A Practical Framework for \(t\)-out-of-\(n\) Oblivious Transfer with Security against Covert Adversaries

Bing Zeng\(^1\), Christophe Tartary\(^2\), Peng Xu\(^1\), Jiandu Jing\(^1\) and Xueming Tang\(^1\)

\(^1\)School of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan, 430074
People’s Republic of China

\(^2\)Institute for Interdisciplinary Information Sciences
Institute for Theoretical Computer Science
Tsinghua University
Beijing, 100084
People’s Republic of China

\{zeng.bing.zb,tang.xueming.txm\}@gmail.com
c tartary@mail.tsinghua.edu.cn
xupeng@mail.hust.edu.cn
jingjd@smail.hust.edu.cn

Abstract

Oblivious transfer plays a fundamental role in the area of secure distributed computation. In particular, this primitive is used to search items in decentralized databases. Using a variant of smooth projective hash previously presented by Zeng \textit{et al.}, we construct a practical framework for \(t\)-out-of-\(n\) oblivious transfer in the plain model without any set-up assumption. It can be implemented under a variety of standard intractability assumptions, including the decisional Diffie-Hellman assumption, the decisional \(N\)-th residuosity assumption, the decisional quadratic residuosity assumption, and the learning with error problem. It is computationally secure in the presence of covert adversaries and only requires four rounds of communication. Compared to existing practical protocols with fully-simulatable security against covert adversaries or malicious adversaries, our framework is generally more efficient.

Keywords: Oblivious transfer, secure multiparty computation, covert adversaries, smooth projective hashing, computational security.

1 Introduction

An oblivious transfer (OT) is a two-party computation protocol between a sender and a receiver where the sender transfers some information to the receiver while remaining oblivious as to which information the receiver obtains [van05]. This cryptographic primitive was introduced by Rabin [Rab81] and plays an important role in database search [NP05]. However, the main strength of OT is the completeness result by Kilian stating that OT could be used to construct a secure multiparty computation (MPC) protocol for any computable function [Kil88]. Given the broad scope of applications of MPC (electronic auctions, anonymous transactions, privacy preserving data mining, electronic cash), the study of OT protocols has attracted a lot of attention from the cryptologic community.

In MPC, two categories of adversaries are usually considered: semi-honest (also called honest-but-curious) and malicious. In both cases, the objective of the adversary 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 [CCD+05]. 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.

In the specific case of OT, it is known how to convert a fully-simulatable protocol \Pi for OT secure against a semi-honest adversary into a fully-simulatable protocol \Pi' for OT secure against a malicious adversary thanks to enhanced trapdoor permutations [Gol04]. Even if this result reduces the problem of OT to the semi-honest case, this solution is based on the use of sub-functionalities for coin tossing and commitment schemes whose security relies on zero-knowledge (ZK) proof systems for NP-statements. Therefore, this approach is rather impractical and the traditional way of tackling the OT problem is still to directly design protocols secure against a malicious adversary. As said above, a malicious adversary is an entity who can arbitrarily deviate from the prescribed protocol steps. However, in many real-world settings, cheating participants do not want to be caught. As a consequence, their adversarial behavior is not arbitrary since they will never use a strategy revealing their deviation. Based on this observation, a new notion called covert security was recently proposed [AL07] (with full version as [AL10]). The objective is to relax the security level by discarding some strategies never used by the adversary in order to obtain computationally efficient protocols. As said by Aumann and Lindell, security against covert adversaries (SACA) does no rule out successful cheating (e.g. biasing the result of computation, recovering some private input). However, it ensures that if cheating occurs, then it will be detected by the honest parties with some desirable probability.

SACA was defined as a fully-simulatable security notion. It exhibits the same reduction to OT as in [Kil88] for the two-party case. Namely, any two-party computation protocol with SACA can be based on OT with SACA [AL10].

As exposed in [AL10], SACA is related to a value \( \epsilon \) called the deterrence factor representing the probability that the honest parties will catch an attempt to cheat by an adversary (in particular, when \( \epsilon = 1 \), a cheating party is always caught). Aumann and Lindell showed that SACA was strictly weaker than fully-simulatable security against malicious adversaries (SAMA) when \( 1/\text{Poly}(k) \leq \epsilon \leq 1 - 1/\text{Poly}(k) \) where Poly(\( \cdot \)) is a positive polynomial. However, when \( \epsilon = 1 - \mu(k) \) (where \( \mu(\cdot) \) is a negligible function), then both notions are identical. Furthermore, it is also showed in [AL10] that SACA implied security against (augmented) semi-honest adversaries (SASHA) for any \( \epsilon > 1/\text{Poly}(k) \). In addition, [DGN10] describes a conversion algorithm allowing to transform a MPC protocol with SASHA into another protocol with SACA where \( \epsilon = 1/4 \). However, this conversion technique requires honest majority and thus it cannot be used in the case of two-party computation. Based on these different properties, one can say that SACA is an intermediate security level between SASHA and SAMA.

In this paper, we are concerned with \emph{t-out-of-n oblivious transfer} (OT\(^t\)) protocols dealing with the following scenario. A database/sender holds \( n \) private values \( m_1, m_2, \ldots, m_n \) and a client/receiver holds \( t \) private indices \( i_1, i_2, \ldots, i_t \). The objective of the receiver is to recover the \( t \) database values corresponding to his index set (i.e. \( m_{i_1}, m_{i_2}, \ldots, m_{i_t} \)) without leaking any information about any of those indexes. In parallel, the sender do not want the receiver to learn anything but the \( t \) values it queried about.

Using the variant of smooth projective hashing (SPH) introduced in [ZTH10], we construct a framework holding SACA for OT\(^t\) with computational security (i.e. the computational power of both participants is assumed to be bounded).

Roughly speaking, this variant of SPH deals with two types of instances (smooth and projective) which are computationally indistinguishable due to a property called hard subset membership. For each instance of each type, it holds witnesses and two types of keys (hash keys and projection keys). A hash value is computed from a hash key and a projection value which is computed from a witness and a projection key. For a smooth instance, the projection value reveals almost no knowledge of 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 a property called projection.

We now present a high level description of our method.

1. Let \( K \) and \( g \) be two predetermined positive integers such that \( g < K \). The receiver generates a hash family parameter and \( K \) 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 \( g \) instance vectors at random to check their legalities (i.e., each one indeed contains at least \( n - t \) smooth instances).

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.

\footnote{Contrary to what one expects the traditional notion of \emph{semi-honest adversary} is not a particular case of SAMA. Indeed, in the light of the ideal/reald model paradigm, simulators for malicious adversaries have more power than simulators for semi-honest enemies. Thus, it is possible to simulate a protocol in the malicious model but not in the semi-honest one. This led to the notion augmented semi-honest adversary first introduced by Goldreich [Gol04]. The difference between augmented and non-augmented semi-honest adversaries is subtle and we refer the reader to for a detailed discussion and some examples on this phenomenon in Section 2.3.3 of [HI10].}
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.

The receiver’s security is guaranteed by the hard subset membership property of the hash family as learning the receiver’s private indices will imply distinguishing smooth instances from projective ones. The sender’s security is ensured by a cut-and-choose technique (in our framework, the receiver “cuts” $K$ instance vectors and the sender “chooses” $g$ ones to check theirs legalities) and the smoothness property of the hash family. Indeed, to learn more than $t$ values, a malicious receiver has to generate $K - g$ illegal instance vectors and those vectors must not have been chosen for the checks.

Our framework is practical. It has the following features.

- It is realizable under generic mathematical assumptions. In [ZTH10], it is shown that the variant of smooth projective hashing used in our construction could be realized under hardness problems such as the decisional Diffie-Hellman (DDH) assumption, the decisional $N$-th residuosity (DNR) assumption, the decisional quadratic residuosity (DQR) assumption and learning with errors (LWE).

- It is usable in practice as it works in the plain model where there is no trusted set-up such as a trusted common reference string (CRS) or a tamper-proof hardware. It only needs an authenticated channel between both participants.

- The deterrence factor to a covert sender always is 1 (i.e., a cheating sender is always caught by the receiver) and the deterrence factor to a covert receiver is $1 - 1/(K - g)$.

- It is computationally efficient. It needs only 4 communication rounds. The main computational overhead is $(K - g) n$ encryptions at the sender and $(K - g) t$ decryptations at the receiver. When compared to existing practical protocols with SAMA or SACA, our framework is generally more efficient (see Section 5.4).

This paper is organized as follows. In Section 2, we present the mathematical and cryptologic background necessary to understand our work. In Section 3, we expose our framework for OT ($^*$) with SACA. Its security is demonstrated in Section 4 while, in Section 5, we analyze its efficiency and provide a comparison to related practical OT protocols. Finally, the last section concludes this work with some open problems.

2 Preliminaries

2.1 Basic Notations and Definitions

We set the following notations for this paper:

- $\mathbb{N}$: set of natural numbers.
- $[i]$: the set $\{1, 2, \ldots, i\}$ where $i$ is any positive natural number.
- $S_n$: the set of all permutations of $[n]$ where $n$ is a natural number.
- $\vec{x}(j)$: the $j$-th entry of the vector $\vec{x}$.
- $\sigma(\vec{x})$: 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: $\forall i \in [n] \; \vec{y}(\sigma(i)) \leftarrow \vec{x}(i)$.
- Poly(.): a positive polynomial.
- $\alpha \in_U D$: an element $\alpha$ obtained from the set $D$ via uniform sampling.
- $\{0, 1\}^*$: set of all bitstrings.

**Definition 1 ([AL10])** A function $\mu(\cdot)$ is called negligible in $k$, if and only if:

$$\exists k_0 \in \mathbb{N} : \forall k > k_0 \; \forall \text{Poly}(\cdot) \; \mu(k) < \frac{1}{\text{Poly}(k)}.$$ 

**Definition 2 ([AL10])** A probability ensemble

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

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 $X(1^k, a)$. 

3
Definition 3 ([AL10]) Let \( X, Y \) be two probability ensembles. We say they are computationally indistinguishable, denoted by \( X \cong Y \), if for any non-uniform probabilistic polynomial-time (PPT) algorithm \( D \), there exists a negligible function \( \mu(\cdot) \) such that for any sufficiently large \( k \) and any \( a \in \{0,1\}^* \), it holds that

\[
|\Pr(D(1^k, a, X(1^k, a)) = 1) - \Pr(D(1^k, a, Y(1^k, a)) = 1)| \leq \mu(k).
\]

Definition 4 Let \( X, Y \) be two probability ensembles. They are said to be asymptotically equal, denoted by \( X \equiv Y \), if for any sufficiently large \( k \) and any \( a \), the distributions of \( X(1^k, a) \) and \( Y(1^k, a) \) are identical.

Remark 1 Let \( X, Y \) be two probability ensembles. We obviously have: \( X \equiv Y \) implies \( X \cong Y \).

In this paper, the security and complexity of our construction will be measured as functions of a value called security parameter and denoted by a natural number \( k \). All parties are assumed to run in time polynomial in \( k \).

2.2 OT with SACA

In this section, we describe the security definition for OT introduced in [AL10]. In that paper, Aumann and Lindell proposed three security levels for SACA: failed-simulation, explicit-cheat, and strong explicit-cheat. The strongest formulation being strong explicit-cheat, this will be the one adopted in our paper when referring to SACA.

The \( \text{OT}^0 \) functionality is defined as follows.

\[
f : \mathbb{N} \times \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*
\]

\[
(1^k, \vec{m}, T) \mapsto (\lambda, (\vec{m}(i))_{i \in T})
\]

where:

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

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

SACA is a fully-simulatable security notion. Before recalling the strong-explicit cheat formulation of SACA introduced by Aumann and Lindell, we first state how the ideal and real worlds are structured.

Before starting the OT protocol, the covert adversary \( A \) corrupts the parties listed in the set \( I \subseteq \{S, R\} \). In our case, we consider that at most one party is corrupted by \( A \). Non-corrupted participants (i.e. honest parties) will faithfully follow the protocol’s instructions. When \( A \) corrupts a party, \( A \) takes control over the party’s actions and \( A \) gets aware of the party’s communication and computation history. In particular, the input of the corrupt participant is known (and controlled) by \( A \). The objective of \( A \) 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 \( A \)’s protocol input.

We are now to recall how the ideal/real world paradigm works. This specification (strong explicit-cheat formulation of SACA) is taken from [AL10] and adapted to the case of OT.

2.2.1 The Ideal World

There is an incorruptible trusted third party (TTP) named ideal functionality as said in Section 1. An execution of \( \text{OT}^0 \) with deterrence factor \( \varepsilon \in (0,1) \) proceeds as follows.

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

Before proceeding to the next stage, \( A \) corrupts parties listed in \( I \) and learns their inputs. For simplicity, when referring to the two parties, we identify \( S \) to 1 and \( R \) to 2.
• **Sending inputs to the TTP:** Each honest party sends its received input to the TTP. For each corrupted party \( i \in I \), \( \mathcal{A} \) sends a string to the TTP on \( i \)'s behalf. This string may either be \( i \)'s received input or another input of the same length. This choice is made by \( \mathcal{A} \) based on the corrupted party's input and the auxiliary information \( z \).

• **Aborting options:** If \( \mathcal{A} \) sends \( \text{Abort}_i \) (respectively, \( \text{Corrupted}_i \)) for some \( i \in I \) to the TTP, then the TTP sends \( \text{Abort}_i \) (respectively, \( \text{Corrupted}_i \)) to the honest parties and halts. If both \( \text{Corrupted}_i \) and \( \text{Abort}_i \) messages are sent, then the TTP ignores the \( \text{Corrupted}_i \) message. Note that, in our two-party computation case, if the TTP gets several \( \text{Corrupted} \) and/or \( \text{Abort} \) messages, then both parties \( S \) and \( R \) are controlled by \( \mathcal{A} \). This case has no security interest whatsoever.

• **Attempted cheat option:** If \( \mathcal{A} \) sends \( \text{Cheat}_i \) for some \( i \in I \) to the TTP, then the TTP works as follows:
  1. With probability \( \epsilon \), the TTP sends \( \text{Corrupted}_i \) to \( \mathcal{A} \) and the honest party.
  2. With probability \( 1 - \epsilon \), the TTP sends Undetected to \( \mathcal{A} \) along with the honest party's input. Following this, \( \mathcal{A} \) sends the TTP output value \( \{c_j\}_{j \notin I} \) of its choice for the honest party. Then, the TTP sends to the honest party his input.

The ideal execution usually terminates at this point. However, if \( \mathcal{A} \) did not send any \( \text{Abort}_i \), \( \text{Corrupted}_i \), or \( \text{Cheat}_i \) messages, the ideal execution continues below.

• **TTP answering the adversary:** Denote by \( (y_1, y_2) \) the inputs received by the TTP. The TTP computes \( f(y_1, y_2) \) (i.e. the TTP faithfully executes \( \text{OT}_t^\mathcal{A} \)) and sends \( \mathcal{A} \) the output of the corrupted parties. That is to say that the TTP sends \( (f(y_1, y_2)(i))_{i \in I} \) to \( \mathcal{A} \).

• **TTP answering the honest party:** After receiving its output, \( \mathcal{A} \) sends either \( \text{Abort}_i \) for some \( i \in I \) or \( \text{Continue} \) to the TTP. If the TTP receives \( \text{Continue} \), then it forward to the honest party his results. Otherwise, if the TTP gets \( \text{Abort}_i \) from \( \mathcal{A} \), then the TTP sends the honest party \( \text{Abort}_i \).

• **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 \( z \), and the messages obtained from the TTP.

The output of the execution is defined/denoted by a vector \( \text{Ideal}_{f,\mathcal{A}(z)}(1^k, \bar{m}, T) \).

### 2.2.2 The Real World

We do not have any TTP and the two parties communicate with each other using an authenticated channel. Let \( \Pi \) be a protocol for \( \text{OT}_t^\mathcal{A} \). An execution of such a protocol works as follows.

• **Initial Inputs:** They are identical to the input in the ideal world.

• **Computation:** Computing \( f \) is done through interactions between both parties. Each honest party strictly follows the prescribed protocol \( \Pi \). The corrupted parties have to follow \( \mathcal{A} \)'s instructions and may arbitrarily deviate from \( \Pi \).

• **Outputs:** Each honest party always outputs what \( \Pi \) 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 \( \Pi \).

The output of the execution is defined/denoted by a vector \( \text{Real}_{\Pi, f, \mathcal{A}(z)}(1^k, \bar{m}, T) \).

Intuitively speaking, the idea behind the ideal/real world paradigm is to say that protocol \( \Pi \) securely computes \( \text{OT}_t^\mathcal{A} \) in the presence of covert adversaries, if and only if, for any covert adversary \( \mathcal{A} \), any actions performed by \( \mathcal{A} \) in the real world can also be done in the ideal world. This intuition is formally captured by the following definition.

**Definition 5** Let \( f \) denote the \( \text{OT}_t^\mathcal{A} \) functionality and let \( \Pi \) be a concrete protocol for computing \( f \). Let \( \Psi \) denote the set of the receiver’s all possible legal private inputs (i.e. any \( t \)-subset of \([n]\)). Let \( \epsilon \in (0, 1] \). We say that \( \Pi \) securely computes \( f \) in the presence of covert adversaries with deterrence factor \( \epsilon \), if and only if for any non-uniform PPT adversary \( \mathcal{A} \) with an auxiliary input \( z \) in the real world, there exists a non-uniform PPT adversary \( \mathcal{G} \) with the same auxiliary input in the ideal world such that, for any \( I \subseteq [2] \), the following equation holds:

\[
\{\text{Real}_{\Pi, f, \mathcal{A}(z)}(1^k, \bar{m}, T)\}_{k \in \mathbb{N}, \bar{m} \in \{0,1\}^n, T \in \Psi} \approx \approx \{\text{Ideal}_{f, \mathcal{A}(z)}(1^k, \bar{m}, T)\}_{k \in \mathbb{N}, \bar{m} \in \{0,1\}^n, T \in \Psi, z \in \{0,1\}^*} \quad (1)
\]

where the parameters input to the two probability ensembles are the same. The adversary \( \mathcal{G} \) in the ideal world is called the simulator of \( \mathcal{A} \).
2.3 A Variant of Smooth Projective Hashing

In this section we present the notion of SPH which is the tool that we will use to construct a framework for $\text{OT}^\pi$. SPH was introduced by Cramer and Shoup [CS02] in order to construct efficient public-key encryption scheme secure against adaptive chosen ciphertext attacks. In our work, we use a variant of this tool proposed in [ZTH10].

As an intuitive description of the variant is given in Section 1, we proceed formally here. A hash family $\mathcal{H}$ is defined by means of the following eight PPT algorithms $\mathcal{H} = (\text{PG}, \text{IS}, \text{Check}, \text{DI}, \text{KG}, \text{Hash}, \text{pHash}, \text{Cheat})$:

- Parameter generator $\text{PG}$: it takes a security parameter $k$ as input and returns a hash parameter $\Lambda$: i.e. $\Lambda \leftarrow \text{PG}(1^k)$.
- Checker Check: it takes a security parameter $k$ and a hash parameter $\Lambda$ as input and returns an indicator bit $b \in \{0, 1\}$: i.e. $b \leftarrow \text{Check}(1^k, \Lambda)$. The objective is to check that $\Lambda$ was correctly generated.
- Instance sampler $\text{IS}$: it takes a security parameter $k$ and a hash parameter $\Lambda$ as input and returns a vector $\vec{a} = \{(\tilde{x}_1, \tilde{w}_1), \ldots, (\tilde{x}_t, \tilde{w}_t), (\tilde{x}_{t+1}, \tilde{w}_{t+1}), \ldots, (\tilde{x}_n, \tilde{w}_n)\}$ (i.e., $\vec{a} \leftarrow \text{IS}(1^k, \Lambda)$) 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 $\text{DI}$: it takes a security parameter $k$, a hash parameter $\Lambda$ and an instance-witness pair $(x, w)$ as input and outputs an indicator value $b$: i.e., $b \leftarrow \text{DI}(1^k, \Lambda, x, w)$. Its goal is to distinguish smooth instances and projective instances.
- Key generator $\text{KG}$: it takes a security parameter $k$, a hash parameter $\Lambda$ and an instance $x$ as input and outputs a hash-projection key pair $(hk, pk)$: i.e., $(hk, pk) \leftarrow \text{KG}(1^k, \Lambda, x)$.
- Hash algorithm $\text{Hash}$: it takes a security parameter $k$, a hash parameter $\Lambda$, an instance $x$ and a hash key $hk$ as input and outputs a value $y$: i.e., $y \leftarrow \text{Hash}(1^k, \Lambda, x, hk)$.
- Projection algorithm $\text{pHash}$: it takes a security parameter $k$, a hash parameter $\Lambda$, an instance $x$, a projection key $pk$ and a witness $w$ of $x$ as input and outputs a value $y$: i.e., $y \leftarrow \text{pHash}(1^k, \Lambda, x, pk, w)$.
- Cheater Cheat: it takes a security parameter $k$, a hash parameter $\Lambda$ as input and outputs $n$ instance-witness pairs: i.e., $((x_1, w_1), \ldots, (x_n, w_n)) \leftarrow \text{Cheat}(1^k, \Lambda)$. The intent is to generate $n$ projective instance-witness pairs.

**Definition 6** For a given hash parameter $\Lambda$, if Check outputs $1$, then $\Lambda$ is said to be legal; otherwise, it is said to be illegal.

**Remark 2** Naturally, any $\Lambda$ generated by $\text{PG}$ is legal.

**Definition 7** Let $R = \{(x, w) : x, w \in \{0, 1\}^*\}$ be a relation. For a legal $\Lambda$, we define its projective relation as $\bar{R}_\Lambda = \{(\tilde{x}, \tilde{w}) : (\tilde{x}, \tilde{w})$ is generated by $\text{IS}(1^k, \Lambda)$} and its smooth relation as $\check{R}_\Lambda = \{(\check{x}, \check{w}) : (\check{x}, \check{w})$ is generated by $\text{IS}(1^k, \Lambda)\}$.

**Definition 8** (Distinguishability [ZTH10]) For any legal hash parameter $\Lambda$, any instance-witness pair $(x, w)$, we require:

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

**Definition 9** If $R$ is a relation, then its language is defined as $L = \{x \in \{0, 1\}^* : \exists w((x, w) \in R)\}$.

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

**Definition 10** (Projection [ZTH10]) For any sufficiently large security parameter $k$, any hash parameter $\Lambda$ generated by $\text{PG}(1^k)$, any projective instance-witness pair $(\check{x}, \check{w})$ generated by $\text{IS}(1^k, \Lambda)$, and any hash-projection key pair $(hk, pk)$ generated by $\text{KG}(1^k, \Lambda, \check{x})$, it holds that

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

**Definition 11** For an instance-witness vector $\vec{a} = \{(x_1, w_1), \ldots, (x_n, w_n)\}$, we define its instance vector as $\vec{x} \overset{\text{def}}{=} (x_1, \ldots, x_n)$ and its witness vector as $\vec{w} \overset{\text{def}}{=} (w_1, \ldots, w_n)$.

**Definition 12** Fix a legal hash parameter $\Lambda$. If a vector $\vec{a}$ contain $n-t$ smooth instance-witness pairs, (i.e., $n-t$ pairs in $\bar{R}_\Lambda$), then $\vec{a}$ is said to be legal.
Remark 3 Note that any $\vec{a}$ generated by 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 $n$ times.

Definition 13 (Smoothness [ZTH10]) For any legal hash parameter $\Lambda$, 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 $S_1 \overset{def}{=} \{S_1(1^k)\}_{k \in \mathbb{N}}$ and $S_2 \overset{def}{=} \{S_2(1^k)\}_{k \in \mathbb{N}}$, specified as follows, are computationally indistinguishable (i.e., $S_1 \equiv S_2$) where:

- $G_\Lambda \overset{def}{=} \{y : x \in \hat{L}_\Lambda \cup \bar{L}_\Lambda, (hk, pk) \leftarrow KG(1^k, \Lambda, x), y \leftarrow Hash(1^k, \Lambda, x, hk)\}$ is a set of all possible hash values.
- Algorithm SmGen$_1(1^k)$ works as follows:
  - $\vec{x} \leftarrow \vec{x}^2$.
  - For each $j \in [n]$, perform: $(hk_j, pk_j) \leftarrow KG(1^k, \Lambda, \vec{x}(j)), y_j \leftarrow Hash(1^k, \Lambda, \vec{x}(j), hk_j$).
  - Set $\vec{p} \leftarrow (pk_j, y_j)$ and output $\vec{p} y$.
- Algorithm SmGen$_2(1^k)$ works as 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 Sm$_i(1^k)$ works as follows: $\vec{p} y \leftarrow SmGen_i(1^k)$, $\vec{p} y \leftarrow \sigma(\vec{p} y)$ and output $\vec{p} y$.

Definition 14 (Hard Subset Membership [ZTH10]) For any $\sigma \in S_n$, the two probability ensembles $HSM_1 \overset{def}{=} \{HSM_1(1^k)\}_{k \in \mathbb{N}}$ and $HSM_2 \overset{def}{=} \{HSM_2(1^k)\}_{k \in \mathbb{N}}$ specified as follows, are computationally indistinguishable (i.e., $HSM_1 \equiv HSM_2$) where:

- Algorithm HSM$_1(1^k)$ works as follows: $\Lambda \leftarrow$ PG$(1^k)$, $\vec{a} \leftarrow IS(1^k, \Lambda)$ and outputs $(\Lambda, \vec{x}^2)$.
- Algorithm HSM$_2(1^k)$ operates as HSM$_1(1^k)$ except that it outputs $(\Lambda, \sigma(\vec{x}^2))$.

Definition 15 (Feasible Cheating [ZTH10]) For any $\sigma, \sigma' \in S_n$, the two probability ensembles $HSM_2$ and $HSM_3$ with $HSM_3 \overset{def}{=} \{HSM_3(1^k)\}_{k \in \mathbb{N}}$ are computationally indistinguishable (i.e., $HSM_2 \equiv HSM_3$) where $HSM_2$ is defined as above and $HSM_3$ works as follows:

- $HSM_3(1^k) : \Lambda \leftarrow$ PG$(1^k)$, $\vec{a} \leftarrow$ Cheat$(1^k, \Lambda)$ and outputs $(\Lambda, \sigma(\vec{x}^2))$.

Remark 4 It is clear that feasible cheating property provides a way to generate an illegal instance-witness vector whose all entries are projective.

In [ZTH10], it is shown that a smooth projective hash family with properties of distinguishability, hard subset membership and feasible cheating can be instantiated under various intractability assumptions (e.g., the DDH assumption, the DNR assumption, the DQR assumption, the LWE problem).

As an example, we restate the DDH-based instantiation from [ZTH10]. We still consider the problem OT$^1$. Let $G$ be a cyclic group with sufficiently large prime order $q$. Under the DDH assumption, it is hard to distinguish tuples of the form $(g_1, g_2, g_1^r, g_2^s)$ from tuples of the form $(g_1, g_2, g_1^r, g_2^s)$ where $g_1$ and $g_2$ are randomly chosen from $G$, and $r$ and $s$ are randomly chosen from $\mathbb{Z}_q$.

- PG$(1^k)$: it generates the cyclic group description: i.e., $(g_1, g_2, q)$, where $g_1, g_2$ are two distinct generators with the same prime order $q$. In this case, we have: $\Lambda = (g_1, g_2, q)$.
- Check$(1^k, \Lambda)$: $(g_1, g_2, q) \leftarrow \Lambda$. If $q$ is a prime number of appropriate size, and $g_1, g_2$ are two distinct generators, then outputs 1: otherwise, outputs 0.
- IS$(1^k, \Lambda)$: $(g_1, g_2, q) \leftarrow \Lambda$. For each $i \in [t]$, do: $r_i \in U \mathbb{Z}_q$, $\tilde{x}_i \leftarrow (g_1^{r_i}, g_2^{r_i})$, $\tilde{w}_i \leftarrow r_i$. For each $i \not\in [t]$, do: $r_i, s_i \in U \mathbb{Z}_q$, $\tilde{x}_i \leftarrow (g_1^{r_i}, g_2^{s_i})$, $\tilde{w}_i \leftarrow r_i$. Finally, output $((\tilde{x}_1, \tilde{w}_1), \ldots, (\tilde{x}_t, \tilde{w}_1), (\tilde{x}_{t+1}, \tilde{w}_{t+1}), \ldots, (\tilde{x}_n, \tilde{w}_n))$.
- DI$(1^k, \Lambda, x, w)$: on input $(g_1, g_2, q) \leftarrow \Lambda$, do: $(\alpha, \beta) \leftarrow x, r \leftarrow w$. If $(\alpha, \beta) = (g_1^r, g_2^s)$, then output 0; if $\alpha = g_1^r$ and $\beta \neq g_2^s$, then output 1; otherwise return 2.
- KG$(1^k, \Lambda, x, h)$: on input $(g_1, g_2, q) \leftarrow \Lambda$, do: $u, v \in U \mathbb{Z}_q$, $hk \leftarrow (u, v)$, $pk \leftarrow g_1^u g_2^v$ and output $(hk, pk)$.
- Hash$(1^k, \Lambda, x, hk)$: do $(\alpha, \beta) \leftarrow (u, v)$, $hk \leftarrow h \alpha^u \beta^v \leftarrow (h, pk)$, then output $y$.
- pHash$(1^k, \Lambda, x, pk, w)$: do $r \leftarrow w$, $y \leftarrow pk^r$ and output $y$. 7
Note that Tauman Kalai also used SPH to construct a framework for OT\(^{\pi}\) (or OT\(^{\gamma}\)) holding half-simulatable SAMA \cite{tau05}. The most essential difference between the two SPH variants is that Tauman Kalai’s provides witnesses only to projective instances while ours provides witnesses to both projective and smooth instances. The modification made by the latter version solves the problem of how to gain the simulation-based security in the case that only the receiver is corrupted and it solves the question of how to deal with general OT\(^{\gamma}\). For more discussion, we refer the reader to Section 3.2 of \cite{zth10}.

3 A Framework for OT\(^{\pi}\) with SACA

For our construction, we will need a PPT algorithm \(\Gamma\) 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. We give an implementation example of \(\Gamma\) in \text{Algorithm 1}.

\textbf{Algorithm 1 Permutation Generator} \(\Gamma\)

\textbf{Input:} \(T_1, T_2\): two legal inputs from the receiver.

1. Initialize \(E \leftarrow \emptyset, C_1 \leftarrow [n] \setminus T_1\) and \(C_2 \leftarrow [n] \setminus T_2\).

2. For each \(j \in T_2\), do the following:
   - Pick \(i \in U\{T_1\} \). Update \(T_1 \leftarrow T_1 \setminus \{i\}\) and \(E \leftarrow E \cup \{j = i\}\).

3. For each \(j \in C_2\), do the following:
   - Pick \(i \in U\{C_1\} \). Update \(C_1 \leftarrow C_1 \setminus \{i\}\) and \(E \leftarrow E \cup \{j = i\}\).

4. Define the \(n\)-set permutation \(\sigma\) such that \(\sigma(i) = j\) if and only if \(j = i \in E\). Return \(\sigma\).

\textbf{Output:} \(\sigma\): a random permutation of \([n]\) sending \(T_1\) to \(T_2\).

We are ready to present our framework II for OT\(^{\pi}\). For clarity, we make the following convention. If \(S\) refuses to send \(R\) a message which is supposed to be sent, or \(S\) sends an invalid message that \(R\) cannot process then \(R\) halts the protocol and outputs Abort\(_1\). Likewise, if \(R\) deviates in a similar way, \(S\) outputs Abort\(_2\). The inputs are described as follows:

- Private Inputs: The sender \(S\) holds a vector \(\vec{m} \in \{\{0,1\}\}^n\) containing \(n\) values and holds a random tape \(r_1 \in \{0,1\}^t\). The receiver \(R\) holds a set \(T \in \Psi\) containing \(t\) indices and holds a random tape \(r_2 \in \{0,1\}^t\). The adversary \(A\) holds a name list \(I \subseteq \{S, R\}\) and a random tape \(r_A \in \{0,1\}^t\).

- Auxiliary Inputs: The adversary \(A\) holds an auxiliary input \(\vec{z} \in \{0,1\}^t\). The security parameter \(t\), a description of a smooth projective hash family \(\mathcal{H}\) and the two positive integers \(K, g\) (with \(g < K\)) are known to \(R, S\) and \(A\).

II proceeds by interactions between \(R\) and \(S\) as described in \text{Protocol 1}.

\textbf{Remark 5} In \text{Algorithm 1}, Step 2 maps \(T_1\) to set \(T_2\) while Step 3 maps the two complement sets to one another.

\textbf{Remark 6} Note that \(S\) 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, \(R\) only needs to send the witnesses of the smooth instances. Formally speaking, \(R\) only needs to send \(((i, j, \tilde{w}_i(j)))_{i \in CS, j \in I}\) to \(S\), where \(J_i \overset{def}{=} \{j : \tilde{v}_i(j) \text{ is smooth}\}\).

Now let us check the correctness of the framework, i.e. the framework works when both \(S\) and \(R\) are honest. For each \(i \in CS\) and each \(j \in T\), we know
\[
\tilde{e}_i(j) = \tilde{m}_i(j) \oplus (\oplus_{i \in CS} \tilde{b}'_i(j)) \text{ and } m'_j = \vec{e}(j) \oplus (\oplus_{i \in CS} \vec{b}'_i(j)).
\]

Because of the projection of \(\mathcal{H}, \vec{b}'_i(j) = \vec{b}'_j\) holds. So, we have: \(\vec{m}(j) = m'_j\). This means that \(R\) gets \((\tilde{m}(i))_{i \in T}\) which is the data it wanted to receive.

4 Security Of The Framework II

Since the demonstration of the framework security is tedious, we first present an intuitive analysis for the reader to get a glimpse at the security proof milestones.
Protocol 1

**R1** (Receiver’s step): $R$ chooses a hash parameter and samples its instance-witness vectors as follows.

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

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

1. If $\text{Check}(1^k, \Lambda) = 0$, then $S$ halts and outputs $\text{Abort}_2$.
2. If $\text{Check}(1^k, \Lambda) = 1$, then $S$ uniformly chooses a random string $r$ containing $g$ 1-bits over $\{0,1\}^K$ and sends it to $R$.

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

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

1. $R$ verifies that the choice indicator $r$ has exactly $g$ 1-bits. If no, $R$ halts and outputs $\text{Abort}_1$; if yes, $R$ proceeds to the next instruction.
2. $R$ sends the witnesses of the chosen instance vectors $(\tilde{a}_i)_{i \in CS}$ to $S$.
3. $R$ shuffles his private input. For each non-chosen vector, $R$ 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 $\tilde{x}_i$ 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., $G_i = \{j : \tilde{x}_i(j) \text{ is projective}\}$). For each $i \in \overline{CS}$, $R$ chooses a permutation $\sigma^2_i \leftarrow \Gamma(G_i, T)$.
4. $R$ sends the permutations, $(\sigma^2_i)_{i \in \overline{CS}}$, to $S$.

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

1. $S$ verifies that each chosen instance vector is legal by checking if it contains $t$ projective instances and $n - t$ smooth instances. Knowing the witness vectors $(\tilde{a}_i)_{i \in CS}$, $R$ invokes algorithm $\text{DI}$ to check the validity of instance vectors $(\tilde{x}_i)_{i \in CS}$. If $R$ did not send the witness vectors at Step R2 or if the check fails, then $S$ halts and outputs $\text{Corrupted}_2$; otherwise $S$ proceeds to the next step.
2. $S$ reorders the non-chosen instance vectors following the way prescribed by $R$: $\forall i \in \overline{CS} \quad \tilde{x}_i \leftarrow \sigma^2_i(\tilde{x}_i)$.
3. $S$ encrypts the value vector $\tilde{m}$. That is, for each $i \in \overline{CS}$ and for each $j \in [n]$, $(hk_{ij}, pk_{ij}) \leftarrow KG(1^k, \Lambda, \tilde{x}_i(j))$, $\beta_{ij} \leftarrow \text{Hash}(1^k, \Lambda, \tilde{x}_i(j), hk_{ij})$, $\tilde{e}_i \leftarrow \tilde{m} \oplus (\oplus_{i \in \overline{CS}} \tilde{\beta}_i), pk_i^{\text{def}} = (pk_{i1}, pk_{i2}, \ldots, pk_{in})$.
4. $S$ sends the encryption of $\tilde{m}$ and the projection keys, $(\tilde{e}, (pk_i)_{i \in \overline{CS}})$, to $R$.

**R3** (Receiver’s step) $R$ decrypts $\tilde{c}$. That is, for each $i \in \overline{CS}$ and $j \in T$, the receiver computes $\beta'_{ij} \leftarrow \text{pHash}(1^k, \Lambda, \tilde{e}_i(j), pk_i(j), \tilde{w}_i(j)), m'_j \leftarrow \tilde{c}(j) \oplus (\oplus_{i \in \overline{CS}} \beta'_ij)$. Finally, $R$ recovers the values $(m'_j)_{j \in T}$.
4.1 An Intuitive Perspective

4.1.1 Sender’s Security

To protect the sender, the framework should prevent the receiver $R$ from obtaining more than $t$ values. The following theorem shows that this guarantee occurs with probability at least $1 - 1/(K-g)$.

**Theorem 1** In the case where $S$ is honest and $R$ is corrupted, the probability that $R$ obtains more than $t$ values is at most $1/(K-g)$.

**Proof.**

According to the framework, the following conditions are necessary for $R$ to obtain more than $t$ values.

1. $R$ has to generate at least one illegal instance vector containing more than $t$ projective instances. Otherwise, it can not correctly decrypt more than $t$ entries of $\tilde{c}$ due to the smoothness of the hash family $H$. Without loss of generality, we assume that the illegal instance vectors are $\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_d$.

2. Not a single illegal instance vector must be chosen and all the non-chosen instance vectors are just the illegal instance vectors: i.e., $\overline{CS} = \{l_1, l_2, \ldots, l_d\}$. We prove this as follows.

(a) Case 1: $\overline{CS} \subseteq \{l_1, l_2, \ldots, l_d\}$. This means that there exists an illegal instance vector chosen by $S$. Thus, $S$ detects $R$’s cheating and $R$ gains nothing.

(b) Case 2: $\overline{CS} \supset \{l_1, l_2, \ldots, l_d\}$. This means that there exists a legal instance vector non-chosen by $S$. Therefore, it is used to encrypt the $n$ values of $S$. Looking at Step S2, the final $n$ encryptions are obtained by XOR-ing the $n$ values and the hash values of all non-chosen instance vectors. Because a legal instance vector holds at least $n-t$ smooth instances, at least $n-t$ encryptions computationally hide their encrypted values. As a result, $R$ knows almost nothing about the at least $n-t$ encrypted values.

3. The number of illegal instance vectors is $K-g$, in other words: $d = K-g$. We prove this as follows. Since $S$ chooses $g$ instance vectors to check, the number of illegal instance vectors has to be no more than $K-g$ to avoid being caught. Following the analysis of the second necessary condition, we know that, to obtain extra values for $R$, the number of illegal instance vectors has to be no less than $K-g$ to make it possible that all non-chosen instance vectors are illegal.

Let us estimate the probability that the last two necessary conditions are met.

$$\text{Prob}(\overline{CS} = \{l_1, l_2, \ldots, l_d\} \land d = K-g) = 1/(\binom{K}{d}) = 1/(\binom{K}{K-g})$$

This means that the probability that $R$ cheats to obtain more than $t$ values is at most $1/(\binom{K}{K-g})$.

Intuitively speaking, Theorem 1 shows that the deterrence factor to $R$ is at least $1 - 1/(K-g)$. The formal security proof, given in Section 4.2.2 of the case where only $R$ is corrupted shows that this intuition is correct.

Due to the properties of binomial coefficients, the maximum lower-bound on the deterrence factor can be achieved by setting $g = K - \lceil \frac{K+1}{2} \rceil$ when $K$ is fixed. This contrasts with our intuition that the more instance vectors are chosen to be checked, the higher the deterrence factor is. The essential reason why our intuition is wrong is that it does not take into account the smoothness property of $H$.

Since $K$ and $g$ are predetermined constant values, the distance between the deterrence factor and 1 cannot be negligibly small. However, a very small distance is still achievable. For example, setting $K = 20$ and $g = 10$, we have a deterrence factor $\epsilon = 1 - 5.41 \times 10^{-6}$.

4.1.2 Receiver’s Security

To protect the receiver $R$, the framework must first prevent the sender $S$ from learning anything about which values $R$ gets (i.e., $R$’s private index set $T$). Intuitively, there might be a possible information leakage in Step R2 where $R$ shuffles $T$. Since $j \in T$ if and only if $\tilde{x}_i$’s $j$-th entry is projective for each $i \in \overline{US}$, learning some $j \in T$ means identifying a projective entry of some $\tilde{x}_i$. Given the fact that $n$ and $t$ are known to both players, $S$ can guess some $j \in T$ with probability $t/n$ and can identify a projective entry of some $\tilde{x}_i$ with probability $t/n$ as well. However, the hard subset membership property of $H$ guarantees that $\tilde{x}_i$’s projective instances and smooth instances are computationally indistinguishable. As a result, the probability of $S$ identifying a projective entry of $\tilde{x}_i$ is negligibly greater than $t/n$. Furthermore, the probability of $S$ learning some $j \in T$ is negligibly greater than $t/n$ too. In a word, $S$ 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.

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

Based on this informal analysis, it seems that $S$ had no effective way of cheating (i.e., the deterrence factor to $S$ seems to be 1). The formal security proof, given in Section 4.2.1 of the case where only $S$ is corrupted shows that this intuition is correct.

### 4.2 Formal Security Proof

We are now to expose the formal demonstration of the scheme security. Specifically, we prove the following theorem.

**Theorem 2** Assume that $\mathcal{H}$ is a smooth projective hash family with properties of distinguishability, hard subset membership and feasible cheating. Assume that $K$ and $g$ are two positive integers such that $g < K$. Then, framework II described in [Protocol 7] securely computes the OT$_n$ functionality in the presence of non-adaptive covert adversaries with computationally bounded power. The deterrence factor to the receiver is $1 - 1/(K^{g})$ and the deterrence factor to the sender is 1.

The remaining of this section is dedicated to the demonstration of the above result. For simplicity, we denote the sender, the receiver and the adversary in the real world by $S$, $R$, $A$ and we denote the corresponding entities in the ideal world by $S'$, $R'$, $\emptyset$.

There are four cases to be considered: only $S$ is corrupted; only $R$ is corrupted; both $S$ and $R$ are corrupted; both $S$ and $R$ are honest. Because the security proofs of the last two cases are trivial, we omit them.

#### 4.2.1 Case 1 – Only $S$ is Corrupted

In this case, $A$ takes the full control of $S$ in the real world. Correspondingly, $A'$s simulator $\emptyset$, takes the full control of $S'$ in the ideal world, where $\emptyset$ is constructed as depicted in Algorithm 2. Recall that $S'$ holds the same value vector $\vec{m}$ as $S$. In addition, to simplify the description of $\emptyset$, we assumed that during the interactions between $A$ and $\emptyset$ (i.e., from Step Sim2 onwards), if $A$ refuses to send some message it is supposed to send or $A$ sends an invalid message that $\emptyset$ cannot process, the $\emptyset$ sends Abort$_1$ to the TTP and halts with outputting whatever $A$ outputs.

**Lemma 1** The simulator $\emptyset$ runs in expected polynomial-time.

**Proof.**
First, let’s focus on Step Sim2. The number of legal choice indicators is $\binom{K}{g}$, thus $\text{Prob}(r = \vec{r}) = 1/(\binom{K}{g})$ and the expected number of repeats of $\Upsilon$ is $\binom{K}{g}$. Because $\binom{K}{g}$ is a predetermined constant value, Step Sim2 runs in expected polynomial-time.

Second, consider the other steps of $\emptyset$. Obviously, these steps run in strict polynomial-time which achieves to prove that $\emptyset$ runs in expected polynomial-time. \hfill $\square$

However, [Definition 5] requires a strict PPT simulator not an expected PPT one. Observe that, the reason why $\emptyset$ is not a strict PPT is that the probability $1/(\binom{K}{g})$ of $\Upsilon$ outputting Success is relatively low. Nonetheless, the following lemma guarantees that we can get a new version of $\Upsilon$ that running in strict polynomial-time and outputting $\text{Success}$ with probability close to 1.

**Lemma 2** Let $q \in (0, 1)$ be a constant value. Given a PPT machine $M$ which, on receiving input $x$, outputs $\text{Success}$ with probability $q$ and outputs $\text{Failure}$ with probability $1 - q$, we can construct a PPT machine $\tilde{M}$ such that, on the same input $x$, we have:

\[
\left\{ \begin{array}{l}
\text{Prob}(\tilde{M}(x) = \text{Failure}) = \mu(k), \\
\text{Prob}(\tilde{M}(x) = \text{Success}) = 1 - \mu(k).
\end{array} \right.
\]

where $\mu(\cdot)$ is a negligible function.
Algorithm 2 Simulator $\mathcal{G}$ when $\mathcal{S}$ is the only Corrupted Player

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

Sim1 (Initialization step):
1. $\mathcal{G}$ corrupts $\mathcal{S}'$ and learns $\mathcal{S}'$'s private input $\vec{m}$.
2. Let $\vec{A}$ be a copy of $\mathcal{A}$. $\mathcal{G}$ invokes $\vec{A}$ as a subroutine with initial inputs of $\vec{A}$ to be $\mathcal{G}$’s initial inputs with the exception that the randomness of $\vec{A}$ to be a uniformly distributed string. In other words, the initial inputs of $\vec{A}$ are set as $k, I, z$ and $r_{\vec{A}}$ with $r_{\vec{A}} \in U \{0, 1\}^*$.
3. $\mathcal{G}$ activates $\vec{A}$.
4. As in the real world, $\vec{A}$ sends a message to corrupt the sender. $\mathcal{G}$ plays the role of $\mathcal{S}$ and supplies $\vec{A}$ with $\vec{m}$.

Comment: To simulate the environment of $\vec{A}$ in the real world, $\mathcal{G}$ is to simultaneously disguises himself as $\mathcal{S}$ and $\mathcal{R}$ to interact with $\mathcal{A}$ during the following steps.

Sim2: $\mathcal{G}$ repeats the following procedure, denoted by $\Upsilon$, until the later outputs Success. At the beginning of a new repetition, all randomness used by $\mathcal{G}$ is refreshed. Upon finishing this repeating process, $\mathcal{G}$ proceeds to the next step.

- Procedure $\Upsilon$:
  1. $\mathcal{G}$ rewinds $\vec{A}$ to the beginning of Step S1 of Protocol 1.
  2. $\mathcal{G}$ uniformly chooses a legal choice indicator $\vec{r}$. $\mathcal{G}$ executes Step R1 of Protocol 1 with the following exception: for each $\vec{r}(i) = 1$, $\mathcal{G}$ honestly generates a legal instance vector $\vec{x}_i$; for each $\vec{r}(i) = 0$, $\mathcal{G}$ calls algorithm Cheat($1^k, \Lambda$) to generate an illegal instance vector $\vec{x}_i$.
  3. $\mathcal{G}$ receives a choice indicator $r$ from $\vec{A}$. If $r$ is illegal, $\mathcal{G}$ sends Abort to the TTP and halts with outputting whatever $\vec{A}$ outputs.
  4. If $r = \vec{r}$, $\Upsilon$ outputs Success; otherwise, it outputs Failure.

Comment: The previous step is a central to extract $\vec{A}$’s real input. Note that, if $\Upsilon$ outputs Success, each non-chosen instance vector holds $n$ projective instances. $\mathcal{H}$’s feasible cheating property guarantees that $\vec{A}$ cannot detect cheating in this case. Due to $\mathcal{H}$’s projection property, $\mathcal{G}$ can extract $\vec{A}$’s real input (i.e., the $n$ values encrypted by $\vec{A}$).

Sim3: Playing the role of $\mathcal{R}$ with an arbitrary legal private input, $\mathcal{G}$ honestly executes Step R2 of Protocol 1 to interact with $\vec{A}$. On receiving $(\vec{c}, (\vec{p}_i)_{i \in S})$ sent by $\vec{A}$, $\mathcal{G}$ decrypts every entry of $\vec{c}$ and gains $\vec{A}$’s real input $\vec{m}'$.

Sim4: $\mathcal{G}$ sends $\vec{m}'$ to the TTP and receives in turn the null string $\lambda$. Then, $\mathcal{G}$ sends Continue to the TTP.

Sim5 (Output step): When $\vec{A}$ halts, $\mathcal{G}$ stops as well and outputs what $\vec{A}$ returned.

Proof.
Let $v(\cdot)$ be an arbitrary positive polynomial. We construct $M$ as follows.

Algorithm 3 Machine $\tilde{M}$

Input: The value $x$ such that $\text{Prob}(M(x) = \text{Success}) = q$, the polynomial $v$ and the security parameter $k$.

1. Set $\alpha \leftarrow \text{Failure}$.
2. For $i$ from 1 to $v(k)$, do the following:
   1. Run $M(x)$.
   2. If $M(x)$ has returned Success then $\alpha \leftarrow \text{Success}$ and break the For loop.
3. Return $\alpha$.

Output: $\alpha$: indicator of the success of at least one iteration of $M(x)$.

Let $M_i(x)$ be the output of $M(x)$ at the $i$-th iteration and denote by $X_i$ be the binary random variable defined as follows.

\[ X_i \overset{\text{def}}{=} \begin{cases} 0 & \text{if } M_i(x) = \text{Failure}, \\ 1 & \text{if } M_i(x) = \text{Success}. \end{cases} \]

12
The variable $X_i$ has a Bernoulli distribution with parameter $q$ standardly denoted by $\mathcal{B}(1,q)$. Consider a random variable $Y$ consisting of the sum of $v(k)$ independent random variables with the $\mathcal{B}(1,q)$ distribution. By definition, $Y$ has a binomial distribution $\mathcal{B}(v(k), q).

The reader should notice that the probabilistic event $\{\tilde{M}(x) = \text{Failure}\}$ corresponds to exactly $v(k)$ independent runs of machine $M$, each of them ending as a failure. As a consequence:

$$\text{Prob}(\tilde{M}(x) = \text{Failure}) = \text{Prob}(Y = 0) = (1 - q)^{v(k)}.$$ 

Since $q \in (0,1)$, there exists a constant $Q > 1$ such that $(1 - q) = 1/Q$. Thus,

$$\text{Prob}(\tilde{M}(x) = \text{Failure}) = 1/Q^{v(k)}$$

which decreases exponentially as a function of the security parameter $k$.

Based on the above proof, we modify Step Sim2 of $\mathcal{G}$ as follows.

**Sim2:** $\mathcal{G}$ executes Procedure $\tilde{\mathcal{T}}$ consisting of at most, say $v(k) = k^3$, iterations of Procedure $\mathcal{T}$ as exposed in Algorithm 3.

If $\tilde{\mathcal{T}}$ outputs Failure, then $\mathcal{G}$ outputs Failure and halts; otherwise, $\mathcal{G}$ proceeds to the next step.

The modified version of $\mathcal{G}$ using the above Sim2 step has the following property.

**Theorem 3** The (modified) simulator $\mathcal{G}$ runs in polynomial-time.

From now on, when referring to the simulator $\mathcal{G}$, we implicitly assume that it uses the modified Sim2 step above.

Note that $\mathcal{G}$ never sends Cheat$_1$ to the TTP. Thus, we actually constructed a standard simulator for $\mathcal{A}$ providing SAMA to $\mathcal{R}$. This means that the deference factor to the corrupted sender is 1. It remains to demonstrate that for any non-uniform PPT adversary $\mathcal{A}$ with an auxiliary input $z$ in the real world, the following equation holds:

$$\{\text{Ideal}_{1(1), \phi(z)}(1^k, \vec{m}, T)\}_{k \in \mathbb{N}, \vec{m} \in \{0,1\}^*} \in \mathcal{H}, z \in \{0,1\}^* \subseteq \{\text{Real}_{\mathcal{H}, 1(1), \mathcal{A}(z)}(1^k, \vec{m}, T)\}_{k \in \mathbb{N}, \vec{m} \in \{0,1\}^*} \in \mathcal{H}, z \in \{0,1\}^* \quad (2)$$

**Theorem 4** When only $S$ is corrupted, Equation (2) holds.

Before proving this result, we first need some intermediate lemmas.

**Lemma 3** The output of $\mathcal{G}$ in the ideal world and the output of $\mathcal{A}$ in the real world are computationally indistinguishable.  

**Proof.**
First, we claim that the outputs of $\mathcal{G}$ and $\tilde{\mathcal{A}}$ are computationally indistinguishable. Indeed, from its design, we know that $\mathcal{G}$ always takes $\tilde{\mathcal{A}}$’s output as its except when $\tilde{\mathcal{T}}$ returns Failure where $\mathcal{G}$ outputs Failure as well. Since this exception arises with negligible probability, our claim holds.

Second, we claim that the views of $\tilde{\mathcal{A}}$ and $\mathcal{A}$ are computationally indistinguishable. The only difference between the two views is that the non-chosen instance vectors in $\tilde{\mathcal{A}}$’s view are generated by Cheat$(1^k, \Lambda)$ and are illegal. The feasible cheating property of the hash family $\mathcal{H}$ guarantees that these instance vectors are computationally indistinguishable from those generated honestly. Therefore, this second claim holds which achieves the demonstration of our lemma.

**Definition 16** A probability ensemble $\{X(1^k, a)\}_{k \in \mathbb{N}, a \in \{0,1\}^*}$ is said to be polynomial-time constructible, if there exists a PPT sampling algorithm $\Delta$ such that for any $a \in \{0,1\}^*$ and any $k \in \mathbb{N}$, the random variables $\Delta(1^k, a)$ and $X(1^k, a)$ are identically distributed.

**Lemma 4** Consider two polynomial-time constructible probability ensembles $X = \{X(1^k, a)\}_{k \in \mathbb{N}, a \in \{0,1\}^*}$ and $Y = \{Y(1^k, a)\}_{k \in \mathbb{N}, a \in \{0,1\}^*}$. Let $F = (f_k)_{k \in \mathbb{N}}$ be an infinite function sequence, where each $f_k : \{0,1\}^* \rightarrow \{0,1\}^*$ is polynomial-time computable. Let $F(X) \overset{\text{def}}{=} \{f_k(X(1^k, a))\}_{k \in \mathbb{N}, a \in \{0,1\}^*}$ and $F(Y) \overset{\text{def}}{=} \{f_k(Y(1^k, a))\}_{k \in \mathbb{N}, a \in \{0,1\}^*}$.

Assume that $X \equiv Y$, then: $F(X) \equiv F(Y)$.

**Proof.**
Assume that the proposition be false. Then, there exists a non-uniform PPT distinguisher $D$ with an auxiliary input $z = (z_k)_{k \in \mathbb{N}}$, a polynomial $\text{Poly}(\cdot)$, an infinite positive integer set $G \subseteq \mathbb{N}$ such that, for each $k \in G$, it holds that

$$|\text{Prob}(D(1^k, z_k, a, f_k(X(1^k, a)))) = 1) - \text{Prob}(D(1^k, z_k, a, f_k(Y(1^k, a)))) = 1)| \geq 1/\text{Poly}(k).$$
We can construct a distinguisher $D'$ with the same auxiliary input $z = (z_k)_{k \in \mathbb{N}}$ for the ensembles $X$ and $Y$ as follows:

$$D'(1^k, z_k, a, \gamma) \overset{\text{def}}{=} D(1^k, z_k, a, f_k(\gamma)).$$

Obviously, we have:

$$D'(1^k, z_k, a, X(1^k, a)) = D(1^k, z_k, a, f_k(X(1^k, a)))$$
$$D'(1^k, z_k, a, Y(1^k, a)) = D(1^k, z_k, a, f_k(Y(1^k, a)))$$

We get:

$$\lvert \text{Prob}(D'(1^k, z_k, a, X(1^k, a)) = 1) - \text{Prob}(D'(1^k, z_k, a, Y(1^k, a)) = 1) \rvert \geq 1/\text{Poly}(k).$$

This contradicts $X \overset{\mu}{=} Y$. 

Now, we can proceed to prove Theorem 4.

Proof.
First, we focus on the real world. Note that $\mathcal{R}$’s output is a deterministic function of $\mathcal{A}$’s real input, denoted by $\gamma$. Without loss of generality, we assume that $\mathcal{A}$ takes its view as output. As as result, $\mathcal{A}$’s output, denoted by $\alpha$, contains its real input. Therefore, $\mathcal{R}$’s output is a deterministic function of $\mathcal{A}$’s output, where the function is given as follows.

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

Since the corrupted sender outputs nothing (i.e., the null string $\lambda$), the whole output of the real world is a deterministic function of $\mathcal{A}$’s output. Formally speaking, let $h(\alpha) \overset{\text{def}}{=} (\alpha, \lambda, g(\alpha))$, then we have:

$$\text{Real}_{\Pi,\{1\},\mathcal{A}(z)}(1^k, \vec{m}, T) \equiv h(\text{Real}_{\Pi,\{1\},\mathcal{A}(z)}(1^k, \vec{m}, T)(0)).$$

Second, in the ideal world, we have:

$$\text{Ideal}_{f_{\{1\},\mathcal{A}(z)}(1^k, \vec{m}, T)} \overset{\text{def}}{=} h(\text{Ideal}_{f_{\{1\},\mathcal{A}(z)}(1^k, \vec{m}, T)(0)}).$$

We use $\equiv$ here rather than $\approx$, because there is a negligible case that $\mathcal{G}$ outputs Failure and in this case $h(\cdot)$ is undefined. Let $X(1^k, \vec{m}, T, z, \{1\}) = \text{Real}_{\Pi,\{1\},\mathcal{A}(z)}(1^k, \vec{m}, T)(0)$ and $Y(1^k, \vec{m}, T, z, \{1\}) = \text{Ideal}_{f_{\{1\},\mathcal{A}(z)}(1^k, \vec{m}, T)(0)}$. Following Lemma 3, we get $X \overset{\mu}{=} Y$. Setting $F \overset{\text{def}}{=} (h)_{k \in \mathbb{N}}$ and applying Lemma 4, we obtain the result we sought.

Pulling together Theorem 3 and Theorem 4 we get that Theorem 2 is true in the case where only $\mathcal{S}$ is corrupted.

4.2.2 Case 2 – Only $\mathcal{R}$ is Corrupted

The adversary $\mathcal{A}$ takes the full control of $\mathcal{R}$ in the real world. Correspondingly, the simulator $\mathcal{G}$ takes the full control of $\mathcal{R}'$ in the ideal world where $\mathcal{G}$ is constructed as depicted in Algorithm 4. Recall that $\mathcal{R}'$ holds the same index set $T$ as $\mathcal{R}$. We set the same convention as in the case where $\mathcal{S}$ was dishonest concerning messages sent during the execution of $\mathcal{G}$.

Theorem 5 The simulator $\mathcal{G}$ runs in polynomial-time.

Proof.
Each step of $\mathcal{G}$ runs in polynomial-time. 

Lemma 5 If Case 1 of Step $\text{Sim}_3$ happens, then the output of $\mathcal{A}$ in the real world and that of $\tilde{\mathcal{A}}$ in the ideal world are computationally indistinguishable.

Proof.
The difference between $\tilde{\mathcal{A}}$'s view and $\mathcal{A}$'s view is that the ciphertext $\vec{c}'$ received by $\tilde{\mathcal{A}}$ is the encryption vector of $\vec{m}'$ while the ciphertext $\vec{c}$ received by $\mathcal{A}$ is the encryption vector of $\vec{m}$.

1. Case 1: $|T'| = t$. The smoothness of the hash family $\mathcal{H}$ directly guarantees that the two views are computationally indistinguishable.
Algorithm 4 Simulator $\mathcal{S}$ when $\mathcal{R}$ is the only Corrupted Player

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

Sim1 (Initialization step):
1. $\mathcal{S}$ corrupts $\mathcal{R}'$ and learns $\mathcal{R}'$’s private input $T$.
2. Let $\bar{\mathcal{A}}$ be a copy of $\mathcal{A}$. $\mathcal{S}$ invokes $\bar{\mathcal{A}}$ as a subroutine with initial inputs of $\bar{\mathcal{A}}$ to be $\mathcal{S}$’s initial inputs with the exception that the randomness of $\bar{\mathcal{A}}$ to be a uniformly distributed string. In other words, the initial inputs of $\bar{\mathcal{A}}$ are set as $\bar{k}, I, z$ and $r_{\bar{\mathcal{A}}}$ with $r_{\bar{\mathcal{A}}} \in_U \{0, 1\}^*$.
3. $\mathcal{S}$ activates $\bar{\mathcal{A}}$.
4. As in the real world, $\bar{\mathcal{A}}$ sends a message to corrupt the receiver. $\mathcal{S}$ plays the role of $\mathcal{R}$ and supplies $\bar{\mathcal{A}}$ with $T$.

Comment: To simulate the environment of $\bar{\mathcal{A}}$ in the real world, $\mathcal{S}$ is to simultaneously disguises himself as $\mathcal{S}$ and $\mathcal{R}$ to interact with $\bar{\mathcal{A}}$ during the following steps.

Sim2: On receiving $(\Lambda, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_K)$ sent by $\bar{\mathcal{A}}$, $\mathcal{S}$ checks the legality of $\Lambda$.

- If $\Lambda$ is illegal, then $\mathcal{S}$ sends $\text{Abort}_2$ to the TTP and receives $\text{Abort}_2$ in return. When $\bar{\mathcal{A}}$ halts, $\mathcal{S}$ stops as well and outputs what $\bar{\mathcal{A}}$ returned.

- If $\Lambda$ is legal, then $\mathcal{S}$ rewinds $\bar{\mathcal{A}}$ to extract $\bar{\mathcal{A}}$’s responses to the distinct choice indicators. Note that there are $v = (|\bar{\mathcal{A}}|)$ legal choice indicators in total. Denote them by $r_1, r_2, \ldots, r_v$. $\mathcal{S}$ rewinds $\bar{\mathcal{A}} v$ times, where in the $i$-th rewind, $\mathcal{S}$ works as follows.
  1. Playing the role of $\mathcal{S}$, $\mathcal{S}$ sends $r_i$ to $\bar{\mathcal{A}}$ and $\mathcal{S}$ records $\bar{\mathcal{A}}$’s response (i.e., either something which serves as the witness vectors of the chosen instance vectors, or a rejection to send anything).
  2. $\mathcal{S}$ rewinds $\bar{\mathcal{A}}$ to the beginning of Step R2 of Protocol 1.

Sim3: $\mathcal{S}$ processes $\bar{\mathcal{A}}$’s responses. For convenience, we say a response is **bad**, if it causes the honest sender to output Corrupted$_2$. $\mathcal{S}$ proceeds as follows.

- Case 1: there are no bad responses. $\mathcal{S}$ proceeds to Step Sim4.

- Case 2: all responses are bad. $\mathcal{S}$ sends Corrupted$_2$ to the TTP and receives Corrupted$_2$ in return. Then, playing the role of $\mathcal{S}$, $\mathcal{S}$ honestly follows Step S1-S2 of Protocol 1 to interact with $\bar{\mathcal{A}}$. When $\bar{\mathcal{A}}$ halts, $\mathcal{S}$ stops as well and outputs what $\bar{\mathcal{A}}$ returned.

- Case 3: some (but not all) responses are bad. Given the number of bad responses, the probability $\epsilon$ that $\bar{\mathcal{A}}$’s cheating can be caught by the honest sender $\mathcal{S}$ in the real world can be figured out. This is the deterrence factor to the corrupted receiver in the real world. $\mathcal{S}$ send Cheat$_2$ to the TTP.
  - Case 3.1: with probability $\epsilon$, the TTP replies $\mathcal{S}$ with Corrupted$_2$. $\mathcal{S}$ uniformly chooses one of legal choice indicators that will cause $\bar{\mathcal{A}}$’s cheating to be caught and sends it to $\bar{\mathcal{A}}$.
  - Case 3.2: with probability $1 - \epsilon$, the TTP replies $\mathcal{S}$ with Undetected and $\mathcal{S}$’s value vector $\bar{m}$. $\mathcal{S}$ determines the result that $\mathcal{S}$ will receive to be the null string $\lambda$ and sends $\lambda$ to the TTP. $\mathcal{S}$ uniformly chooses one of legal choice indicators that will not cause $\bar{\mathcal{A}}$’s cheating to be caught and sends it to $\bar{\mathcal{A}}$. Then, playing the role of $\mathcal{S}$ with private input $\bar{m}$, $\mathcal{S}$ honestly follows Step S2 to interact with $\bar{\mathcal{A}}$.

When $\bar{\mathcal{A}}$ halts, $\mathcal{S}$ stops as well and outputs what $\bar{\mathcal{A}}$ returned.

Sim4: Playing the role of $\mathcal{S}$, $\mathcal{S}$ honestly executes Step S1 to Step S2.2 of Protocol 1. At this point, $\mathcal{S}$ extracts $\bar{\mathcal{A}}$’s real input using the witness vectors recorded in Step Sim3 and the permutations $(\sigma_i^j)_{i,j\in\mathcal{N}}$ received in the current interaction. To do so, $\mathcal{S}$ operates as follows: for each $i \in \mathcal{CS}$, set $T_i^1 \leftarrow \emptyset$; for each $i \in \mathcal{CS}$ and $j \in [n]$, if the $j$-th entry of the instance vector $\bar{x}_i$ is projective, then $T_i^l \leftarrow T_i^l \cup \{j\}$; finally: $T_i^l \leftarrow \cap_{i \in \mathcal{CS}} T_i^l$.

Comment: The above extraction process works since we are in the same situation as Case 1 of Step Sim3 where $\bar{\mathcal{A}}$’s all possible responses are good. This implies that all the instance vectors sent by $\bar{\mathcal{A}}$ are legal.

Sim5: $\mathcal{S}$ sends $T^l$ to the TTP, receives $(\bar{m}(i))_{i \in T^l}$ in return and sends back Continue. To carry on interactions with $\bar{\mathcal{A}}$, $\mathcal{S}$ builds a vector $\bar{m}'$ as follows. For each $i \in T^l$, set $\bar{m}'(i) \leftarrow \bar{m}(i)$. For each $i \notin T^l$, set $\bar{m}'(i)$ to be a value uniformly chosen from $G_A$ where $G_A$ denotes the set of all possible hash values (see Definition 13).

Sim6 (Output step): Playing the role of $\mathcal{S}$ with private input $\bar{m}'$, $\mathcal{S}$ follows Steps S2.3 and S2.4 to complete the interaction with $\bar{\mathcal{A}}$. When $\bar{\mathcal{A}}$ halts, $\mathcal{S}$ stops as well and outputs what $\bar{\mathcal{A}}$ returned.
2. Case 2: $|T'| < t$. Looking at Step S2.3 of Protocol 1 where the sender encrypts its values, we know that the encryption vector is obtained by XOR-ing the $n$ components of $\vec{m}$ with the hash values of all non-chosen instance vectors. For each $j \in T'$, $\vec{m}'(j)$ (resp, $\vec{m}(j)$) is hidden by the hash values of the projective instances. Because of $\mathcal{H}$’s projection property, $\mathcal{A}$ knows $\vec{m}'(j)$ (resp, $\mathcal{A}$ knows $\vec{m}(j)$). Note that $(\vec{m}(i))_{i \in T'} = (\vec{m}'(i))_{i \in T'}$. Thus, $(\vec{c}(i))_{i \in T'} \equiv (\vec{c}'(i))_{i \in T'}$. For each $j \notin T'$, $\vec{m}'(j)$ (resp, $\vec{m}(j)$) is hidden by the hash values of the smooth instances. Due to the smoothness property of $\mathcal{H}$, $\vec{c}(j)$ (resp, $\vec{c}'(j)$) only leaks negligible knowledge about $\vec{m}'(j)$ (resp, $\vec{m}(j)$). Therefore: $(\vec{c}(i))_{i \in T'} \approx (\vec{c}'(i))_{i \notin T'}$. Thus, $\vec{c} \approx \vec{c}'$ and the two views are computationally indistinguishable. □

**Remark 7** In the previous proof, it was impossible to have $|T'| > t$ since all instance vectors generated by $\mathcal{A}$ were legal.

**Remark 8** The proof of Lemma 5 also shows that both $\mathcal{A}$’s effective private input and $\mathcal{A}$’s were $T'$.

**Lemma 6** If Case 2 or Case 3 of Step Sim3 happens, then the output of $\mathcal{A}$ in the real world and that of $\mathcal{A}$ in the ideal world are identical. 

**Proof.**
Looking at Algorithm 4 if $\mathcal{G}$ receives back Corrupted, from the TTP, then $\mathcal{G}$ can perfectly play the role of $\mathcal{S}$ to interact with $\mathcal{A}$ although $\mathcal{G}$ ignores $\vec{m}$. If $\mathcal{G}$ receives Undetected from the TTP, then $\mathcal{G}$ knows $\vec{m}$ and perfectly plays the role of $\mathcal{S}$ in this case as well. Therefore, $\mathcal{A}$’s view and $\mathcal{A}$’s view are identical in the two cases. This implies their outputs to be identical in both cases. □

**Lemma 7** The output of $\mathcal{A}$ in the real world and that of $\mathcal{G}$ in the ideal world are computationally indistinguishable.

**Proof.**
First, we claim that the outputs of $\mathcal{G}$ and $\mathcal{A}$ are identical. Indeed, $\mathcal{G}$ always takes $\mathcal{A}$’s output as its own. Second, we assert that the outputs of $\mathcal{A}$ and $\mathcal{A}$ are computationally indistinguishable. Indeed, looking at Step Sim3, the probability that each case happens is identical to that in the real world. Thus, according to Lemma 5 and Lemma 6 this second claim holds. □

**Theorem 6** When only $\mathcal{R}$ is corrupted, Equation (1) required by Definition 3 holds.

**Proof.**
Note that the honest senders $\mathcal{S}$ (in the real world) and $\mathcal{S}'$ (in the ideal world) end up with outputting nothing. In addition, $\mathcal{R}$ and $\mathcal{R}'$ do not output anything either. Thus, the fact that the outputs of $\mathcal{G}$ and $\mathcal{A}$ are computationally indistinguishable, shown by Lemma 7 directly proves this theorem. □

Pulling together Theorem 5 and Theorem 6 we get that Theorem 2 is true in the case where only $\mathcal{R}$ is corrupted. This achieves our demonstration of Theorem 2.

## 5 Efficiency Of The Framework II

It is clear that II 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 consideration is justified since, in II, 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 SPH was to construct public-key encryption schemes [CS02]. With this consideration in mind, we can see that the main computational overhead of our framework is $(K - g) \cdot n$ encryptions at the sender (Step S2) and $(K - g) \cdot t$ decryptions at the receiver (Step R3). For instance, setting $K = 20$ and $g = 10$, our protocol costs $10 \cdot n$ encryptions and $10 \cdot t$ decryptions.

The deterrence factor to a covert sender is 1 while and the deterrence factor to a covert receiver is $1 - 1/(K - g)$ (roughly equal to $1 - 5.41 \times 10^{-6}$ when $K = 20$ and $g = 10$). Since the deterrence factor to a covert sender is independent of both $K$ and $g$, we will implicitly refer the deterrence factor to a receiver when discussing about the deterrence factor of our scheme from now on.

---

1. 20
Since our framework for OT$_n^m$ provides non-adaptive SACA, we compare it to protocols offering either SACA or SAMA such as those from [GH07, Lin08, AL10, LP11, PVW08, Kol10]. Table 1 shows an efficiency comparison of the instantiation of these protocols. Because [GH07] relies on the Decisional Bilinear Diffie-Hellman (DBDH) assumption and [LP11] is DDH-based, we instantiated other protocols under the DDH assumption as well for fairness. Note that the protocols from [PVW08, Kol10] are not listed in this table since they use a stronger assumption (namely, the existence of a trusted set-up). Nonetheless, we will still discuss these approaches in Section 5.3.

In Table 1, we instantiated II with hash family presented at the end of Section 2.3. In this hash family, algorithms IS, DI, KG, Hash, pHash respectively cost $2n$, $2$, $1.25$, $1.25$, $1$ modular exponentiations. Note that each double exponentiation of the form $g^{t_1}g^{t_2}$ costs $1.25$ times the cost of a standard exponentiation [MvOV96]. In II, IS is invoked $K$ times while Check is run once during Step R1. DI is called $g(n-t)$ times, KG $(K-g)n$ times and Hash $(K-g)n$ times during Step R2 and pHash is run $(K-g)n$ times in Step R3. In summary, the protocol costs $4.5K + Kt - 0.5gn - 3gt$ exponentiations in addition to the overhead of Check.

To illustrate our comparative analysis, we set $K = 2$ and $g = 1$. The resulting deterrence factor to the receiver for II is $1/2$. This instantiation costs $8.5n - t$ exponentiations in addition to the overhead of Check. In Table 1, we omitted the overhead of Check (i.e., the overhead of checking the legality of the description of a group of prime order). This is due to the fact that all other protocols listed in the table also need to perform their own checking. Overlooking this overhead does not hurt the correctness of the comparison and it allows to simplify the content of Table 1.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Functionality</th>
<th>Assumption</th>
<th>Security</th>
<th>Communication round</th>
<th>Exponentiation</th>
<th>Bilinear map</th>
</tr>
</thead>
<tbody>
<tr>
<td>[GH07]</td>
<td>OT$_1^{n}$</td>
<td>DDH</td>
<td>SAMA</td>
<td>$\geq 12$</td>
<td>$\geq 4n + 17.5t + 23.25$, $\geq n$</td>
<td>$\geq 3n + 5t$</td>
</tr>
<tr>
<td>Lin08</td>
<td>OT$_1^{n}$</td>
<td>DDH</td>
<td>SAMA</td>
<td>$6$</td>
<td>$426.5$</td>
<td></td>
</tr>
<tr>
<td>LP11</td>
<td>OT$_1^{n}$</td>
<td>DDH</td>
<td>SAMA</td>
<td>$6$</td>
<td>$26$</td>
<td></td>
</tr>
<tr>
<td>[AL10]</td>
<td>OT$_1^{n}$</td>
<td>DDH</td>
<td>SACA</td>
<td>$4$</td>
<td>$23$</td>
<td></td>
</tr>
<tr>
<td>Our work</td>
<td>OT$_1^{n}$ (OT$_1^{2}$)</td>
<td>DDH</td>
<td>SACA</td>
<td>$4$</td>
<td>$8.5n - t (16)$</td>
<td></td>
</tr>
</tbody>
</table>

Let $\hat{e} : G \times G \rightarrow \hat{G}$ be a bilinear map. For the pairing-based protocol of [GH07], we use two values to separately show the cost of exponentiations in $G$ and in $\hat{G}$. The first row shows that protocol of [GH07] costs at least $4n + 17.5t + 23.25$ exponentiations of $G$ and at least $n$ exponentiations of $\hat{G}$.

Table 1: A Comparison with Known Practical Protocols in the Plain Model

### 5.1 Comparison to Protocols with SAMA

#### 5.1.1 Green-Hohenberger [GH07]

In [GH07], the construction of [Sch91] is used as a ZK proof of knowledge of discrete logarithm modulo a prime. However, as pointed out in [CDM00], Schnorr’s technique cannot be proved to be ZK against a dishonest verifier (at least not by any known repeatable simulation technique). For this reason, we replaced it with the ZK version presented in [CDM00] for our comparison. This version costs 4 communication rounds and 10 exponentiations.

Green and Hohenberger’s protocol is described at a high level making it difficult to figure out the exact numbers of communication rounds, exponentiations and bilinear maps costed by this protocol. However, based on the properties of $\Sigma$-protocol [CDM00, Dam11], we can get a lower bound on these overheads. These bounds are exposed in Table 1.

Since the fact that our computational overhead is smaller does not clearly appear, a detailed analysis is needed. The protocol from [GH07] is based on pairings. In [GPS08], it is shown that the efficiency of cryptography using pairings scales more-or-less like RSA [RSA78] rather than like elliptic curve cryptography. This means that pairing-based schemes need keys that are similar in size to RSA’s. Therefore, using elliptic curves, our solution II based on the DDH assumption is considerably more efficient.

#### 5.1.2 Lindell [Lin08]

Lindell presents two protocols for OT$_1^2$. One relies on the DDH assumption and the other assumes the existence of homomorphic encryption schemes. The ideas behind the two protocols are identical. Let’s set the statistical error parameter of [Lin08] to be $40$ which is the value suggested in the paper. Because the computational overhead is a random variable, we figured out its expected value. The overhead of the first protocol, which is the more efficient, is shown in the comparison
table. The second protocol costs 6 communication, 122 encryptions, 1 decryptions, 40 homomorphic operations. The table clearly indicates that II is more efficient than Lindell’s protocols under the DDH assumption. Recalling our overhead stated previously, our protocol is still more efficient under the other intractability assumptions underlined previously, even when we raise the deterrence factor to be a high value such as \(1 - 5.41 \times 10^{-6}\).

5.1.3 Lindell-Pinkas \([\text{LP11}]\)

The values shown in the comparison table come from \([\text{LP11}]\). Making use of a highly-efficient ZK proof of knowledge protocol to generate the CRS, Lindell and Pinkas presented a DDH-based instantiation of \([\text{PVW08}]\)’s framework in the plain mode. However, this instantiation loses the universal composability property. It needs 2 communication rounds and 10 exponentiations more than our DDH-based instantiation.

As Lindell and Pinkas’ scheme provides SAMA and its cost is not significantly larger than our instantiation, it seems that our framework II has little interest. However, it is unknown if every instantiation of \([\text{PVW08}]\)’s framework has such a highly efficient generation of the CRS in the plain mode (in fact, only the DDH-based instantiation is formally proved to have such a CRS generation). For example, its lattice-based instantiation. Considering this, our framework II still makes sense, because it can provide efficient protocols for OT_{1}^{\ast} in the plain model under the same intractability assumptions as \([\text{PVW08}]\). This is important, especially in the quantum setting, since the DDH assumption is false due to Shor’s result on the factoring and discrete logarithm problems \([\text{Sho97}]\).

5.2 Comparison to Protocols with SACA

The protocol from \([\text{AL10}]\) requires the existence of homomorphic encryption schemes. It costs \(4K\) encryptions (where \(K\) can be any polynomial in the security parameter), 1 decryption and 2 homomorphic operations. It provides SACA with deterrence factor 1 and \(1 - 1/K\) to the sender and the receiver respectively. For its fairness, we assume that ElGamal’s public key encryption \([\text{ElG85}]\) is used as an homomorphic cryptosystem and set \(K = 2\). The exact overhead of this instantiation is shown in the table.

The protocol from \([\text{AL10}]\) exhibits several differences with ours. First, it is based on homomorphic encryptions while we use a variant of smooth projective hashing. Second, it is restricted to the OT_{1}^{\ast} functionality while we consider the general OT_{t}^{\ast} problem. Third, its deterrence factor is of the form \(1 - 1/K\) while ours is of the form \(1 - 1/(K - \varepsilon)\) whose granularity is better. Fourth, II is more efficient. Indeed, when instantiate our protocol as previously stated in the case \(n = 2\) and \(t = 1\), Table 1 shows that our framework II costs 7 exponentiations less than \([\text{AL10}]\).

In fact, the higher the deterrence factor is, the bigger the computational efficiency gap between II and \([\text{AL10}]\) becomes. For example, fixing deterrence factor to be \(1 - 1/(\log K) = 1 - 5.41 \times 10^{-6}\), then our framework costs 20 encryptions and 10 decryptions while the protocol by Aumann and Lindell needs \(4 \times K = 739024\) encryptions, 1 decryption, and 2 homomorphic operations. This computational efficiency gap between the two approaches is large. This example also shows that II can achieve considerably higher deterrence factor under the same computational cost. We remark that the idea of this paper also can be used to extend the protocol of \([\text{AL10}]\) to deal with general OT_{t}^{\ast} and to gain better granularity for the deterrence factor.

5.3 Comparison to Protocols with Trusted Set-Up

This comparison is a bit tricky. Indeed, on one hand, such protocols (very likely) are more efficient than those listed in Table 1. On one hand, their practical usages are more restricted since one needs to first establish the trusted set-up.

5.3.1 Peikert-Vaikuntanathan-Waters \([\text{PVW08}]\)

In \([\text{PVW08}]\), Peikert et al. presented a framework for OT_{1}^{\ast} with SAMA under the set-up assumption that a trusted CRS is available. Their framework can be instantiated under assumptions such as the DDH, the DQR and LWE. It is remarkable that the framework of \([\text{PVW08}]\) may be the most efficient one for OT_{1}^{\ast} in the setting where a trusted CRS is available, and that it preserves SAMA under universal composition which is not the case of ours or other works mentioned in the section.

To use such a protocol in practice, one has to solve the problem of providing the needed CRS first. This problem seems difficult. In \([\text{CF01, CKL06}]\), Canetti et al. showed that even given a authenticated communication channel, it is impossible to implement a universal composable protocol that provides useful CRS in the presence of malicious adversaries. In \([\text{PVW08}]\), it is suggested to get the CRS from a natural processes. However, no formal proof is given and Katz pointed out that although certain natural events can be viewed as producing bit-sources with high min-entropy, the resulting bit-sources may not be uniformly random \([\text{Kat07}]\). Therefore, the usage of this framework in practice is very restricted.

18
5.3.2 Kolesnikov [Kol10]

Based on the set-up assumption that a resettable (actually, stateless) tamper-proof hardware implementing a strong pseu-
dorandom permutation (SPRPG) is available, Kolesnikov presents a protocol for OT\textsuperscript{2} with SACA. Though this scheme
is based on a set-up assumption, it is not universal composable but sequentially composable as protocols listed in the table.

There are several additional differences between Kolesnikov’s protocol and our approach. First, it is secure against a covert
sender and a malicious receiver while, in contrast, our framework is secure against a malicious sender and a covert receiver.
Second, the instantiation security of Kolesnikov’s protocol depends not only on the security of the hardware but also on the
security of the SPRPG instantiations. Kolesnikov suggests to instantiate the SPRPG with heuristic block ciphers, such as
AES [DR02]. As a result, though the whole protocol is highly efficient, its security is not provable at present. Concerning II
however, the security of its instantiations only depends on the security of the underlying intractability assumptions, and
so, our construction has provable security.

Kolesnikov’s protocol requires the receiver to get a hardware created by the sender before executing the protocol. He
envisioned his protocol to be used in a context where cheap or free tamper-proof hardwares (SIM cards for cell phones,
TV cable smart cards) are available to the users. However, in many settings, (e.g., peer-to-peer network), this is unpractical.

5.4 Global Comparison

Since the communication rounds of the protocols listed in Table 1 are independent of their concrete instantiations, it is
clear that our protocol costs the minimum number of communication rounds among the protocols that work in the plain
model.

Considering the case under the DDH assumption, the table and the previous analysis of [GH07] show that our protocol
with deterrence factor \(1/2\) costs the minimum computational overhead among the protocols listed in the table. Considering
the cases under other intractability assumptions, only \([\text{Lin08}][\text{AL10}]\) are known to have instantiations. Following the
previous analysis of \([\text{Lin08}][\text{AL10}]\), our protocol is more efficient than them too, even when raise deterrence factor to be
a high value such as \(1 - 5.41 \times 10^{-6}\). Therefore, in general, our protocol costs the minimum computational overhead
among the protocols that work in the plain model.

We conclude that we the following facts:

- If no trusted set-up is available and the adversaries are covert, then our framework II is the best solution to OT\textsuperscript{2}.
- If no trusted set-up is available and the adversaries are malicious, then the protocol of [GH07] is the best solution to
  OT\textsuperscript{ni} and the protocols of \([\text{Lin08}][\text{LPT11}]\) are the best solutions to OT\textsuperscript{i};
- If a trusted CRS is available, then the framework of [PYW08] is the best solution to OT\textsuperscript{2}.
- If cheap or free tamper-proof hardware are available and the adversaries are covert, then the protocol of [Kol10] is
  the best solution to OT\textsuperscript{2}.

6 Conclusion

In this paper, we presented a new framework II realizing the OT\textsuperscript{ni} functionality with SACA. Our computationally secure
construction is based on the use of SPH. It does not need any set-up assumption and only uses a small number of rounds
(four). When II is instantiated using a hash family defined over a (multiplicative) group of prime order, it is shown that the
number of group exponentiation for II is much smaller than what is needed by some recent OT protocols providing SACA
and SAMA. As SACA is a very novel security notion, one may wonder if it would be possible to reduce the computational
cost for realizing the above MPC functionality even further by using a different cryptographic primitive than projective
smooth hashing even at a slightly larger number of rounds (yet, still constant).

Acknowledgment

The authors would like to acknowledge Yehuda Lindell for valuable discussions on the relations amongst security notions
for MPC. We would like to thank the reviewers for their suggestions to improve the quality of this paper. Christophe
Tartary’s research was funded by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301,
the National Natural Science Foundation of China Grant 61033001, 61073174. Christophe Tartary also acknowledges
support from the Danish National Research Foundation and the National Natural Science Foundation of China (under
the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation (CTIC) within which part
of this work was performed. Christophe Tartary’s work was also financed by the International Young Scientists program.
of the Natural Science Foundation of China (61050110147 and 61150110344). Peng Xu acknowledges support from the National Natural Science Foundation of China (Grant No: 61100222) and China Postdoctoral Science Foundation (Grant No: 20100480900).

References


