. Then*

$$\overrightarrow{XY} \overset{c}{=} \Phi(\overrightarrow{\widetilde{XY}})$$

where $\overrightarrow{XY}$ *and* $\Phi(\overrightarrow{\widetilde{XY}})$ *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* $Y_i(1^k, a) = Y(1^k, a)$, $poly_1(.) \le poly(.)$, *all* $X_i(1^k, a)$ *and* $Y_i(1^k, a)$ *are independent;*
- $\Phi(\overrightarrow{\widetilde{XY}}) \overset{def}{=} \{\Phi_k(\overrightarrow{\widetilde{XY}}(1^k, a))\}_{k\in\mathbb{N}, a\in\{0,1\}^*}$, $\overrightarrow{\widetilde{XY}} = \overrightarrow{XY}$, $\Phi \overset{def}{=} (\Phi_k)_{k\in\mathbb{N}}$, *each* $\Phi_k : [poly(k)] \to [poly(k)]$ *is a permutation.*

Proof. In case $\Phi_k([poly_1(k)]) \subseteq [poly_1(k)]$, it obviously holds. We proceed to prove it also holds in case $\Phi_k([poly_1(k)]) \nsubseteq [poly_1(k)]$. Assume it does not hold, then there exists a non-uniform probabilistic polynomial-time 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 $k \in G$,

$$|Pr(D(1^k, z_k, a, \overrightarrow{XY}(1^k, a)) = 1) - Pr(D(1^k, z_k, a, \Phi_k(\overrightarrow{\widetilde{XY}}(1^k, a)) = 1)| \ge 1/poly_2(k) \quad (7)$$

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

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

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

$$\Phi_k^j(i) = \begin{cases} i & i \in (F - F_j) \cup \Phi_k(F - F_j) \\ \Phi_k(i) & i \in [poly(k)] - (F - F_j) - \Phi_k(F - F_j) \end{cases} \quad j \in [\#F]$$

It is easy to see that $\overrightarrow{\widetilde{XY}}(1^k, a) = \Phi_k^{0'}(\overrightarrow{\widetilde{XY}}(1^k, a)) \equiv \Phi_k^0(\overrightarrow{\widetilde{XY}}(1^k, a))$, and $\Phi_k = \Phi_k^{\#F}$. Since $\overrightarrow{XY}(1^k, a) = \overrightarrow{\widetilde{XY}}(1^k, a)$, so we have

$$\begin{aligned} &|Pr(D(1^k, z_k, a, \overrightarrow{XY}(1^k, a)) = 1) - Pr(D(1^k, z_k, a, \Phi_k(\overrightarrow{\widetilde{XY}}(1^k, a))) = 1)| \\ &= |Pr(D(1^k, z_k, a, \Phi_k^0(\overrightarrow{\widetilde{XY}}(1^k, a))) = 1) - Pr(D(1^k, z_k, a, \Phi_k^{\#F}(\overrightarrow{\widetilde{XY}}(1^k, a))) = 1)| \end{aligned} \quad (8)$$

Following triangle inequality, we have

$$|Pr(D(1^k, z_k, a, \Phi_k^0(\overrightarrow{\widetilde{XY}}(1^k, a))) = 1) - Pr(D(1^k, z_k, a, \Phi_k^{\#F}(\overrightarrow{\widetilde{XY}}(1^k, a))) = 1)| \leq \sum_{j=1}^{\#F} |Pr(D(1^k, z_k, a, \Phi_k^{j-1}(\overrightarrow{\widetilde{XY}}(1^k, a))) = 1) - Pr(D(1^k, z_k, a, \Phi_k^{j}(\overrightarrow{\widetilde{XY}}(1^k, a))) = 1)| \tag{9}$$

Combining equation (7) (8) (9), we have

$$\sum_{j=1}^{\#F} |Pr(D(1^k, z_k, a, \Phi_k^{j-1}(\overrightarrow{\widetilde{XY}}(1^k, a))) = 1) - Pr(D(1^k, z_k, a, \Phi_k^{j}(\overrightarrow{\widetilde{XY}}(1^k, a))) = 1)| \geq 1/poly_2(k)$$

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

$$|Pr(D(1^k, z_k, a, \Phi_k^{j-1}(\overrightarrow{\widetilde{XY}}(1^k, a))) = 1) - Pr(D(1^k, z_k, a, \Phi_k^{j}(\overrightarrow{\widetilde{XY}}(1^k, a))) = 1)| \geq 1/(\#F \cdot poly_2(k)) \tag{10}$$

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 $\Phi_k^{j-1}(\overrightarrow{\widetilde{XY}}(1^k, a))$ and $\Phi_k^{j}(\overrightarrow{\widetilde{XY}}(1^k, a))$ are the i_j-th and $\Phi_k(i_j)$-th entries, i.e. $\Phi_k^{j-1}(\overrightarrow{\widetilde{XY}}(1^k, a))\langle i_j \rangle = X(1^k, a)$, $\Phi_k^{j-1}(\overrightarrow{\widetilde{XY}}(1^k, a))\langle \Phi_k(i_j) \rangle = Y(1^k, a)$, $\Phi_k^{j}(\overrightarrow{\widetilde{XY}}(1^k, a))\langle i_j \rangle = Y(1^k, a)$, $\Phi_k^{j}(\overrightarrow{\widetilde{XY}}(1^k, a))\langle \Phi_k(i_j) \rangle = X(1^k, a)$.
Let $\overrightarrow{MXY} \overset{def}{=} \{\overrightarrow{MXY}(1^k, a)\}_{k \in \mathbb{N}, a \in \{0,1\}^*}$, where $\overrightarrow{MXY}(1^k, a)$ is defined as follows. For each $d \in [poly(k)]$,

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

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

$Y(1^k, a)$. Following triangle inequality, we have

$$|Pr(D(1^k, z_k, a, \Phi_k^{j-1}(\overrightarrow{\widetilde{XY}}(1^k, a))) = 1) - Pr(D(1^k, z_k, a, \overrightarrow{MXY}(1^k, a)) = 1)| +$$
$$|Pr(D(1^k, z_k, a, \overrightarrow{MXY}(1^k, a)) = 1) - Pr(D(1^k, z_k, a, \Phi_k^{j}(\overrightarrow{\widetilde{XY}}(1^k, a))) = 1)|$$
$$\geq |Pr(D(1^k, z_k, a, \Phi_k^{j-1}(\overrightarrow{\widetilde{XY}}(1^k, a))) = 1) - Pr(D(1^k, z_k, a, \Phi_k^{j}(\overrightarrow{\widetilde{XY}}(1^k, a))) = 1)| \tag{11}$$

Combining (10) (11), we know that

$$|Pr(D(1^k, z_k, a, \Phi_k^{j-1}(\overrightarrow{\widetilde{XY}}(1^k, a))) = 1) - Pr(D(1^k, z_k, a, \overrightarrow{MXY}(1^k, a)) = 1)|$$
$$\geq 1/(2\#F \cdot poly_2(k)) \tag{12}$$

or

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

holds. Without loss of generality, we assume equation (12) holds (in case equation (13), the proof can be done in the 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.

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

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

$$|Pr(D'(1^k, z_k, a, X(1^k, a)) = 1) - Pr(D'(1^k, z_k, a, Y(1^k, a)) = 1)| =$$
$$|Pr(D(1^k, z_k, a, \overrightarrow{MXY}(1^k, a)) = 1) - Pr(D(1^k, z_k, a, \Phi_k^{c-1}(\overrightarrow{\widetilde{XY}}(1^k, a))) = 1)| \tag{14}$$

Combining (12) (14), we have

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

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

Proposition 24. *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* $X \overset{c}{=} Y$*. Then*

$$\overrightarrow{X} \overset{c}{=} \overrightarrow{XY}$$

where $\overrightarrow{Y}$ *is defined in lemma 18 and* $\overrightarrow{XY}$ *is defined in proposition 23. All random variables* $\overrightarrow{X}(1^k, a)\langle i\rangle$ *and* $\overrightarrow{XY}(1^k, a)\langle i\rangle$ *are independent.*

Proof. Assume the proposition is false, then there exists a non-uniform probabilistic polynomial-time distinguisher D with an infinite sequence $z = (z_k)_{k\in\mathbb{N}}$, a polynomial $poly_2(.)$, an infinite positive integer set $G \subseteq \mathbb{N}$ such that, for each $k \in G$,

$$|Pr(D(1^k, z_k, a, \overrightarrow{Y}(1^k, a)) = 1) - Pr(D(1^k, z_k, a, \overrightarrow{XY}(1^k, a)) = 1)| \geq 1/poly_2(k) \quad (15)$$

Let $Hybird_j \overset{def}{=} \{Hybird_j(1^k, a)\}_{k\in\mathbb{N}, a\in\{0,1\}^*}$, $Hybird_j(1^k, a) \overset{def}{=} (X_1(1^k, a), \ldots, X_{poly_1(k)+j}(1^k, a), Y_{poly_1(k)+j+1}(1^k, a), \ldots, Y_{poly(k)}(1^k, a))$. Let $d \overset{def}{=} poly(k) - poly_1(k)$. Obviously, $Hybird_0(1^k, a) = \overrightarrow{XY}(1^k, a)$, $Hybird_d(1^k, a) = \overrightarrow{X}(1^k, a)$, so we have

$$|Pr(D(1^k, z_k, a, \overrightarrow{Y}(1^k, a)) = 1) - Pr(D(1^k, z_k, a, \overrightarrow{XY}(1^k, a)) = 1)| = |Pr(D(1^k, z_k, a, Hybird_0(1^k, a)) = 1) - Pr(D(1^k, z_k, a, Hybird_d(1^k, a)) = 1)| \quad (16)$$

Following triangle inequality, we have

$$\sum_{j=1}^{d} |Pr(D(1^k, z_k, a, Hybird_{j-1}(1^k, a)) = 1) - Pr(D(1^k, z_k, a, Hybird_j(1^k, a)) = 1)| \geq |Pr(D(1^k, z_k, a, Hybird_0(1^k, a)) = 1) - Pr(D(1^k, z_k, a, Hybird_d(1^k, a)) = 1)| \quad (17)$$

Combining (15) (16) (17), we know that there exists a constant $j \in [d]$ such that

$$|Pr(D(1^k, z_k, a, Hybird_{j-1}(1^k, a)) = 1) - Pr(D(1^k, z_k, a, Hybird_j(1^k, a)) = 1)| \geq 1/(d \cdot poly_2(k)) \quad (18)$$

The difference between $Hybird_{j-1}(1^k, a)$ and $Hybird_{j-1}(1^k, a)$ is the $poly_1(k)+j$-th entry, i.e. $Hybird_{j-1}(1^k, a)\langle poly_1(k)+j\rangle = Y(1^k, a)$, $Hybird_j(1^k, a)\langle poly_1(k)+$

$j\rangle = X(1^k, a)$. 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.

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

Obviously,

$$\begin{gathered}|Pr(D^{'}(1^k, z_k, a, Y(1^k, a)) = 1) - Pr(D^{'}(1^k, z_k, a, X(1^k, a))| = \\ |Pr(D(1^k, z_k, a, Hybird_{j-1}(1^k, a)) = 1) - Pr(D(1^k, z_k, a, Hybird_j(1^k, a)) = 1)|\end{gathered} \tag{19}$$

Combining (18) (19), we have

$$|Pr(D^{'}(1^k, z_k, a, Y(1^k, a)) = 1) - Pr(D^{'}(1^k, z_k, a, X(1^k, a))| \geq 1/(d \cdot poly_2(k)$$

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

Proposition 25. *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,* $X \overset{c}{=} Y$, $F = (f_k)_{k\in\mathbb{N}}$, $f_k : \{0,1\}^* \to \{0,1\}^*$ *is a polynomial time computable function, then*

$$F(\vec{X}) \overset{c}{=} F(\vec{Y})$$

where $F(\vec{X}) \overset{def}{=} \{f_k(\vec{X}(1^k, a))\}_{k\in\mathbb{N}, a\in\{0,1\}^*}$,$F(\vec{Y}) \overset{def}{=} \{f_k(\vec{Y}(1^k, a))\}_{k\in\mathbb{N}, a\in\{0,1\}^*}$, $\vec{X}(1^k, a)$ *and* $\vec{Y}(1^k, a)$ *are defined as lemma 18.*

Proof. Following lemma 18, $\vec{X} \overset{c}{=} \vec{Y}$ holds. Since the probability ensemble X is polynomial time constructible, so we can gain a PPT sampling algorithm for the probability ensemble $\vec{X}$ by invocating $S_X(.)$ $poly(k)$ times, thus $\vec{X}$ also is polynomial time constructible. We can prove $\vec{Y}$ is polynomial time constructible in the same way. Following proportion 19, we have $F(\vec{X}) \overset{c}{=} F(\vec{Y})$. □

Proposition 26. *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,* $X \overset{c}{=} Y$, $F = (f_k)_{k\in\mathbb{N}}$, $f_k : \{0,1\}^* \to \{0,1\}^*$ *is a polynomial time computable function, then*

$$F(\vec{X}) \overset{c}{=} F(\overrightarrow{XY})$$

where the probability ensemble $F(\vec{X})$ *and* $F(\overrightarrow{XY})$ *are defined as proposition 25. All variables* $F(\vec{X}(1^k, a))\langle i\rangle$ *and* $F(\overrightarrow{XY}(1^k, a))\langle i\rangle$ *are independent.*

Proof. Following proposition 24, we know $\vec{X} \overset{c}{=} \overrightarrow{XY}$. As in the proof of proposition 25, we can prove that $\vec{X}$ and $\overrightarrow{XY}$ are polynomial time constructible. Following proposition 25, we have $F(\vec{X}) \overset{c}{=} F(\overrightarrow{XY})$. □

Theorem 27. *Let $\mathcal{H} = (PG, IS, VF, HG, Hash, pHash)$ be a hash family. Let $n \overset{def}{=} h + t$. For each $i \in [n]$, $HSM^i \overset{def}{=} \{HSM^i(1^k)\}_{k\in\mathbb{N}}$, $HSM^i(1^k) \overset{def}{=} (HSM_1(1^k)\langle 1\rangle, HSM_1(1^k)\langle i+1\rangle)$, 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. *$HSM^1 \overset{c}{=} HSM^{h+1}$.*

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

Proof. First, we prove $\mathcal{H}$ has property hard subset membership 3a.
Let $\pi \in \Pi$, $X \overset{def}{=} HSM^1$, $Y \overset{def}{=} HSM^{h+1}$, $\Phi = (\pi)_{k\in\mathbb{N}}$, $poly_1(.) \overset{def}{=} h$, $poly(.) \overset{def}{=} n$. Following proposition 23, we know

$$\overrightarrow{XY} \overset{c}{=} \Phi(\overrightarrow{\widetilde{XY}})$$

That is

$$((HSM_1(1^k)\langle 1\rangle, HSM_1(1^k)\langle 2\rangle), \ldots (HSM_1(1^k)\langle 1\rangle, HSM_1(1^k)\langle n+1\rangle)) \overset{c}{=} (HSM_2(1^k)\langle 1\rangle, HSM_2(1^k)\langle 2\rangle), \ldots (HSM_2(1^k)\langle 1\rangle, HSM_2(1^k)\langle n+1\rangle))$$

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

$$(HSM_1(1^k)\langle 1\rangle, HSM_1(1^k)\langle 2\rangle, \ldots, HSM_1(1^k)\langle n+1\rangle) \overset{c}{=} (HSM_2(1^k)\langle 1\rangle, HSM_2(1^k)\langle 2\rangle, \ldots, HSM_2(1^k)\langle n+1\rangle)$$

i.e.

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

Second, we prove $\mathcal{H}$ has property hard subset membership 3b. Let $\pi' \in \Pi$, $F = (\pi')_{k\in\mathbb{N}}$. Following proposition 26, we have

$$F(\vec{X}) \overset{c}{=} F(\overrightarrow{XY})$$

That is

$$((HSM_3(1^k)\langle 1\rangle, HSM_3(1^k)\langle 2\rangle), \ldots (HSM_3(1^k)\langle 1\rangle, HSM_3(1^k)\langle n+1\rangle)) \overset{c}{=} \pi'((HSM_1(1^k)\langle 1\rangle, HSM_1(1^k)\langle 2\rangle), \ldots (HSM_1(1^k)\langle 1\rangle, HSM_1(1^k)\langle n+1\rangle))$$

Since $HSM_3(1^k)\langle 1\rangle = HSM_1(1^k)\langle 1\rangle$, so

$$\begin{aligned}(HSM_3(1^k)\langle 1\rangle, HSM_3(1^k)\langle 2\rangle, \ldots, HSM_3(1^k)\langle n+1\rangle) &\overset{c}{=} \\ (HSM_1(1^k)\langle 1\rangle, \pi'(HSM_1(1^k)\langle 2\rangle, \ldots, &HSM_1(1^k)\langle n+1\rangle))\end{aligned}$$

We further have

$$\begin{aligned}(HSM_3(1^k)\langle 1\rangle, \pi'^{-1}(HSM_3(1^k)\langle 2\rangle, \ldots, HSM_3(1^k)\langle n+1\rangle)) &\overset{c}{=} \\ (HSM_1(1^k)\langle 1\rangle, HSM_1(1^k)\langle 2\rangle, \ldots, &HSM_1(1^k)\langle n+1\rangle) \quad (20)\end{aligned}$$

Because $HSM_3(1^k)\langle 2\rangle = \ldots = HSM_3(1^k)\langle 1+n\rangle$, we have

$$\begin{aligned}(HSM_3(1^k)\langle 1\rangle, \pi'^{-1}(HSM_3(1^k)\langle 2\rangle, \ldots, HSM_3(1^k)\langle n+1\rangle)) &\equiv \\ (HSM_3(1^k)\langle 1\rangle, HSM_3(1^k)\langle 2\rangle, \ldots, &HSM_3(1^k)\langle n+1\rangle) \quad (21)\end{aligned}$$

Combining equation (20) (21) and recalling the definitions of HSM_1 and HSM_2, we have

$$HSM_1 \overset{c}{=} HSM_3$$

Remember that we have proven $HSM_1 \overset{c}{=} HSM_2$, so we have

$$HSM_2 \overset{c}{=} HSM_3$$

□

Loosely speaking, Theorem 27 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.

6.2 A Construction Under Lattice

6.2.1 Background

Learning with errors (LWE) is an average-case problem. [36] 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 $x \mod y \overset{def}{=} x - \llcorner x/y \lrcorner y$. Then we know $x \mod 1 \overset{def}{=} x - \llcorner x \lrcorner$. Let β be an arbitrary positive real number. Let Ψ_β be a density function whose probability 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

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

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

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

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. LWE can be formulated as follows.

Definition 28 (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, $\chi : Z_q \rightarrow R^+$ is a probability density function.

With respect to the hardness of LWE, [36] 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 29 ([36])**.** *Setting security parameter k to be a value such that q is a prime, $\alpha \leftarrow \alpha(1^k)$, and $\alpha \cdot q > 2\sqrt{k}$. Then the lattice problems $SIVP$ and $GapSVP$ can be reduced to $LWE_{q,\bar{\Psi}_\alpha}$. More specifically, if there exists an efficient (possibly quantum) algorithm that solves $LWE_{q,\bar{\Psi}_\alpha}$, then there exists an efficient quantum algorithm for solving the following worst-case lattice problems in the l_2 norm.*

- *SIVP: In any lattice Λ of dimension k, and a set of k linearly independent lattice vectors of length within at most $\tilde{O}(k/\alpha)$ of optimal.*

- *GapSVP: In any lattice Λ of dimension m, approximate the length of a shortest nonzero lattice vector to within a $\tilde{O}(k/\alpha)$ factor.*

We emphasize the fact that the reduction of Lemma 29 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 29 is beyond the scope of this paper. To see more details and examples, we recommend [36] and [34].

To construct a $SPWH_{h,n}$, we need to use a public key cryptosystem based on LWE. [36] and [34] respectively presents such an cryptosystem. Considering the cost, we choose the one presented by the latter and slightly tailor it to our need. The LWE-based cryptosystem with message space Z_p is defined as follow, where $p \geq 2$ is polynomial in k.

- $Setup(1^k, p)$: Generates the public parameters as follows. $q \in_U \{q| q \in , q$ is polynomial in $k, q > p\}$, $m \leftarrow poly(1^k)$, $\chi \leftarrow \chi(1^k)$ and χ is a density function over Z_q, $para \leftarrow (q, p, m, \chi)$, finally outputs $para$.

- $KeyGen(1^k, para)$: $A \in_U (Z_q)^{m\times k}$, $\vec{s} \in_U (Z_q)^k$, $\vec{e} \in_\chi (Z_q)^m$, $\vec{p} \leftarrow A\vec{s} + \vec{e} \mod q$, $pubk \leftarrow (A, \vec{p})$, $sk \leftarrow \vec{s}$, finally outputs a public-secret key pair $(pubk, sk)$.

- $Enc(.), Dec(.)$: Since $Enc(.)$, $Dec(.)$ are immaterial to understand this paper, we omit their detailed procedure here.

6.2.2 Detailed Construction

We now present our construction of a $SPWH_{h,t}$ based on LWE.

- $PG(1^k)$: $para \leftarrow Setup(1^k, p)$, $(q, p, m, \chi) \leftarrow para$, $A \in_U (Z_q)^{m\times k}$, $\Lambda \leftarrow (p, q, m, k, A, \chi)$, finally outputs Λ.

- $IS(1^k, \Lambda)$: $((p, q, m, k, A, \chi) \leftarrow \Lambda$, $\forall i \in [n]$ $\vec{s_i} \in_U (Z_q)^k$, $\forall i \in [n]$ $\vec{e_i} \in_\chi (Z_q)^m$, $\forall i \in [h]$ $\dot{x}_i \leftarrow A\vec{s_i} + \vec{e_i} \mod q$, $\forall i \in [h]$ $\dot{w}_i \leftarrow \vec{s_i}$, $\forall i \in [n] - [h]$ $\ddot{x}_i \leftarrow A\vec{s_i} + \vec{e_i} + (1, 1, \ldots, 1)^T \mod q$, $\forall i \in [n] - [h]$ $\ddot{w}_i \leftarrow (\vec{s_i}, \vec{e_i})$, finally outputs $((\dot{x}_1, \dot{w}_1), \ldots, (\dot{x}_h, \dot{w}_h), (\ddot{x}_{h+1}, \ddot{w}_{h+1}), \ldots, (\ddot{x}_n, \ddot{w}_n))$.

- $VF(1^k, \Lambda, x, w)$: $(p, q, m, k, A, \chi) \leftarrow \Lambda$, $(\vec{s_i}, \vec{e_i}) \leftarrow w$, if $x = A\vec{s_i} + \vec{e_i} + (1, 1, \ldots, 1)^T \mod q$ holds, then outputs 1; otherwise outputs 0.

- $KG(1^k, \Lambda, x)$: $(p, q, m, k, A, \chi) \leftarrow \Lambda$, $a \in_U Z_p$, $\vec{s} \in_U (Z_q)^k$, $\vec{p} \leftarrow A\vec{s} + x \mod q$, $\alpha \leftarrow Enc_{A,\vec{p}}(a)$, $hk \leftarrow a$, $pk \leftarrow (\vec{s}, \alpha)$, finally outputs (hk, pk).

- $Hash(1^k, \Lambda, x, hk)$: $(p, q, m, k, A, \chi) \leftarrow \Lambda$, $a \leftarrow hk$, finally outputs a.
- $pHash(1^k, \Lambda, x, pk, w)$: $(k, m, p, q, \chi, A) \leftarrow \Lambda$, $(\vec{s}, \alpha) \leftarrow pk$, $\vec{u} \leftarrow \vec{s} + w$, $a \leftarrow Dec_{\vec{u},A}(\alpha)$, finally outputs a.

We remark that the choice of $(1, 1, \ldots, 1)^T$ is arbitrary. It is used to separate $\dot{R}$ from $\ddot{R}$. From the proof of the following proposition, we can see that any constant vector $\vec{c} \in (Z_p)^m - \{(0, 0, \ldots, 0)\}$ is good too.

Proposition 30. *Assume LWE is a hard problem, then VF computes the function ζ defined in Definition 4.*

Proof. It is easy to see that in case $(x, w) \in \ddot{R}_\Lambda$ VF correctly computes ζ. It remains to prove that in case $(x, w) \in \dot{R}_\Lambda$ VF correctly computes ζ. Assume that VF outputs 1 in the latter case. Then there exists an efficient adversary such that on receiving $(1^k, (k, m, p, q, \chi, A), A\vec{s}_i + \vec{e}_i, \vec{s})$ outputs $(\vec{s}^{\,\prime}_i, \vec{e}^{\,\prime}_i)$, where $A\vec{s}_i + \vec{e}_i \mod q = A\vec{s}^{\,\prime}_i + \vec{e}^{\,\prime}_i + (1, 1, \ldots, 1)^T \mod q$. That is,

$$A\vec{s}_i + \vec{e}_i - (1, 1, \ldots, 1)^T \mod q = A\vec{s}^{\,\prime}_i + \vec{e}^{\,\prime}_i \mod q$$

which implies that the adversary is an efficient algorithm breaking LWE. □

Proposition 31. *Assume LWE is a hard problem, the hash system holds the property projection.*

Proof. Let $\dot{x}_i \in Range(IS(1^k, \Lambda))$. Looking at $IS(1^k, \Lambda)$, $\dot{x}_i$ in fact is a public key whose corresponding secret key is $\vec{s}_i$. The ciphertext α in $KG(1^k, \Lambda, x)$ is encrypted using the public key whose corresponding secret key is $\vec{s}_i + \vec{s}$. The value of $Hash(1^k, \Lambda, x, hk)$ is the plaintext of α. Using $\vec{s}_i + \vec{s}$ as a secret key, $pHash(1^k, \Lambda, x, pk, w)$ correctly outputs α's plaintext. This means that for any $(\dot{x}, \dot{w}, \Lambda)$ generated by the hash system, it holds that

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

□

Proposition 32. *The hash system holds the property smoothness.*

Proof. We are to use Theorem 20 to prove this. It is easy to see that this $SPWH_{h,n}$ meets the first two requirements, so it remains to prove that $SPWH_{h,n}$ also meets the last requirement, i.e. $Sm_1^{h+1} \overset{c}{=} Sm_2^{h+1}$. For this case, Sm_1^{h+1}, Sm_2^{h+1} are

- $Sm_1^{h+1}(1^k)$: $\Lambda \leftarrow PG(1^k)$, $(p, q, m, k, A, \chi) \leftarrow \Lambda$, $\vec{s}_{h+1} \in_U (Z_q)^k$, $\vec{e}_{h+1} \in_\chi (Z_q)^m$, $\ddot{x}_{h+1} \leftarrow A\vec{s}_{h+1} + \vec{e}_{h+1} + (1, 1, \ldots, 1)^T \mod q$, $a \in_U Z_p$, $\vec{s} \in_U (Z_q)^k$, $\vec{p} \leftarrow A\vec{s} + \ddot{x}_{h+1} \mod q$, $\alpha \leftarrow Enc_{A,\vec{p}}(a)$, $hk \leftarrow a$, $pk \leftarrow (\vec{s}, \alpha)$. Finally outputs $(\Lambda, \ddot{x}_{h+1}, pk, a)$.

- $Sm_2^{h+1}(1^k)$: Outputs $(\Lambda, \ddot{x}_{h+1}, pk, y)$, where $(\Lambda, \ddot{x}_{h+1}, pk)$ is generated in the same way as $Sm_1^{h+1}(1^k)$ and $y \in_U Z_q$.

Obviously, $Sm_1^{h+1}(1^k)$ and $Sm_2^{h+1}(1^k)$ are identically distributed, which implies that $Sm_1^{h+1} \overset{c}{=} Sm_2^{h+1}$. □

Proposition 33. *Assume LWE is a hard problem, then hash system holds the property hard subset membership.*

Proof. We are to use Theorem 27 to prove this proposition holds. It is easy to see that this $SPWH_{h,n}$ meets the first two requirements, so it remains to prove that $SPWH_{h,n}$ also holds the last requirement, i.e. $HSM^1 \overset{c}{=} HSM^{h+1}$. For this case, HSM^1, HSM^{h+1} are where

- $HSM^1(1^k)$: $\Lambda \leftarrow PG(1^k)$, $(p, q, m, k, A, \chi) \leftarrow \Lambda$, $\vec{s}_1 \in_U (Z_q)^k$, $\vec{e}_1 \in_\chi (Z_q)^m$, $\dot{x}_1 \leftarrow A\vec{s}_1 + \vec{e}_1 \mod q$. Finally outputs $(\Lambda, \dot{x}_1)$.
- $HSM^{h+1}(1^k)$: $\Lambda \leftarrow PG(1^k)$, $(p, q, m, k, A, \chi) \leftarrow \Lambda$, $(p, q, m, k, A, \chi) \leftarrow \Lambda$, $\vec{s}_{h+1} \in_U (Z_q)^k$, $\vec{e}_{h+1} \in_\chi (Z_q)^m$, $\ddot{x}_{h+1} \leftarrow A\vec{s}_{h+1} + \vec{e}_{h+1} + (1, 1, \ldots, 1)^T \mod q$. Finally outputs $(\Lambda, \ddot{x}_{h+1})$.

Obviously, HSM^1 and HSM^{h+1} are identically distributed, which implies that $HSM^1 \overset{c}{=} HSM^{h+1}$. □

Combining propositions above and Lemma 29, we have the following theorem.

Theorem 34. *If $SIVP$ or $GapSVP$ is a hard problem, then the hash system is a $SPWH_{h,t}$.*

6.2.3 An Instantiation of OT_h^n Against Any Quantum Algorithm

The security proof of the framework guarantees that, any algorithm breaking the framework is an algorithm breaking at least one of cryptographic tools used in the framework. Therefore, to gain an instantiation of our framework against any quantum algorithm, it suffices that adopting the instantiations of commitment schemes and $SPWH_{h,t}$, which are secure against any quantum algorithm, in our framework.

[36] shows that the problems $SIVP$ and $GapSVP$ are hard for quantum algorithm at present. Combining Theorem 34, our LWE-based $SPWH_{h,t}$ is secure against any quantum algorithm. It remains to find a PHC and a PBC with security against any quantum algorithm. [3] presents such a commitment scheme, which is provably unbreakable by both parties with unlimited computation power and algorithmic sophistication. So we have,

Theorem 35. *Assuming that one of the problems $SIVP$ and $GapSVP$ is hard for any quantum algorithm, instantiating the OT_h^n framework with our LWE-based $SPWH_{h,t}$ and the commitment scheme presented by [3], the resulted protocol for OT_h^n is secure against any quantum algorithm.*

[36] points out that the problem of LWE and the problem of decoding random linear code (DRLC) are essentially the same. This implies that replacing the commitment scheme with the PHC and PBC based on DRLC, Theorem 35 also holds. What is more, [14] shows that, first, assuming that DRLC is hard, there exists a one-way function; second, assuming the existence of a one-way function, then there exists perfectly binding scheme and perfectly hiding scheme. Therefore, we have

Theorem 36. *Assuming that one of the problems $SIVP$ and $GapSVP$ is hard for any quantum algorithm, then there exists a protocol for OT_h^n with security against any quantum algorithm.*

6.3 A Construction Under The Decisional Diffie-Hellman Assumption

6.3.1 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 are the generator, the order, the operation of the group.

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

6.3.2 Detailed Construction

We describe our construction of hash system based on DDH as follows.

- $PG(1^k)$: $\Lambda \leftarrow Gen(1^k)$, finally outputs Λ.

- $IS(1^k, \Lambda)$: $(g, q, *) \leftarrow \Lambda$, $a_i \in_U Z_q \ \forall \in [n]$, $b_i \in_U Z_q \ \forall i \in [n]$, $c_i \leftarrow a_i b_i$ $\forall i \in [h]$, $\dot{x}_i \leftarrow (g^{a_i}, g^{b_i}, g^{c_i})$ $\forall i \in [h]$, $\dot{w}_i \leftarrow (a_i, b_i)$ $\forall i \in [h]$, $c_i \in_U Z_q$ $\forall i \in [n]-[h]$, $\ddot{x}_i \leftarrow (g^{a_i}, g^{b_i}, g^{c_i})$ $\forall i \in [n]-[h]$, $\ddot{w}_i \leftarrow (a_i, b_i)$ $\forall i \in [n]-[h]$, finally outputs $((\dot{x}_1, \dot{w}_1), \ldots, (\dot{x}_h, \dot{w}_h), (\ddot{x}_{h+1}, \ddot{w}_{h+1}), \ldots, (\ddot{x}_n, \ddot{w}_n))$.

- $VF(1^k, \Lambda, x, w)$: $(g, q, *) \leftarrow \Lambda$, $(\alpha, \beta, \gamma) \leftarrow x$, $(a, b) \leftarrow w$, 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.

Theorem 37. *Assuming DDH is a hard problem, the hash system is a* $SPWH_{h,t}$.

The proof of this theorem can be done in the same way as that of Theorem 34. So we don't iterate here.

To gain a concrete protocol for OT^n_h based only on DDH, it remains to instantiate PHC and PBC with the ones builded on DDH. The commitment scheme [33] presents is an concrete PHC we need. The encryption scheme [10] presents 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. What is more, this encryption scheme can be used as an concrete PBC. Therefore, using those two commitment schemes and our DDH-based $SPWH_{h,t}$, we gain an protocol for OT^n_h based only on DDH. Considering the efficiency, we recommend DDH of the group which is on elliptic curves.

6.4 A Construction Under The Decisional N-th Residuosity Assumption

6.4.1 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), presented in [32], is how to construct an algorithm to distinguish the two probability ensembles $DNR_1 \overset{def}{=} \{DNR_1(1^k)\}_{k\in\mathbb{N}}$ and $DNR_2 \overset{def}{=} \{DNR_2(1^k)\}_{k\in\mathbb{N}}$ 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 $DNR_1 \overset{c}{=} DNR_2$.

The hash system we will construct is an instantiation of ϵ-$UPH_{h,t}$. We will build it on a DNR-based instantiation, presented by [21], of ε-VUPH. ϵ-$UPH_{1,1}$ is different from ε-VUPH in a similar way that $SPWH_{1,1}$ is different from $VSPH$. Thus, the definition of ε-VUPH is easy to deduced, and we omit them here. Please see [21] for its detailed definition.

The instantiation of ε-VUPH is stated as follows [21], where $\varepsilon < 1$.

- $PG(1^k)$: $N \leftarrow Gen(1^k)$, $a \in_U Z^*_{N^2}$, $T \leftarrow N^{\ulcorner 2\log N \urcorner}$, $g \leftarrow a^{N\cdot T} \mod N^2$, $\Lambda \leftarrow (N, g)$, 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.
- $KG(1^k, \Lambda, x)$: $(N, g) \leftarrow \Lambda$, $hk \in_U Z_{N^2}$, $pk \leftarrow g^{hk} \mod N^2$, 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.

The IT holds a property called verifiability, which is described as follows.

1. For any $\Lambda \in Rang(PG(.))$, any $(w, \dot{x}, \ddot{x}) \in Rang(IS(1^k.\Lambda))$, it holds that $IT(\Lambda, \dot{x}, \ddot{x}) = IT(\Lambda, \ddot{x}, \dot{x}) = 1$.
2. For any Λ, $\dot{x}$, $\ddot{x}$ such that $IT(\Lambda, \ddot{x}, \dot{x}) = 1$, it holds that the hash system is either $\varepsilon(|\Lambda|)$- universal on $(\Lambda, \dot{x})$ or $\varepsilon(|\Lambda|)$- universal on $(\Lambda, \ddot{x})$.

Note that in the hash system, $\dot{x}$ is projective and $\dot{x}$ is universal. What is more, the ε-universality, projection are contradictory. Therefore, [21] indirectly proves the following lemma.

Lemma 38. *Let* $L \overset{def}{=} \{x | (N, g) \leftarrow PG(1^k), r \in_U Z_N^*, w \leftarrow r, x \leftarrow g^w \mod N^2\}$ *and* $L' \overset{def}{=} \{x | (N, g) \leftarrow PG(1^k), r, v \in_U Z_N^*, w \leftarrow r, x \leftarrow g^w(1+vN) \mod N^2\}$. *Then*

$$L \cap L' = \emptyset$$

6.4.2 Detailed Construction

Recalling theorem 22, to gain a $SPWH_{h,n}$, what we need to do is to construct a ϵ-$UPWH_{h,n}$ first, then transform it into a $SPWH_{h,n}$ using the algorithm guaranteed by the theorem. In this section, we construct an instantiation of ϵ-$UPWH_{h,n}$ based on DNR.

We now present our construction of hash system under the DNR assumption as follows.

- $PG(1^k)$: $N \leftarrow Gen(1^k)$, $a \in_U Z_{N^2}^*$, $T \leftarrow N^{\lceil 2 \log N \rceil}$, $g \leftarrow a^{N \cdot T} \mod N^2$, $\Lambda \leftarrow (N, g)$, finally outputs Λ.

- $IS(1^k, \Lambda)$: $(N, g) \leftarrow \Lambda$, $r_i \in_U Z_N^*$ $\forall i \in [n]$, $\dot{x}_i \leftarrow g^{r_i} \mod N^2$ $\forall i \in [h]$, $\dot{w}_i \leftarrow (r_i, 0)$ $\forall i \in [n] - [h]$, $v_i \in_U Z_N^*$ $\forall i \in [n] - [h]$, $\ddot{x}_i \leftarrow g^{r_i}(1 + v_i N) \mod N^2$ $\forall i \in [n] - [h]$, $\ddot{w}_i \leftarrow (r_i, v_i)$ $\forall i \in [n] - [h]$, finally outputs $((\dot{x}_1, \dot{w}_1), \ldots, (\dot{x}_h, \dot{w}_h), (\ddot{x}_{h+1}, \ddot{w}_{h+1}), \ldots, (\ddot{x}_n, \ddot{w}_n))$.

- $VF(1^k, \Lambda, x, w)$: $(N, g) \leftarrow \Lambda$, $(r, v) \leftarrow w$,
  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.

We now prove that the scheme above is a ϵ-$UPWH_{h,n}$, where $\epsilon < 1$.

Proposition 39. *Assume that DNR is a hard problem, then VF computes the function ζ defined in Definition 4.*

According to lemma 38, it is easy to see this proposition holds.

Proposition 40. *The hash system holds the property projection and ε-universality, where $\varepsilon < 1$.*

These properties holded by the hash system are directly inherited from the instantiation of ϵ-$UPH_{h,t}$.

Proposition 41. *Assuming DNR is a hard problem, the hash system holds the property hard subset membership.*

Proof. We are to use Theorem 27 to prove this proposition holds. It is easy to see that this $SPWH_{h,n}$ meets the first two requirements, so it remains to prove that $SPWH_{h,n}$ also holds the last requirement, i.e. $HSM^1 \overset{c}{=} HSM^{h+1}$. For this case, HSM^1, HSM^{h+1} are where

- $HSM^1(1^k)$: $\Lambda \leftarrow PG(1^k)$, $(N,g) \leftarrow \Lambda$, $r_1 \in_U Z_N^*$, $\dot{x}_1 \leftarrow g^{r_1} \mod N^2$. Finally outputs $(\Lambda, \dot{x}_1)$.
- $HSM^{h+1}(1^k)$: $\Lambda \leftarrow PG(1^k)$, $(N,g) \leftarrow \Lambda$, $r_{h+1}, v_{h+1} \in_U Z_N^*$, $\ddot{x}_{h+1} \leftarrow g^{r_{h+1}}(1+v_{h+1}N) \mod N^2$. Finally outputs $(\Lambda, \ddot{x}_{h+1})$.

It is clear that $HSM^1 \overset{c}{=} HSM^{h+1}$. □

Combining propositions above, we have the following theorem.

Theorem 42. *Assuming DNR is a hard problem, the hash system is a ϵ-$UPWH_{h,n}$, where $\epsilon < 1$.*

6.5 A Construction Under The Decisional Quadratic Residuosity Assumption

We reuse $Gen(1^k)$ defined in section 6.4.1. The problem decisional quadratic residuosity (DQR) is how to construct an algorithm to distinguish the two probability ensembles $QR_1 \overset{def}{=} \{QR_1(1^k)\}_{k\in\mathbb{N}}$ and $QR_2 \overset{def}{=} \{QR_2(1^k)\}_{k\in\mathbb{N}}$ which are formulated as follows.

- $QR_1(1^k)$: $N \leftarrow Gen(1^k)$, $x \in_U Z_N^*$, finally outputs (N, x).
- $QR_2(1^k)$: $N \leftarrow Gen(1^k)$, $r \in_U Z_N^*$, $x \leftarrow r^2 \mod N$, finally outputs (N, x).

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

As in section 6.4, the hash system we aim to achieve is an instantiation of ϵ-$UPH_{h,t}$. We will build it on an instantiation of ε-VUPH presented by [21] which is constructed under DQR assumption.

Our hash system is described 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.

- $IS(1^k, \Lambda)$: $(N, g) \leftarrow \Lambda$, $r_i \in_U Z_N \ \forall i \in [n]$, $\dot{x}_i \leftarrow g^{r_i} \mod N \ \forall i \in [h]$, $\dot{w}_i \leftarrow r_i \ \forall i \in [n]-[h]$, $\ddot{x}_i \leftarrow N - g^{r_i} \mod N \ \forall i \in [n]-[h]$, $\ddot{w}_i \leftarrow r_i \ \forall i \in [n]-[h]$, finally outputs $((\dot{x}_1, \dot{w}_1), \ldots, (\dot{x}_h, \dot{w}_h), (\ddot{x}_{h+1}, \ddot{w}_{h+1}), \ldots, (\ddot{x}_n, \ddot{w}_n))$.

- $VF(1^k, \Lambda, x, w)$: $(N, g) \leftarrow \Lambda$, $(r, v) \leftarrow w$; 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.

The fact that the hash system above is a ϵ-$UPH_{h,t}$ ($\epsilon < 1$) can be proven in a similar way in which Theorem 42 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] B. Barak and Y. Lindell. Strict Polynomial-time in Simulation and Extraction. *SIAM Journal on Computing*, 33(4):783–818, 2004.

[3] G Brassard, C Crepeau, R Jozsa, and D Langlois. A quantum bit commitment scheme provably unbreakable by both parties. In *FOCS 1993*, volume 1, page 362. IEEE Computer Society, 1993. 34th Annual Symposium on Foundations of Computer Science: November 3-5, 1993, Palo Alto, California: proceedings [papers].

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

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

[6] 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.

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

[8] 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.

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

[10] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

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

[12] 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.

[13] 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.

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

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

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

[17] 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.

[18] 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.

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

[20] 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.

[21] 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. .

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

[23] 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.

[24] 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.

[25] 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.

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

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

[28] 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.

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

[30] 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.

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

[32] 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.

[33] 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.

[34] 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.

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

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

[37] 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.

[38] 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.

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

[40] 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.

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