An Efficient Outsourced Oblivious Transfer Extension Protocol and Its Applications

Shengnan Zhao,1 Xiangfu Song,1 Han Jiang,1 Ming Ma,1 Zhihua Zheng,2 and Qiuliang Xu1

1School of Software, Shandong University, Jinan 250100, China
2School of Information Science and Engineering, Shandong Normal University, Jinan 250358, China

Correspondence should be addressed to Han Jiang; jianghan@sdu.edu.cn and Qiuliang Xu; xql@sdu.edu.cn

Received 20 August 2020; Revised 2 November 2020; Accepted 18 November 2020; Published 5 December 2020

Oblivious transfer (OT) is a cryptographic primitive originally used to transfer a collection of messages from the sender to the receiver in an oblivious manner. OT extension protocol reduces expensive asymmetric operations by running a small number of OT instances first and then cheap symmetric operations. While most earlier works discussed security model or communication and computation complexity of OT in general case, we focus on concrete application scenarios, especially where the sender in the OT protocol is a database with less computation and limited interaction capability. In this paper, we propose a generic outsourced OT extension protocol (OText) that outsources all the asymmetric operations of the sender to a semihonest server so as to adapt to specific scenarios above. We give OText a standard security definition, and the proposed protocol is proven secure in the semihonest model. In OText, the sender works on the fly and performs only symmetric operations locally. Whatever the number of rounds OT to be executed and the length of messages in OT to be sent, our protocol realizes optimal complexity. Besides, OText can be used to construct high-level protocols, such as private membership test (PMT) and private set intersection (PSI). We believe our OText construction may be a building block in other applications as well.

1. Introduction

Oblivious transfer (OT) is one of the most important primitives in secure computation. It is wildly used in Yao’s protocol [1], GMW construction [2], and preprocessing phase of SPDZ-like [3] protocols. With the development of big data, cloud computing, and mobile computing, the demand for joint computation grows rapidly between different organizations and individuals. In order to ensure the security of such computing tasks against complicated external environment, cryptographic components have to be used to design protocols, in which OT plays a pivotal role together with homomorphic encryption, secret sharing, and garbled circuit.

However, OT is public-key primitive centered, which makes it computational expensive for secure computation. Many privacy-preserving protocols, such as private membership test (PMT) and private set intersection (PSI), rely heavily on huge number of OT instances for secure computation to get the trade-off between computation and communication. The most efficient way to produce many OT instances is through OT extension protocol [4, 5]. In such protocol, two participants collectively run few “base” OT instances and then perform some cheap symmetric operations to produce many OT instances.

1.1. Motivation. In an OT extension protocol, the sender S needs to interact with the receiver R for each step during the protocol, which involves exponential calculation and intensive interaction. In some application scenarios, however, S could be a mobile device with less computation power or a database holder with limited interaction capability. When invoking OT extension as a subprotocol in some more complex computation tasks, S needs to respond requests from R as fast as possible.

Research Article

An Efficient Outsourced Oblivious Transfer Extension Protocol and Its Applications
Nowadays, many applications are rapidly transferred to cloud-based service, and it would be desired to seek some server-aided OT extension protocol to relieve the burden of $\mathcal{DR}$ under reasonable security assumption. A considerable literature [6–12] has grown up around the theme of fertilizing functionality of OT or optimizing communication cost of the receiver $\mathcal{DR}$. However, far too little attention has been paid to investigate sender side of OT adapting to specific scenarios.

To this end, we propose a generic outsourced oblivious transfer extension protocol (OTex) in the semihonest model. In OTex, the sender $\mathcal{DS}$ first outsources all expensive asymmetric operations to a third party who runs a sub-protocol called “base” OT instances with the receiver $\mathcal{DR}$. Based on the corresponding outputs of the subprotocol and other auxiliary information, $\mathcal{DS}$ generates symmetric keys used to encrypt sending messages in OT. As a result, the sender $\mathcal{DS}$ works on the fly and sends its inputs encrypted by symmetric key generated from OTex to the receiver $\mathcal{DR}$, and thus, it enables two parties to complete the whole OT extension protocol.

Recent trends in OT extension have led to a proliferation of studies showing how to design an efficient PSI [13–17] or PSI-based protocols [18–21] in different secure models since OT extension protocol is an important component in secure computation and plays a key role in set operations. Take PSI as an example, without violating individuals’ privacy, and the use of PSI in contact tracing [21] can help prevent the further spread of COVID-19. Therefore, OTex framework has a wide range of applications in outsourced scenarios, and as a building block, we think OTex can be applied conveniently to high-level protocols.

1.2. Related Work. Rabin [22] first introduced the notion of OT that the receiver receives a message sent from the sender with probability 1/2 and the sender does not know whether the receiver has received the message or not. Then, a line of works seek to enrich functionality of OT, and they mainly consist of 1-out-of-2 OT [6], 1-out-of-$n$ OT [7, 8], and $k$-out-of-$n$ OT [9, 10], in which the first two functionalities are considered in this paper. Some researchers [23–25] proposed OT protocols in different secure models although we focus on semihonest model that is sufficient enough to deploy our framework in future among including malicious model, covert adversary model, and universally composable model.

Because OT is public-key primitive centered, the issue of efficiency has received considerable critical attention after Rabin’s work [22]. Beaver [26] showed that OT can be precomputed using only prior transfers. Studies over the past two decades have proved that extending a small number of OT to huge number of OT instances can be achieved by one-way function [27–29]. Until 2003, Ishai et al. [4] proposed an efficient method to extend 1-out-of-2 OTs by running few “base” OT instances, which is also known as the IKNP OT extension protocol. Kolesnikov and Kumaresan [5] generalized IKNP from a coding understanding and proposed an improved OT extension protocol allowing to do 1-of-out-$n$ OTs with less communication and computation cost. Lindell et al. [11] studied input-size hiding two-party computation based on fully homomorphic encryption (FHE) and proposed a secure OT extension protocol to reduce the communication cost of both the sender and receiver. Cho et al. [12] focused on the receivers’ communication cost in OT and proposed laconic OT protocol based on the decisional Diffie–Hellman (DDH) assumption. Carter et al. [30] proposed outsourced OT protocol specifically for the mobile use-case where the cloud receives outputs of OT. Recently, Mansy and Rindal proposed [31] noninteractive OTs from noninteractive key exchange. However, the resulting OT extension would require 3 rounds.

The research to date has tended to focus more on the cost of the receiver and less on the sender in OT, and there are few studies that have investigated computation and communication complexity on sender side. The aim of this work is to explore the cost of sender in OT and construct efficient OT extension framework assisted by a third party. In addition, OT extension provides a brief but useful account of the construction of oblivious pseudorandom function (PRF). Also, oblivious PRF has been attracting a lot of interest in very recent years, such as multiparty PSI [14], PSI cardinality [21], and private set union [32]. This indicates a need to adapt OT extension to outsourcing scenarios due to practical constraints.

1.3. Our Contribution. In this paper, we focus on server-aided OT to reduce the sender’s public-key computation and rounds of interaction with the receiver. Main contributions of our work go as follows:

(i) We propose a generic outsourced OT extension protocol (OTex). In OTex, the server and the receiver cooperatively run a small number of OT instances, and at this moment, the sender can be offline. After the first phase, the sender can fetch necessary correlated randomness from the server whenever needed; then, the sender can send their inputs encrypted only by a symmetric key to the receiver, which completes an OT instance. We design a novel mechanism for this purpose and formally prove its security under semihonest model.

(ii) We analyze the complexity of our construction and perform implementation, and the experiment shows that our construction is practical and efficient.

(iii) Our OTex construction can be applied to improve the efficiency of OT-based privacy-preserving primitives in server-aided setting, such as oblivious pseudorandom function, and high-level protocols, such as PMT and PSI, which is of independent interest.

2. Preliminaries

2.1. Notation. Unless otherwise stated, we use OT to denote 1-out-of-2 OT and OT$^k$ to denote $k$ instances of 1-out-of-2 OT of $l$-bit strings. For simplicity, let $H(\cdot)$ denote random
oracles with a suitable secure output length, which will be well defined in an actual protocol. Matrices are denoted by capital letters, and vectors are denoted by small bold letters; that is, \( t_i \) denotes the \( i \)-th row of a matrix \( T \), and \( \mathbf{v} \) denotes the \( j \)-th column of \( T \). Small light front with subscript \( s_j \) denotes the \( j \)-th bit of a string \( s \). Notably, we regard strings as vectors and do not distinguish the difference between strings and vectors. If \( s = s_1 \cdots s_n \) and \( t = t_1 \cdots t_n \) are two strings, then the notation \( s \odot t \) denotes \((s_1 \oplus t_1) \cdots (s_n \oplus t_n)\). Similarly, the notation \( s \odot t \) denotes the vector \((s_1 \cdot t_1) \cdots (s_n \cdot t_n)\). Specially, when \( c \in \{0,1\}, c \cdot s \) denotes the vector \( (c \cdot s)_1 \cdots (c \cdot s)_n \).

2.2. Secure Computation and Security Model. The formal definition of security of a secure multiparty protocol [33] is based on comparing two output distributions coming from an ideal world and a real world, respectively. The functionality of three parties \( P_1, P_2, P_3 \) is denoted as \( f: \{0,1\}^\ast \times \{0,1\} \times \{0,1\}^\ast \mapsto \{0,1\}^\ast \times \{0,1\}^\ast \times \{0,1\}^\ast \), where \( f = (f_1,f_2,f_3) \) and \( P_i \) gets \( f_i \) as output.

Ideal/reality Simulation Paradigm. In an ideal world, participants send the input to an external trusted party who computes the functionality and sends each participant the corresponding output. Suppose there exists an adversary who has the inputs and outputs of a protocol in the ideal world and executes attack against a real protocol, then there always be an adversary executing the same attack in the ideal world. In a real protocol, if no adversary can do more harm than the execution of the protocol in the ideal world, the protocol in the real world is said to secure.

Computationally Indistinguishability. Two distribution probability ensembles \( X = \{X(a,n)\}_{a \in \{0,1\}^\ast \cap \mathbb{N}} \) and \( Y = \{Y(a,n)\}_{a \in \{0,1\}^\ast \cap \mathbb{N}} \) are said to be computationally indistinguishable, denoted by \( X \equiv Y \), if for every non-uniform polynomial-time algorithm \( D \), there exists a negligible function \( \mu(\cdot) \) such that for every \( a \in \{0,1\}^\ast \) and every \( n \in \mathbb{N} \):

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

Semi-honest Adversary Model. In the semi-honest adversary model, corrupted participant must execute the protocol correctly. However, the adversary can comprehensively obtain the internal status of the corrupted party, e.g., the transcripts of all received messages, and then tries to obtain additional information that should be kept confidential. Semi-honest model is sufficient and captures many scenarios in practice although it is a very weak adversary model.

In this paper, we focus on semi-honest model and honest majority case where an adversary can corrupt at most one participant and any two participants will not get colluded. In the following, the formal security definition is proposed.

Definition 1. Let \( f = (f_1,f_2,f_3) \) be a deterministic functionality and \( \pi \) be a three-party protocol for computing \( f \). Given the security parameter \( \kappa \) and triple inputs \( (x, y, z) \), where \( x \) is from \( P_1 \), \( y \) is from \( P_2 \), and \( z \) is from \( P_3 \), the view of \( P_i \) \( i = 1, 2, 3 \) in the protocol \( \pi \) is denoted as \( \text{view}_i^\pi(x, y, z, \kappa) = (w_i, r_i, m_i, \ldots, m_i) \), where \( w \in \{x, y, z\}, r_i \) is the randomness used by \( P_i \), and \( m_i \) is the \( i \)-th message received by \( P_i \); the output of \( P_i \) is denoted as \( \text{out}_i^\pi(x, y, z, \kappa) \), and the joint output of the parties is \( \text{out}^\pi(x, y, z, \kappa) = (\text{out}_1^\pi(x, y, z, \kappa), \text{out}_2^\pi(x, y, z, \kappa), \text{out}_3^\pi(x, y, z, \kappa)) \). We say that \( \pi \) securely computes \( f \) in the semi-honest model if the following holds:

(i) The correctness holds:

\[
\text{out}_i^\pi(x, y, z, \kappa) = f(x, y, z) \] (2)

(ii) There exist probabilistic polynomial-time simulators \( \delta_1, \delta_2, \) and \( \delta_3 \) such that

\[
\begin{align*}
\delta_1 \{1^\kappa, x, f_1(x, y, z)\} &\equiv \text{view}_1^\pi(x, y, z, \kappa), \\
\delta_2 \{1^\kappa, y, f_2(x, y, z)\} &\equiv \text{view}_2^\pi(x, y, z, \kappa), \\
\delta_3 \{1^\kappa, z, f_3(x, y, z)\} &\equiv \text{view}_3^\pi(x, y, z, \kappa).
\end{align*}
\] (3)

2.2.1. OTex Security Model. In OTex, a simpler definition can be used since two of three parties output nothing (see \( \mathcal{F}_{\text{OTex}} \) in Figure 1). Specifically, three parties \( P_1, P_2, P_3 \) stand for server \( S \), sender \( S' \), and receiver \( R \), respectively. The functionality of OTex can be given by simply writing \( f: \{0,1\}^\ast \times \{(x_0, x_1), c\} \mapsto (\lambda, \lambda', c') \), where \( \lambda \) denotes the empty string. We still require correctness described above, and security meaning that there exist three simulators \( \text{Sim}_{S}, \text{Sim}_{S'}, \) and \( \text{Sim}_{R} \) such that

\[
\begin{align*}
\text{Sim}_{S} \{1^\kappa, s, \bot\} &\equiv \text{view}_0^\pi(x, s, c, \kappa), \\
\text{Sim}_{S'} \{1^\kappa, x, \bot\} &\equiv \text{view}_1^\pi(x, s, c, \kappa), \\
\text{Sim}_{R} \{1^\kappa, c, \chi\} &\equiv \text{view}_2^\pi(x, s, c, \kappa),
\end{align*}
\] (4)

where \( x \) denotes input set \( \{x_0, x_1\} \) from \( S' \), \( \chi \) denotes output \( \{x_0, x_1\} \) of \( R \), and \( \bot \) denotes null value.

2.3. OT Extension. We start by introducing the definition of standard 1-out-of-2 OT, where a sender holding two messages \( (m_0, m_1) \) interacts with a receiver holding a choice bit \( b \). The 1-out-of-2 OT protocol guarantees that the receiver obtains \( m_b \) without knowing anything about \( m_{1-b} \), while the sender knows nothing about \( b \). The ideal functionality for 1-out-of-2 OT, denoted as \( \mathcal{F}_{\text{OT}} \), is described in Figure 2.

In most settings, it is necessary to run a large number of OT instances at the same time. The multiexecution of OT is
Let $k$ be security parameter.

Three parties: sender $S$, receiver $R$, server $\mathcal{S}$.

(i) Input:
1. $S$ inputs $m$ pairs of messages $(x_i^0, x_i^1)$, each of length $l$, $1 \leq i \leq m$.
2. $S$ inputs random string $r \in \{0, 1\}^n$.
3. $R$ inputs $m$-bit selection vector $c_1||...||c_m$.

(ii) Output:
1. $S$ outputs nothing.
2. $S$ outputs nothing.
3. $R$ outputs $[x_i^1 | i \in [m]]$ and learns nothing about $[x_i^{1-c_i} | i \in [m]]$.

Figure 1: Outsourced oblivious transfer functionality $F_{OTex}$.

**Functionality $F_{OT}$**

Let $l \in \mathbb{N}$. Two parties: sender $S$ and receiver $R$.

(i) Input:
1. $S$ inputs two messages $x_0, x_1 \in \{0, 1\}^l$.
2. $R$ inputs a selection bit $\sigma \in \{0, 1\}$.

(ii) Output:
1. $S$ has no outputs.
2. $R$ outputs $x_{\sigma}$ and learns nothing about $x_{1-\sigma}$.

Figure 2: 1-out-of-2 oblivious transfer functionality $F_{OT}$.

**Functionality $F_{OT}^m$**

Let $l, m \in \mathbb{N}$. Two parties: sender $S$ and receiver $R$.

(i) Input:
1. $S$ inputs $m$ pairs of messages $\{(x_i^0, x_i^1) | i \in [m], |x_i^0| = |x_i^1| = l\}$.
2. $R$ inputs $m$-bit selection vector $c = c_1||...||c_m$.

(ii) Output:
1. $S$ has no outputs.
2. $R$ outputs $[x_i^c | i \in [m]]$ and learns nothing about $[x_i^{1-c} | i \in [m]]$.

Figure 3: Batch oblivious transfer functionality $F_{OT}^m$.

called batch OT (see Figure 3) denoted as $F_{OT}^m$. The IKNP OT extension protocol [4] is a real milestone in the development of OT research computing $F_{OT}^m$ efficiently. It is trivial to compute $F_{OT}^m$ by simultaneously running $F_{OT}$ $m$ times although it leads to large costs on computation and communication. Therefore, OT extension is the most efficient way for executing $F_{OT}^m$ instead of running $m$ instances of OT in parallel.

As shown in Figure 3, the functionality of IKNP is that $R$ receives messages $\{x_i^c | it \in n[m]\}$ without knowing anything about $\{x_i^{1-c} | it \in n[m]\}$, while $S$ knows nothing about $c_i$, where $c_i$ denotes the $i$-th bit of $c$. In the IKNP protocol, after acting as the receiver in $F_{OT}$ and running it $k$ times, $S$ computes $m$ pairs of symmetric keys denoted as $\{K_i^0, K_i^1\}, \ldots, \{K_m^0, K_m^1\}$, which are used to encrypt each pair of messages $\{(x_i^0, x_i^1) | i \in [m]\}$. For each pair of symmetric keys $(K_0, K_1)$, $R$ only knows the exact one according to his selection string, i.e., $\{K_i^1 | tncq \in hc x 7IC \in [m]\}$. The key insight of IKNP OT extension is to execute such an OT-based key agreement between $S$ and $R$ in the following way:

$R$ forms $m \times k$ matrix $T$ at random and then computes matrix $U$ such that $t_i \oplus u_i = (c_i||...||c_m)$, where $c_i$ is generated by $R$ locally. In addition, due to the randomness of $s$ chosen by $S$, $R$ learns $K_i^{1-s_i} = H(t_i || s)$ with no more than a negligible probability $(1/2^k)$. Then, $S$ can execute just symmetric operations so as to encrypt each pair of messages using key pairs $\{K_i^0, K_i^1\}$ and sends $m$ pairs of encrypted messages to $R$. Finally, $R$ can decrypt the corresponding message for each message pair under $K_i^{1-s_i} = H(t_i || s)$.

As described above, the IKNP protocol begins with running $F_{OT}^k$ $k$ times and then executes lots of symmetric operations to “extend” these OT instances, which are also called as “base” OTs.

**2.4. Oblivious PRF.** Freedman et al. [34] proposed oblivious evaluation of pseudorandom function (OPRF) and gave a general construction of OPRF from OT. An OPRF is a two-party protocol where a sender $P_1$ inputting a random seed $s$ obtains nothing while a receiver $P_2$ inputting an evaluation point $x$ obtains $f_s(x)$ for some pseudorandom function family $f_s$. The functionality of OPRF (see Figure 4) can be defined by $(s, x) \rightarrow (\lambda, f_s(x))$.

A general definition of OPRF is that the receiver $P_2$ inputs an evaluation set $X = \{x_i\}$ and obtains the evaluations on a PRF in an oblivious manner, i.e., $\{f_s(x_i)\}$. The functionality in Figure 4 can be regarded as a special and simplified case although it is sufficient to construct such an
efficient OPRF for some scenarios. In this paper, we consider only the definition where \( receiver\ P_2 \) evaluates the PRF on a single point \( x \), which can be constructed massively in an efficient manner.

### 2.4.1. Batched OPRF Based on OT Extension

Kolesnikov and Kumaresan [5] generalized IKNP and proposed KK protocol realizing \( 1 \rightarrow \text{out} \rightarrow \text{of} \rightarrow 2^l \) OTs in an efficient way. In IKNP, the equation \( T \oplus U \) is either all zeros or all ones. This feature was interpreted as a repetition code in KK, and they improved it using a linear error correcting code denoted by \( \mathbf{C} \). Now for each row in \( T \) and \( U \), it holds that \( T \oplus U = \mathbf{C}(c_i) \), where \( \mathbf{C} \) is a public linear error correcting code of dimension \( l \) and codeword length \( k \). Therefore, in KK, equation (5) becomes

\[
q_i = T_i \oplus \left[ \mathbf{C}(c_i) \circ s \right]. \tag{6}
\]

Notably, \( c_i \) stands for the \( i \)-th element in vector \( c \), and now, it is no longer a binary bit in equation (5) but an \( l \)-bit string. The codeword length of \( \mathbf{C}(c_i) \) determines the length of \( s \) and the number of columns of matrices \( T \) and \( U \), instead in IKNP of \( k \) (\( k \) is relative to a security parameter \( k \)). In KK, to reach the same security requirement, the codeword length \( k \) is about twice as much as \( k \). That is to say, \( k_{\text{KK}} \approx 2.5k_{\text{IKNP}} \). As a consequence, the number of "base" OT in KK is doubled than that in IKNP.

Based on 1-out-of-\( m \) OT extension, Kolesnikov et al. [13] proposed a variant of OPRF and described an efficient protocol to generate batched OPRF instances (known as \( \text{BaRK-OPRF} \)). From the point of adaption of OT extension, the variant OPRF functionality based on equation (4) is that

\[
((q_i, s), c_i) \rightarrow (\lambda, H(i, q_i \oplus [\mathbf{C}(c_i) \circ s])), \tag{7}
\]

where \( \mathbf{C} \) now is a pseudorandom code instead of a linear correcting code.

Notably, the random seed \( s \) consists of \( (q_i, s) \) in \( \text{BaRK-OPRF} \) and \( i \in [m] \), indicating that equation (6) essentially instantiates \( m \) different OPRFs in total, and this is why \( \text{BaRK-OPRF} \) is called batched and key-related OPRF. In addition, the codeword length of pseudorandom code in \( \text{BaRK-OPRF} \) is approximately 2 times longer than the output length of linear error correcting code in KK, which is necessary to reach security requirement. Based on OPRF, it is trivial to construct the private membership test, which will be illustrated later.

### 3. Overview of Our Construction

The functionality of \( \text{OT}^{ex} \) is described in Figure 1. While the essential difference between \( \mathcal{F}^{\text{OTm}} \) in Figure 3 and \( \mathcal{F}^{\text{OTex}} \) in Figure 1 may appear to be unimpressive and unnecessary, the distinction becomes more pronounced in the case of practical OT extension applications and even more so in outsourced OT scenarios.

The codeword length of code schemes in equations (6) and (7), i.e., pseudorandom code and linear error correcting code, determines the number of "base" OT instances to be evaluated. This gives us the intuition that we could use a specific number of OTs to extend any large amount of OT instances we need. Both Ishai et al. [4] and Kolesnikov et al. [5, 13] show \( n \) OTs of long strings that can be reduced to \( k \) "base" OTs of shorter strings. That is, given pseudo-random generator and a small number of OTs, we can implement any \( \mathcal{F}^{\text{OTm}} \) we want. Therefore, we make OT execute in a program-iteration-like way (see Figure 5) to reach the final functionality \( \mathcal{F}^{\text{OText}} \). Here, for clarity, we denote selection string in \( \mathcal{F}^{\text{OText}} \) by bold letter \( r_i \) instead of \( c \) and \( j \)-th element of \( r_i \) is denoted by \( r_i[j] \).

To better understand our work, let us review IKNP OT extension where \( \delta \) holds \( m \) pairs of \( l \)-length messages \( \{x_1^i, x_2^i, \ldots, x_m^i\} \), and the receiver \( R \) holds \( m \)-bit selection vector \( c \). According to Figure 5, we now focus on \( \text{OT}^{(k)}_{x_1^i} \) and \( \text{OT}^{(k)}_{x_2^i} \) only. For \( i \in [m] \), each row of matrices \( T_2 \) and \( T_1 \) consists of \( x_1^i \) and \( x_2^i \), respectively; that is, both \( T_2 \) and \( T_1 \) are \( m \times l \) matrices and \( r_i = c \). Now, \( \text{OT}^{(k)}_{x_1^i} \) serves as "base" OT in IKNP, while the final functionality is to realize \( \text{OT}^{(k)}_{x_2^i} \), where \( k_1 \) is the security parameter, \( k_2 = m \), and \( k_3 = l \). After \( \text{OT}^{(k)}_{x_1^i} \), \( \delta \) computes equation (5) and sends encrypted \( T_2 \) and \( T_1 \) to \( R \). Using symmetric-key operations only, \( R \) receives \( V_2 \) from \( T_2 \) and \( T_1 \) in an oblivious way. For \( i \in [m] \), each row of matrix \( V_2 \) is \( x_1^i \). We have reviewed the whole framework of IKNP protocol so far, and it does matter the relationships among \( T_1 \), \( T_2 \), and \( r_i \). If we write the algorithm implementing \( \text{OT}^{(k)}_{x_1^i} \) as recursive function \( \text{F}_{\text{OT}(k, k_1)} \) in programming language, then the key step in IKNP is that \( \text{F}_{\text{OT}(m)} \) invokes \( \text{F}_{\text{OT}(k,m)} \); i.e., \( \text{OT}^{(k)} \) is reduced to \( \text{OT}^{(m)} \), where \( k \) is the security parameter in IKNP. Thus, we implement \( \text{F}_{\text{OT}(m)} \) by invoking \( \text{F}_{\text{OT}(k,m)} \) first and then executing some symmetric-key encryption operations.

In more general case, two parties \( P_1 \) and \( P_2 \) prepare to run protocol \( \text{OT}^{(k)}_{x_1^i} \), where \( P_1 \) acts as sender \( \delta \) holding messages matrices \( T_1 \) and \( T_2 \) while \( P_2 \) holds chosen vector \( r_i \). After \( \text{OT}^{(k)}_{x_1^i} \), \( P_1 \) gets outputs \( V_i \) consisting of those messages he/she chooses to receive. According to OT extension protocol, \( P_1 \) and \( P_2 \) first run \( \text{OT}^{(k)}_{x_1^i} \), where \( P_1 \) acts as a \( receiver \) and \( P_2 \) acts as a \( sender \). For each \( \text{OT}^{(k)}_{x_1^i} \) and \( \text{OT}^{(k)}_{x_1^i} \), it holds

\[
T_{i-1}[j] \oplus T_{i-1}[j] = f(r_i[j]). \tag{8}
\]

\( T_{i-1}[j] \) denotes the \( j \)-th row of matrix \( T_{i-1}[j] \), \( r_i[j] \) denotes \( j \)-th element of \( r_i \), and \( f(\cdot) \) for some encoding scheme, such as repetition code in equation (5) and linear error correcting code in equation (6). We present detailed...
shows $OT_{\text{ex}}$ construction in a program-iteration-like way, where a small number of $OT$ instances, the innermost $OT^{k_1}_{\text{ex}}$, are first to be executed cooperatively. In fact, we let $k_1 = k$, and it becomes $OT^{k}_{\text{ex}}$, where $k$ is the security parameter in real protocol. From the global perspective, we give procedure of $OT_{\text{ex}}$ in Figure 7 so as to have a better understanding of roles three parties play in every phase.

First, in the “outsourced phase,” $R$ and $S$ run $OT^{k}_{\text{ex}}$ as follows. $R$ inputs random selection string $r$, while $S$ inputs two random $k \times k$ matrices $T$ and $T'$. After $OT^{k}_{\text{ex}}$, $R$ gets outputs $D$ and prepares $V$ (and $V'$) for $OT^{k}$. In the second phase, $\delta$ aims to “respond to the request” for $OT^{k}_{\text{ex}}$. This phase can be concluded as $F_{OT^{k}_{\text{ex}}}$ invoking $F_{OT^{k}}$ and $\delta$ sending pairs of messages $(T, T')$ encrypted by $E$ to $R$. In the last phase, $R$ gets outputs from $OT^{k}$, which can be seen as $F_{OT^{k} (m, l)}$ invoking $F_{OT^{k} (m, l)}$.

We describe $OT_{\text{ex}}$ in Figure 8 which realizes functionality $F_{OT_{\text{ex}}}$ in Figure 4. The invoking procedure in $OT_{\text{ex}}$ can be written as $F_{OT^{k}_{\text{ex}}} \gg F_{OT^{k}_{\text{ex}}} \gg F_{OT^{k} (m, l)}$.

According to equation (8), each time $F_{OT^{k}_{\text{ex}}}$ invokes $F_{OT^{k}}$ and $\delta$ sending pairs of messages $(T, T')$ encrypted by $V$ to $R$. In the last phase, $R$ gets outputs from $OT^{k}$, which can be seen as $F_{OT^{k} (m, l)}$ invoking $F_{OT^{k} (m, l)}$.

4. Outsourced Oblivious Transfer Extension

In this section, we show how to construct an outsourced oblivious transfer extension protocol $OT_{\text{ex}}$, where three-party functionality $F_{OT_{\text{ex}}}$ can be securely computed in the presence of semihonest adversaries.

Our $OT_{\text{ex}}$ protocol consists of two major phases among three parties, $\text{sender} \delta$, $\text{receiver} R$, and $\text{server} S$. Figure 6 shows $OT_{\text{ex}}$ construction in a program-iteration-like way, where a small number of $OT$ instances, the innermost $OT^{k_1}_{\text{ex}}$, are first to be executed cooperatively. In fact, we let $k_1 = k$, and it becomes $OT^{k}_{\text{ex}}$, where $k$ is the security parameter in real protocol. From the global perspective, we give procedure of $OT_{\text{ex}}$ in Figure 7 so as to have a better understanding of roles three parties play in every phase.

First, in the “outsourced phase,” $R$ and $S$ run $OT^{k}_{\text{ex}}$ as follows. $R$ inputs random selection string $r$, while $S$ inputs two random $k \times k$ matrices $T$ and $T'$. After $OT^{k}_{\text{ex}}$, $R$ gets outputs $D$ and prepares $V$ (and $V'$) for $OT^{k}$. In the second phase, $\delta$ aims to “respond to the request” for $OT^{k}_{\text{ex}}$. This phase can be concluded as $F_{OT^{k}_{\text{ex}}}$ invoking $F_{OT^{k}}$ and $\delta$ sending pairs of messages $(T, T')$ encrypted by $E$ to $R$. In the last phase, $R$ gets outputs from $OT^{k}$, which can be seen as $F_{OT^{k} (m, l)}$ invoking $F_{OT^{k} (m, l)}$.

We describe $OT_{\text{ex}}$ in Figure 8 which realizes functionality $F_{OT_{\text{ex}}}$ in Figure 4. The invoking procedure in $OT_{\text{ex}}$ can be written as $F_{OT^{k}_{\text{ex}}} \gg F_{OT^{k}_{\text{ex}}} \gg F_{OT^{k} (m, l)}$.

According to equation (8), each time $F_{OT^{k}_{\text{ex}}}$ invokes $F_{OT^{k}}$ and $\delta$ sending pairs of messages $(T, T')$ encrypted by $V$ to $R$. In the last phase, $R$ gets outputs from $OT^{k}$, which can be seen as $F_{OT^{k} (m, l)}$ invoking $F_{OT^{k} (m, l)}$.

Notably, in more general case, the outputs of $f_1 (r_1 [j])$ and $f_1 (r_{11} [j])$ may be not equal length, which is determined by $OT^{k_1}_{\text{ex}}$ and $OT^{k_0}_{\text{ex}}$ respectively. Figures 7 and 8 illustrate $OT_{\text{ex}}$ framework and procedure, where the symbols are consistent and the output length of repetition code is embodied in the number of columns in the matrices $T$ and $V$.

**Theorem 1.** The $OT_{\text{ex}}$ protocol in Figure 8 securely computes the functionality $F_{OT_{\text{ex}}}$ (Figure 4) in semihonest setting, as described in Definition 1, given random oracle and functionality $F_{OT}$ (Figure 1).

**Proof.** We begin by proving the correctness. After $OT_{\text{ex}}$, we prove that $R$ only receives $x_i^R$ and knows nothing about $x_i^{l \to i}$ when computing $\tilde{x}_i = z_i^r \oplus H (i, v_i)$. In the “outsourced phase,” it holds that

\[ f_1 (s) = s_i \oplus \cdots \oplus s_i, \]
\[ f_2 (c) = c_i \oplus \cdots \oplus c_i. \]
INPUT OF sender $S$: $m$ pairs $(x_i^0, x_i^1)$ of $l$-bit binary string.

INPUT OF server $S'$: random string $r \in \{0, 1\}^s$.

INPUT OF receiver $R$: $m$ selection bit $c = c_1, \ldots, c_m$.

COMMON INPUT: a security parameter $\kappa$.

ORACLE: random oracles $H$: $\{m\} \times \{0, 1\}^s \rightarrow \{0, 1\}^l$, $H_j : \{\kappa\} \times \{0, 1\}^s \rightarrow \{0, 1\}^m$.

CRYPTOGRAPHIC PRIMITIVE: an ideal OT$_{\kappa}$ primitive.

(i) Outsourced phase:

1. $R$ chooses $r \in \{0, 1\}^s$ at random.
2. $S$ computes $\kappa \times \kappa$ matrix $T$ at random and sets matrix $T'$ such that $t_i \oplus t'_i = (s_1, \ldots, s_m)$.
3. For each $j \in \{\kappa\}$, $S$ and $R$ invoke $F_{\text{OT}}$ in the following way:
   - (a) $R$ acts as receiver with input $r_j$.
   - (b) $S$ acts as sender with input $(t'_i, l_j)$.
   - (c) $R$ receives output $d_j$.

$R$ forms $\kappa \times \kappa$ matrix $D$ such that the $j$-th column of $D$ is the vector $d_j$.

(ii) Responding phase:

4. $S$ sends string $s$ and matrix $T$ to $S$.
5. $R$ forms $m \times \kappa$ matrix $V$ at random and sets matrix $T'$ such that $v_j \oplus v'_j = (c_1, \ldots, c_m)$, where $i \in \{m\}$. For $j \in \{\kappa\}$, $R$ computes $y_j^2 = v_j \oplus H_1 (j, d_j)$ and $y_j^3 = v_j \oplus H_1 (j, d_j \oplus r)$ and then sends messages $(y_j^2, y_j^3)$ to $S$.
6. $S$ recovers $m \times \kappa$ matrix $E$ by computing $d_j' = y_j^2 \oplus H_1 (j, t_j)$, $j \in \{\kappa\}$.
   - For $i \in \{m\}$, $S$ computes $z_i^2 = x_i^0 \oplus H (i, c_i)$ and $z_i^1 = x_i^1 \oplus H (i, c_i \oplus s)$ and then sends messages $(z_i^2, z_i^3)$ to $R$.

(iii) Outputting phase:

7. For $i \in \{m\}$, after receiving $(d_i^0, z_i^2)$ to $R$ outputs $\xi_i = x_i^0 \oplus H (i, v_i)$.

FIGURE 8: Outsourced oblivious transfer extension protocol.

\[ d_i = t_i \oplus [r_j : s], \quad d_i = t_i \oplus [s_j : r], \quad \text{where } i, j \in \{\kappa\}. \]  

Then, in the responding phase, Step 5, $R$ essentially computes

\[ y_j^0 = v_j \oplus H_1 (j, t_j \oplus [s_j : r]), \]

\[ y_j^1 = v_j \oplus H_1 (j, t_j \oplus [(s_j \oplus 1) : r]), \quad \text{where } j \in \{\kappa\}. \]  

Therefore, in Step 6, for $j \in \{\kappa\}$, by computing $e_i = y_j^0 \oplus H_1 (j, t_j)$, $R$ gets $e_i = v_i$ when $s_j = 0$ and gets $e_i = v_i^j$ when $s_j = 1$. Due to $v_i \oplus v'_i = (c_1, \ldots, c_i)$, we have $e_i = v_i \oplus [s_j : c]$ and

\[ e_i = v_i \oplus [c_i : s], \quad \text{where } i \in \{m\}, j \in \{\kappa\}. \]

For $\sigma \in \{0, 1\}$, $R$ computes $z_i^\sigma = x_i^\sigma \oplus H (i, e_i \oplus [\sigma : s])$ and essentially sends to $S$ the following messages:

\[ z_i^\sigma = x_i^\sigma \oplus H (i, v_i \oplus [c_i \oplus \sigma : s]). \]  

It holds that $H (i, v_i \oplus [c_i \oplus \sigma : s]) = H (i, v_i)$ if and only if $c_i = \sigma$. Concretely, when $c_i = 0$, $R$ computes
\[
\bar{\chi}_i = \bar{z}_i^0 \oplus H(i, v_i) = x_i^0 \oplus H(i, v_i \oplus [0 \oplus 0] \cdot s) \oplus H(i, v_i) = x_i^0.
\]

(14)

When \(c_i = 1\), \(\mathcal{R}\) computes
\[
\bar{\chi}_i = \bar{z}_i^1 \oplus H(i, v_i) = x_i^1 \oplus H(i, v_i \oplus [1 \oplus 1] \cdot s) \oplus H(i, v_i) = x_i^1.
\]

(15)

In summary, \(\mathcal{R}\) only receives \(x_i^0\) in \(\text{OTex}\) and cannot recover \(x_i^{1-c_i}\) by computing \(\bar{z}_i^{1-c_i} \oplus H(i, v_i)\) because \(\mathcal{R}\) knows nothing about \(H(i, v_i \oplus [0 \oplus 1] \cdot s)\), i.e., \(H(i, v_i \oplus [c_i \oplus (1 - c_i)]) \cdot s\).

This concludes the correctness of \(\text{OTex}\).

We now construct three simulators \(\text{Sim}_S\), \(\text{Sim}_R\), and \(\text{Sim}_{\mathcal{R}}\) for simulating corrupt \(\delta\), \(\mathcal{S}\), and \(\mathcal{R}\) such that the produced transcript and the view of real execution are computationally indistinguishable. That is,
\[
\begin{align*}
\{\text{Sim}_S(1^k, x, \bot)\}_{x \in \text{acc}} & \equiv \{\text{view}_S^C(x, s, c, \kappa)\}_{x \in \text{acc}} , \\
\{\text{Sim}_R(1^k, x, \bot)\}_{x \in \text{acc}} & \equiv \{\text{view}_R^C(x, s, c, \kappa)\}_{x \in \text{acc}} , \\
\{\text{Sim}_{\mathcal{R}}(1^k, c, \bar{x})\}_{x \in \text{acc}} & \equiv \{\text{view}_{\mathcal{R}}^C(x, s, c, \kappa)\}_{x \in \text{acc}} ,
\end{align*}
\]

where \(x\) denotes input set \((x^0, x^1)\) from \(\delta\), \(\bar{x}\) denotes output \([\bar{z}_i]\) of \(\mathcal{R}\), and \(\bot\) denotes the null value.

\textbf{Corrupt} \(S\). Given \(\mathcal{F}_{\text{OTex}}\), it is easy to perfectly simulate the view of the \textit{Server} \(S\) because \(S\) only has input, neither receives messages nor outputs during the execution of the protocol. That is, \(\{\text{Sim}_S(1^k, x, \bot)\}_{x \in \text{acc}}\) is computationally indistinguishable with \(\{\text{view}_S^C(x, s, c, \kappa)\}_{x \in \text{acc}}\).

\textbf{Corrupt} \(\delta\). In protocol, Sender \(\delta\) works only in the responding phase, when \(\delta\) receives messages from \(S\) and \(\mathcal{R}\) then sends encrypted inputs to \(\mathcal{R}\). The messages obtained by \(\delta\) are \(\{s, T_i, (y^0_j, y^1_j)\}\). That is to say, \(\text{view}_S^C(x, s, c, \kappa)\) consists of input \(x\) and messages \(\{s, T_i, (y^0_j, y^1_j)\}\), where \(j \in [k]\). It is trivial for \(\text{Sim}_S\) to generate a simulation of \([s, T]\) since both are random values from the point of view of \(\delta\).

Now, \(\text{Sim}_{\delta}\) simulates a simulation of \(\{(y^0_j, y^1_j) | j \in [k]\}\) by choosing a \(k\)-length string \(s\), \(m\)-length strings \(\bar{T}_j, k \times k\) matrix \(T\), and \(m \times k\) matrix \(E\) at random and computing
\[
\begin{align*}
\bar{y}_j^0 &= \bar{e}^g \oplus H_1(j, \bar{T}_j), \\
\bar{y}_j &= \bar{e}^g \oplus H_1(j, \bar{T}_j).
\end{align*}
\]

(17)

Let \(\{\bar{s}, t, \bar{T}_j, q(y^0_j, y^1_j)\}\) be the output of \(\text{Sim}_{\delta}(1^k, x, \bot)\). Therefore, we claim that the view generated by \(\text{Sim}_{\delta}\) and the view of corrupt \(\delta\) is in a real protocol are computationally indistinguishable.

\textbf{Corrupt} \(\mathcal{R}\). We construct a simulator \(\text{Sim}_{\mathcal{R}}\) that simulates the view of corrupt \(\mathcal{R}\) in the real protocol execution. We first analyze \(\mathcal{R}\)’s view \(\text{view}_{\mathcal{R}}^C(x, s, c, \kappa)\) in \(\text{OTex}\). \(\mathcal{R}\) obtains matrix \(D\) in the outsourced phase and \((\bar{z}_i^0, \bar{z}_i^1)\) in the outputting phase. Therefore, \(\text{view}_{\mathcal{R}}^C(x, s, c, \kappa)\) consists of \(\mathcal{R}\)’s input \(c\) and messages \(\{D, (\bar{z}_i^0, \bar{z}_i^1)\}\). To construct the simulation of the view, \(\text{Sim}_{\mathcal{R}}\) works as follows.

(i) In the outsourced phase, given the security parameter \(\kappa\), \(\mathcal{R}\)’s input \(c\), and the randomness \(r\), \(\text{Sim}_{\mathcal{R}}\) calls simulation \(\text{Sim}_{\text{OTex}}\) with input \((\bar{T}, \bar{T}', r)\) and gets output \(D\). Then, \(\text{Sim}_{\mathcal{R}}\) appends the output of \(\text{Sim}_{\text{OTex}}\), i.e., matrix \(D\), to its own output.

(ii) In the outputting phase, given the security parameter \(\kappa\), \(\mathcal{R}\)’s input \(c\), the randomness \(V\), and output \(x\), \(\text{Sim}_{\mathcal{R}}\) simulates a simulation of \((\bar{z}_i^0, \bar{z}_i^1) | i \in [n(m)]\) by choosing \(m\)-length strings \(\bar{r}_j\) and computing
\[
\begin{align*}
\bar{z}_i^0 &= \bar{z}_i^1 \oplus H(i, v_i), \\
\bar{z}_i^1 &= \bar{z}_i^1 \oplus H(i, v_i).
\end{align*}
\]

(18)

Then, \(\text{Sim}_{\mathcal{R}}\) appends \(\{(\bar{z}_i^0, \bar{z}_i^1) | i \in [n(m)]\}\) to its output.

Combining two phases described above in sequence, we finally claim that the output of \(\text{Sim}_{\mathcal{R}}\) is computationally indistinguishable from the real execution, i.e., \(\{\text{Sim}_{\mathcal{R}}(1^k, c, \bar{x})\}_{x \in \text{acc}} \equiv \{\text{view}_{\mathcal{R}}^C(x, s, c, \kappa)\}_{x \in \text{acc}}\).

This completes the construction of three simulators: \(\text{Sim}_S, \text{Sim}_{\mathcal{R}}, \text{Sim}_{\mathcal{R}}\). In summary, \(\text{OTex}\) is secure under semihonest model.

\[\square\]

4.1. Performance Analysis. We now analyze the performance of \(\text{OTex}\) in semihonest model. The complexities of \(\text{OTex}\) are presented in Table 1.

(i) \textbf{Computation complexity}. The number of symmetric and asymmetric operation to be executed in \(\text{OTex}\) essentially depends on the size of matrices \(T\), \(V\), and \(E\), respectively. In the outsourced phase, \(\mathcal{F}_{\text{OTex}}\) consists of \(O(\kappa)\) public-key and \(O(\kappa)\) private-key operations, for both \(S\) (acting as sender) and \(\mathcal{R}\) (acting as receiver). In the responding phase, \(\mathcal{R}\) invokes random oracle and performs XOR operation for \(O(\kappa)\) times. Then, \(\delta\) recovers matrix \(E\) and encrypts messages \((\bar{z}_i^0, \bar{z}_i^1)\), which only consists of \(O(m + k^2)\) symmetric operations. In the outputting phase, \(\mathcal{R}\) performs symmetric operations \(O(m)\) times to compute its outputs.

(ii) \textbf{Communication complexity}. In the outsourced phase, the number of bits transferred between \(S\) and \(\mathcal{R}\) is bounded by \(O(\kappa^2)\). In the responding phase, \(\delta\) receives messages from \(S\) and \(\mathcal{R}\) and sends encrypted messages to \(\mathcal{R}\), which consist of \(O(ml + \kappa^2)\). In the outputting phase, \(\mathcal{R}\) receives \(2ml\)-bit messages in total from \(\delta\).

(iii) \textbf{Round complexity}. The only interaction in \(\text{OTex}\) exists in the outsourced phase between \(S\) and \(\mathcal{R}\), i.e., \(\mathcal{F}_{\text{OTex}}\). The number of interaction round in \(\mathcal{F}_{\text{OTex}}\) is bounded by 1.
4.2. Efficiency Comparison. Since we focus on the efficiency of the sender \( \delta \) in OT, we provide comparisons to prior classical protocols from the \( \delta \)'s point of view in Table 2. In OTex, all asymmetric operations are operated between the receiver \( \mathcal{R} \) and the cloud server \( \mathcal{S} \), and \( \delta \) works on the fly and conducts symmetric computations locally.

5. Performance

In this section, we test the performance of OTex. The experiments were performed on a Linux machine equipped with 4 cores 3.40 GHz Intel Core i5-7500 CPU and 8 GB RAM.

Our tests refer to the implementation on GitHub: https://github.com/emp-toolkit/emp-ot. We simulated main asymmetric operations in OTex on a single machine. We compared OTex with the OT protocol used in recent work [32], where they compute at least 450 OT instances locally from 128 “base” OTs. Therefore, we regarded \( n = 2^8 \) as a reference and simulated the outsourced phase. After setting the length of security parameter equal to 128, we simulated OTex for \( n = 2^7 \), \( 2^9 \), and \( 2^{10} \), respectively. We repeated each simulation 20 times, and the result is shown in Figure 9. Although OTex shows an almost identical performance to Vladimir’s, the running time tends to be steady sooner when \( n = 2^9 \). In addition, in Figure 9, the sender \( \delta \) of OTex works offline for most of the time during simulation, and this is one of the advantages of OTex.

The other advantage of OTex found in performance tests is the reduction of communication bottleneck caused by the sender \( \delta \). The main idea of OT extension protocol is to extend a small number of base OTs to perform many OTs. As a consequence, we summed from the time of base OT on setup to the time of protocol finish and then computed average time for each original OT.

The averaged time is illustrated in Figure 10. We tested \( n' = 2^{8x} \) OT instances for different \( x \) values and computed the average running time for one single OT, i.e., \( OT^1 \). From Figure 10, we can see that the average time keeps steady when \( x \) is less than 8. If we extend base OT to an adequate number of OTs, for example, \( n' = 2^{18} \), the average time has a sharp increase. It does not matter in OTex, however, since the extension process is conducted between \( \mathcal{R} \) and \( \mathcal{S} \) in the outsourced phase.

6. Applications of OTex

The OTex framework has a wide range of applications in outsourced scenarios. In this section, we take private set intersection (PSI) as a case study and demonstrate that, as a

---

**Table 1: Complexity of OTex.**

| Party    | Asymmetric | Symmetric | Communication | Round |
|----------|------------|-----------|---------------|-------|
| Server \( \mathcal{S} \) | \( O(k) \) | \( O(k) \) | \( O(k^2) \) | 1     |
| Sender \( \delta \) | \( O(m + k) \) | \( O(ml + k^2) \) | \( - \) | -     |
| Receiver \( \mathcal{R} \) | \( O(k) \) | \( O(m + k) \) | \( O(ml + k^2) \) | 1     |

**Table 2: Efficiency comparison.**

| Protocol | Round | Communication | Asymmetric computation | Security model                     |
|----------|-------|---------------|------------------------|-----------------------------------|
| [4]      | 2     | \( O(ml + mk) \) | \( O(k) \) | Semihonest and malicious          |
| [5]      | 2     | \( O((mk/log n) * ln log n) \) | \( O(k) \) | Semihonest                        |
| [16]     | 2     | \( O(ml + k^2) \) | \( O(k + ln) \) | Semihonest and one-sided malicious|
| [30]     | 2     | \( O(ml + mk) \) | \( O(k) \) | Semihonest                        |
| [31]     | 3     | \( O(ml + log[G] + k) \) | \( O(k) \) | Malicious                         |
| Ours     |       | \( O(ml + k^2) \) | \( - \) | Semihonest                        |

Note: the efficiency of sender in OT is presented here, who inputs \( m \) pairs of \( l \)-bit length messages in protocol. Specially, \( k \) is security parameter, \( G \) denotes a group, and \( k = 2.5k \) in 1-out-of-\( n \) OT.
building block, OText can be applied conveniently to high-level protocol.

We first introduce a private membership test (PMT) protocol to estimate whether an element \( x \) belongs to a set \( Y = \{y_1, \ldots, y_m\} \) or not and then describe how to efficiently implement it via OText. The resulting protocol can then be simply extended to compute functionality \( \mathcal{F}_{\psi} \) in Figure 11 by applying the OText-based PMT protocol.

### 6.1. OPRF-Based PMT Protocol

PMT protocol involves two parties: Alice who holds an element \( x \) and Bob who holds a set \( Y \). After PMT, Bob knows nothing about \( x \), and Alice only knows whether her element \( x \) belongs to the set \( Y \) or not.

Based on OPRF, we can construct PMT protocol as follows. First, during the OPRF phase, Bob acts as sender with a random seed \( s \). And Alice, acting as receiver, inputs her element \( x \). After OPRF, Alice obtains \( f_s(x) \) and Bob has \( f_s(\cdot) \). Then, Bob sends all \( \{f_s(y_i)\} \) to Alice who compares \( f_s(x) \) with \( \{f_s(y_i)\} \) one by one. In conclusion, Alice outputs 1 if and only if there exits an \( i \) such that \( f_s(y_i) = f_s(x) \) and else outputs 0.

The security of PMT via OPRF relies on the fact that an OPRF protocol is secure. In a secure OPRF, it guarantees that \( f_s(\cdot) \) is a one-way pseudorandom function and the length of \( s \) equals to secure parameter. Therefore, Alice receives \( \{f_s(y_i)\} \) and then learns whether there exists an \( i \) such that \( f_s(y_i) = f_s(x) \), but she will not learn the exact values \( \{y_i | y_i \in n \forall q, h f_s(x, y_i, 7) \in \mathbb{C}: f_s(x)\} \) with non-negligible probability. We omit concrete security proof here.

### 6.2. OText-Based PMT Protocol

Given functionality \( \mathcal{F}_{\text{OPRF}} \), we can construct PMT protocol in a simple manner. In this section, we introduce how to design OPRF protocol based on OText. Equation (7) indicates that 1-out-of-\( n \) OT extension implies \( \text{BarK-OPRF} \); therefore, we focus on the transformation from 1-out-of-\( n \) OT extension to OText. As a result, OText can be easily used to construct PMT protocol.

We can apply pseudorandom code to equation (8) for defining new relationship among \( T_2 \) and \( T_2' \) in Figure 6, which implements functionality \( \mathcal{F}_{\text{OPRF}} \) in Figure 3. In OText, however, we stress that it always holds \( T_1[i] \oplus T_1'[i] = f_1(r_2[i]) \) and \( T_2[j] \oplus T_2'[j] = f_1(r_1[j]) \), where \( f_1(\cdot) \) is the repetition code. This means that we implement INKP protocol in an outsourced manner since we use the same repetition code \( f_1(\cdot) \) in \( F_{\text{OPRF}}(k_1,k_2) \) and \( F_{\text{OPRF}}(k_2,k_3) \). That is to say, three parties, \( S, \delta, \) and \( \delta' \), evaluate corporately \( m \) instances of 1-out-of-2 OT of \( l \)-bit messages where repetition code is used twice in total, each by \( S \) and \( \delta \), respectively. Now, suppose Alice and Bob would like to execute \( m \) instances of 1-out-of-\( n \) OT where Bob inputs \( \{(y'_1, y'_2, \ldots, y'_{l}) \in [m], y'_i = 1\} \) and Alice inputs her selection vector \( r_x \) and they need make the following adjustments:

(i) Bob organizes the inputs in OText with \( m \times l \) matrices \( T_1^{\{1\}}, T_1^{\{2\}}, \ldots, T_1^{\{m\}} \) where the \( j \)-th row of matrix \( T_1^{\{s\}} \) is \( y'_j \). (ii) Alice makes her selection vector in OText be \( \{ r_x \}_{i=1} \cap n [m] \).

Then, the only adaption they need take in OText is that \( T_2[j] \oplus T_2'[j] = f_2(r_1[j]) \), where \( f_2(\cdot) \) is the linear error correcting code \( \mathcal{E} \). To reach the security requirement defined in [13], in OText, we need to reset security parameter \( k \) rather than \( k \)-concretely \( 3k < k < 4k \).

Given 1-out-of-\( n \) OT protocol designed from OText, it is trivial to design OText-based OPRF so is to OText-based PMT. Again, Alice prepares her element \( x = r_2[1] \), while Bob holds set \( Y = \{y'_1, y'_2, \ldots, y'_m\} \). In brief, if it holds in OText that

\[
T_1[i] \oplus T_1'[i] = f_1(r_2[i]),
\]

\[
T_2[j] \oplus T_2'[j] = \mathcal{E}(r_1[j]),
\]

where \( f_1(\cdot) \) is the repetition code and \( \mathcal{E} \) is the pseudorandom code; OText implies the functionality \( \mathcal{F}_{\text{OPRF}} \) where the seed \( s \) consists of \((V_2, r_2)\) generated by Bob and Alice gets \( f_s(x) \). Meanwhile, this essentially completes the main construction of OText-based PMT protocol. All needed to do next is that Bob sends \( \{ f_s(y'_i) \}_{i=1} \) to Alice who compares \( f_s(x) \) with \( \{ f_s(y'_i) \}_{i=1} \) locally.

### 6.3. OText-Based PSI Protocol

To obtain the final PSI protocol that computes \( X \cap Y \), Alice simply invokes the PMT protocol for each \( x_i \in X \). The protocol in Figure 12 computes \( X \cap Y \) in an outsourced manner. The intuition is that
Alice holds a set $X$, while Bob holds $Y$, and after PMT, Alice knows whether her element $x_i$ belongs to $Y$ or not, i.e., the intersection of two sets.

Some details on method of preprocessing items in set are simply omitted in Figure 12. We can hash the item to bins and then operate the OTex-based PSI protocol on each bin separately. Specifically, we use Cuckoo hashing [35] for PSI in the following way. First, Alice and Bob agree on 3 random hash functions $h_1, h_2, h_3 : [0, 1]^* \rightarrow [m^*]$ suitable for 3-way Cuckoo hashing. Alice places her items into $m$ in either $h_1(x)$, $h_2(x)$, or $h_3(x)$, and each bin contains at most one item. Bob places each of his items $y$ in locations $h_1(y)$, $h_2(y)$, and $h_3(y)$. At this point, Alice pads her input with dummy items so that each bin contains exactly 1 item. Note that when we use cuckoo hashing, then there will be some items which cannot be placed into the table and have to be moved to a stash, which does not matter because of the usage of stash-less cuckoo hashing [20]. Finally, Alice and Bob perform a PSI in each bin.

7. Conclusions and Future Work

In this paper, we proposed a generic outsourced OT extension protocol (OTex) which can outsource all the “base” OT from the sender to a semihonest server. The proposed protocol realizes optimal computational and communication complexity relative to security parameter. In addition, we showed that OTex can be efficiently used in private membership test and private set intersection. In the future, we will consider malicious model and construct efficient outsourced OT extension protocols secure against malicious adversary.

Data Availability

The performance test data used to support the findings of this study are included within the article.

Conlicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant nos. 61572294 and 61632020), Major Innovation Project of Science and Technology of Shandong Province (Grant no. 2018CXGC0702), Natural Science Foundation of Shandong Province (Grant no. ZR2017MF021), and Fundamental Research Funds of Shandong University (Grant no. 2017JC019).

References

[1] C.-C. Y. Andrew, “How to generate and exchange secrets,” in Proceedings of the 27th Annual Symposium on Foundations of Computer Science (SFC 1986), pp. 162–167, Washington, DC, USA, October 1986.
[2] O. Goldreich, S. Micali, and A. Wigderson, “How to play any mental game, or a completeness theorem for protocols with honest majority,” in Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali, pp. 307–328, Weizmann Institute of Science, Rehovot, Israel, 2019.
[3] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, “Multiparty computation from somewhat homomorphic encryption,” in Proceedings of the Annual Cryptology Conference, pp. 643–662, Santa Barbara, CA, USA, August 2012.
[4] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, “Extending oblivious transfers efficiently,” in Proceedings of the Annual International Cryptology Conference, pp. 145–161, Santa Barbara, CA, USA, August 2003.
[5] V. Kolesnikov and R. Kumaresan, “Improved ot extension for transferring short secrets,” in Proceedings of the Annual Cryptology Conference, pp. 54–70, Santa Barbara, CA, USA, 2013.
[6] S. Even, O. Goldreich, and A. Lempel, “A randomized protocol for signing contracts,” Communications of the ACM, vol. 28, no. 6, pp. 637–647, 1985.

[7] G. Brassard, C. Claude, and J.-M. Robert, “All-or-nothing disclosure of secrets,” in Proceedings of the Conference on the Theory and Application of Cryptographic Techniques, pp. 234–238, Santa Barbara, CA, USA, 1986.

[8] W.-G. Zeng, “Efficient 1-out-n oblivious transfer schemes,” in Proceedings of the International Workshop on Public Key Cryptography, pp. 159–171, Paris, France, February 2002.

[9] Y. Mu, J. Zhang, and V. Varadharajan, “in out of n oblivious transfer,” in Proceedings of the Australasian Conference on Information Security and Privacy, pp. 395–405, Melbourne, Australia, 2002.

[10] C.-K. Chu and W.-G. Zeng, “Efficient k-out-n oblivious transfer schemes with adaptive and non-adaptive queries,” in Proceedings of the International Workshop on Public Key Cryptography, pp. 172–183, New York, NY, USA, March 2005.

[11] Y. Lindell, K. Nissim, and C. Orlandi, “Hiding the input-size in secure two-party computation,” in Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, pp. 421–440, Bengaluru, India, December 2013.

[12] C. Cho, N. Döttling, S. Garg, D. Gupta, P. Miao, and A. Polychnioudou, “Laconic oblivious transfer and its applications,” in Proceedings of the Annual International Cryptology Conference, pp. 33–65, Santa Barbara, CA, USA, 2017.

[13] V. Kolesnikov, R. Kumaresan, M. Rosulek, and T. Ni, “Efficient batched oblivious PRF with applications to private set intersection,” in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 818–829, Vienna, Austria, October 2016.

[14] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and T. Ni, “Practical multi-party private set intersection from symmetric-key techniques,” IACR Cryptology ePrint Archive, vol. 799, p. 2017, 2017.

[15] B. Pinkas, T. Schneider, and M. Zohner, “Scalable private set intersection based on OT extension,” ACM Transactions on Privacy and Security, vol. 21, no. 2, p. 7, 2018.

[16] B. Pinkas, M. Rosulek, T. Ni, and A. Yanai, “Spot-light: lightweight private set intersection from sparse extension,” in Proceedings of the Annual International Conference, pp. 401–431, Santa Barbara, CA, USA, August 2019.

[17] M. Chase and P. Miao, “Private set intersection in the internet setting from lightweight oblivious PRF,” in Proceedings of the Annual International Cryptology Conference, pp. 34–63, Santa Barbara, CA, USA, August 2020.

[18] B. Pinkas, T. Schneider, C. Weinert, and U. Wieder, “Efficient circuit-based PSI via cuckoo hashing,” in Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 125–157, Darmstadt, Germany, May 2018.

[19] M. Ciampi and C. Orlandi, “Combining private set intersection with secure two-party computation,” in Proceedings of the International Conference on Security and Cryptography for Networks, pp. 464–482, Amalfi, Italy, September 2018.

[20] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai, “Efficient circuit-based PSI with linear communication,” in Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 122–153, Darmstadt, Germany, May 2019.

[21] T. Duong, D. H. Phan, and T. Ni, Catalic: Delegated PSI Cardinality with Applications to Contact Tracing, IACR Cryptology ePrint Archive, vol. 1105, p. 2020, 2020.

[22] M. O. Rabin, “How to exchange secrets with oblivious transfer,” IACR Cryptology ePrint Archive, vol. 187, p. 2005, 2005.

[23] A. Y. Lindell, “Efficient fully-simulatable oblivious transfer,” in Proceedings of the Cryptographers’ Track at the RSA Conference, pp. 52–70, San Francisco, CA, USA, April 2008.

[24] B. Zeng, C. Tartary, P. Xu, J. Jing, and X. Tang, “A Practical Framework for t-out-of-n oblivious transfer with security against covert adversaries,” IEEE Transactions on Information Forensics and Security, vol. 7, no. 2, pp. 465–479, 2012.

[25] M. Green and S. Hohenberger, “Universally composable adaptive oblivious transfer,” in Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, pp. 179–197, Melbourne, Australia, December 2008.

[26] D. Beaver, “Precomputing oblivious transfer,” in Proceedings of the Annual International Cryptology Conference, pp. 97–109, Santa Barbara, CA, USA, August 1995.

[27] D. Beaver, “Correlated pseudorandomness and the complexity of private computations,” in Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, pp. 479–488, Philadelphia, PA, USA, May 1996.

[28] Y. Gertner, S. Kannan, T. Malkin, O. Reingold, and M. Viswanathan, “The relationship between public key encryption and oblivious transfer,” in Proceedings 41st Annual Symposium on Foundations of Computer Science, pp. 325–335, Redondo Beach, CA, USA, November 2000.

[29] Y. Gertner, T. Malkin, and O. Reingold, “On the impossibility of basing trapdoor functions on trapdoor predicates,” in Proceedings 42nd IEEE Symposium on Foundations of Computer Science, pp. 126–135, Heraklion, Greece, 2001.

[30] H. Carter, B. Mood, P. Traynor, and K. Butler, “Secure outsourced garbled circuit evaluation for mobile devices,” Journal of Computer Security, vol. 24, no. 2, pp. 137–180, 2016.

[31] D. Mansy and P. Rindal, “Endemic oblivious transfer,” in Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 309–326, London, UK, November 2019.

[32] V. Kolesnikov, M. Rosulek, T. Ni, and X. Wang, “Scalable private set union from symmetric-key techniques,” in Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, pp. 636–666, Kobe, Japan, December 2019.

[33] O. Goldreich, Foundations of Cryptography: Volume 2, Basic Applications, Cambridge University Press, Cambridge, UK, 2009.

[34] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold, “Keyword search and oblivious pseudorandom functions,” in Proceedings of the Theory of Cryptography Conference, pp. 303–324, Cambridge, MA, USA, February 2005.

[35] R. Pagh and F. F. Rodler, “Cuckoo hashing,” Journal of Algorithms, vol. 51, no. 2, pp. 122–144, 2004.