A Framework for Fully-Simulatable t-out-of-n Oblivious Transfer

Bing Zeng[a], Christophe Tartary[b], Chingfang Hsu[c], Xueming Tang[d]

[a]*School of Software Engineering, South China University of Technology, Guangzhou, 510006, China*
[b]*School of Architecture, Computing and Engineering, University of East London, London, United Kingdom*
[c]*Computer School, Central China Normal University, Wuhan, 430079, China*
[d]*School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China*

Abstract

Oblivious transfer is a fundamental building block for multiparty computation protocols. In this paper, we present a generally realizable framework for fully-simulatable t-out-of-n oblivious transfer (OT_t^n) with security against non-adaptive malicious adversaries in the plain mode. Our construction relies on a single cryptographic primitive which is a variant of smooth projective hashing (SPH). A direct consequence of our work is that the existence of protocols for OT_t^n is reduced to the existence of this SPH variant. Before this paper, the only known reductions provided half-simulatable security and every known efficient protocol involved at least two distinct cryptographic primitives. We show how to instantiate this new SPH variant under not only the decisional Diffie-Hellman assumption, the decisional N-th residuosity assumption and the decisional quadratic residuosity assumption as currently existing SPH constructions, but also the learning with errors problem. Our framework only needs 4 communication rounds, which implies that it is more round-efficient than known protocols holding identical features.

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

1. Introduction

1.1. Secure Oblivious Transfer

Oblivious transfer (OT), introduced in [1], is a fundamental cryptographic primitive allowing the secure multiparty computation (MPC) of *any* computable function [2]. Beside this completeness result by Kilian, OT exhibits interests on its own since this primitive is generally used as a building block in a variety of security protocols: secure computation of the median [3], privacy-preserving data publishing [4], privacy-preserving set operations [5], electronic commerce [6], private mutual authentication [7], privacy preserving data mining [8], database search [9, 10], oblivious keyword search [11], oblivious polynomial evaluation [12], contract signing [13], oblivious sampling [14, 9].

In MPC, two categories of adversaries are usually considered: *semi-honest* (also called *honest-but-curious*) and *malicious*. In both cases, the adversary's goal is to learn more information using the transcript of the computation than what is inferred by his private input and the result of the computation. However, there is a fundamental difference between these two models: a semi-honest adversary always executes the steps of the MPC protocol faithfully while a malicious enemy can arbitrarily deviate from them [15]. A standard way of proving the security of a MPC protocol is to use the ideal/real model paradigm. In this context, adversaries in the real world are demonstrated to be equivalent to enemies in the ideal world where the computation is executed by an incorruptible entity named *ideal functionality*. As the MPC protocol (in the real world) simulates the ideal world, the security is said to be *fully-simulatable*. A relaxation of this model appears in the 2-party case with the notion of *half-simulatability* where simulation-based security is guaranteed if the sender is corrupted while privacy is ensured when the receiver is corrupted. However, this simpler

Email addresses: zeng.bing.zb@gmail.com (Bing Zeng), c.tartary@uel.ac.uk (Christophe Tartary), cherryjingfang@gmail.com (Chingfang Hsu), tang.xueming.txm@gmail.com (Xueming Tang)

security model exhibits some major issues. In particular, half-simulatable protocols cannot offer the same composability properties as fully-simulatable ones since composability depends on full-simulation. As a consequence, such protocols are hard to use as building blocks of more complex functionalities. In addition, half-simulatable protocols for OT are vulnerable to selective-failure attacks where the sender causes a failure depending on the receiver's selection [9]. Such weaknesses do not appear in fully-simulatable protocols as the ideal/real world paradigm only makes assumptions about the adversary's computational power rather than about some concrete attacks.

OT is a particular family of 2-party functionalities. This primitive has the property that we can convert a fully-simulatable OT protocol Π having *security against a semi-honest adversary* (SASHA) into a fully-simulatable OT protocol Π' with *security against a malicious adversary* (SAMA) thanks to enhanced trapdoor permutations [16]. The issue with this approach is the fact that it relies on the use of sub-functionalities for coin tossing and commitment schemes whose security relies on *zero-knowledge* (ZK) proof systems for $\mathcal{NP}$-statements. Thus, for efficiency purposes, one usually prefers designing an OT protocol with SAMA from the scratch rather than first exclusively focussing on passive adversaries and then using the above conversion technique.

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

The first efficient protocols for OT were independently presented in [17] and [6]. The first fully-simulatable protocol for OT_t^n with adaptive adversaries was presented in [18]. However, the underlying intractability assumptions were non-standard. Recently, Green and Hohenberger presented a protocol for adaptive OT_t^n under a simple, static assumption [10]. The first fully-simulatable protocol based on a standard assumption was proposed in [19]. Later, Lindell constructed two protocols for OT_1^2. One relies on the decisional Diffie-Hellmann (DDH) assumption and the other assumes the existence of homomorphic encryption schemes with some additional properties, perfectly binding commitments and perfectly hiding commitments. Recently, Lindell and Pinkas [20] proposed an efficient DDH-based instantiation of [21] in the *plain model* (i.e., with no set-up assumptions beyond that of authenticated communication). Despite the loss of the universal composability property, it is remarkably the most efficient protocol for fully-simulatable OT_1^2 in the plain model. In a practical point of view, the plain model approach is needed. Indeed, protocols with set-up require this trusted phase to occur before the actual computation. In many settings (e.g., peer-to-peer network), this is impractical. To illustrate this issue, consider the protocol for OT_1^2 appearing in [21]. It is based on the *common reference string* (CRS) model. However, [22, 23] showed that, even given an authenticated communication channel, it was impossible to implement a universally composable protocol providing a useful CRS in the presence of malicious adversaries. In addition, despite [21] suggested to get a CRS from natural processes, no formal proof of security was provided and [24] also pointed out that, although certain natural events could be viewed as producing bit-sources with high min-entropy, the resulting bit-sources might not be uniformly random.

1.2. Our Contribution

We are to present a framework Π for OT_t^n with fully-simulatable SAMA. Our construction is efficient enough for practical use and it works in the plain model. Our starting point is the construction proposed in [25] (preliminary version [26]) for OT_1^2 with half-simulatable SAMA. To save space in the current paper, we assume that the reader is familiar with this work.

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

1. We present a generally realizable framework for fully-simulatable t-out-of-n oblivious transfer (OT_t^n) with SAMA. Our framework is more round-efficient than known protocols holding identical features, and is more computationally efficient than them under the decisional N-th residuosity assumption, the decisional quadratic residuosity assumption, and the learning with errors problem. See Section 1.2.4.

2. We present a new smooth projective hash (SPH) variant, and the first reduction OT_t^n to SPH with fully-simulatable security.
3. We instantiate this SPH variant under not only the decisional Diffie-Hellman assumption, the decisional N-th residuosity assumption and the decisional quadratic residuosity assumption as currently existing SPH constructions, but also the learning with errors problem.

1.2.1. Cryptographic Approach

The main cryptographic tool used by Halevi and Tauman Kalai is a *smooth projective hash family* (SPH), a notion introduced by Cramer and Shoup [27] to construct efficient public-key encryption scheme secure against adaptive chosen ciphertext attacks. The variant used in [25] was called *verifiably smooth projective hash family*. It deals with two types of instances (smooth and projective) which are computationally indistinguishable due to a property called *hard subset membership*. Nonetheless, another property called *verifiable smoothness* provides a way to verify whether at least one of a two instances is smooth. In the remaining of this paper, we are to denote verifiably smooth projective hash family with hard membership property by VSPH-HM.

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

Despite the notion of VSPH-HM can be used to deal with OT_1^n, it seems difficult to extend it to handle the general case OT_t^n. The reason is that, to hold verifiable smoothness, both types of instances have to be generated in a dependent way. This makes it difficult to design an algorithm checking that at least t of n arbitrary instances are smooth without leaking any information to the adversaries which could be used to distinguish smooth instances from projective instances. Therefore, even constructing a protocol for OT_t^n that is half-simulatable as in [25] seems difficult.

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

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

We would like to recall that, in [25], the receiver learns the value it queried about via a projective instance. For a smooth-projective instance pair, if the witness pair is available, the simulator can identify the projective instance, and hence the simulator can learn which value the adversary chooses (i.e. it learns the adversary's real input). Employing a cut-and-choose technique as in [28, 29], the simulator can see the witnesses by rewinding the adversary's computation. Our idea is for the receiver to "cut" some instance vectors (where each one contains t projective instances and $n - t$ smooth instances) and for the sender to "choose" some instance vectors at random to check their *legalities* (i.e., the sender checks that each vector indeed contains at least $n - t$ smooth instances). The receiver then sends the chosen instance vectors' witnesses. Combining the previous analysis, we can see that for a protocol constructed following this idea, the simulator can extract the adversary's real input, and hence simulation-based security can be gained in the case where only the receiver is corrupted.

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

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

1.2.2. Reducing OT to SPH

One of our theoretical contributions is that we reduce the existence of OT_t^n to the existence of SPH. This follows the fact that Π only employs one cryptographic primitive (namely, SPH-DHM). Before this paper, only one such a reduction was known and it only guaranteed half-simulatable security [25]. We would like to point out that all known efficient protocols for OT in the plain model [18, 19, 28, 20, 10] use at least two different cryptographic primitives, such as ZK proof of knowledge, commitment and signature.

Though we use a cut-and-choose approach as [28, 29], we removed the need of the coin-tossing protocol which involved two additional cryptographic primitives (a perfectly binding commitment scheme and a perfectly hiding commitment scheme). In [28, 29], the objects are chosen by the sender and receiver via a coin-tossing protocol. The underlying motivations were that, in the case where only the sender was corrupted, the simulator could gain a good coin-tossing result by rewinding the adversary in order to extract the adversary's real input through one time. Thus, these work could achieve simulation-based security in this case.

However, in [25], it is feasible for the simulator to extract the adversary's real input through multiple times rather than one time. The views of the sender interacting with the receiver with distinct private inputs are computationally indistinguishable. Therefore, Halevi and Tauman Kalai gave an idea (without proof) that let the simulator take distinct indexes as private input in distinct rewinds and extracts one of the adversary's two private values in each rewind.

Observing this, we will remove the coin-tossing protocol and allow only the sender to choose objects in Π. However, we emphasize that Halevi and Tauman Kalai's idea does not work well to provide the simulation-based security in the case where only the sender is corrupted as one may expect. This is due to the fact there are two subtle problems to be considered. First, the adversary's private values may be distinct in different rewinds. Second, the adversary may refuse to interact with the simulator in some rewinds and may not in some rewind. Our solutions are very technical. We solve the first problem by formally proving that the simulation-based security does not depend on that the adversary's private values are the same in distinct rewinds. Concerning dealing with the second problem, we use a technique originating from [30]

1.2.3. Instantiation of the Framework

We will present an instantiation of a SPH-DHM based on the learning with errors (LWE) assumption following the original design of the public key cryptosystem from [31]. Prior to our construction, it was thought to be technically difficult to instantiate a SPH under this assumption. We observed that the algorithm $\mathsf{Hash}(.)$ (computing hash values) could be viewed as an encryption algorithm while $\mathsf{pHash}(.)$ (computing projection values) could be interpreted as a decryption algorithm. Our instantiation idea is to take public-private key pairs as projective instance-witness pairs

and take messy public-private key pairs[1] as smooth instance-witness pairs. With this identification, we can see that normal public keys and messy public keys can provide projection and smoothness, respectively. To implement this idea, we let the key generator (i.e., the algorithm generating the hash-projection key pairs) take an instance as a part of its input. This is a technical difference between a SPH-DHM and a SPH. Besides the LWE assumption, we also show that a SPH-DHM can be instantiated under the DDH assumption, the decisional N-th residuosity (DNR) assumption and the decisional quadratic residuosity (DQR) assumption as for previous existing SPH.

1.2.4. Efficiency of the Construction

Our protocol Π for OT_t^n costs 4 communication rounds. In practice, it expectedly costs $20\,n$ encryptions and $20\,t$ decryptions. In the particular case of OT_1^2, its communication rounds (respectively, expected computational overhead) is twice (respectively, 20 times) of [25].

Known constructions with identical features as ours appear in [19] (OT_t^n), [28] (OT_1^2) and [20] (OT_1^2). Considering the number of communication rounds, our framework Π is more efficient as it respectively costs 8, 2, 2 rounds *less* than these schemes. On the computational overhead side, Π is *more efficient* than [19, 28] but *less efficient* than [20]. This follows the fact that Lindell's scheme is already more efficient than [19] and that our protocol improves the cut-and-choose technique used in [28, 29]. Since Lindell and Pinkas' scheme [20] is only DDH-based and Green and Hohenberger's relies on the decisional bilinear Diffie-Hellman assumption, we conclude that, under the DDH assumption, the construction from [20] is the best approach for OT_1^2, while under other intractability assumptions quoted above, our framework Π is the most efficient one for OT_t^n.

1.2.5. Related Approach

In [32], Zeng *et al.* presented a framework for OT_t^n secure against covert adversaries whose design is close to the protocol presented in this paper. In [33], Aumann and Lindell proved that, for any protocol, if its deterrence factor to covert adversaries was $\epsilon = 1 - \mu(k)$ (where $\mu(\cdot)$ is a negligible function of the security parameter), then the protocol was also secure against malicious adversaries. With this result in mind, one might naturally wonder whether it would be possible to get such a good deterrence factor for [32] via a simple parameter setting to ensure [32]'s security against malicious adversaries. Unfortunately, this straightforward idea does not allow for [32]'s formal security proof to hold. For example, in the case where only the sender is corrupted, following Theorem 23 and the proof of Lemma 24 in [32], the expected running time of the simulator would be $\binom{K}{K-g} = 1/(1-\epsilon)$, where K and g are statistical security parameters. If $\epsilon = 1 - \mu(k)$, then the expected running time $\binom{K}{K-g} = 1/\mu(k)$ is greater than any positive polynomial in the security parameter. As a consequence, in the current paper, to obtain security against malicious adversaries, the ideas behind formal security proofs, which are discussed above, are totally different from [32].

In [34], which is a early version of parts of this paper, we claim that if the instance vectors to be open are chosen by only the sender, it *seems* impossible to provide simulation-based proof of security level against malicious adversaries, which contradicts this paper. However, as explicitly stated in [34], the precondition of this impossibility claim is to follow [32]'s idea to construct the simulator. As clarified above, the current ideas are totally different from [32]. Thus, this paper does not contradict the impossibility claim.

Table1 summarizes the relations between [32, 34] and this paper.

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

[1] A key is called *messy* if a ciphertext generated under such a key carries no information (statistically) about the corresponding plaintext.

Table 1: Relations between [32, 34] and this paper

Paper	Protocol	Type of adversaries	Simulatability	Relations to others
[32]	OT	covert	full	
[34]	the first OT	covert	full	Published in [32]. Uses this paper's SPH
	the second OT	malicious	half	Not published. Early version of parts of this paper
This paper	OT	malicious	full	
	a SPH variant			

1.3. Paper Organization

In the next section, we describe the notations used throughout our work and the security definition for OT_t^n. In Section 3, we define a new variant of smooth projective hash (i.e. SPH-DHM). Our framework Π for OT_t^n is exposed in Section 4 and its security is demonstrated in Section 5. In Section 6, we reduce the construction of SPH-DHM to considerably simpler hash families. In Section 7, we instantiate those simpler families under the DDH, LWE, DQR, DNR assumptions, respectively. Finally, the last section concludes with some open problems.

2. Preliminaries

Most notations and concepts mentioned in this section come from [35, 16, 36] and we tailored them to deal with OT_t^n.

2.1. Basic Notations and Definitions

We set the following notations for this paper:

- $\mathbb{N}$: set of natural numbers.
- k: *security parameter* where $k \in \mathbb{N}$. It is used to measure the security of the underlying computational assumptions (e.g., the DDH assumption).
- K: *statistical security parameter* where $K \in \mathbb{N}$. This is the number of instance vectors that the receiver "cuts", and so K defines the size of our "cut-and-choose" test. Thus, K is used to measure the probability that the adversary is not caught in cut-and-choose type checks (see Lemma 25). Note that this probability does not depend on any computational intractability assumption.
- $[i]$: the set $\{1, 2, \ldots, i\}$ where $i \in \mathbb{N}$.
- $\vec{x}\langle j\rangle$: the j-th entry of the vector $\vec{x}$.
- S_n: the set of all permutations of $[n]$ where $n \in \mathbb{N}$.
- $\sigma(\vec{x})$: the vector gained by shifting the i-th entry of the n-vector $\vec{x}$ to the $\sigma(i)$-th entry where $\sigma \in S_n$. In other words, $\sigma(\vec{x})$ denotes the vector $\vec{y}$ such that: $\forall i \in [n] \quad \vec{y}\langle\sigma(i)\rangle = \vec{x}\langle i\rangle$.
- $\mathsf{Poly}(.)$: an unspecified positive polynomial.
- $\{0,1\}^*$: set of all bitstrings.
- $\alpha \in_U D$: an element α chosen uniformly at random from a domain D.
- $\alpha \in_\chi D$: an element α chosen from a domain D according the probabilistic distribution χ.
- $|X|$: the cardinality of a finite set X.

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

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

Definition 2. *A* probability ensemble

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

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

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

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

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

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

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

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

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

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

The OT_t^n functionality is defined as follows.

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

where:

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

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

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

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

2.2.1. The Ideal World

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

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

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

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

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

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

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

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

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

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

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

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

where

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

2.2.2. The Real World

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

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

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

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

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

2.2.3. Security Definition

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

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

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

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

We point out that the security definitions presented in [16, 36] require the simulator $\mathfrak{S}$ to run in strictly polynomial-time but those from [37, 29, 28] allow $\mathfrak{S}$ to run in expected polynomial-time. Definition 9 follows the latter. We argue about our choice as follows. First, allowing the simulator to run in expected polynomial-time is essential for achieving (non-trivial) constant-round protocols (our framework Π has constant round complexity) as Barak and Lindell showed that there was no (non-trivial) constant-round ZK proof of argument having a strictly polynomial-time black-box simulator [38]. Second, in many cases (also when strictly polynomial-time simulators exist), the expected running time of the simulator provides a better bound than the worst-case running time [39].

3. A New Smooth Projective Hash

As said in Section 1, SPH was introduced to design chosen-ciphertext secure encryption schemes and Halevi and Tauman Kalai applied a variant of this cryptographic primitive to construct a protocol for OT. However, these definitions does not suffice for our application. Rather, we define another variant of SPH.

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

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

- *Key generator* KG*: it takes a security parameter k, a hash parameter* Λ *and an instance x as input and outputs a hash-projection key pair* (hk, pk)*: i.e.,* $(hk, pk) \leftarrow \mathsf{KG}(1^k, \Lambda, x)$.
- *Hash algorithm* Hash*: it takes a security parameter k, a hash parameter* Λ*, an instance x and a hash key hk as input and outputs a value y: i.e.,* $y \leftarrow \mathsf{Hash}(1^k, \Lambda, x, hk)$.
- *Projection algorithm* pHash*: it takes a security parameter k, a hash parameter* Λ*, an instance x, a projection key pk and a witness w of x as input and outputs a value y: i.e.,* $y \leftarrow \mathsf{pHash}(1^k, \Lambda, x, pk, w)$.

Definition 11. *For a given hash parameter* Λ*, if* Check *outputs* 1*, then* Λ *is said to be* legal*; otherwise, it is said to be* illegal.

Remark 12. *It is obvious that any* Λ *generated by* PG *is legal.*

Definition 13. *Let* $R = \{(x, w) : x, w \in \{0,1\}^*\}$ *be a relation. For a legal* Λ*, we define its* projective relation *as* $\dot{R}_\Lambda = \{(\dot{x}, \dot{w}) : (\dot{x}, \dot{w})$ *is generated by* $\mathsf{IS}(1^k, \Lambda)\}$ *and its* smooth relation *as* $\ddot{R}_\Lambda = \{(\ddot{x}, \ddot{w}) : (\ddot{x}, \ddot{w})$ *is generated by* $\mathsf{IS}(1^k, \Lambda)\}$.

Definition 14 (Distinguishability)**.** *For any legal hash parameter* Λ*, any instance-witness pair* (x, w)*, we require:*

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

Definition 15. *If R is a relation, then its* language *is defined as* $L \stackrel{def}{=} \{x \in \{0,1\}^* : \exists w((x, w) \in R)\}$.

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

Definition 16 (Projection)**.** *For any hash parameter* Λ *generated by* $\mathsf{PG}(1^k)$*, any projective instance-witness pair* $(\dot{x}, \dot{w})$ *generated by* $\mathsf{IS}(1^k, \Lambda)$*, and any hash-projection key pair* (hk, pk) *generated by* $\mathsf{KG}(1^k, \Lambda, \dot{x})$*, it holds that*

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

Definition 17. *For an instance-witness vector* $\vec{a} = ((x_1, w_1), \ldots, (x_n, w_n))$*, we define its* instance vector *as* $\vec{x}^{\vec{a}} \stackrel{def}{=} (x_1, \ldots, x_n)$ *and its* witness vector *as* $\vec{w}^{\vec{a}} \stackrel{def}{=} (w_1, \ldots, w_n)$.

Definition 18. *Fix a legal hash parameter* Λ*. If a vector* $\vec{a}$ *contain at least* $n - t$ *smooth instance-witness pairs, (i.e., at least* $n - t$ *pairs in* $\ddot{R}_\Lambda$*), then* $\vec{a}$ *is said to be* legal.

Remark 19. *Note that any* $\vec{a}$ *generated by* $\mathsf{IS}(1^k, \Lambda)$ *is legal, and the legality of any* $\vec{a}$ *that may be maliciously generated can be checked by invoking algorithm* DI *at most n times.*

Definition 20 (Smoothness)**.** *For any legal hash parameter* Λ*, any legal instance-witness vector* $\vec{a}$ *(without loss of generality, we assume that the last* $n - t$ *entries of* $\vec{a}$ *are smooth), any permutation* $\sigma \in S_n$*,* smoothness *holds if the two probability ensembles* $\mathsf{SM}_1 \stackrel{def}{=} \{\mathsf{SM}_1(1^k)\}_{k\in\mathbb{N}}$ *and* $\mathsf{SM}_2 \stackrel{def}{=} \{\mathsf{SM}_2(1^k)\}_{k\in\mathbb{N}}$*, specified as follows, are statistically indistinguishable: i.e.,* $\mathsf{SM}_1 \stackrel{s}{=} \mathsf{SM}_2$. Perfect smoothness *holds, if* $\mathsf{SM}_1 \equiv \mathsf{SM}_2$.

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

- *For $i \in [2]$, algorithm $\mathsf{SM}_i(1^k)$ works as follows: $\overrightarrow{pky} \leftarrow SmGen_i(1^k)$, $\widetilde{\overrightarrow{pky}} \leftarrow \sigma(\overrightarrow{pky})$ and output $\widetilde{\overrightarrow{pky}}$.*

Definition 21 (Hard Subset Membership)**.** *For any $\sigma \in S_n$, the two probability ensembles $\mathsf{HS}_1 \stackrel{def}{=} \{\mathsf{HS}_1(1^k)\}_{k\in\mathbb{N}}$ and $\mathsf{HS}_2 \stackrel{def}{=} \{\mathsf{HS}_2(1^k)\}_{k\in\mathbb{N}}$, specified as follows, are computationally indistinguishable, i.e., $\mathsf{HS}_1 \stackrel{c}{=} \mathsf{HS}_2$.*

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

Remark 22. *In this paper, for a projective instance $\dot{x}$, its witness $\dot{w}$ is mainly used to gain the instance's hash value while, for a smooth instance $\ddot{x}$, its witness $\ddot{w}$ serves as a proof of smoothness (i.e., a proof of $\ddot{x} \in \ddot{L}_\Lambda$). The distinguishability property guarantees that, given the needed witness-vector, projective instances and smooth instances are distinguishable.*

As both our SPH-DHM notion and Halevi and Tauman Kalai's VSPH-HM [25] are used to construct protocols for OT, it is necessary to discuss the differences between these two variants of SPH.

1. The major difference between these two notions is that a SPH-DHM not only provides a witness to each projective instance (as a VSPH-HM) but also to every smooth instance.
2. The key generation algorithm KG of a SPH-DHM takes an additional parameter: an instance x. This technical modification makes instantiating such a hash family easier as we will see in Section 7 using the LWE assumption.
3. The instance sampling algorithm IS of a VSPH-HM generates tuples consisting of a smooth instance, a projective instance and its witness. To deal with OT_t^n, in a SPH-DHM, the sampling algorithm returns vectors containing t projective instance-witness pairs and $n - t$ smooth instance-witness pairs. As a natural result, the properties of smoothness and hard subset membership are extended to consider instance vectors of n entries.
4. A SPH-DHM does not need the verifiable smoothness property of a VSPH-HM (implemented by algorithm IT in [25]). This property was used by Halevi and Tauman Kalai to verify whether at least one of two instances is smooth. Instead, a SPH-DHM exhibits the distinguishability property (implemented by algorithm DI).

4. A Framework for OT_t^n with Fully-Simulatable SAMA

4.1. Description of the Protocol

Let $\mathcal{H} = (\mathsf{PG}, \mathsf{IS}, \mathsf{Check}, \mathsf{DI}, \mathsf{KG}, \mathsf{Hash}, \mathsf{pHash})$ be a SPH-DHM. For our construction, we will need a PPT algorithm Γ taking two legal inputs $T_1, T_2 \in \Psi$ of the receiver as input and outputting a uniformly chosen permutation $\sigma \in S_n$ mapping T_1's indexes to T_2's. We will use the notation $\sigma \leftarrow \Gamma(T_1, T_2)$ to denote a call to this algorithm. For an implementation example of Γ, please see [32].

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

- Private Inputs: The sender $\mathcal{S}$ holds a vector $\vec{m} \in (\{0,1\}^*)^n$ containing n values and holds a random tape $r_1 \in \{0,1\}^*$. The receiver $\mathcal{R}$ holds a set $T \in \Psi$ containing t indices and holds a random tape $r_2 \in \{0,1\}^*$. The adversary $\mathcal{A}$ holds a name list $I \subseteq \{\mathcal{S}, \mathcal{R}\} = \{1, 2\}$ and a random tape $r_{\mathcal{A}} \in \{0,1\}^*$.
- Auxiliary Inputs: The adversary $\mathcal{A}$ holds an auxiliary input $z \in \{0,1\}^*$. The security parameter k, the statistical security parameter K, and the description of $\mathcal{H}$ are known to $\mathcal{R}, \mathcal{S}$ and $\mathcal{A}$.

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

Remark 23. *Note that $\mathcal{S}$ only cares about whether each chosen instance vector contains at least $n-t$ smooth instances. Thus, in Step R2, to prove the legalities of the chosen instance vectors, $\mathcal{R}$ only needs to send the witnesses of the smooth instances. Formally speaking, $\mathcal{R}$ only needs to send $((i, j, \tilde{\vec{w}}_i\langle j\rangle))_{i\in CS, j\in J_i}$ to $\mathcal{S}$, where $J_i \stackrel{def}{=} \{j : \tilde{\vec{x}}_i\langle j\rangle$ is smooth$\}$.*

Protocol 1 Π

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

1. $\mathcal{R}$ chooses a hash parameter Λ by running $\mathsf{PG}(1^k)$.
2. $\mathcal{R}$ samples K instance-witness vectors: $\forall i \in [K] \quad \vec{a}_i \leftarrow \mathsf{IS}(1^k, \Lambda)$.
3. $\mathcal{R}$ shuffles each of these K vectors. That is, for each $i \in [K]$, $\mathcal{R}$ uniformly chooses a permutation $\sigma_i^1 \in_U S_n$ and $\tilde{\vec{a}}_i \leftarrow \sigma_i^1(\vec{a}_i)$.
4. $\mathcal{R}$ sends the hash parameter Λ and the instance vectors $\tilde{\vec{x}}_1, \tilde{\vec{x}}_2, \ldots, \tilde{\vec{x}}_K$ to $\mathcal{S}$, where $\tilde{\vec{x}}_i$ is the instance vector of $\tilde{\vec{a}}_i$ (i.e., $\tilde{\vec{x}}_i = \vec{x}^{\tilde{\vec{a}}_i}$).

S1 (Sender's step): $\mathcal{S}$ verifies that the hash parameter is legal by calling $\mathsf{Check}(1^k, \Lambda)$.

1. If $\mathsf{Check}(1^k, \Lambda) = 0$, then $\mathcal{S}$ halts and outputs Abort_2.
2. If $\mathsf{Check}(1^k, \Lambda) = 1$, then $\mathcal{S}$ uniformly chooses a random string $r \in \{0,1\}^K$ and sends it to $\mathcal{R}$.

Comment: The string r indicates that the instance vectors whose indices fall into $CS \stackrel{def}{=} \{i : r\langle i\rangle = 1\}$ (similarly said, $\overline{CS} \stackrel{def}{=} \{i : r\langle i\rangle = 0\}$) are chosen by $\mathcal{S}$. In other words, r indicates that instance vectors $(\tilde{\vec{x}}_i)_{i\in CS}$ are chosen. For this reason, we call *r choice indicator*.

R2 (Receiver's step): According to the value of the string r, $\mathcal{R}$ sends the witness vectors corresponding to the chosen instance vectors and the different permutations of T to $\mathcal{S}$ as follows.

1. $\mathcal{R}$ sends the witnesses of the chosen instance vectors $(\tilde{\vec{w}}_i)_{i\in CS}$ to $\mathcal{S}$.
2. $\mathcal{R}$ shuffles his private input. For each non-chosen vector, $\mathcal{R}$ reorders it by applying a permutation, so that the set of indexes of projective instances of the resulting vector and set T be identical. Formally speaking, let $\tilde{\vec{x}}_i$ be a non-chosen vector (i.e. $i \in \overline{CS}$) and denote by G_i the set of indexes of its projective instances (i.e., G_i=$\{j : \tilde{\vec{x}}_i\langle j\rangle$ is projective$\}$). For each $i \in \overline{CS}$, $\mathcal{R}$ chooses a permutation $\sigma_i^2 \leftarrow \Gamma(G_i, T)$.
3. $\mathcal{R}$ sends the permutations, $(\sigma_i^2)_{i\in\overline{CS}}$, to $\mathcal{S}$.

S2 (Sender's step): $\mathcal{S}$ checks the legality of the chosen instance vectors and sends the encryption of $\vec{m}$ to $\mathcal{R}$ via the following instructions.

1. $\mathcal{S}$ verifies that each chosen instance vector is legal by checking if it contains at least $n - t$ smooth instances. Knowing the witness vectors $(\tilde{\vec{w}}_i)_{i\in CS}$, $\mathcal{R}$ invokes algorithm DI to check the validity of instance vectors $(\tilde{\vec{x}}_i)_{i\in CS}$. If $\mathcal{R}$ did not send the witness vectors at Step R2 or if the check fails, then $\mathcal{S}$ halts and outputs Abort_2; otherwise $\mathcal{S}$ proceeds to the next step.
2. $\mathcal{S}$ reorders the non-chosen instance vectors following the way prescribed by $\mathcal{R}$: $\forall i \in \overline{CS} \quad \tilde{\tilde{\vec{x}}}_i \leftarrow \sigma_i^2(\tilde{\vec{x}}_i)$.
3. $\mathcal{S}$ encrypts the value vector $\vec{m}$. That is, for each $i \in \overline{CS}$ and for each $j \in [n]$, $(hk_{ij}, pk_{ij}) \leftarrow \mathsf{KG}(1^k, \Lambda, \tilde{\tilde{\vec{x}}}_i\langle j\rangle)$, $\beta_{ij} \leftarrow \mathsf{Hash}(1^k, \Lambda, \tilde{\tilde{\vec{x}}}_i\langle j\rangle, hk_{ij})$, $\vec{\beta}_i \stackrel{def}{=} (\beta_{i1}, \beta_{i2}, \ldots, \beta_{in})$, $\vec{c} \leftarrow \vec{m} \oplus (\oplus_{i\in\overline{CS}}\vec{\beta}_i)$, $\overrightarrow{pk}_i \stackrel{def}{=} (pk_{i1}, pk_{i2}, \ldots, pk_{in})$.
4. $\mathcal{S}$ sends the encryption of $\vec{m}$ and the projection keys, $(\vec{c}, (\overrightarrow{pk}_i)_{i\in\overline{CS}})$, to $\mathcal{R}$.

R3 (Receiver's step) $\mathcal{R}$ decrypts $\vec{c}$. That is, for each $i \in \overline{CS}$ and $j \in T$, the receiver computes $\beta'_{ij} \leftarrow \mathsf{pHash}(1^k, \Lambda, \tilde{\tilde{\vec{x}}}_i\langle j\rangle, \overrightarrow{pk}_i\langle j\rangle, \tilde{\tilde{\vec{w}}}_i\langle j\rangle)$, $m'_j \leftarrow \vec{c}\langle j\rangle \oplus (\oplus_{i\in\overline{CS}}\beta'_{ij})$. Finally, $\mathcal{R}$ recovers the values $(m'_j)_{j\in T}$.

4.2. Analysis of Π

4.2.1. Correctness of the Protocol

We now check that, when both the sender $\mathcal{S}$ and the receiver $\mathcal{R}$ are honest, Π is sound. For each $i \in \overline{CS}$, each $j \in [n]$, we know:

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

For each $j \in T$, $\vec{\tilde{x}}_i\langle j\rangle$ is a projective instance. $\mathcal{H}$'s property projection guarantees that its projection value equals its hash value. That is to say: $\vec{\beta}_i\langle j\rangle = \beta'_{ij}$. Thus, we get: $\vec{m}\langle j\rangle = m'_j$ which proves that $\mathcal{R}$ gets $(\vec{m}\langle i\rangle)_{i\in T}$ which is the sought data.

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

4.2.2. Security of the Scheme

The security of Π can be stated as follows.

Theorem 24. *Assume that $\mathcal{H}$ is a smooth projective hash family that holds properties distinguishability and hard subset membership. Then,* Π *(i.e., Protocol 1) securely computes functionality OT^n_t in the presence of non-adaptive malicious adversaries where the sender's security is statistical and the receiver's security is computational.*

We defer the strict proof of Theorem 24 to Section 5 and simply give an intuitive analysis here. First, let's focus on the sender's security. The framework should prevent any malicious adversary controlling the corrupted receiver from learning more than t values. This is achieved by using a cut and choose technique. The following lemma shows that the probability that the malicious adversaries learn extra values is negligible.

Lemma 25. *In the case where the sender is honest and the receiver is corrupted by a malicious adversary, the probability that the adversary learns more than t values is at most* $1/2^K$.

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

1. $\mathcal{A}$ has to generate at least one illegal instance vector containing more than t projective instances. If not, $\mathcal{A}$ cannot correctly decrypt more than t encryptions due to the smoothness of $\mathcal{H}$. Without loss of generality, we assume the illegal instance vectors are $\vec{x}_{\ell_1}, \vec{x}_{\ell_2}, \ldots, \vec{x}_{\ell_d}$.
2. All illegal instance vectors are lucky not to be chosen by the sender and all the unchosen instance vectors are just the illegal instance vectors: i.e., $\overline{CS} = \{\ell_1, \ell_2, \ldots, \ell_d\}$. We prove this claim in the following two cases.
   (a) If $\overline{CS} \subset \{\ell_1, \ell_2, \ldots, \ell_d\}$, then there exists an illegal instance vector chosen by the sender who detects the cheating and $\mathcal{A}$ gains nothing.
   (b) If $\overline{CS} \supset \{\ell_1, \ell_2, \ldots, \ell_d\}$, then there exists a legal instance vector not chosen by the sender. Following the instructions of Π, this instance vector is used to encrypt the n values. Looking at Step S2, the final n encryptions are gained by XOR-ing the n values and the hash values of all unchosen instance vectors. Because a legal instance vector holds at least $n - t$ smooth instances, at least $n - t$ encryptions statistically hide their encrypted values. As a result, $\mathcal{A}$ knows almost nothing about at least $n - t$ encrypted values.

Now, let us estimate the probability that the second necessary condition is met. We have:

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

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

Second, consider the receiver's security. It is necessary for Π to prevent any malicious adversary controlling the corrupted sender from learning which values the receiver chose. Intuitively, there might be a possible information leakage in Step R2 where $\mathcal{R}$ shuffles T. Since $j \in T$ if and only if $\vec{\tilde{x}}_i$'s j-th entry is projective for each $i \in \overline{CS}$, learning some $j \in T$ means identifying a projective entry of some $\vec{\tilde{x}}_i$. Given the fact that n and t are known to both players, $\mathcal{A}$

can guess some $j \in T$ with probability t/n and can identify a projective entry of some $\tilde{\tilde{\vec{x}}}_i$ with probability t/n as well. However, $\mathcal{H}$'s hard subset membership property guarantees that $\tilde{\tilde{\vec{x}}}_i$'s projective instances and smooth instances are computationally indistinguishable. As a result, the probability of $\mathcal{A}$ identifying a projective entry of $\tilde{\tilde{\vec{x}}}_i$ is negligibly greater than t/n. Furthermore, the probability of $\mathcal{A}$ learning some $j \in T$ is negligibly greater than t/n too. In a word, $\mathcal{A}$ only knows that T is a set containing t indexes and its probability of learning extra knowledge about T is negligible in the security parameter k.

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

4.2.3. Efficiency of the Framework

It is clear that Π needs 4 communication rounds as Step R3 can be performed without communication.

Abstractly, we can see an invocation of Hash as a call to an encryption algorithm and an invocation of pHash as a request to the corresponding decryption algorithm. Such a consideration is justified since, in Π, Hash plays an encrypting role to hide $\mathcal{S}$'s values while pHash can be considered as decryption machine recovering the values that $\mathcal{R}$ wants. This parallel to a cryptosystem is not fortuitous as the first usage of a SPH was to construct public-key encryption schemes [27].

Looking at Step S1, it is easy to see that the expected number of chosen instance vector is $K/2$. Therefore, the main expected computational overhead is $K\,n/2$ encryptions at the sender (Step S2) and $K\,t/2$ decryptions at the receiver (Step R3).

Since K determines the computational overhead of Π, a natural question is how to set it in practice. In the proof of Theorem 24, the simulator $\mathfrak{S}$ does not simulate the case where the adversary learns more than t values. This is justified, because Lemma 25 shows this case arises with negligible probability $1/2^K$ (see Remark 35 for the details). Thus, we must set K to be a value so that $1/2^K$ is small enough to be negligible in the security parameter.

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

Remark 26. *How to set the security parameter k or the statistical security parameter K in practice? This is essentially the question of bridging theoretical cryptography and practice in order to translate a guarantee of asymptotic security into a concrete security one. The task of determining the value of the (statistical) security parameter to use, generally speaking, is complex and depends on the scheme in question as well as other considerations. This problem is discussed in [35, 40] for instance but no concrete standard solution is suggested.*

5. Formal Security Proof

This section is dedicated to the demonstration of Theorem 24. For notational clarity, we denote the sender, the receiver and the adversary in the real world by $\mathcal{S}$, $\mathcal{R}$, $\mathcal{A}$ and their corresponding entities in the ideal world by $\mathcal{S}'$, $\mathcal{R}'$, $\mathfrak{S}$.

Based on the parties corrupted by $\mathcal{A}$, there are four cases to be considered: only $\mathcal{S}$ is corrupted, only $\mathcal{R}$ is corrupted, both $\mathcal{S}$ and $\mathcal{R}$ are corrupted, both parties are honest. Because the security proofs of the last two cases are trivial, we omit them to save space.

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

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

To simplify the description of $\mathfrak{S}$, we assumed that during the interactions between $\overline{\mathcal{A}}$ and $\mathfrak{S}$ (i.e., from Step Sim2 onwards), if $\overline{\mathcal{A}}$ refuses to send some message it is supposed to send or $\overline{\mathcal{A}}$ sends an invalid message that $\mathfrak{S}$ cannot process, the simulator $\mathfrak{S}$ sends Abort_1 to the TTP and halts with outputting whatever $\overline{\mathcal{A}}$ returns.

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

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

Sim1 (Initialization step):

1. $\mathfrak{S}$ corrupts $\mathcal{S}'$ and learns $\mathcal{S}'$'s private input $\vec{m}$.
2. Let $\overline{\mathcal{A}}$ be a copy of $\mathcal{A}$. $\mathfrak{S}$ invokes $\overline{\mathcal{A}}$ as a subroutine with initial inputs of $\overline{\mathcal{A}}$ to be $\mathfrak{S}$'s initial inputs with the exception that the randomness of $\overline{\mathcal{A}}$ to be a uniformly distributed string. In other words, the initial inputs of $\overline{\mathcal{A}}$ are set as k, I, z_k and $r_{\overline{\mathcal{A}}}$ with $r_{\overline{\mathcal{A}}} \in_U \{0,1\}^*$.
3. $\mathfrak{S}$ activates $\overline{\mathcal{A}}$. As in the real world, $\overline{\mathcal{A}}$ sends a message to corrupt the sender. $\mathfrak{S}$ plays the role of $\mathcal{S}$ and supplies $\overline{\mathcal{A}}$ with $\vec{m}$.

Comment: To simulate the environment of $\overline{\mathcal{A}}$ in the real world, $\mathfrak{S}$ is to simultaneously plays the role of $\mathcal{S}$ and $\mathcal{R}$ to interact with $\overline{\mathcal{A}}$ during the following steps.

Sim2: Playing the role of $\mathcal{R}$, $\mathfrak{S}$ honestly follows the receiver's Step R1-R2.1 of Π to interact with $\overline{\mathcal{A}}$.

Sim3: Let $T_1, T_2, \ldots, T_d$ be arbitrary legal index sets such that $[n] = \cup_{j=1}^{d} T_j$. $\mathfrak{S}$ repeats d times the following procedure denoted by Υ.

- Procedure Υ: $\mathfrak{S}$ rewinds $\overline{\mathcal{A}}$ to the beginning of Step S2 of Π. $\mathfrak{S}$ honestly follows the receiver's Step R2.2-R3 to interact with $\overline{\mathcal{A}}$.

In each repetition, $\mathfrak{S}$ uses fresh randomness. In the j-th repetition, $\mathfrak{S}$ plays the role of $\mathcal{R}$ with private input T_j. If $\mathfrak{S}$ receive t values in this repetition, it records theses values. Otherwise, $\mathfrak{S}$ sends Abort_1 to the TTP, stops repeating procedure Υ, and outputs whatever $\overline{\mathcal{A}}$ returned.

Sim4: Using the recorded values, $\mathfrak{S}$ builds a value vector $\tilde{\vec{m}}$ as follows. $\mathfrak{S}$ takes the values obtained in the j-th repetition as $(\tilde{\vec{m}}\langle i\rangle)_{i\in T_j}$. If there are two distinct sets $T_j, T_{j'}$ such that $i \in T_j \cap T_{j'}$ for some i, then there are multiple values corresponding to the same index i and $\mathfrak{S}$ takes arbitrary one of these as $\tilde{\vec{m}}\langle i\rangle$.

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

The simulator construction (Algorithm 2) follows the ideas appearing in [25] in the case of half-simulatable OT_1^2. Nonetheless, it ignores two subtle problems.

P1 $\tilde{\vec{m}}$ is not obtained in one shot. Indeed, $\overline{\mathcal{A}}$'s private value vector may be distinct during different repetitions of procedure Υ. This may involve that $\tilde{\vec{m}}$ be different from $\mathcal{A}$'s real value vector (remember that $\mathfrak{S}$ has to simulate $\mathcal{A}$) or they may even be computationally distinguishable.

P2 $\overline{\mathcal{A}}$'s responses may not always be good[2] in all repetitions. Note that if $\overline{\mathcal{A}}$ make a good response in a repetition, $\mathfrak{S}$ will get t values in the repetition.

[2]We say an adversary's response in an execution is *good*, if it does not cause an honest receiver to output Abort_1.

To illustrate P2, consider the following. Let p_j be the probability that $\overline{\mathcal{A}}$ makes a good response when it interacts with an honest receiver whose private input is T_j. Similarly, we use notation p for $\mathcal{A}$ and T. The probability of $\mathfrak{S}$ successfully constructing $\tilde{\vec{m}}$ is $\prod_{j=1}^{d} p_j$. Thus, the probability of $\mathcal{R}'$ learning t values in the ideal world is $\prod_{j=1}^{d} p_j$. It is easy to see that the probability of $\mathcal{R}$ learning t values in the real world is p. Therefore, the real world's output and the ideal world's output may be distinguished with probability at least $|p - \prod_{j=1}^{d} p_j|$. Unfortunately, this difference may not be negligible: e.g., $p = p_1 = \ldots = p_d = 0.5$, $d = 3$, $|p - \prod_{j=1}^{d} p_j| = 0.375$. We will now show how to solve both problems.

Lemma 27. *Let* $\mathsf{View}^j_{\overline{\mathcal{A}}}$ *denote the view of* $\overline{\mathcal{A}}$ *when it interacts with an honest receiver whose private input is* T_j*. Similarly, we use notation* $\mathsf{View}_{\mathcal{A}}$ *for* $\mathcal{A}$ *and* T*. Then, we have:*

$$\mathsf{View}^j_{\overline{\mathcal{A}}} \stackrel{c}{=} \mathsf{View}_{\mathcal{A}}.$$

Proof. In the ideal world, the honest receiver's private input T is unknown to the simulator. Therefore, it may occur that $T_j \neq T$. If $T_j = T$, then obviously $\mathsf{View}^j_{\overline{\mathcal{A}}} \equiv \mathsf{View}_{\mathcal{A}}$. When $T_j \neq T$, there are two differences between $\mathsf{View}^j_{\overline{\mathcal{A}}}$ and $\mathsf{View}_{\mathcal{A}}$.

First Difference. In $\mathsf{View}^j_{\overline{\mathcal{A}}}$, each permutation σ_i^2 relates to T_j while, in $\mathsf{View}_{\mathcal{A}}$, each σ_i^2 is related to T. In both views, permutation σ_i^1 is uniformly chosen from S_n. As a result, for set G_i (defined in Step R2.2 of Π), its elements are uniformly distributed over $[n]$. Looking at the implementation of Γ as depicted in [32], we know that for any image $c \in [n]$, its pre-image under permutation σ_i^2 is uniformly distributed over $[n]$. This implies that each σ_i^2 is uniformly distributed over S_n. Therefore, any σ_i^2 in $\mathsf{View}^j_{\overline{\mathcal{A}}}$ and any σ_i^2 in $\mathsf{View}_{\mathcal{A}}$ are identically distributed.

Second Difference. In $\mathsf{View}^j_{\overline{\mathcal{A}}}$, for each instance vector $\tilde{\vec{x}}_i$, its projective entries correspond to index set T_j, and in $\mathsf{View}_{\mathcal{A}}$, the projective entries of $\tilde{\vec{x}}_i$ are indexed by T. Nonetheless, the fact that any $\tilde{\vec{x}}_i$ in $\mathsf{View}^j_{\overline{\mathcal{A}}}$ and any $\tilde{\vec{x}}_i$ in $\mathsf{View}_{\mathcal{A}}$ are computationally indistinguishable immediately follows from the hard subset membership property of $\mathcal{H}$.

Combining the above two points, we obtain that $\mathsf{View}^j_{\overline{\mathcal{A}}} \stackrel{c}{=} \mathsf{View}_{\mathcal{A}}$ also holds in the case of $T_j \neq T$. □

Corollary 28. *Let* p_j*,* p *be the probabilities introduced above. Let* $\vec{m}_j^*$ *be a random variable describing the distribution of the real value vector of* $\overline{\mathcal{A}}$ *when it makes a good response during its interaction with an honest receiver whose private input is* T_j*. Similarly, we use notation* $\vec{m}^*$ *for* $\mathcal{A}$ *and* T*. Then, we have:*

$$\vec{m}_j^* \stackrel{c}{=} \vec{m}^* \qquad \textit{and} \qquad |p_j - p| \leq \mu(k),$$

where $\mu(k)$ *is a negligible function of the security parameter* k*.*

Following Corollary 28, even if the value vector $\tilde{\vec{m}}$ built by $\mathfrak{S}$ is not equal to $\mathcal{A}$'s real value vector, their distributions are computationally indistinguishable. This is good enough to solve problem P1.

To overcome problem P2, the simulator $\mathfrak{S}$ must allow $\mathcal{R}'$ to get t values with probability close to p (e.g., p_1). This is due to the fact that $\mathcal{R}$ gets t values with probability p. Note that $\mathcal{R}'$ gets t values if and only if $\mathfrak{S}$ successfully constructs $\tilde{\vec{m}}$. This means that, if $\mathcal{A}$ make a good response in the first repetition (occurring with probability p_1), then $\mathfrak{S}$ must successfully extract t values for each $T_j (j \geq 2)$. Given $T_j (j \geq 2)$, a straightforward way to extract values for this index set is for $\mathfrak{S}$ to repeat procedure Υ until it receives t values. The expected number of times Υ is to be iterated is p_1/p_j. Following Corollary 28, the difference $|p_1 - p_j|$ is negligible in k. However, the ratio p_1/p_j might not be polynomial in k (e.g., $p_j = 1/2^k$, $p_1 = 1/2^{2k}$, $p_1/p_j = 2^k$). Our solution to this issue is for $\mathfrak{S}$ to first estimate the value of p_1 and then to appropriately upper bound the number of times Υ is to be repeated. This technique was previously presented by Goldreich in the context of ZK [30].

Based on the above discussion, we modify Algorithm 2 by replacing Step Sim3 with the following three sub-steps.

- Sim3.1: Playing the role of $\mathcal{R}$ with private input T_1, $\mathfrak{S}$ executes Υ once. If $\overline{\mathcal{A}}$ gives a bad response in this execution, $\mathfrak{S}$ sends Abort_1 to the TTP and outputs what $\overline{\mathcal{A}}$ returned when $\overline{\mathcal{A}}$ halted.
- Sim3.2: (Estimating p_1) Let $\mathsf{Poly}_1(.)$ be a sufficiently large polynomial. Playing the role of $\mathcal{R}$ with private input T_1, $\mathfrak{S}$ repeats Υ until the number of $\overline{\mathcal{A}}$'s good responses received is up to $\mathsf{Poly}_1(k)$. In each repetition, $\mathfrak{S}$ uses fresh randomness. Let ℓ denote the number of the times Υ is repeated. The estimated value of p_1 is set to $\tilde{p}_1 = \mathsf{Poly}_1(k)/\ell$.
- Sim3.3: $\mathfrak{S}$ tries to extract values for each T_j ($j \geq 2$). Let $\mathsf{Poly}_2(.)$ be another sufficiently large polynomial. For T_j, $\mathfrak{S}$ operates as follows:

  Playing the role of $\mathcal{R}$ with private input T_j, $\mathfrak{S}$ repeats Υ at most $\mathsf{Poly}_2(k)/\tilde{p}_1$ times. If $\overline{\mathcal{A}}$ makes a good response, $\mathfrak{S}$ records the received t values and stops the iterations for T_j. If $\overline{\mathcal{A}}$ always makes bad responses, $\mathfrak{S}$ halts and outputs $\mathsf{Timeout}$.

Lemma 29. *The modified simulator $\mathfrak{S}$ outputs* $\mathsf{Timeout}$ *with (at most) negligible probability.*

Proof. We introduce the following probabilistic event notations:

- $\mathsf{TO} \stackrel{def}{=} \{\mathfrak{S}$ outputs $\mathsf{Timeout}\}$,
- $\mathsf{TO}_j \stackrel{def}{=} \{\mathfrak{S}$ outputs $\mathsf{Timeout}$ when $\mathfrak{S}$ tries to extract values for private input $T_j\}$.

We have:

$$\mathrm{Prob}(\mathsf{TO}) \leq \sum_{j=2}^{d} \mathrm{Prob}(\mathsf{TO}_j),$$

$$\mathrm{Prob}(\mathsf{TO}_j) \leq p_1 \sum_{i=1}^{+\infty} \mathrm{Prob}\left(\left\lfloor \frac{1}{\tilde{p}_1} \right\rfloor = i\right)(1-p_j)^{i\,\mathsf{Poly}_2(k)}. \tag{2}$$

In the proof of Claim 2 of [30], Goldreich and Kahan showed that the right hand side of Equation (2) is negligible in k. Therefore, our lemma holds. □

Combining Lemma 29 and the previous discussion, we know that $\mathcal{R}'$ learns t values with probability $p_1 - \mu(k)$. Thus, the difference between the probability of $\mathcal{R}'$ learning t values and the probability of $\mathcal{R}$ learning t values is negligible. This solves the problem P2. Now, we are ready to prove Theorem 24 in the case where only $\mathcal{S}$ is corrupted.

Proposition 30. *The modified simulator $\mathfrak{S}$ runs in expected polynomial-time.*

Proof. All steps of simulator $\mathfrak{S}$ are polynomial-time except Step $\mathsf{Sim3}$. Let ET, T_Υ, ETP, TV_j denote the expected running time of whole Step $\mathsf{Sim3}$, the running time of Υ, the expected running time of Step $\mathsf{Sim3.2}$ and the running time of extracting values for T_j, respectively. We have the following three relations:

$$\mathsf{ET} = (1-p_1)\mathsf{T}_\Upsilon + p_1(\mathsf{T}_\Upsilon + \mathsf{ETP} + \sum_{j=2}^{d} \mathsf{TV}_j) \qquad \mathsf{ETP} = \frac{\mathsf{Poly}_1(k)}{p_1} \times \mathsf{T}_\Upsilon \qquad \mathsf{TV}_j \leq \frac{\mathsf{Poly}_2(k)}{\tilde{p}_1} \times \mathsf{T}_\Upsilon.$$

Furthermore:

$$\mathsf{ET} \leq \mathsf{T}_\Upsilon + \mathsf{Poly}_1(k) \times \mathsf{T}_\Upsilon + \frac{p_1}{\tilde{p}_1} \times \mathsf{Poly}_2(k) \times (d-1)\mathsf{T}_\Upsilon. \tag{3}$$

Therefore, it remains to consider the ratio $p_1/\tilde{p}_1$. Let's focus on Step $\mathsf{Sim3.2}$ and denote by Y_i a binary random variable that equals 1 if $\overline{\mathcal{A}}$ makes a good response in the i-th repetition and equals 0 otherwise. Let $0 \leq \delta \leq 1$. We have:

$$\mathrm{Prob}\left(\frac{p_1}{\tilde{p}_1} > \frac{1}{1-\delta}\right) = \mathrm{Prob}\left(\frac{p_1}{\sum_{i=1}^{\ell} Y_i/\ell} > \frac{1}{1-\delta}\right) = \mathrm{Prob}\left(\sum_{i=1}^{\ell} Y_i < (1-\delta)\ell p_1\right] \leq \frac{1}{e^{\delta^2 \ell p_1/2}}$$

where the right hand side inequality follows the Chernoff bound [41]. Setting $\delta = 1/2$, we get:

$$\text{Prob}\left(\frac{p_1}{\tilde{p}_1} > 2\right) \leq \frac{1}{e^{\ell p_1/8}} \leq \frac{1}{e^{\mathsf{Poly}_1(k)\, p_1/8}}, \tag{4}$$

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

Combining Inequality (3) and Inequality (4), it is easy to see that Step Sim3 runs in expected polynomial-time. □

Proposition 31. *In the case where only the sender is corrupted, Equation* (1) *in Definition 9 holds.*

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

$$g(\alpha) = \begin{cases} \mathsf{Abort}_1 & \text{if } \gamma \text{ causes } \mathcal{R} \text{ to output } \mathsf{Abort}_1, \\ (\gamma\langle i\rangle)_{i\in T} & \text{if } \gamma \text{ is a vector containing } n \text{ values.} \end{cases}$$

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

$$\mathsf{Real}_{\Pi,\{1\},\mathcal{A}(z_k)}(1^k, \vec{m}, T) \equiv h(\mathsf{Real}_{\Pi,\{1\},\mathcal{A}(z_k)}(1^k, \vec{m}, T)\langle 0\rangle). \tag{5}$$

Now, we concentrate our attention to the ideal world. We claim that $\mathfrak{S}$'s output and $\mathcal{A}$'s output are computationally indistinguishable:

$$\mathsf{Real}_{\Pi,\{1\},\mathcal{A}(z_k)}(1^k, \vec{m}, T)\langle 0\rangle \stackrel{c}{=} \mathsf{Ideal}_{f,\{1\},\mathfrak{S}(z_k)}(1^k, \vec{m}, T)\langle 0\rangle. \tag{6}$$

Note that $\mathfrak{S}$ almost always takes $\overline{\mathcal{A}}$'s output as its own and, with at most negligible probability, it takes Timeout as its own output (Lemma29). Lemma 27 implies that $\overline{\mathcal{A}}$'s output in each repetition in the ideal world and $\mathcal{A}$'s output in the real world are computationally indistinguishable. Therefore, the claim holds.

We claim that $\mathcal{R}'$'s output is a deterministic function of $\mathfrak{S}$'s output, specifically,

$$\mathsf{Ideal}_{f,\{1\},\mathfrak{S}(z_k)}(1^k, \vec{m}, T)\langle 2\rangle \stackrel{c}{=} g(\mathsf{Ideal}_{f,\{1\},\mathfrak{S}(z_k)}(1^k, \vec{m}, T)\langle 0\rangle). \tag{7}$$

We argue about this claim as follows:

1. In the case where $\overline{\mathcal{A}}$ makes a bad response in Step Sim3.1, $\mathcal{R}'$ receives Abort_1, and so Equation (7) holds.
2. In the case that $\overline{\mathcal{A}}$ makes a good response in Step Sim3.1 and $\mathfrak{S}$ does not output Timeout, $\mathcal{R}'$ receives t values of $\tilde{\vec{m}}$. Though $\tilde{\vec{m}}$ is gained by $\mathfrak{S}$ playing the role of $\mathcal{R}$ with distinct private inputs, Corollary 28 implies that the distribution of $\vec{m}$ and the distribution of the value vector in $\mathfrak{S}$'s output ($\vec{m}^*_d$ defined in Corollary 28) are computationally indistinguishable. Thus, Equation (7) holds.
3. Following Lemma 29, we do not need to consider the case where $\overline{\mathcal{A}}$ makes a good response in Step Sim3.1 and $\mathfrak{S}$ outputs Timeout.

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

$$\mathsf{Ideal}_{f,\{1\},\mathfrak{S}(z_k)}(1^k, \vec{m}, T) \stackrel{c}{=} h(\mathsf{Ideal}_{f,\{1\},\mathfrak{S}(z_k)}(1^k, \vec{m}, T)\langle 0\rangle) \tag{8}$$

Combining Equations (5), (6), (8), we deduce that Equation (1) holds which proves our proposition. □

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

In this case, $\mathcal{A}$ takes the full control of $\mathcal{R}$ in the real world. Correspondingly, the simulator $\mathfrak{S}$ takes the full control of $\mathcal{R}'$ in the ideal world. It works as represented in Algorithm 3 where we adopted the same policy as before concerning the incorrect message transmission behavior of $\overline{\mathcal{A}}$.

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

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

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

Sim1 (Initialization step):

1. $\mathfrak{S}$ corrupts $\mathcal{R}'$ and learns $\mathcal{R}'$'s private input T.
2. Let $\overline{\mathcal{A}}$ be a copy of $\mathcal{A}$. $\mathfrak{S}$ invokes $\overline{\mathcal{A}}$ as a subroutine with initial inputs of $\overline{\mathcal{A}}$ to be $\mathfrak{S}$'s initial inputs with the exception that the randomness of $\overline{\mathcal{A}}$ to be a uniformly distributed string. In other words, the initial inputs of $\overline{\mathcal{A}}$ are set as k, I, z_k and $r_{\overline{\mathcal{A}}}$ with $r_{\overline{\mathcal{A}}} \in_U \{0,1\}^*$.
3. $\mathfrak{S}$ activates $\overline{\mathcal{A}}$.
4. As in the real world, $\overline{\mathcal{A}}$ sends a message to corrupt the receiver. $\mathfrak{S}$ plays the role of $\mathcal{R}$ and supplies $\overline{\mathcal{A}}$ with T.

Sim2: Playing the role of $\mathcal{S}$, the simulator $\mathfrak{S}$ honestly executes Step S1-S2.2 of Π. If $\overline{\mathcal{A}}$'s response in the interaction is good (i.e., it does not cause an honest sender to output Abort_2), then $\mathfrak{S}$ records the choice indicator r and proceeds to the next step. Else, $\mathfrak{S}$ halts outputting what $\overline{\mathcal{A}}$ returned.

Sim3: The simulator repeats the following procedure Ξ until $\overline{\mathcal{A}}$ gives a good response. In each iteration, $\mathfrak{S}$ uses fresh randomness.

- Procedure Ξ: $\mathfrak{S}$ rewinds $\overline{\mathcal{A}}$ to the beginning of Step R2 of Π. Playing the role of $\mathcal{S}$, the simulator $\mathfrak{S}$ honestly follows Step S1-S2.2 to interact with $\overline{\mathcal{A}}$.

$\mathfrak{S}$ records the choice indicator $\tilde{r}$ and the received witness vectors in the last iteration.

Comment: The chosen instance vectors of $\overline{\mathcal{A}}$ in the last iteration are legal as they passed the legality check.

Sim4:

- Case 1: $r = \tilde{r}$. The simulator $\mathfrak{S}$ outputs Failure and halts.
- Case 2: $r \neq \tilde{r}$ and $\forall i\,(r\langle i\rangle \neq \tilde{r}\langle i\rangle \Rightarrow r\langle i\rangle = 1$ and $\tilde{r}\langle i\rangle = 0)$. The simulator is reset to the beginning of Step Sim1 with a new randomness $r_{\mathfrak{S}} \in_U \{0,1\}^*$.
- Case 3: $r \neq \tilde{r}$ and $\exists i\ :\ r\langle i\rangle = 0$ and $\tilde{r}\langle i\rangle = 1$. The simulator $\mathfrak{S}$ arbitrarily chooses one such instance i and records it as e. Then, it proceeds to the next step.

Comment: The aim of Step Sim3 and Sim4 is to extract one of $\overline{\mathcal{A}}$'s possible real private inputs related to the interaction with $\mathfrak{S}$ in Step Sim2. If Case 3 happens, $\mathfrak{S}$ knows $\tilde{\vec{x}}_e$'s witness vector $\tilde{\vec{w}}_e$. Note that $\tilde{\vec{x}}_e$ is a legal instance vector. Then, $\mathfrak{S}$ can easily deduce $\overline{\mathcal{A}}$'s private index set corresponding to permutation σ_e^2 received in Step Sim2. We stress the fact that this private input is one of $\overline{\mathcal{A}}$'s possible private inputs as distinct permutations may correspond to distinct private inputs. However, as we will see in the proof of Lemma 34, this inconsistence does not affect the correctness of our simulation. $\overline{\mathcal{A}}$'s initial inputs are fixed by $\mathfrak{S}$ in Step Sim1. Thus, if $\overline{\mathcal{A}}$ receives the same messages, it responds in the same way. Therefore, $\mathfrak{S}$ can reproduce the scenario of Step Sim2 by rewinding $\overline{\mathcal{A}}$ to the beginning of Step R2 and sending the choice indicator r to $\overline{\mathcal{A}}$.

Sim5: The simulator rewinds $\overline{\mathcal{A}}$ to the beginning of Step R2 of Π, chooses r as choice indicator, honestly executes Step S1-S2.2 and deduces $\overline{\mathcal{A}}$'s private input (denoted by T'_e) corresponding to permutation σ_e^2.

Sim6: The simulator sends T'_e to the TTP and receives $(\vec{m}\langle i\rangle)_{i\in T'_e}$ in return. Then, $\mathfrak{S}$ sends Continue to the TTP and builds a value vector $\vec{m}'$ as follows. For each $i \in T'_e$, $\vec{m}'\langle i\rangle \leftarrow \vec{m}\langle i\rangle$. For each $i \notin T'_e$, set $\vec{m}'\langle i\rangle$ to be a value uniformly chosen from G_Λ, where G_Λ is a set of all possible hash values (see Definition 20).

Sim7 (Output step): Playing the role of $\mathcal{S}$ with private input $\vec{m}'$, $\mathfrak{S}$ follows Step S2.3 of Π to complete the interaction with $\overline{\mathcal{A}}$. When $\overline{\mathcal{A}}$ halts, $\mathfrak{S}$ outputs what $\overline{\mathcal{A}}$ returned.

Proof. First, let's focus on Step Sim3. In each iteration of Ξ, the chosen instance vectors are uniformly distributed. This leads to the probability that $\overline{\mathcal{A}}$ gives a good response in each repetition to be the same. We denote this value by p. Let ET_3 and T_Ξ denote expected the running time of Step Sim3 and the running time of Ξ respectively. We have:

$$\mathsf{ET}_3 = (1/p)\,\mathsf{T}_\Xi.$$

For the same reason, the probability that $\overline{\mathcal{A}}$ gives a good response in Step Sim2 is p as well. Let OncET_{1-3} denote the expected running time of one run of the sequence Step Sim1 to Step Sim3. Let T_j denote the running time of $\mathfrak{S}$'s j-th step. We have:

$$\mathsf{OncET}_{1-3} \le \mathsf{T}_1 + \mathsf{T}_2 + p\,\mathsf{ET}_3 \le \mathsf{T}_1 + \mathsf{T}_2 + \mathsf{T}_\Xi$$

Second, consider Step Sim4 and especially Case 2. Note that the initial inputs of $\mathfrak{S}$ (apart from the refreshed randomness) remain the same in each iteration. Thus, the probability that $\mathfrak{S}$ runs from scratch in each iteration is the same. We denote this probability by $1 - q$. As a consequence, the expected running time of the sequence Step Sim1 to Step Sim4, denoted by ET_{1-4}, is:

$$\mathsf{ET}_{1-4} \le (1 + 1/q)\,(\mathsf{OncET}_{1-3} + \mathsf{T}_4) \le (1 + 1/q)\,(\mathsf{T}_1 + \mathsf{T}_2 + \mathsf{T}_\Xi + \mathsf{T}_4).$$

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

$$\mathsf{ET}_{\mathfrak{S}} \le \mathsf{ET}_{1-4} + \mathsf{T}_5 + \mathsf{T}_6 + \mathsf{T}_7 \le (1 + 1/q)\,(\mathsf{T}_1 + \mathsf{T}_2 + \mathsf{T}_\Xi + \mathsf{T}_4) + \mathsf{T}_5 + \mathsf{T}_6 + \mathsf{T}_7. \tag{9}$$

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

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

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

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

We introduce the following notations to denote probabilistic events:

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

It is easy to see that U_1, U_2, U_3 are mutually disjoint, $(\{0,1\}^K)^2 = U_1 \cup U_2 \cup U_3$, $|U_1| = 2^K$ and $|U_2| = |U_3| = 2^{K-1}$. Since both R and $\tilde{R}$ are uniformly distributed, we have:

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

and

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

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

$$q = \mathrm{Prob}(\mathcal{E}) + \mathrm{Prob}(\overline{\mathcal{E}})\,(1/2 + 1/2^{K+1}) = 1/2 + 1/2^{K+1} + (1/2 - 1/2^{K+1})\,\mathrm{Prob}(\mathcal{E}) \tag{13}$$

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

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

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

Lemma 33. *The simulator $\mathfrak{S}$ outputs* Failure *with probability at most* $1/2^{K-1}$.

Proof. Let X be the random variable defined as the number of the trials of $\mathfrak{S}$ in a whole execution. From the proof of Lemma32, we know two facts. First, we have: $\text{Prob}(X = i) = (1-q)^{i-1}q < 1/2^{i-1}$. Second, in each trial the event $\{\mathfrak{S}$ outputs Failure$\}$ is the combined event of $\overline{\mathcal{E}}$ and $R = \tilde{R}$. This combined event happens with probability:

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

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

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

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

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

□

Lemma 34. *The output of the adversary $\mathcal{A}$ in the real world and that of the simulator $\mathfrak{S}$ in the ideal world are statistically indistinguishable, i.e.:*

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

Proof. First, we claim that $\mathfrak{S}$'s output and $\overline{\mathcal{A}}$'s output are statistically indistinguishable. Indeed, $\mathfrak{S}$ always returns $\overline{\mathcal{A}}$'s output with the exception of returning Failure when Case 1 of Step Sim4 happens. Since Lemma 33 shows that this situation arises with negligible probability, our claim holds.

Second, we claim that the outputs of $\mathcal{A}$ and $\overline{\mathcal{A}}$ are statistically indistinguishable. The only difference between $\overline{\mathcal{A}}$'s view and $\mathcal{A}$'s view is that the encryption received by $\overline{\mathcal{A}}$ is the encryption of the constructed $\vec{m}'$ rather than $\vec{m}$.

Let T'_i be the private input corresponding to permutation σ_i^2 where $i \in \overline{CS}$. Let $T' \stackrel{def}{=} \cap_{i\in\overline{CS}} T'_i$. From the proof of Lemma 25, we know that the probability of the event $\{|T'| > t\}$ is negligible. As a result, it suffices to prove that $\overline{\mathcal{A}}$'s view and $\overline{\mathcal{A}}$'s view are statistically indistinguishable in the case where $|T'| \leq t$.

Looking at Step S2.3 of Π (i.e., Protocol 1), we know that the encryption is generated by XOR-ing $\vec{m}$ with the hash values of the unchosen instance vectors.

For each $i \in T'$, the value is hidden by hash values of projective instances. The projection property of $\mathcal{H}$ guarantees that $\overline{\mathcal{A}}$ (resp., $\mathcal{A}$) learns $(\vec{m}'\langle i\rangle)_{i\in T'}$ (resp., $(\vec{m}\langle i\rangle)_{i\in T'}$). Seeing Step Sim6 of $\mathfrak{S}$, we know:

$$(\vec{m}'\langle i\rangle)_{i\in T'} = (\vec{m}\langle i\rangle)_{i\in T'}.$$

For each $i \notin T'$, the value is hidden by hash values of smooth instances. The smoothness property of $\mathcal{H}$ guarantees that $\overline{\mathcal{A}}$ (resp., $\mathcal{A}$) learns nothing about $(\vec{m}'\langle i\rangle)_{i\notin T'}$ (resp., $(\vec{m}\langle i\rangle)_{i\notin T'}$) and

$$(\vec{c}'\langle i\rangle)_{i\notin T'} \stackrel{s}{=} (\vec{c}\langle i\rangle)_{i\notin T'}.$$

Therefore, $\overline{\mathcal{A}}$'s view and $\overline{\mathcal{A}}$'s view are statistically indistinguishable in the case of $|T'| \leq t$. Thus, our second claim holds which achieves our lemma demonstration. □

Remark 35. *Because $\mathfrak{S}$ filters out the case that all chosen instance vectors are legal and all unchosen ones are illegal by repeating procedure Ξ in Step* Sim3*, $\mathfrak{S}$ does not simulate the real world case where $\mathcal{A}$ learns more than t values. Correspondingly, in the proof of Lemma 34, we do not consider the corresponding case where $|T'| > t$. This is justified by the fact that Lemma 25 shows this situation arises with probability at most* $1/2^{K}$.

Proposition 36. *In the case where only the receiver is corrupted, Equation* (1) *in Definition 9 holds.*

Proof. Note that the senders $\mathcal{S}$ and $\mathcal{S}'$ end up with outputting nothing as they honestly follow Π. On the other hand, the receivers $\mathcal{R}$ and $\mathcal{R}'$ end up with no output either bcause they are corrupted and so are dummy players. As a consequence, the fact that the outputs of $\mathfrak{S}$ and $\mathcal{A}$ are statistically indistinguishable (Lemma 34) immediately proves this proposition. □

6. Reducing the Construction Conditions of a SPH-DHM

In order to obtain a SPH-DHM, we need to satisfy a large number of properties. In this section, we reduce the design of such a family to the construction of considerably simpler hash families. The basic idea is to generate the instances of each type (projective/smooth) independently.

6.1. Basic Hash Family

Definition 37. A basic hash family $\mathcal{H}$ *is defined by means of the same seven PPT algorithms* $\mathcal{H} = (\mathsf{PG}, \mathsf{IS}, \mathsf{Check}, \mathsf{DI}, \mathsf{KG}, \mathsf{Hash}, \mathsf{pHash})$ *as in Definition 10 with the exception of algorithm* IS *is defined as follows.*

- Instance sampler IS*: it takes a security parameter k, a hash parameter* Λ*, a* work mode $\delta \in \{0, 1\}$ *as input and outputs an instance-witness pair* (x, w) *(i.e.,* $(x, w) \leftarrow \mathsf{IS}(1^k, \Lambda, \delta)$*). Its goal is to generate projective instance-witness pairs under mode* 0 *and smooth ones under mode* 1.

To be consistent with Definition 10, we write the instance-witness pair generated under mode $\delta = 0$ as $(\dot{x}, \dot{w})$, and under mode $\delta = 1$ as $(\ddot{x}, \ddot{w})$. The properties of distinguishability and projection are defined as in Section 3. However, the properties of smoothness and hard subset membership are defined in considerably simpler way.

Definition 38 (Smoothness)**.** *For any legal hash parameter* Λ*, any instance-witness* $(\ddot{x}, \ddot{w})$ *generated by* $\mathsf{IS}(1^k, \Lambda, 1)$*,* smoothness *holds if the two probability ensembles* $\mathsf{BSM}_1 \stackrel{def}{=} \{\mathsf{BSM}_1(1^k)\}_{k\in\mathbb{N}}$ *and* $\mathsf{BSM}_2 \stackrel{def}{=} \{\mathsf{BSM}_2(1^k)\}_{k\in\mathbb{N}}$*, specified as follows, are statistically indistinguishable, i.e.,* $\mathsf{BSM}_1 \stackrel{s}{=} \mathsf{BSM}_2$. Perfect smoothness *holds, if* $\mathsf{BSM}_1 \equiv \mathsf{BSM}_2$.

- $\mathsf{BSM}_1(1^k)$*: Generate* $(hk, pk) \leftarrow \mathsf{KG}(1^k, \Lambda, \ddot{x})$ *and compute* $y \leftarrow \mathsf{Hash}(1^k, \Lambda, \ddot{x}, hk)$*. Output* (pk, y)*.*
- $\mathsf{BSM}_2(1^k)$*: Compared to* $\mathsf{BSM}_1(1^k)$*, the only difference is that* $y \in_U G_\Lambda$ *where* G_Λ *is a set of all possible hash values (see Definition 20).*

Definition 39 ([27, 25])**.** *Let* $0 < \varepsilon < 1$*. For any legal hash parameter* Λ*, any instance-witness* $(\ddot{x}, \ddot{w})$ *generated by* $\mathsf{IS}(1^k, \Lambda, 1)$*,* ε-universality *holds if for any string* $pk \in \{0, 1\}^*$*, and any value* $y \in G_\Lambda$*, it holds that*

$$\mathrm{Prob}(\mathsf{Hash}(1^k, \Lambda, \ddot{x}, HK) = y \,|\, PK = pk) \leq \varepsilon,$$

where $(HK, PK) \leftarrow \mathsf{KG}(1^k, \Lambda, \ddot{x})$*. The probability is taken over the randomness used by* KG*.*

Intuitively speaking, universality requires that, for any instance-witness $(\ddot{x}, \ddot{w})$ generated by $\mathsf{IS}(1^k, \Lambda, 1)$, the probability of guessing its hash value is at most ε. Compared with smoothness, ε-universality relaxes the upper bound of this probability to be a higher value. However, we can efficiently gain smoothness from universality.

Lemma 40 ([27, 25])**.** *Given a basic hash family* $\widetilde{\mathcal{H}}$ *holding universality, then we can efficiently construct a basic hash family* $\mathcal{H}$ *holding smoothness.*

The constructive proof of this lemma proceeds in two steps as follows. First, it reduces ε-universality to ε^c-universality by simple c-fold "parallelization". Second, smoothness is obtained from ε^c-university by simply applying the Leftover Hash Lemma (see [42] for this lemma). We refer the reader to [27, 25] for the proof details.

Definition 41 (Hard subset membership)**.** *The two probability ensembles* $\mathsf{BHS}_1 \stackrel{def}{=} \{\mathsf{BHS}_1(1^k)\}_{k\in\mathbb{N}}$ *and* $\mathsf{BHS}_2 \stackrel{def}{=} \{\mathsf{BHS}_2(1^k)\}_{k\in\mathbb{N}}$ *defined as follows, are computationally indistinguishable (i.e.,* $\mathsf{BHS}_1 \stackrel{c}{=} \mathsf{BHS}_2$*).*

- $\mathsf{BHS}_1(1^k)$*: Generate* $\Lambda \leftarrow \mathsf{PG}(1^k)$ *and compute* $(\dot{x}, \dot{w}) \leftarrow \mathsf{IS}(1^k, \Lambda, 0)$*. Output* $(\Lambda, \dot{x})$*.*
- $\mathsf{BHS}_2(1^k)$*: Generate* $\Lambda \leftarrow \mathsf{PG}(1^k)$ *and compute* $(\ddot{x}, \ddot{w}) \leftarrow \mathsf{IS}(1^k, \Lambda, 1)$*. Output* $(\Lambda, \ddot{x})$*.*

Construction 1 Reduction Procedure

Input: A basic hash family $\mathcal{H} = (\mathsf{PG}, \mathsf{IS}, \mathsf{Check}, \mathsf{DI}, \mathsf{KG}, \mathsf{Hash}, \mathsf{pHash})$.

We construct a hash family $\overline{\mathcal{H}}$ (as defined in Definition 10) having the same algorithms $\mathcal{H}$ with the exception of the instance sampling algorithm $\overline{\mathsf{IS}}$ which is defined as follows.

- $\overline{\mathsf{IS}}(1^k, \Lambda)$: For each $i \in [t]$, $\vec{a}\langle i\rangle \leftarrow \mathsf{IS}(1^k, \Lambda, 0)$. For each $i \in [n] \setminus [t]$, $\vec{a}\langle i\rangle \leftarrow \mathsf{IS}(1^k, \Lambda, 1)$. Return $\vec{a}$.

6.2. The Reduction Process

Theorem 42. *In Construction 1, if the basic hash family $\mathcal{H}$ has (perfect) smoothness, then $\overline{\mathcal{H}}$ holds this property as well.*

Theorem 43. *In Construction 1, if the basic hash family $\mathcal{H}$ has the property hard subset membership property, then the hash family $\overline{\mathcal{H}}$ holds this property as well.*

We will prove the above two theorems in Section 6.2.1 and Section6.2.2 respectively. A consequence of these two theorems and Lemma 40, we can reduce the construction of a SPH-DHM to the design of a basic hash family having smoothness (or universality), projection, distinguishability and hard subset membership.

6.2.1. Proof of Theorem 42

Before starting our demonstration of Theorem 42, we need some preliminary results.

Definition 44. *For any probability ensemble $X = \{X(1^k, a)\}_{k\in\mathbb{N}, a\in\{0,1\}^*}$, we define its* multiple-sampled probability ensemble $\vec{X}$ *as follows:*

$$\vec{X} = \{\vec{X}(1^k, a)\}_{k\in\mathbb{N}, a\in\{0,1\}^*} \text{ with } \vec{X}(1^k, a) = (X_1(1^k, a), X_2(1^k, a), \ldots, X_n(1^k, a)),$$

where each $X_i(1^k, a) \equiv X(1^k, a)$ and each random variable $X_i(1^k, a)$ is independent.

Lemma 45 ([35])**.** *Let X, Y be two probability ensembles, and $\vec{X}$, $\vec{Y}$ their multiple-sampled probability ensembles, respectively.*

- *If $X \overset{s}{=} Y$, then $\vec{X} \overset{s}{=} \vec{Y}$.*
- *If X and Y are polynomial-time constructible and $X \overset{c}{=} Y$, then $\vec{X} \overset{c}{=} \vec{Y}$.*

Definition 46. *Let $F \overset{def}{=} (f_k)_{k\in\mathbb{N}}$ be a function family. For a probability ensemble $X = \{X(1^k, a)\}_{k\in\mathbb{N}, a\in\{0,1\}^*}$, we define its* image probability ensemble $F(X)$ *under F as follows:*

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

Lemma 47. *Let X, Y be two probability ensembles and $F(X)$, $F(Y)$ their image probability ensembles, respectively. If $X \overset{s}{=} Y$, then $F(X) \overset{s}{=} F(Y)$.*

Proof.

$$\frac{1}{2}\sum_{\alpha\in\{0,1\}^*} |\text{Prob}[f_k(X(1^k, a)) = f_k(\alpha)] - \text{Prob}[f_k(Y(1^k, a)) = f_k(\alpha)]| \leq \frac{1}{2}\sum_{\alpha\in\{0,1\}^*} |\text{Prob}[X(1^k, a) = \alpha] - \text{Prob}[Y(1^k, a) = \alpha]|.$$

This shows that the statistical difference between $F(X)$ and $F(Y)$ is less than that between X and Y. □

We can now expose our proof of Theorem 42.

Proof. We only consider smoothness as the proof in the case of perfect smoothness is similar and simpler. We reuse notations $\mathsf{SmGen}_i(1^k)$, SM_i from Definition 20 and BSM_i from Definition 38.

For the hash family $\overline{\mathcal{H}}$ in Construction 1, we define the following probability ensemble for each $i \in [2]$ and $j \in [n]$.

$$\mathsf{SM}_i^j = \{\mathsf{SM}_i^j(1^k)\}_{k\in\mathbb{N}} \stackrel{def}{=} \{\mathsf{SmGen}_i(1^k)\langle j\rangle\}_{k\in\mathbb{N}}.$$

It is easy to see that: $\forall j \in \{t+1, t+2, \cdots, n\}$ $\mathsf{SM}_i^j \equiv \mathsf{BSM}_i$. Combining the fact that basic hash family $\mathcal{H}$ holds smoothness (i.e., $\mathsf{BSM}_1 \stackrel{s}{=} \mathsf{BSM}_2$) and Lemma 45, we have:

$$\{(\mathsf{SM}_1^{t+1}(1^k), \ldots, \mathsf{SM}_1^n(1^k))\}_{k\in\mathbb{N}} \stackrel{s}{=} \{(\mathsf{SM}_2^{t+1}(1^k), \ldots, \mathsf{SM}_2^n(1^k))\}_{k\in\mathbb{N}}.$$

We introduce the following notations: $\vec{X} \stackrel{def}{=} \{(\mathsf{SM}_1^1(1^k), \ldots, \mathsf{SM}_1^n(1^k))\}_{k\in\mathbb{N}}$ and $\vec{Y} \stackrel{def}{=} \{(\mathsf{SM}_2^1(1^k), \ldots, \mathsf{SM}_2^n(1^k))\}_{k\in\mathbb{N}}$. Looking at the definition of $\mathsf{SmGen}_i(1^k)$, we notice: $\forall j \in [t]$ $\mathsf{SM}_1^j(1^k) \equiv \mathsf{SM}_2^j(1^k)$. So, it holds: $\vec{X} \stackrel{s}{=} \vec{Y}$.

Let $F \stackrel{def}{=} (\sigma)_{k\in\mathbb{N}}$ where σ is an arbitrary permutation over set $[n]$. Following Lemma 47, we have $F(\vec{X}) \stackrel{s}{=} F(\vec{Y})$, i.e.:

$$\{\sigma(\mathsf{SM}_1^1(1^k), \ldots, \mathsf{SM}_1^n(1^k))\}_{k\in\mathbb{N}} \stackrel{s}{=} \{\sigma(\mathsf{SM}_2^1(1^k), \ldots, \mathsf{SM}_2^n(1^k))\}_{k\in\mathbb{N}}.$$

This means: $\mathsf{SM}_1 \stackrel{s}{=} \mathsf{SM}_2$. □

6.2.2. Proof of Theorem 43

We start by some preliminary results to be used in our main proof.

Definition 48. *For any probability ensemble $X = \{X(1^k, a)\}_{k\in\mathbb{N},a\in\{0,1\}^*}$ and $Y=\{Y(1^k, a)\}_{k\in\mathbb{N},a\in\{0,1\}^*}$, we define their* hybrid probability ensemble $\overrightarrow{XY}$ *as follows:*

$$\overrightarrow{XY}=\{\overrightarrow{XY}(1^k, a)\}_{k\in\mathbb{N},a\in\{0,1\}^*} \text{ with } \overrightarrow{XY}(1^k, a)=(X_1(1^k, a), \ldots, X_t(1^k, a), Y_{t+1}(1^k, a), \ldots, Y_n(1^k, a)),$$

where each $X_i(1^k, a) \equiv X(1^k, a)$, each $Y_i(1^k, a) \equiv Y(1^k, a)$, and all random variables are independent.

Lemma 49. *Let X, Y be two probability ensembles and $\overrightarrow{XY}$ their hybrid probability ensemble. Let $\sigma(\overrightarrow{XY})$ be the image probability ensemble of $\overrightarrow{XY}$ under the function family $F=(\sigma)_{k\in\mathbb{N}}$ where σ is an arbitrary element of S_n. If $X \stackrel{c}{=} Y$ and X, Y are polynomial-time constructible, then $\overrightarrow{XY} \stackrel{c}{=} \sigma(\overrightarrow{XY})$.*

Proof. If $\{\sigma\langle i\rangle\}_{i\in[t]} \subseteq [t]$, then the relation $\overrightarrow{XY} \stackrel{c}{=} \sigma(\overrightarrow{XY})$ holds. It remains to consider the case where $\{\sigma\langle i\rangle\}_{i\in[t]} \nsubseteq [t]$.

Assume that the lemma is false in this case. Then, there exists a non-uniform PPT distinguisher D with auxiliary input $z = (z_k)_{k\in\mathbb{N}}$, a polynomial $\mathsf{Poly}(.)$, an infinite positive integer set $G \subseteq \mathbb{N}$ such that, for each $k \in G$, we have:

$$|\text{Prob}(D(1^k, z_k, a, \overrightarrow{XY}(1^k, a)) = 1) - \text{Prob}(D(1^k, z_k, a, \sigma(\overrightarrow{XY}(1^k, a)) = 1)| \geq 1/\mathsf{Poly}(k). \tag{14}$$

Consider $V \stackrel{def}{=} \{i : i \in [t], \sigma(i) \notin [t]\}$. We list the elements of V in order as $i_1 < ... < i_j ... < i_{|V|}$. Let $V_j \stackrel{def}{=} \{i_1, \ldots, i_j\}$. We define the following $|V| + 1$ permutations over $[n]$.

$$\phi_0(i) = \begin{cases} i & \text{if } i \in V \cup \{\sigma(i')\}_{i'\in V}, \\ \sigma(i) & \text{otherwise.} \end{cases}$$

For each $j \in [|V|]$, we set:

$$\phi_j(i) = \begin{cases} i & \text{if } i \in (V \setminus V_j) \cup \{\sigma(i')\}_{i'\in V\setminus V_j}, \\ \sigma(i) & \text{otherwise.} \end{cases}$$

In particular: $\sigma = \phi_{|V|}$. Note that $\phi_j(\overrightarrow{XY}(1^k, a))$ is well defined as $\sigma(\overrightarrow{XY}(1^k, a))$, because ϕ_j is a permutation over $[n]$ as σ. As $\phi_0(\overrightarrow{XY}(1^k, a))$ is obtained by not swapping positions between $X(1^k, a)$ and $Y(1^k, a)$, it holds that $\overrightarrow{XY}(1^k, a) \equiv \phi_0(\overrightarrow{XY}(1^k, a))$. So, we have:

$$\begin{aligned}|\text{Prob}(D(1^k, z_k, a, \overrightarrow{XY}(1^k, a)) = 1) - \text{Prob}(D(1^k, z_k, a, \sigma(\overrightarrow{XY}(1^k, a))) = 1)| \\ = |\text{Prob}(D(1^k, z_k, a, \phi_0(\overrightarrow{XY}(1^k, a))) = 1) - \text{Prob}(D(1^k, z_k, a, \phi_{|V|}(\overrightarrow{XY}(1^k, a))) = 1)|. \quad (15)\end{aligned}$$

Using the triangle inequality, we get:

$$\begin{aligned}|\text{Prob}(D(1^k, z_k, a, \phi_0(\overrightarrow{XY}(1^k, a))) = 1) - \text{Prob}(D(1^k, z_k, a, \phi_{\#V}(\overrightarrow{XY}(1^k, a))) = 1)| \leq \\ \sum_{j=1}^{|V|} |\text{Prob}(D(1^k, z_k, a, \phi_{j-1}(\overrightarrow{XY}(1^k, a))) = 1) - \text{Prob}(D(1^k, z_k, a, \phi_j(\overrightarrow{XY}(1^k, a))) = 1)| \quad (16)\end{aligned}$$

Combining Inequality (14) to Inequality (16), we have:

$$\sum_{j=1}^{|V|} |\text{Prob}(D(1^k, z_k, a, \phi_{j-1}(\overrightarrow{XY}(1^k, a))) = 1) - \text{Prob}(D(1^k, z_k, a, \phi_j(\overrightarrow{XY}(1^k, a))) = 1)| \geq 1/\mathsf{Poly}(k).$$

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

$$|\text{Prob}(D(1^k, z_k, a, \phi_{j-1}(\overrightarrow{XY}(1^k, a))) = 1) - \text{Prob}(D(1^k, z_k, a, \phi_j(\overrightarrow{XY}(1^k, a))) = 1)| \geq 1/(|V|\,\mathsf{Poly}(k)). \quad (17)$$

For any pair of permutations (ϕ_{j-1}, ϕ_j), they differ at values of points i_j and $\sigma(i_j)$. Similarly, for probability ensembles $\phi_{j-1}(\overrightarrow{XY}(1^k, a))$ and $\phi_j(\overrightarrow{XY}(1^k, a))$ differ at their i_j-th entry and $\sigma(i_j)$-th entry. Specifically,

$$\begin{aligned}\phi_{j-1}(\overrightarrow{XY}(1^k, a))\langle i_j\rangle &\equiv X(1^k, a), \\ \phi_{j-1}(\overrightarrow{XY}(1^k, a))\langle \sigma(i_j)\rangle &\equiv Y(1^k, a), \\ \phi_j(\overrightarrow{XY}(1^k, a))\langle i_j\rangle &\equiv Y(1^k, a), \\ \phi_j(\overrightarrow{XY}(1^k, a))\langle \sigma(i_j)\rangle &\equiv X(1^k, a).\end{aligned}$$

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

$$\forall d \in [n] \quad \overrightarrow{MXY}(1^k, a)\langle d\rangle \equiv \begin{cases}\phi_{j-1}(\overrightarrow{XY}(1^k, a))\langle d\rangle & \text{if } d \neq \sigma(i_j) \\ X(1^k, a) & \text{if } d = \sigma(i_j)\end{cases}$$

Using the triangle inequality, we get:

$$\begin{aligned}|\text{Prob}(D(1^k, z_k, a, \phi_{j-1}(\overrightarrow{XY}(1^k, a))) = 1) - \text{Prob}(D(1^k, z_k, a, \overrightarrow{MXY}(1^k, a)) = 1)| + \\ |\text{Prob}(D(1^k, z_k, a, \overrightarrow{MXY}(1^k, a)) = 1) - \text{Prob}(D(1^k, z_k, a, \phi_j(\overrightarrow{XY}(1^k, a))) = 1)| \\ \geq |\text{Prob}(D(1^k, z_k, a, \phi_{j-1}(\overrightarrow{XY}(1^k, a))) = 1) - \text{Prob}(D(1^k, z_k, a, \phi_j(\overrightarrow{XY}(1^k, a))) = 1)| \quad (18)\end{aligned}$$

Combining Inequality (17) and Inequality(18), we know that either

$$|\text{Prob}(D(1^k, z_k, a, \phi_{j-1}(\overrightarrow{XY}(1^k, a))) = 1) - \text{Prob}(D(1^k, z_k, a, \overrightarrow{MXY}(1^k, a)) = 1)| \geq 1/(2\,|V|\,\mathsf{Poly}(k)) \quad (19)$$

or

$$|\text{Prob}(D(1^k, z_k, a, \overrightarrow{MXY}(1^k, a)) = 1) - \text{Prob}(D(1^k, z_k, a, \phi_j(\overrightarrow{XY}(1^k, a))) = 1)| \geq 1/(2\,|V|\,\mathsf{Poly}(k)) \quad (20)$$

Without loss of generality, we assume that Inequality (19) holds (in the case where Inequality (20) holds, the proof is similar). The difference between $\overrightarrow{MXY}(1^k, a)$ and $\phi_{j-1}(\overrightarrow{XY}(1^k, a))$ is on their $\sigma(i_j)$-th entry. Specifically:

$$\overrightarrow{MXY}(1^k, a)\langle\sigma(i_j)\rangle \equiv X(1^k, a),$$
$$\phi_{j-1}(\overrightarrow{XY}(1^k, a))\langle\sigma(i_j)\rangle \equiv Y(1^k, a).$$

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

- $D'(1^k, z_k, a, \gamma)$: Sample an instance vector $\overrightarrow{mxy}$ according to the random variable $\overrightarrow{MXY}(1^k, a)$ and set γ to be its $\sigma(i_j)$-th entry. Return $D'(1^k, z_k, a, \overrightarrow{mxy})$.

Obviously, we have:

$$|\text{Prob}(D'(1^k, z_k, a, X(1^k, a)) = 1) - \text{Prob}(D'(1^k, z_k, a, Y(1^k, a)) = 1)| = |\text{Prob}(D(1^k, z_k, a, \overrightarrow{MXY}(1^k, a)) = 1) - \text{Prob}(D(1^k, z_k, a, \phi_{j-1}(\overrightarrow{XY}(1^k, a))) = 1)|. \quad (21)$$

Combining Inequality (19) and Inequality(21), we obtain:

$$|\text{Prob}(D'(1^k, z_k, a, X(1^k, a)) = 1) - \text{Prob}(D'(1^k, z_k, a, Y(1^k, a)) = 1)| \geq 1/(2\,|V|\,\mathsf{Poly}(k))$$

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

Remark 50. *Let* $\mathsf{Poly}_1(.), \mathsf{Poly}_2(.)$ *be two arbitrary positive polynomials such that* $\mathsf{Poly}_1(.) < \mathsf{Poly}_2(.)$*. It is easy to verify that Lemma 49 also holds in the case where* $t = \mathsf{Poly}_1(k)$*,* $n = \mathsf{Poly}_2(k)$*.*

We are ready to prove Theorem 43.

Proof. We reuse the notations HS_i from Definition 21, and BHS_i from Definition 41. For a hash family $\overline{\mathcal{H}}$ in Construction 1, we define the following probability ensemble for each $j \in [n]$.

$$\mathsf{HS}^j{=}\{\mathsf{HS}^j(1^k)\}_{k\in\mathbb{N}} \stackrel{def}{=} \{(\mathsf{HS}_1(1^k)\langle 1\rangle, \mathsf{HS}_1(1^k)\langle 2\rangle\langle j\rangle)\}_{k\in\mathbb{N}},$$

Recall that $\mathsf{HS}_1(1^k)\langle 1\rangle$ corresponds to the hash parameter Λ, and $\mathsf{HS}_1(1^k)\langle 2\rangle$ corresponds to the instance vector $\vec{x}^{\vec{a}}$. It is easy to see that: $\forall j \in [t]\ \mathsf{HS}^j \equiv \mathsf{BHS}_1$ and: $\forall j \notin [t]\ \mathsf{HS}^j \equiv \mathsf{BHS}_2$. Combining the fact that basic hash family $\mathcal{H}$ holds the hard subset membership property (i.e., $\mathsf{BHS}_1 \stackrel{c}{=} \mathsf{BHS}_2$) and Lemma 49, we have:

$$((\mathsf{HS}_1(1^k)\langle 1\rangle, \mathsf{HS}_1(1^k)\langle 2\rangle\langle 1\rangle), \ldots (\mathsf{HS}_1(1^k)\langle 1\rangle, \mathsf{HS}_1(1^k)\langle 2\rangle\langle n\rangle)) \stackrel{c}{=} (\mathsf{HS}_2(1^k)\langle 1\rangle, \mathsf{HS}_2(1^k)\langle 2\rangle\langle 1\rangle), \ldots (\mathsf{HS}_2(1^k)\langle 1\rangle, \mathsf{HS}_2(1^k)\langle 2\rangle\langle n\rangle)).$$

Since $\mathsf{HS}_1(1^k)\langle 1\rangle \equiv \mathsf{HS}_2(1^k)\langle 1\rangle$, we have:

$$(\mathsf{HS}_1(1^k)\langle 1\rangle, \mathsf{HS}_1(1^k)\langle 2\rangle\langle 1\rangle, \ldots, \mathsf{HS}_1(1^k)\langle 2\rangle\langle n\rangle) \stackrel{c}{=} (\mathsf{HS}_2(1^k)\langle 1\rangle, \mathsf{HS}_2(1^k)\langle 2\rangle\langle 1\rangle, \ldots, \mathsf{HS}_2(1^k)\langle 2\rangle\langle n\rangle),$$

That is: $\mathsf{HS}_1 \stackrel{c}{=} \mathsf{HS}_2$. □

7. Constructing Basic Hash Families

In the previous section, we reduced the existence of a SPH-DHM to the construction of a basic hash family having smoothness (or universality), projection, distinguishability and hard subset membership. In this section, we show how to create such families under several mathematical hardness assumptions: DDH, LWE, DNR and DQR.

7.1. Instantiation under the Decisional Diffie-Hellman Assumption

Let $\mathsf{Gen}(1^k)$ be a PPT algorithm taking a security parameter k as input and outputting the description of a cyclic group G of prime order: $(g_1, g_2, q) \leftarrow \mathsf{Gen}(1^k)$ where g_1, g_2 and q are respectively two distinct generators of G and the group order.

Assumption 51. *The* decisional Diffie-Hellman (DDH) assumption *assumes that there is no non-uniform PPT algorithm distinguishing the following two probability ensembles* DDH_1=$\{\mathsf{DDH}_1(1^k)\}_{k\in\mathbb{N}}$ *and* DDH_2=$\{\mathsf{DDH}_2(1^k)\}_{k\in\mathbb{N}}$ *defined as:*

- $\mathsf{DDH}_1(1^k)$*: Set* $(g_1, g_2, q) \leftarrow \mathsf{Gen}(1^k)$ *and choose* $a \in_U \mathbb{Z}_q$, $b \in_U \mathbb{Z}_q$. *Output* $((g_1, g_2, q), g_1^a, g_2^b)$.
- $\mathsf{DDH}_2(1^k)$*: Operate as* $\mathsf{DDH}_1(1^k)$ *but return* $((g_1, g_2, q), g_1^a, g_2^a)$.

Our DDH-based basic hash family whose syntax is specified in Section 6.1 is described in Construction 2.

Construction 2 A DDH-based basic hash family

- $\mathsf{PG}(1^k)$: Set $\Lambda \leftarrow \mathsf{Gen}(1^k)$ and return Λ.
- $\mathsf{IS}(1^k, \Lambda, \delta)$: Parse Λ as (g_1, g_2, q) and choose $a \in_U \mathbb{Z}_q$, $b \in_U \mathbb{Z}_q$. Set $\dot{x} \leftarrow (g_1^a, g_2^a)$, $\dot{w} \leftarrow a$, $\ddot{x} \leftarrow (g_1^a, g_2^b)$ and $\ddot{w} \leftarrow a$. Return $(\dot{x}, \dot{w})$ if $\delta = 0$ or $(\ddot{x}, \ddot{w})$ if $\delta = 1$.
- $\mathsf{Check}(1^k, \Lambda)$: Parse Λ as (g_1, g_2, q). If q is a prime number of appropriate bitlength, and g_1, g_2 are two distinct generators, then outputs 1. Otherwise, output 0.
- $\mathsf{DI}(1^k, \Lambda, x, w)$: Parse Λ as (g_1, g_2, q) and x as (α, β). If $(\alpha, \beta) = (g_1^w, g_2^w)$, then output 0. If $\alpha = g_1^w$ and $\beta \neq g_2^w$, then output 1. Otherwise return 2.
- $\mathsf{KG}(1^k, \Lambda, x)$: Parse Λ as (g_1, g_2, q) and choose $u \in_U \mathbb{Z}_q$, $v \in_U \mathbb{Z}_q$. Set $hk \leftarrow (u, v)$ and compute $pk \leftarrow g_1^u g_2^v$. Output (hk, pk).
- $\mathsf{Hash}(1^k, \Lambda, x, hk)$: Parse x as (α, β) and hk as (u, v). Compute $y \leftarrow \alpha^u \beta^v$ and return y.
- $\mathsf{pHash}(1^k, \Lambda, x, pk, w)$: Compute $y \leftarrow pk^w$ and output y.

Lemma 52. *The hash family obtained from Construction 2 has perfect smoothness.*

Proof. Let $\Lambda = (g_1, g_2, q)$ be a legal hash parameter. Let $(\ddot{x}, a) = ((g_1^a, g_2^b), a)$ be an instance-witness pair generated by $\mathsf{IS}(1^k, \Lambda, 1)$. For this family, the probability ensembles BSM_1, BSM_2 in Definition 38 can be described as follows.

- $\mathsf{BSM}_1(1^k)$: Choose $u \in_U \mathbb{Z}_q$ and $v \in_U \mathbb{Z}_q$. Set $hk \leftarrow (u, v)$, $pk \leftarrow g_1^u g_2^v$ and $y \leftarrow g_1^{au} g_2^{bv}$. Output (pk, y).
- $\mathsf{BSM}_2(1^k)$: Run as $\mathsf{BSM}_1(1^k)$ with the exception that $y \in_U G$, where G is the group described by (g_1, g_2, q).

The difference between the two ensembles is the generation of y. Let's consider the value y generated by BSM_1. Since g_1 is a generator of the cyclic group G, there exists a constant c such that $g_2 = g_1^c$. Then: $g_1^{au} g_2^{bv} = g_1^{au+cbv}$. Since u and v are chosen uniformly from $\mathbb{Z}_q$, $au + cbv$ is also uniformly distributed over $\mathbb{Z}_q$. Since q is prime, $g_1^{au+cbv} (= y)$ is uniformly distributed over the group G. Therefore: $\mathsf{BSM}_1 \equiv \mathsf{BSM}_2$ since the value y created by $\mathsf{BSM}_2(1^k)$ is, by construction, uniformly uniformly distributed over G. □

Lemma 53. *The hash family obtained from Construction 2 has the projection property.*

Proof. Write $\Lambda = (g_1, g_2, q)$ and let $(\dot{x}, \dot{w}) = ((g_1^a, g_2^a), a)$ denote an instance-witness pair generated by $\mathsf{IS}(1^k, \Lambda, 0)$. Let $(hk, pk) = ((u, v), g_1^u g_2^v)$ be a hash-projection key pair generated by $\mathsf{KG}(1^k, \Lambda, \dot{x})$. We have:

$$\mathsf{Hash}(1^k, \Lambda, \dot{x}, hk) = \mathsf{Hash}(1^k, \Lambda, (g_1^a, g_2^a), (u, v)) = g_1^{au} g_2^{av},$$
$$\mathsf{pHash}(1^k, \Lambda, \dot{x}, pk, \dot{w}) = \mathsf{pHash}(1^k, \Lambda, (g_1^a, g_2^a), g_1^u g_2^v, a) = g_1^{au} g_2^{av}.$$

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

Lemma 54. *The hash family obtained from Construction 2 has the distinguishability property.*

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

Lemma 55. *Assuming that the DDH assumption holds, the hash family obtained from Construction 2 has hard subset membership.*

Proof. For this family, the probability ensembles BHS_1, BHS_2 in Definition 41 can be described as follows.

- $\mathsf{BHS}_1(1^k)$: Generate $\Lambda \leftarrow \mathsf{PG}(1^k)$, and parse Λ as (g_1, g_2, q). Choose $a \in_U \mathbb{Z}_q$ and compute $\dot{x} \leftarrow (g_1^a, g_2^a)$. Return $(\Lambda, \dot{x})$.
- $\mathsf{BHS}_2(1^k)$: Generate $\Lambda \leftarrow \mathsf{PG}(1^k)$, and parse Λ as (g_1, g_2, q). Choose $a \in_U \mathbb{Z}_q$, $b \in_U \mathbb{Z}_q$ and compute $\ddot{x} \leftarrow (g_1^a, g_2^b)$. Return $(\Lambda, \ddot{x})$.

Obviously: $\mathsf{BHS}_1 \stackrel{c}{=} \mathsf{BHS}_2$. □

7.2. Instantiation under the Learning with Errors Assumption

We start this section with some preliminaries related to the discretization of probabilistic distributions.

We consider the quotient ring $\mathbb{R}/\mathbb{Z}$ representing the segment $[0, 1)$ with addition modulo 1. Let β be an arbitrary positive real number. We denote by Ψ_β the distribution over $\mathbb{R}/\mathbb{Z}$ obtained by sampling from a normal variable with mean 0 and standard deviation $\beta/\sqrt{2\pi}$ and reducing the result modulo 1.

$$\begin{array}{rccl} \Psi_\beta: & \mathbb{R}/\mathbb{Z} & \longrightarrow & \mathbb{R}^+ \\ & r & \longmapsto & \displaystyle\sum_{k=-\infty}^{\infty} \frac{1}{\beta} \exp\left(-\pi\left(\frac{r-k}{\beta}\right)^2\right). \end{array}$$

Definition 56 ([43])**.** *Given an arbitrary integer $q \geq 2$, an arbitrary distribution $\phi : \mathbb{R}/\mathbb{Z} \to \mathbb{R}^+$, the* discretization *of ϕ over $\mathbb{Z}_q$ is defined as:*

$$\begin{array}{rccl} \overline{\phi}: & \mathbb{Z}_q & \longrightarrow & \mathbb{R}^+ \\ & i & \longmapsto & \displaystyle\int_{(i-1/2)/q}^{(i+1/2)/q} \phi(x)\, dx. \end{array}$$

Let $\mathsf{Gen}(1^k)$ be a PPT algorithm taking a security parameter k as input and outputs a description (q, m, χ), where q is a prime, $m = \mathsf{Poly}(k)$ and $\chi = \overline{\Psi}_\alpha$ where $\alpha \in (2\sqrt{k}/q, 1)$.

Assumption 57 ([43])**.** *The* decision version of the learning with errors (LWE) assumption *assumes that that there is no non-uniform PPT algorithm distinguishing the following two probability ensembles LWE_1=$\{\mathsf{LWE}_1(1^k)\}_{k\in\mathbb{N}}$ and LWE_2=$\{\mathsf{LWE}_2(1^k)\}_{k\in\mathbb{N}}$ defined as:*

- $\mathsf{LWE}_1(1^k)$*: Set $(q, m, \chi) \leftarrow \mathsf{Gen}(1^k)$ and choose $A \in_U (\mathbb{Z}_q)^{m\times k}$, $\vec{s} \in_U (\mathbb{Z}_q)^k$, $\vec{e} \in_\chi (\mathbb{Z}_q)^m$ (which means each entry of $\vec{e}$ is independently drawn from $\mathbb{Z}_q$ according to χ). Compute $\vec{b} \leftarrow A\vec{s} + \vec{e}$ mod q and output $((q, m, \chi), A, \vec{b})$.*
- $\mathsf{LWE}_2(1^k)$*: Operate as $\mathsf{LWE}_1(1^k)$ except that $\vec{b} \in_U (\mathbb{Z}_q)^m$.*

Remark 58. *Assumption 57 describes only the decision version of LWE problem. Nonetheless, the hardness of this decision problem is the same as the LWE problem itself [43].*

Remark 59. *The LWE problem is an average-case problem. However, Regev showed that its hardness is implied by both the worst-case hardness of the* shortest independent vector problem *(SIVP) and the worst-case hardness of the* decision version of the shortest vector problem *(GapSVP) (see Theorem 1 of [43]). In other words, if the LWE problem is efficiently solvable, so are both the SIVP and GapSVP. Since these two problems are believed to be hard even for quantum algorithms [44], so is the LWE problem.*

Before presenting our LWE-based instantiation of basic hash family, we need to recall the key generation algorithm of a LWE-based public key cryptosystem presented by Gentry *et al.* to be used as a building block [31].

- Message space: $\{0,1\}$.
- $\mathsf{Setup}(1^k)$: Uniformly chooses a prime q such that $q \in [k^2, 2k^2]$. Set $m \leftarrow (1+\varepsilon)(k+1)\log q$ (where $\varepsilon > 0$ is an arbitrary constant and arbitrarily choose α such that $\alpha = o(1/(\sqrt{k}\log k))$ (e.g., $\alpha = \frac{1}{\sqrt{k}(\log k)^2}$). Set $\chi \leftarrow \overline{\Psi}_\alpha$ and $\Lambda \leftarrow (q, m, \chi)$. Output Λ.
- $\mathsf{KeyGen}(1^k, \Lambda)$: Parse Λ as (q, m, χ). Choose $A \in_U (\mathbb{Z}_q)^{m\times k}$, $\vec{s} \in_U (\mathbb{Z}_q)^k$ and $\vec{e} \in_\chi (\mathbb{Z}_q)^m$. Compute $\vec{b} \leftarrow A\vec{s} + \vec{e} \bmod q$ and set $pubk \leftarrow (A, \vec{b})$ and $sk \leftarrow \vec{s}$. Return the public-private key pair $(pubk, sk)$.

Definition 60 ([31])**.** *A public key $(A, \vec{b})$ is* messy *if and only if, for all $m_0, m_1 \in \{0,1\}$, the statistical difference between the distributions of $\mathsf{Enc}_{A,\vec{b}}(m_0)$ and $\mathsf{Enc}_{A,\vec{b}}(m_1)$ is negligible where Enc denotes the encryption algorithm of the cryptosystem.*

We would like to point out that any ciphertext produced under messy public keys carries no information (statistically) about the encrypted message. Therefore, for any (even unbounded) adversary, recovery of the corresponding plaintext is impossible. Gentry *et al.* showed that, if the construction parameters were appropriately chosen, their cryptosystem held the following properties.

P1 It provides security against chosen plaintext attack. [For our construction, we only need semantic security.]

P2 All but an at most $2/q^k$-fraction of $(A, \vec{b}) \in ((\mathbb{Z}_q)^{m\times k}, (\mathbb{Z}_q)^m)$ is messy.

P3 There exists an efficient algorithm $\mathsf{IsMessy}$ that identifies messy public keys. It takes a public key $(A, \vec{b})$, a trapdoor T as input and outputs an indicator. $\mathsf{IsMessy}$ has the following properties.

- $\mathsf{IsMessy}$ outputs 1 on an overwhelming fraction of all possible public keys which may be maliciously generated.
- If $\mathsf{IsMessy}(A, T, \vec{b})$ outputs 1 on a public key $(A, \vec{b})$, then $(A, \vec{b})$ is indeed messy.

We do not know how to gain distinguishability as defined in Definition 14. Instead, we will be able to obtained a relaxed version defined as follows.

Definition 61 (Relaxed Distinguishability)**.** *For any legal hash parameter Λ, any instance-witness pair $(x, w) \in (\{0,1\}^*)^2$ which may be maliciously generated, we require:*

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

In our framework for OT_t^n, the only usage of DI is to check the legalities of instance-witness vectors. According to the legality definition (Definition 18), an instance-witness vector is called legal if and only if it has at least $n - t$ entries that on receiving them DI outputs 1. Therefore, this relaxed distinguishability is sufficient for our objective.

Our LWE-based basic hash family whose syntax is specified in Section 6.1 is described in Construction 3.

Lemma 62. *The hash family obtained from Construction 3 has smoothness.*

Proof. Let $\Lambda = (q, m, \chi)$ be a legal hash parameter. Let $(\ddot{x}, \ddot{w}) = ((A, \vec{b}), (1, T))$ be an instance-witness pair generated by $\mathsf{IS}(1^k, \Lambda, 1)$. For this family, the probability ensembles BSM_1 and BSM_2 from Definition 38 can be described as follows.

- $\mathsf{BSM}_1(1^k)$: Choose $a \in_U \{0,1\}$ and compute $pk \leftarrow \mathsf{Enc}_{A,\vec{b}}(a)$. Set $y \leftarrow a$ and output (pk, y).
- $\mathsf{BSM}_2(1^k)$: Choose $a' \in_U \{0,1\}$ and compute $pk \leftarrow \mathsf{Enc}_{A,\vec{b}}(a')$. Pick $y \in_U \{0,1\}$ and output (pk, y).

Construction 3 A LWE-based basic hash family

(Setup, KeyGen, Enc, Dec) representing the cryptosystem from [31] where the parameters of Setup were tuned so that properties P1, P2 and P3 hold.

- $\mathsf{PG}(1^k)$: Set $\Lambda \leftarrow \mathsf{Setup}(1^k)$ and return Λ.
- $\mathsf{IS}(1^k, \Lambda, \delta)$: Parse Λ as (q, m, χ) and choose $A \in_U (\mathbb{Z}_q)^{m \times k}$ along with its trapdoor T, $\vec{s} \in_U (\mathbb{Z}_q)^k$ and $\vec{e} \in_\chi (\mathbb{Z}_q)^m$. Compute $\dot{x} \leftarrow (A, A\vec{s}+\vec{e} \bmod q)$ and set $\dot{w} \leftarrow (0, \vec{e}, \vec{s})$. Pick $\vec{b} \in_U (\mathbb{Z}_q)^m$ such that $\mathsf{IsMessy}(A, T, \vec{b}) = 1$ and set $\ddot{x} \leftarrow (A, \vec{b})$, $\ddot{w} \leftarrow (1, T)$. Output $(\dot{x}, \dot{w})$ if $\delta = 0$, $(\ddot{x}, \ddot{w})$ if $\delta = 1$.
- $\mathsf{Check}(1^k, \Lambda)$: Parse Λ as (q, m, χ). If q is a prime number, m is a positive integer, and χ is a discretization of an appropriate distribution then output 1. Otherwise, return 0.
- $\mathsf{DI}(1^k, \Lambda, x, w)$: Parse Λ as (q, m, χ) and x as $(A, \vec{b})$. Let i denote the value of w's first entry. If $i = 0$, then $(0, \vec{e}, \vec{s}) \leftarrow w$; if $i = 1$, then $(1, T) \leftarrow w$; otherwise output 2. Perform the following checks:
  - if $i = 0$: Check $\vec{b}, \vec{e} \in (\mathbb{Z}_q)^m$, $A \in (\mathbb{Z}_q)^{m \times k}$, $\vec{s} \in (\mathbb{Z}_q)^k$ and $\vec{b} = A\vec{s} + \vec{e} \bmod q$. If the checks are all successful then output 0; else output 2.
  - if $i = 1$: Check $\vec{b} \in (\mathbb{Z}_q)^m$, $A \in (\mathbb{Z}_q)^{m \times k}$, T is a trapdoor and $\mathsf{IsMessy}(A, T, \vec{b}) = 1$. If the checks are all successful then output 1; otherwise output 2.
- $\mathsf{KG}(1^k, \Lambda, x)$: Parse Λ as (q, m, χ) and x as $(A, \vec{b})$. Choose $a \in_U \{0, 1\}$ and compute $\alpha \leftarrow \mathsf{Enc}_{A,\vec{b}}(a)$. Set $hk \leftarrow a$, $pk \leftarrow \alpha$ and output (hk, pk).
- $\mathsf{Hash}(1^k, \Lambda, x, hk)$: Output hk.
- $\mathsf{pHash}(1^k, \Lambda, x, pk, w)$: Parse Λ as (m, q, χ) and w as $(0, \vec{e}, \vec{s})$. Compute $a \leftarrow \mathsf{Dec}_{\vec{s}}(pk)$ and output a.

It is easy to see that the second component of each ensemble is uniformly distributed. The difference between these two ensembles is their first value denoted by pk. In the case of $\mathsf{BSM}_1(1^k)$, it is the encryption of y while, for $\mathsf{BSM}_2(1^k)$, it corresponds to the encryption of a'. Because $(A, \vec{b})$ is messy, the two distributions of this value pk are statistically indistinguishable. Therefore, $\mathsf{BSM}_1 \stackrel{s}{=} \mathsf{BSM}_2$. □

Lemma 63. *The hash family obtained from Construction 3 has the projection property.*

Proof. Let $(\dot{x}, \dot{w}) = ((A, \vec{b}), (0, \vec{e}, \vec{s}))$ be an instance-witness generated by $\mathsf{IS}(1^k, \Lambda, 0)$. Obviously, $((A, \vec{b}), \vec{s})$ is a normal public-private key pair. Then, we have:

$$\mathsf{Hash}(1^k, \Lambda, \dot{x}, hk) = a \qquad \text{and} \qquad \mathsf{pHash}(1^k, \Lambda, \dot{x}, pk, \dot{w}) = \mathsf{Dec}_{\vec{s}}(\alpha) = \mathsf{Dec}_{\vec{s}}(\mathsf{Enc}_{A,\vec{b}}(a)) = a$$

This means that, for any $(\Lambda, \dot{x}, \dot{w})$ generated by the hash family, it holds: $\mathsf{Hash}(1^k, \Lambda, \dot{x}, hk) = \mathsf{pHash}(1^k, \Lambda, \dot{x}, pk, \dot{w})$. □

Lemma 64. *The hash family obtained from Construction 3 has the distinguishability property.*

Proof. Let Λ be a legal hash parameter, (x, w) an instance-witness pair that may be maliciously generated. It is trivial to see that DI outputs 0 if (x, w) can be generated by $\mathsf{IS}(1^k, \Lambda, 0)$ (i.e., $(x, w) \in \dot{R}_\Lambda$).

Recalling the property of algorithm $\mathsf{IsMessy}$, we know that if DI outputs 1, then $(x, w) \in \ddot{R}_\Lambda$; if $(x, w) \in \ddot{R}_\Lambda$, then DI outputs 1 with probability almost 1. □

Lemma 65. *Assuming the LWE assumption holds, the hash family obtained from Construction 3 has hard subset membership.*

Proof. For this family, the probability ensembles BHS_1, BHS_2 from Definition 41 can be described as follows.

- $\mathsf{BHS}_1(1^k)$: Generate $\Lambda \leftarrow \mathsf{PG}(1^k)$ and parse it as (q, m, χ). Choose $A \in_U (\mathbb{Z}_q)^{m \times k}$ along with its trapdoor T, $\vec{s} \in_U (\mathbb{Z}_q)^k$ and $\vec{e} \in_\chi (\mathbb{Z}_q)^m$. Compute $\dot{x} \leftarrow (A, A\vec{s} + \vec{e} \bmod q)$ and output $(\Lambda, \dot{x})$.
- $\mathsf{BHS}_2(1^k)$: Generate $\Lambda \leftarrow \mathsf{PG}(1^k)$ and parse it as (q, m, χ). Choose $A \in_U (\mathbb{Z}_q)^{m \times k}$ along with its trapdoor T and pick $\vec{b} \in_U (\mathbb{Z}_q)^m$ such that $\mathsf{IsMessy}(A, T, \vec{b}) = 1$. Set $\ddot{x} \leftarrow (A, \vec{b})$ and output $(\Lambda, \ddot{x})$.

Obviously, $\mathsf{BHS}_1 \stackrel{c}{=} \mathsf{BHS}_2$. □

Since, in our basic hash family, each instance holds a matrix A, so does the hash family resulting from Construction 1. It is trivial to show that this hash family can be improved by each instance vector sharing a common matrix A. However, to securely instantiate our framework for OT^n_t, it cannot be improved by all instance vectors sharing a common matrix A. The reason is that, in this case, seeing A's trapdoor T in Step S2 of the framework Π, the sender can distinguish smooth instances and projective instances of the unchosen instance vectors and thus he can deduce the receiver's private input.

7.3. Instantiation under the Decisional Quadratic Residuosity Assumption

Let $\mathsf{Gen}(1^k)$ be a PPT algorithm taking a security parameter k as input and returning an integer N being the product of two distinct k-bit odd primes. Let J_N be the subgroup of $\mathbb{Z}^*_N$ of elements having Jacobi symbol +1.

Assumption 66 ([45])**.** *The* decisional quadratic residuosity (DQR) assumption *assumes that there is no non-uniform PPT algorithm distinguishing the following two probability ensembles* $\mathsf{DQR}_1 = \{\mathsf{DQR}_1(1^k)\}_{k \in \mathbb{N}}$ *and* $\mathsf{DQR}_2 = \{\mathsf{DQR}_2(1^k)\}_{k \in \mathbb{N}}$ *defined as:*

- $\mathsf{DQR}_1(1^k)$*: Generate* $N \leftarrow \mathsf{Gen}(1^k)$*, choose* $r \in_U \mathbb{Z}^*_N$ *and compute* $x \leftarrow r^2 \bmod N$*. Output* (N, x)*.*
- $\mathsf{DQR}_2(1^k)$*: Operate as* $\mathsf{DQR}_1(1^k)$ *except* $x \in_U J_N$*.*

Definition 67 ([25]). *A hash family* $\mathcal{H} = (\mathsf{PG}, \widehat{\mathsf{IS}}, \widehat{\mathsf{IT}}, \widehat{\mathsf{KG}}, \mathsf{Hash}, \mathsf{pHash})$ *is a* verifiably-ε-universal projective hashing family*, if for any* Λ *generated by* $PG(1^k)$*, any* $(\dot{x}, \dot{w}, \ddot{x})$ *generated by* $\widehat{\mathsf{IS}}(1^k, \Lambda)$*,* $\mathcal{H}$ *is projective on* $\dot{x}$*, and universal on* $\ddot{x}$*. In addition, it holds verifiable-*ε*-universallity, which is defined as follows.*

- *For any* Λ *generated by* $\mathsf{PG}(1^k)$*, any* $(\dot{w}, \dot{x}, \ddot{x})$ *generated by* $\widehat{\mathsf{IS}}(1^k, \Lambda)$*, we have:* $\widehat{\mathsf{IT}}(1^k, \Lambda, \dot{x}, \ddot{x}) = \widehat{\mathsf{IT}}(1^k, \Lambda, \ddot{x}, \dot{x}) = 1$.
- *For any tuple* (Λ, x_1, x_2) *which may be maliciously generated, if* $\widehat{\mathsf{IT}}(1^k, \Lambda, x_1, x_2) = 1$*, then* $\mathcal{H}$ *is* ε*-universal on at least one of* x_1 *or* x_2.

We can see that the syntax of a hash family appearing in [25] is different from our basic hash family notion. It does not have algorithms like Check or DI. However, it possesses an algorithm $\widehat{\mathsf{IT}}$. There are some additional syntaxic differences not reflected by above definition. In a Halevi and Tauman Kalai hash family, $\widehat{\mathsf{IS}}$ does not take any work mode parameter δ and $\widehat{\mathsf{IS}}(1^k, \Lambda)$ outputs a tuple $(\dot{x}, \dot{w}, \ddot{x})$ where typically $\dot{x}$ and $\ddot{x}$ are generated in dependent way. In addition, their key generating algorithm $\widehat{\mathsf{KG}}$ does not take any instance parameter x.

Under the DQR assumption, Halevi and Tauman Kalai proposed the following verifiably-ε-universal projective hash family.

Construction 4 A verifiably-universal projective hash family [25]

- $\mathsf{PG}(1^k)$: Choose uniformly two k-bit prime p, q such that $p = q = 3 \bmod 4$. Set $N \leftarrow pq$, choose $a \in_U Z_N^*$ and compute $g \leftarrow a^2 \bmod N$. Set $\Lambda \leftarrow (N, g)$ and output Λ.
- $\widehat{\mathsf{IS}}(1^k, \Lambda)$: Parse Λ as (N, g). Set $T \leftarrow 2^{\lceil \log N \rceil}$, choose $\dot{w} \in_U \mathbb{Z}_N$ and compute $\dot{x} \leftarrow (g^T)^{\dot{w}} \bmod N$ and $\ddot{x} \leftarrow N - \dot{x}$. Output $(\dot{w}, \dot{x}, \ddot{x})$.
- $\widehat{\mathsf{IT}}(1^k, \Lambda, \dot{x}, \ddot{x})$: Parse Λ as (N, g). Checks that $N > 2^{2n}$, $g, \dot{x} \in Z_N^*$, and $\ddot{x} = N - \dot{x}$. Outputs 1 if all these tests pass and 0 otherwise.
- $\widehat{\mathsf{KG}}(1^k, \Lambda)$: Parse Λ as (N, g). Set $T \leftarrow 2^{\lceil \log N \rceil}$, choose $hk \in_U \mathbb{Z}_N$ and compute $pk \leftarrow (g^T)^{hk} \bmod N$. Output (hk, pk).
- $\mathsf{Hash}(1^k, \Lambda, x, hk)$: Parse Λ as (N, g) and compute $y \leftarrow x^{hk} \bmod N$. Output y.
- $\mathsf{pHash}(1^k, \Lambda, x, pk, w)$: Parse Λ as (N, g) and compute $y \leftarrow pk^w \bmod N$. Output y.

Our DQR-based basic hash family whose syntax is specified in Section6.1 is described in Construction 5.

Construction 5 A DQR-based basic hash family

- Algorithms PG, KG, Hash, pHash operate as in Construction 4.
- $\mathsf{IS}(1^k, \Lambda, \delta)$: Parse Λ as (N, g). Set $T \leftarrow 2^{\lceil \log N \rceil}$, choose $r \in_U \mathbb{Z}_N$ and compute $\dot{x} \leftarrow (g^T)^r \bmod N$. Set $\dot{w} \leftarrow r$, compute $\ddot{x} \leftarrow N - (g^T)^r \bmod N$ and assign $\ddot{w} \leftarrow r$. Output $(\dot{x}, \dot{w})$ if $\delta = 0$; output $(\ddot{x}, \ddot{w})$ if $\delta = 1$.
- $\mathsf{Check}(1^k, \Lambda)$: Parse Λ as (N, g). If it holds that $N > 2^{2k}$ and $g \in Z_N^*$, then output 1; otherwise, output 0.
- $\mathsf{DI}(1^k, \Lambda, x, w)$: Parse Λ as (N, g). Set $T \leftarrow 2^{\lceil \log N \rceil}$ and $r \leftarrow w$. If $x = (g^T)^r \bmod N$, then output 0. If $x = N - (g^T)^r \bmod N$, then output 1. Otherwise, output 2.

It is easy to see that the basic hash family obtained from Construction 5 inherits the universality and projection properties from Construction 4.

Lemma 68. *The hash family obtained from Construction 5 has the distinguishability property.*

Proof. Let Λ be a legal hash parameter and (x, w) an instance-witness pair that may be maliciously generated. It is easy seen that $\mathsf{DI}(1^k, \Lambda, x, w) = 0$ if and only if (x, w) can be generated by $\mathsf{IS}(1^k, \Lambda, 0)$ (i.e., $(x, w) \in \dot{R}_\Lambda$) and $\mathsf{DI}(1^k, \Lambda, x, w) = 1$, if and only if (x, w) can be generated by $\mathsf{IS}(1^k, \Lambda, 1)$ (i.e., $(x, w) \in \ddot{R}_\Lambda$). □

Lemma 69. *Assuming that the DQR assumption holds, the hash family obtained from Construction 5 has hard subset membership.*

Proof. For this family, the probability ensembles BHS_1 and BHS_2 from Definition 41 can be described as follows.

- $\mathsf{BHS}_1(1^k)$: Generate $(N, g) \leftarrow \mathsf{PG}(1^k)$. Set $T \leftarrow 2^{\lceil \log N \rceil}$, choose $r \in_U \mathbb{Z}_N$ and compute $\dot{x} \leftarrow (g^T)^r \bmod N$. Output $(\Lambda, \dot{x})$.
- $\mathsf{BHS}_2(1^k)$: Operate as $\mathsf{BHS}_1(1^k)$ except that it outputs $(\Lambda, \ddot{x})$, where $\ddot{x} \leftarrow N - (g^T)^r \bmod N$.

It is easy to see that $(g^T)^r \bmod N$ is a quadratic residue modulo N and N is a Blum integer. Following the properties of Blum integers, $N - (g^T)^r \bmod N$ is not a quadratic residue and $N - (g^T)^r \bmod N \in J_N$. Therefore, if the DQR assumption holds, we have: $\mathsf{BHS}_1 \stackrel{c}{=} \mathsf{BHS}_2$. □

7.4. Instantiation under the Decisional N-th Residuosity Assumption

We use the same parameter generator $\mathsf{Gen}(1^k)$ as in Section 7.3.

Assumption 70 ([46])**.** *The* decisional N-th residuosity (DNR) assumption *assumes that there is no non-uniform PPT algorithm distinguishing the following two probability ensembles* $\mathsf{DNR}_1 = \{\mathsf{DNR}_1(1^k)\}_{k\in\mathbb{N}}$ *and* $\mathsf{DNR}_2 = \{\mathsf{DNR}_2(1^k)\}_{k\in\mathbb{N}}$ *defined as follows:*

- $\mathsf{DNR}_1(1^k)$*: Generate* $N \leftarrow \mathsf{Gen}(1^k)$ *and choose* $a \in_U \mathbb{Z}^*_{N^2}$*. Compute* $b \leftarrow a^N \bmod N^2$ *and output* (N, b)*.*
- $\mathsf{DNR}_2(1^k)$*: Operate as* $\mathsf{DNR}_1(1^k)$ *except that* $b \in_U \mathbb{Z}^*_{N^2}$*.*

Similarly to our DQR-based basic hash family, our DNR-based basic hash family is built from Halevi and Kalai's DNR-based verifiably-ε-universal projective hash family [25]. The modifications and security proofs are similar too. To save space, we omit these details and only describe our basic hash family in Construction 6.

Construction 6 A DNR-based basic hash family

- $\mathsf{PG}(1^k)$: Generate $N \leftarrow \mathsf{Gen}(1^k)$, choose $a \in_U Z^*_{N^2}$ and compute $g \leftarrow a^N \bmod N$. Set $\Lambda \leftarrow (N, g)$ and return Λ.
- $\mathsf{IS}(1^k, \Lambda, \delta)$: Parse Λ as (N, g). Set $T \leftarrow N^{\lceil 2 \log N \rceil}$, choose $r \in_U \mathbb{Z}^*_N$ and compute $\dot{x} \leftarrow (g^T)^r \bmod N^2$. Set $\dot{w} \leftarrow (r, 0)$, $v \in_U \mathbb{Z}^*_N$, choose $\ddot{x} \leftarrow (g^T)^r(1 + vN) \bmod N^2$ and $\ddot{w} \leftarrow (r, v)$. Output $(\dot{x}, \dot{w})$ if $\delta = 0$; output $(\ddot{x}, \ddot{w})$ if $\delta = 1$.
- $\mathsf{Check}(1^k, \Lambda)$: Parse Λ as (N, g). If $N > 2^{2k}$ and $g \in \mathbb{Z}^*_{N^2}$, then output 1. Otherwise, output 0.
- $\mathsf{DI}(1^k, \Lambda, x, w)$: Parse Λ as (N, g) and w as (r, v). If $v = 0$, $r \in \mathbb{Z}^*_N$ and $x = (g^T)^r \bmod N^2$, then output 0. If $v \neq 0$, $r, v \in \mathbb{Z}^*_N$ and $x = (g^T)^r(1 + vN) \bmod N^2$, then output 1. Otherwise, output 2.
- $\mathsf{KG}(1^k, \Lambda, x)$: Parse Λ as (N, g). Set $T \leftarrow N^{\lceil 2 \log N \rceil}$, choose $hk \in_U \mathbb{Z}_{N^2}$ and compute $pk \leftarrow (g^T)^{hk} \bmod N^2$. Output (hk, pk).
- $\mathsf{Hash}(1^k, \Lambda, x, hk)$: Output $x^{hk} \bmod N^2$.
- $\mathsf{pHash}(1^k, \Lambda, x, pk, w)$: Output $pk^w \bmod N^2$.

8. Conclusion

In this paper, we proposed a framework Π for OT_t^n with fully-simulatable SAMA working in the plain model. Our construction is based on a new SPH variant called SPH-DHM. This approach reduced the existence of protocols for OT_t^n to this primitive. We showed how to generate a SPH-DHM under several hardness assumption such as DDH, DRN, DQR and LWE. In an efficiency point of view, our framework is quite performant as Π only needs 4 communication rounds and it expectedly costs $20\,n$ encryptions and $20\,t$ decryptions. This work leads to two interesting open problems. First, can the existence of SPH be reduced to the existence of OT? Since both SPH and OT are public-key cryptographic primitives, we conjecture this might be the case. Second, before this paper, known OT protocols required at least 6 communication rounds. Since our protocol Π only has 4 rounds, one may wonder if this value is necessary to realize OT with fully-simulatable SAMA in the plain model. In other words, is it possible to design such a secure framework with at most 3 rounds?

References

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

[2] J. Kilian, Founding cryptography on oblivious transfer, in: 20th Annual ACM Symposium on Theory of Computing (STOC'88), ACM Press, Chicago, USA, 1988, pp. 20 – 31.

[3] G. Aggarwal, N. Mishra, B. Pinkas, Secure computation of the median (and other elements of specified ranks), Journal of Cryptology 23 (3) (2010) 373–401.

[4] N. Mohammed, D. Alhadidi, B. C. M. Fung, M. Debbabi, Secure two-party differentially private data release for vertically partitioned data, Dependable and Secure Computing, IEEE Transactions on 11 (1) (2014) 59–71.

[5] C. Hazay, K. Nissim, Efficient set operations in the presence of malicious adversaries, Journal of Cryptology 25 (3) (2012) 383–433.

[6] B. Aiello, Y. Ishai, O. Reingold, Priced oblivious transfer: How to sell digital goods, in: B. Pfitzmann (Ed.), Advances in Cryptology - Eurocrypt'01, Vol. 2045 of Lecture Notes in Computer Science, Springer - Verlag, Innsbruck, Austria, 2001, pp. 119 – 135.

[7] S. Jarecki, X. M. Liu, Private mutual authentication and conditional oblivious transfer, in: S. Halevi (Ed.), Advances in Cryptology - Crypto 2009, Vol. 5677 of Lecture Notes in Computer Science, 2009, pp. 90–107.

[8] Lindell, Pinkas, Privacy preserving data mining, Journal of Cryptology 15 (2002) 177–206.

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

[10] M. Green, S. Hohenberger, Practical adaptive oblivious transfer from simple assumptions, in: Y. Ishai (Ed.), 8th Theory of Cryptography Conference (TCC'11), Vol. 6597, Springer - Verlag, Providence, USA, 2011, pp. 347 – 363.

[11] W. Ogata, K. Kurosawa, Oblivious keyword search, Journal of complexity 20 (2) (2004) 356–371.

[12] M. Naor, B. Pinkas, Oblivious polynomial evaluation, SIAM Journal on Computing 35 (2006) 1254 – 1281.

[13] S. Even, O. Goldreich, A. Lempel, A randomized protocol for signing contracts, Communications of the ACM 28 (6) (1985) 637–647.

[14] K. Bharat, A. Broder, A technique for measuring the relative size and overlap of public web search engines, Computer Networks and ISDN Systems 30 (1) (1998) 379–388.

[15] D. Catalano, R. Cramer, I. Damgård, G. Di Crescenzo, D. Poincheval, T. Takagi, Contemporary Cryptology, Advanced Courses in Mathematics - CRM Barcelona, Birkhäuser, 2005.

[16] O. Goldreich, Foundations of Cryptography: Volume II - Basic Applications, Cambridge University Press, 2004.

[17] M. Naor, B. Pinkas, Efficient oblivious transfer protocols, in: 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01), Society for Industrial and Applied Mathematics, Washington, USA, 2001, pp. 448 – 457.

[18] J. Camenisch, G. Neven, abhi shelat, Simulatable adaptive oblivious transfer, in: M. Naor (Ed.), Advances in Cryptology - Eurocrypt'07, Vol. 4515 of Lecture Notes in Computer Science, Springer - Verlag, Barcelona, Spain, 2007, pp. 573 – 590.

[19] M. Green, S. Hohenberger, Blind identity-based encryption and simulatable oblivious transfer, in: K. Kurosawa (Ed.), Advances in Cryptology - Asiacrypt'07, Vol. 4833 of Lecture Notes in Computer Science, Springer - Verlag, Kuching, Malaysia, 2007, pp. 265 – 282.

[20] Y. Lindell, B. Pinkas, Secure two-party computation via cut-and-choose oblivious transfer, Journal of Cryptology 25 (4) (2012) 680–722.

[21] C. Peikert, V. Vaikuntanathan, B. Waters, A framework for efficient and coposable oblivious transfer, in: D. Wagner (Ed.), Advances in Cryptology - Crypto'08, Vol. 5157 of Lecture Notes in Computer Science, Springer - Verlag, Santa Barbara, USA, 2008, pp. 554 – 571.

[22] R. Canetti, M. Fischlin, Universally composable commitments, in: J. Kilian (Ed.), Advances in Cryptology - Crypto'01, Vol. 2139 of Lecture Notes in Computer Science, Springer - Verlag, Santa Barbara, USA, 2001, pp. 19 – 40.

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

[24] J. Katz, Universally composable multi-party computation using tamper-proof hardware, in: M. Naor (Ed.), Advances in Cryptology - Eurocrypt'07, Vol. 4515 of Lecture Notes in Computer Science, Springer - Verlag, Barcelona, Spain, 2007, pp. 115 – 128.

[25] S. Halevi, Y. Tauman Kalai, Smooth projective hashing and two-message oblivious transfer, Journal of Cryptology 25 (1) (2012) 158–193.

[26] Y. Tauman Kalai, Smooth projective hashing and two-message oblivious transfer, in: R. Cramer (Ed.), Advances in Cryptology - Eurocrypt'05, Vol. 3494 of Lecture Notes in Computer Science, Springer - Verlag, Aarhus, Denmark, 2005, pp. 78 – 95.

[27] R. Cramer, V. Shoup, Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption, in: L. R. Knudsen (Ed.), Advances in Cryptology - Eurocrypt'02, Vol. 2332 of Lecture Notes in Computer Science, Springer - Verlag, Amsterdam, The Netherlands, 2002, pp. 45 – 64.

[28] A. Y. Lindell, Efficient fully-simulatable oblivious transfer, in: T. Malkin (Ed.), Topics in Cryptology - CT-RSA 2008, Vol. 4964 of Lecture Notes in Computer Science, Springer - Verlag, San Francisco, USA, 2008, pp. 52 – 70.
[29] Y. Lindell, B. Pinkas, An efficient protocol for secure two-party computation in the presence of malicious adversaries, in: M. Naor (Ed.), Advances in Cryptology - Eurocrypt'07, Vol. 4515 of Lecture Notes in Computer Science, Springer - Verlag, Barcelona, Spain, 2007, pp. 52 – 78.
[30] O. Goldreich, A. Kahan, How to construct constant-round zero-knowledge proof systems for $\mathcal{NP}$, Journal of Cryptology 9 (3) (1996) 167 – 189.
[31] C. Gentry, C. Peikert, V. Vaikuntanathan, Trapdoors for hard lattices and new cryptographic constructions, in: C. Dwork (Ed.), 40th Annual ACM Symposium on Theory of Computing (STOC'08), ACM Press, Victoria, Canada, 2008, pp. 197 – 206.
[32] B. Zeng, C. Tartary, P. Xu, J. Jing, X. Tang, A practical framework for t-out-of-n oblivious transfer with security against covert adversaries, IEEE Transactions on Information Forensics and Security 7 (2) (2012) 465–479.
[33] Y. Aumann, Y. Lindell, Security against covert adversaries: Efficient protocols for realistic adversaries, Journal of Cryptology 23 (2) (2010) 281–343.
[34] B. Zeng, X. Tang, P. Xu, J. Jing, Practical frameworks for h-out-of-n oblivious transfer with security against covert and malicious adversaries, Cryptology ePrint Archive, Report 2011/001, `http://eprint.iacr.org/` (2011).
[35] O. Goldreich, Foundations of Cryptography: Volume I - Basic Tools, Cambridge University Press, 2001.
[36] R. Canetti, Security and composition of multiparty cryptographic protocols, Journal of Cryptology 13 (1) (2000) 143 – 202.
[37] R. Canetti, I. Damgård, S. Dziembowski, Y. Ishai, T. Malkin, Adaptive versus non-adaptive security of multi-party protocols, Journal of Cryptology 17 (3) (2004) 153 – 207.
[38] B. Barak, Y. Lindell, Strict polynomial-time in simulation and extraction, SIAM Journal on Computing 33 (4) (2004) 783 – 818.
[39] O. Goldreich, On expected probabilistic polynomial-time adversaries: A suggestion for restricted definitions and their benefits, Journal of Cryptology 23 (1) (2010) 1 – 36.
[40] J. P. Degabriele, K. Paterson, G. Watson, Provable security in the real world, Security & Privacy, IEEE 9 (3) (2011) 33–41.
[41] A. Blum, Lecture notes on randomized algorithms, `www.cs.cmu.edu/~avrim/Randalgs11/lectures/lect0124.pdf` (January 2011).
[42] M. Luby, Pseudorandomness and Cryptographic Applications, Princeton University Press, 1996.
[43] O. Regev, On lattices, learning with errors, random linear codes, and cryptography, Journal of the ACM (JACM) 56 (6).
[44] D. J. Bernstein, J. Buchmann, E. Dahmen (Eds.), Post-Quantum Cryptography, Springer, 2009.
[45] S. Goldwasser, S. Micali, Probabilistic encrytion, Journal of Computer and System Sciences 28 (2) (1984) 270 – 299.
[46] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in: J. Stern (Ed.), Advances in Cryptology - Eurocrypt'99, Vol. 1592 of Lecture Notes in Computer Science, Springer - Verlag, Prague, Czech Republic, 1999, pp. 223 – 238.