Hindawi
Security and Communication Networks
Volume 2020, Article ID 8847487, 12 pages
https://doi.org/10.1155/2020/8847487

Research Article

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 Xu**[1]

[1]*School of Software, Shandong University, Jinan 250100, China*
[2]*School 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

Academic Editor: Zhihua Xia

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 ($\mathbb{O}Tex$) that outsources all the asymmetric operations of the sender to a semihonest server so as to adapt to specific scenarios above. We give $\mathbb{O}Tex$ a standard security definition, and the proposed protocol is proven secure in the semihonest model. In $\mathbb{O}Tex$, 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, $\mathbb{O}Tex$ can be used to construct high-level protocols, such as private membership test (PMT) and private set intersection (PSI). We believe our $\mathbb{O}Tex$ 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 $\mathcal{S}$ needs to interact with the receiver $\mathcal{R}$ for each step during the protocol, which involves exponential calculation and intensive interaction. In some application scenarios, however, $\mathcal{S}$ 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, $\mathcal{S}$ needs to respond requests from $\mathcal{R}$ as fast as possible.

Nowadays, many applications are rapidly transferred to cloud-based service, and it would be desired to seek some server-aided OT extension protocol to relief the burden of $\mathcal{S}$ 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 $\mathcal{R}$. 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 ($\mathbb{O}Tex$) in the semihonest model. In $\mathbb{O}Tex$, the sender $\mathcal{S}$ first outsources all expensive asymmetric operations to a third party who runs a subprotocol called "base" OT instances with the receiver $\mathcal{R}$. Based on the corresponding outputs of the subprotocol and other auxiliary information, $\mathcal{S}$ generates symmetric keys used to encrypt sending messages in OT. As a result, the sender $\mathcal{S}$ works on the fly and sends its inputs encrypted by symmetric key generated from $\mathbb{O}Tex$ to the receiver $\mathcal{R}$, 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, $\mathbb{O}Tex$ framework has a wide range of applications in outsourced scenarios, and as a building block, we think $\mathbb{O}Tex$ 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 security 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 ($\mathbb{O}Tex$). In $\mathbb{O}Tex$, 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 corelated 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 $\mathbb{O}Tex$ 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_l^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, $\mathbf{t}_i$ denotes the i-th row of a matrix T, and $\mathbf{t}^j$ denotes the j-th column of T. Small light front with subscript s_i denotes the i-th bit of a string $\mathbf{s}$. 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 $s \oplus t$ 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 $c \in \{0,1\}$, $c \cdot \mathbf{s}$ 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: $\{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^*$, 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 $X = \{X(a,n)\}_{a \in \{0,1\}^*; n \in \mathbb{N}}$ and $Y = \{Y(a,n)\}_{a \in \{0,1\}^*; n \in \mathbb{N}}$ are said to be *computationally indistinguishable*, denoted by $X^c \equiv Y$, if for every nonuniform polynomial-time algorithm D, there exists a negligible function $\mu(\cdot)$ such that for every $a \in \{0,1\}^*$ and every $n \in \mathbb{N}$:

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

Semihonest Adversary Model. In the semihonest 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. Semihonest model is sufficient and captures many scenarios in practice although it is a very weak adversary model.

In this paper, we focus on semihonest 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 π be a three-party protocol for computing f. Given the security parameter κ 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 π is denoted as $view_i^\pi(x, y, z, \kappa) = (w, r_i, m_i^1, \ldots, m_i^t)$, where $w \in \{x, y, z\}$, r_i is the randomness used by P_i, and m_i^j is the j-th message received by P_i; the output of P_i is denoted as $output_i^\pi(x, y, z, \kappa)$, and the joint output of the parties is $output^\pi(x, y, z, \kappa) = (output_1^\pi(x, y, z, \kappa), output_2^\pi(x, y, z, \kappa), output_3^\pi(x, y, z, \kappa))$. We say that π securely computes f in the semihonest model if the following holds:

(i) The correctness holds:

$$\text{output}^\pi(x, y, z, \kappa)^c \equiv \{f(x, y, z)\}_{x,y,z,\kappa}. \tag{2}$$

(ii) There exist probabilistic polynomial-time simulators $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$ such that

$$\begin{aligned} \{\mathcal{S}_1(1^\kappa, x, f_1(x, y, z))\}_{x,y,z,\kappa}{}^c &\equiv \{view_1^\pi(x, y, z, \kappa)\}_{x,y,z,\kappa}, \\ \{\mathcal{S}_2(1^\kappa, y, f_2(x, y, z))\}_{x,y,z,\kappa}{}^c &\equiv \{view_2^\pi(x, y, z, \kappa)\}_{x,y,z,\kappa}, \\ \{\mathcal{S}_3(1^\kappa, z, f_3(x, y, z))\}_{x,y,z,\kappa}{}^c &\equiv \{view_3^\pi(x, y, z, \kappa)\}_{x,y,z,\kappa}. \end{aligned} \tag{3}$$

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

$$\begin{aligned} \{\text{Sim}_{\mathbb{S}}(1^\kappa, \mathbf{s}, \perp)\}_{\mathbf{x},\mathbf{s},\mathbf{c},\kappa}{}^c &\equiv \{\text{view}_{\mathbb{S}}^\pi(\mathbf{x}, \mathbf{s}, c, \kappa)\}_{\mathbf{x},\mathbf{s},\mathbf{c},\kappa}, \\ \{\text{Sim}_{\mathcal{S}}(1^\kappa, \mathbf{x}, \perp)\}_{\mathbf{x},\mathbf{s},\mathbf{c},\kappa}{}^c &\equiv \{\text{view}_{\mathcal{S}}^\pi(x, \mathbf{s}, \mathbf{c}, \kappa)\}_{\mathbf{x},\mathbf{s},\mathbf{c},\kappa}, \\ \{\text{Sim}_{\mathcal{R}}(1^\kappa, \mathbf{c}, \tilde{x})\}_{\mathbf{x},\mathbf{s},\mathbf{c},\kappa}{}^c &\equiv \{\text{view}_{\mathcal{R}}^\pi(\mathbf{x}, \mathbf{s}, \mathbf{c}, \kappa)\}_{\mathbf{x},\mathbf{s},\mathbf{c},\kappa}, \end{aligned} \tag{4}$$

where $\mathbf{x}$ denotes input set $\{(x_i^0, x_i^1)\}$ from $\mathcal{S}$, $\tilde{x}$ denotes output $\{x_i^{c_i}\}$ of $\mathcal{R}$, and $\perp$ 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

Functionality $F_{\mathbb{O}\mathrm{Tex}}$

Let k be security parameter.
Three parties: sender S, receiver R, server $\mathbb{S}$.

(i) Input:
  (1) S inputs m pairs of messages (x_i^0, x_i^1), each of length l, $1 \le i \le m$.
  (2) $\mathbb{S}$ inputs random string $r \leftarrow \{0,1\}^{\kappa}$.
  (3) R inputs m-bit selection vector $c = c_1 || \ldots || c_m$.
(ii) Output:
  (1) S outputs nothing.
  (2) $\mathbb{S}$ outputs nothing.
  (3) R outputs $\{x_i^{c_i} \mid i \in [m]\}$ and learns nothing about $\{x_i^{1-c_i} \mid i \in [m]\}$.

Figure 1: Outsourced oblivious transfer functionality $\mathcal{F}_{\mathbb{O}Tex}$.

Functionality F_{OT}

Let $l \in$ 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_σ and learns nothing about $x_{1-\sigma}$.

Figure 2: 1-out-of-2 oblivious transfer functionality $\mathcal{F}_{\mathrm{OT}}$.

Functionality $F_{\mathrm{OT}_l^m}$

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

(i) Input:
  (1) S inputs m pairs of messages $\{(x_i^0, x_i^1) \mid i \in [m], |x_i^0| = |x_i^1| = l\}$.
  (2) R inputs m-bit selection string $c = c_1 || \ldots || c_m$.
(ii) Output:
  (1) S has no outputs.
  (2) R inputs $\{x_i^{c_i} \mid i \in [m]\}$ and learns nothing about $\{x_i^{1-c_i} \mid i \in [m]\}$.

Figure 3: Batch oblivious transfer functionality $\mathcal{F}_{\mathrm{OT}_l^m}$.

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

As shown in Figure 3, the functionality of IKNP is that $\mathcal{R}$ receives messages $\{x_i^{c_i} | it \in n[m]\}$ without knowing anything about $\{x_i^{1-c_i} | it \in n[m]\}$, while $\mathcal{S}$ knows nothing about $\mathbf{c}$, where c_i denotes the i-th bit of $\mathbf{c}$. In the IKNP protocol, after acting as the *receiver* in $\mathcal{F}_{\mathrm{OT}}$ and running it k times, $\mathcal{S}$ computes m pairs of symmetric keys denoted as $\{(\mathcal{K}_1^0, \mathcal{K}_1^1), \ldots, (\mathcal{K}_m^0, \mathcal{K}_m^1)\}$, which are used to encrypt each pair of messages $\{(x_i^0, x_i^1)\}$, where $i \in [m]$. For each pair of symmetric keys $(\mathcal{K}_i^0, \mathcal{K}_i^1)$, $\mathcal{R}$ only knows the exact one according to his selection string, *i.e.*, $\{\mathcal{K}_i^{c_i} | t\, nc_i q \in hc, x\, 7iC \in ; [m]\}$. The key insight of IKNP OT extension is to execute such an OT-based key agreement between $\mathcal{S}$ and $\mathcal{R}$ in the following way:

$\mathcal{R}$ forms $m \times k$ matrix T at random and then computes matrix U such that $\mathbf{t}_i \oplus \mathbf{u}_i = (c_i \| \cdots \| c_i)$. $\mathcal{S}$ chooses $\mathbf{s} \leftarrow \{0,1\}^k$ at random. For each $j \in k$, $\mathcal{R}$ invokes $\mathcal{F}_{\mathrm{OT}}$ with input $(\mathbf{t}^j, \mathbf{u}^j)$, and $\mathcal{S}$ acts as the *receiver* with selection bit s_j. $\mathcal{S}$ receives $\mathbf{q}^j$ after each computation of $\mathcal{F}_{\mathrm{OT}}$ and then forms matrix Q. For each column of matrix Q, it implies $\mathbf{q}^j = [(s_j \oplus 1) \cdot \mathbf{t}^j] \oplus (s_j \cdot \mathbf{u}^j) = \mathbf{t}^j \oplus (s_j \cdot \mathbf{c})$. The essential observation is that, for each row of matrix Q, it holds

$$\mathbf{q}_i = \mathbf{t}_i \oplus (c_i \cdot \mathbf{s}). \tag{5}$$

$\mathcal{S}$ prepares key pairs by $\mathcal{K}_i^0 = H(\mathbf{q}_i)$ and $\mathcal{K}_i^1 = H(\mathbf{q}_i \oplus \mathbf{s})$ and $\mathcal{R}$ exactly knows $\mathcal{K}_i^{c_i} = H(\mathbf{t}_i)$, where $\mathbf{t}_i$ is generated by $\mathcal{R}$ locally. In addition, due to the randomness of $\mathbf{s}$ chosen by $\mathcal{S}$, $\mathcal{R}$ learns $\mathcal{K}_i^{1-c_i} = H(\mathbf{t}_i \oplus \mathbf{s})$ with no more than a negligible probability $(1/2^k)$. Then, $\mathcal{S}$ can execute just symmetric operations so as to encrypt each pair of messages using key pairs $(\mathcal{K}_i^0, \mathcal{K}_i^1)$ and sends m pairs of encrypted messages to $\mathcal{R}$. Finally, $\mathcal{R}$ can decrypt the corresponding message for each message pairs under $\mathcal{K}_i^{c_i} = H(\mathbf{t}_i)$.

As described above, the IKNP protocol begins with running $\mathcal{F}_{\mathrm{OT}}$ 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 $(s, x) \mapsto (\lambda, f_s(x))$.

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

Functionality F_{OPRF}

Let f be a pseudorandom function. Two parties: *sender* S and *receiver* R.

(i) Input:
  (1) S inputs a random seed s.
  (2) R inputs a value r.

(ii) Output:
  (1) S has no outputs.
  (2) R outputs $f_s(r)$.

Figure 4: Oblivious PRF functionality $\mathcal{F}_{\text{OPRF}}$.

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 $1-\text{out}-\text{of}-2^l$ OTs in an efficient way. In IKNP, the equation $\mathbf{t}_i \oplus \mathbf{u}_i = (c_i \| \cdots \| c_i)$ means that each row of matrix $T \oplus U$ 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 $\mathscr{C}$. Now for each row in T and U, it holds that $\mathbf{t}_i \oplus \mathbf{u}_i = \mathscr{C}(c_i)$, where $\mathscr{C}$ is a public linear error correcting code of dimension l and codeword length k. Therefore, in KK, equation (5) becomes

$$\mathbf{q}_i = \mathbf{t}_i \oplus [\mathscr{C}(c_i) \odot \mathbf{s}]. \tag{6}$$

Notably, c_i stands for the i-th element in vector $\mathbf{c}$, and now, it is **no longer** a binary bit in equation (5) but an l-bit string. The codeword length of $\mathscr{C}(c_i)$ determines the length of $\mathbf{s}$ and the number of columns of matrices T and U, instead in IKNP of k (k is relative to a security parameter κ). 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-n OT extension, Kolesnikov et al. [13] proposed a variant of OPRF and described an efficient protocol to generate batched OPRF instances (known as *BaRK-OPRF*). From the point of adaption of OT extension, the variant OPRF functionality based on equation (6) is that

$$((\mathbf{q}_i, \mathbf{s}), c_i) \longmapsto (\lambda, H(i, \mathbf{q}_i \oplus [\mathscr{C}(c_i) \odot \mathbf{s}])), \tag{7}$$

where $\mathscr{C}$ now is a *pseudorandom code* instead of a linear error correcting code.

Notably, the random seed s consists of $(\mathbf{q}_i, \mathbf{s})$ in *BaRK-OPRF* and $i \in [m]$, indicating that equation (6) essentially instantiates m different OPRFs in total, and this is why *BaRK-OPRF* is called bathed and key-related OPRF. In addition, the codeword length of pseudorandom code in *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 $\mathbb{O}Tex$ is described in Figure 1. While the essential difference between $\mathcal{F}_{\text{OT}_l^m}$ in Figure 3 and $\mathcal{F}_{\mathbb{O}Tex}$ 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 $\mathcal{F}_{\text{OT}_{l'}^{m'}}$ we want. Therefore, we make OT execute in a program-iteration-like way (see Figure 5) to reach the final functionality $\mathcal{F}_{\text{OT}_l^m}$. Here, for clarity, we denote selection string in $\mathcal{F}_{\text{OT}_{k_i}^{k_{i+1}}}$ by bold letter $\mathbf{r}_i$ instead of $\mathbf{c}$, and j-th element of $\mathbf{r}_i$ is denoted by $\mathbf{r}_i[j]$.

To better understand our work, let us review IKNP OT extension where $\mathcal{S}$ holds m pairs of l-length messages $\{(x_1^0, x_1^1), \ldots, (x_m^0, x_m^1)\}$, and the receiver $\mathcal{R}$ holds m-bit selection vector $\mathbf{c}$. According to Figure 5, we now focus on $\text{OT}_{k_2}^{k_1}$ and $\text{OT}_{k_3}^{k_2}$ only. For $i \in [m]$, each row of matrices T_2 and T_2' consists of x_i^0 and x_i^1, respectively; that is, both T_2 and T_2' are $m \times l$ matrices and $\mathbf{r}_2 = \mathbf{c}$. Now, $\text{OT}_{k_2}^{k_1}$ serves as "base" OT in IKNP, while the final functionality is to realize $\text{OT}_{k_3}^{k_2}$, where k_1 is the security parameter, $k_2 = m$, and $k_3 = l$. After $\text{OT}_{k_2}^{k_1}$, $\mathcal{S}$ computes equation (5) and sends encrypted T_2 and T_2' to $\mathcal{R}$. Using symmetric-key operations only, $\mathcal{R}$ receives V_2 from T_2 and T_2' in an oblivious way. For $i \in [m]$, each row of matrix V_2 is $x_i^{c_i}$. We have reviewed the whole framework of IKNP protocol so far, and it does matter the relationships among T_1, T_1', and $\mathbf{r}_2$. If we write the algorithm implementing $\text{OT}_{k_{i+1}}^{k_i}$ as recursive function $F_{\text{OT}(k_i, k_{i+1})}$ in programming language, then the key step in IKNP is that $F_{\text{OT}(m,l)}$ invokes $F_{\text{OT}(\kappa,m)}$; *i.e.,* OT_l^m is reduced to OT_m^κ, where κ is the security parameter in IKNP. Thus, we implement $F_{\text{OT}(m,l)}$ by invoking $F_{\text{OT}(\kappa,m)}$ first and then executing some symmetric-key encryption operations.

In more general case, two parties P_1 and P_2 prepare to run protocol $\text{OT}_{k_{i+1}}^{k_i}$, where P_1 acts as sender $\mathcal{S}$ holding messages matrices T_i and T_i' while P_2 holds chosen vector $\mathbf{r}_i$. After $\text{OT}_{k_{i+1}}^{k_i}$, P_2 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 $OT_{k_i}^{k_{i-1}}$, where P_1 acts as a *receiver* and P_2 acts as a *sender*. For each $\text{OT}_{k_{i+1}}^{k_i}$ and $\text{OT}_{k_i}^{k_{i-1}}$, it holds

$$T_{i-1}[j] \oplus T_{i-1}'[j] = f(\mathbf{r}_i[j]). \tag{8}$$

$T_{i-1}[j]$ denotes the j-th row of matrix $T_{i-1}[j]$, $\mathbf{r}_i[j]$ denotes j-th element of $\mathbf{r}_i$, 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

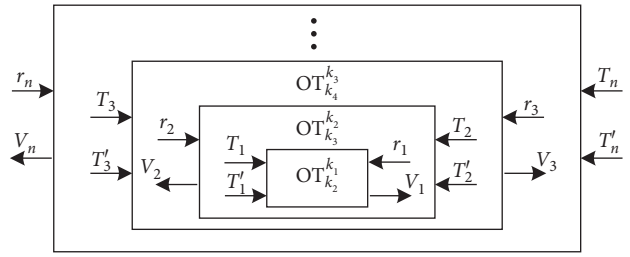

Figure 5: Overview of OT iteration.

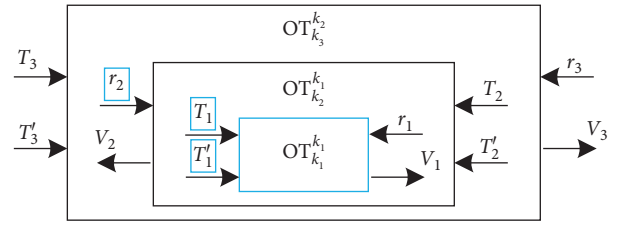

Figure 6: Construction of $\mathbb{O}Tex$.

$f(\cdot)$ in concrete protocol and just emphasize the connections between T_{i-1} and $\mathbf{r}_i$ here.

$\mathrm{OT}_{k_{i+1}}^{k_i}$ and $\mathrm{OT}_{k_i}^{k_{i-1}}$ are named as *outer* and *inner* OT, respectively. Specially, $\mathrm{OT}_{k_2}^{k_1}$ is the innermost one. Then, we find that the two input vectors T_{i-1} and $T_{i-1}{}'$ of the inner $\mathrm{OT}_{k_i}^{k_{i-1}}$ are determined by the input $\mathbf{r}_i$ of the outer $\mathrm{OT}_{k_{i+1}}^{k_i}$. The role of P_1 and P_2 get reversed each time $F_{\mathrm{OT}(k_i,k_{i+1})}$ invokes $F_{\mathrm{OT}(k_{i-1},k_i)}$; that is to say, $\mathcal{S}$ in outer OT becomes the *receiver* in inner OT, so as to $\mathcal{R}$. Furthermore, $\mathrm{OT}_{k_i}^{k_{i+1}}$ (if exists) is the *outer* OT related to $\mathrm{OT}_{k_{i-1}}^{k_i}$; therefore, the input vectors of T_i and T_i' are determined by $\mathbf{r}_{i+1}$.

The transformation among $\mathrm{OT}_{k_{i+2}}^{k_{i+1}}$, $\mathrm{OT}_{k_{i+1}}^{k_i}$, and $\mathrm{OT}_{k_i}^{k_{i-1}}$ is the main part of our work. The key observation is that the inputs of both two parties in $\mathrm{OT}_{k_i}^{k_{i-1}}$ and $\mathrm{OT}_{k_{i+2}}^{k_{i+1}}$ are independent. That is, T_{i+1} and T_{i-1}, $T_{i+1}{}'$ and $T_{i-1}{}'$, and $\mathbf{r}_{i+1}$ and $\mathbf{r}_{i-1}$ are independent. If $\mathrm{OT}_{k_{i+2}}^{k_{i+1}}$ is a necessary component in protocol, we can precompute $\mathrm{OT}_{k_i}^{k_{i-1}}$ assisted with a semihonest server. We focus on a special-but-sufficient case where $\mathcal{S}$ and $\mathcal{R}$ prepare to execute $\mathrm{OT}_{k_3}^{k_2}$; notably, the innermost is $\mathrm{OT}_{k_1}^{k_1}$ (see Figure 6) instead of $\mathrm{OT}_{k_2}^{k_1}$. In such a scenario, k_1 should not be less than security parameter. If $\mathcal{R}$ is a normal user acting as the receiver in $\mathrm{OT}_{k_3}^{k_2}$ and $\mathcal{S}$ is a database with limited computation power and interaction capability, the framework becomes more useful and efficient. Therefore, we consider a server-aided oblivious transfer reducing the sender's all public-key computation and numbers of interaction with other parties. As shown in Figure 6, $\mathcal{S}$ outsources the computation marked with blue rectangle to a server.

Our $\mathbb{O}Tex$ protocol consists of two major phases. First, in the **outsourced phase**, $\mathcal{R}$ and *server* $\mathbb{S}$ run $\mathrm{OT}_{k_1}^{k_1}$ as follows. $\mathcal{R}$ inputs random selection string $\mathbf{r}_1$, while $\mathbb{S}$ inputs two random $k_1 \times k_1$ matrices T_1 and T_1'. After $\mathrm{OT}_{k_1}^{k_1}$, $\mathcal{R}$ gets output V_1 and prepares T_2 (and T_2') for $\mathrm{OT}_{k_2}^{k_1}$. In the second phase, $\mathcal{S}$ aims to **respond to the request** for $\mathrm{OT}_l^{k_2}$. This phase can be concluded as $F_{\mathrm{OT}(k_1,k_2)}$ invoking $F_{\mathrm{OT}(k_1,k_1)}$ and $\mathcal{S}$ sending pairs of messages (T_3 and T_3' encrypted by V_2) to $\mathcal{R}$. In the last phase, $\mathcal{R}$ gets outputs from $OT_l^{k_2}$, which can be seen as $F_{\mathrm{OT}(k_2,l)}$ invoking $F_{\mathrm{OT}(k_1,k_2)}$.

4. Outsourced Oblivious Transfer Extension

In this section, we show how to construct an outsourced oblivious transfer extension protocol $\mathbb{O}Tex$, where three-party functionality $\mathcal{F}_{\mathbb{O}Tex}$ can be securely computed in the presence of semihonest adversaries.

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

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

We describe $\mathbb{O}Tex$ in Figure 8 which realizes functionality $F_{\mathbb{O}Tex}$ in Figure 4. The invoking procedure in $\mathbb{O}Tex$ can be written as $F_{\mathrm{OT}(m,l)} \gg F_{\mathrm{OT}(\kappa,m)} \gg F_{\mathrm{OT}(\kappa,\kappa)}$.

According to equation (8), each time $F_{\mathrm{OT}(k_i,k_{i+1})}$ invokes $F_{\mathrm{OT}(k_{i-1},k_i)}$, and it holds $T_{i-1}[j] \oplus T_{i-1}{}'[j] = f(\mathbf{r}_i[j])$. Let $f(\cdot)$ represent different encoding schemes each time iteration occurs. In $\mathbb{O}Tex$, we have $T[i] \oplus T'[i] = f_1(\mathbf{s})$ and $V[j] \oplus V'[j] = f_2(\mathbf{c})$, where both $f_1(\cdot)$ and $f_2(\cdot)$ are repetition code with output length κ, *i.e.*,

$$\begin{aligned} f_1(\mathbf{s}) &= s_i \| \cdots \| s_i, \\ f_2(\mathbf{c}) &= c_i \| \cdots \| c_i. \end{aligned} \tag{9}$$

Notably, in more general case, the outputs of $f_1(\mathbf{r}_i[j])$ and $f_1(\mathbf{r}_{i+1}[j])$ may be not equal length, which is determined by $\mathrm{OT}_{k_i}^{k_{i-1}}$ and $\mathrm{OT}_{k_{i+1}}^{k_i}$, respectively. Figures 7 and 8 illustrate $\mathbb{O}Tex$ 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 $\mathbb{O}Tex$ protocol in Figure 8 securely computes the functionality $F_{\mathbb{O}Tex}$ (Figure 4) in semihonest setting, as described in Definition 1, given random oracle and functionality $\mathcal{F}_{OT}$(Figure 1).*

Proof. We begin by proving the correctness. After $\mathbb{O}Tex$, we prove that $\mathcal{R}$ only receives $x_i^{c_i}$ and knows nothing about $x_i^{1-c_i}$ when computing $\tilde{x}_i = \tilde{z}_i^{c_i} \oplus H(i, \mathbf{v}_i)$. In the *outsourced phase*, it holds that

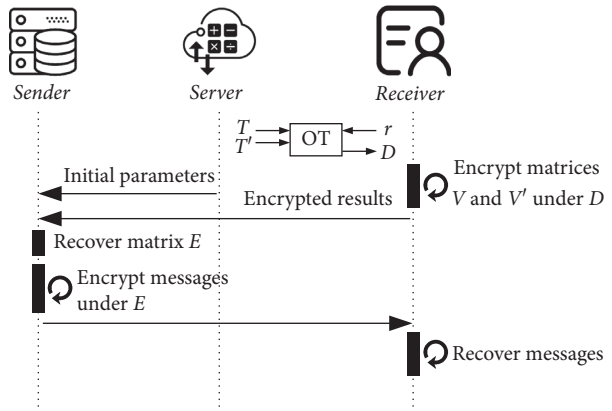

Figure 7: Procedure of outsourced oblivious transfer.

INPUT OF *sender* S: m pairs (x_i^0, x_i^1) of l-bit binary string.
INPUT OF *server* $\mathbb{S}$: random string $s \leftarrow \{0,1\}^\kappa$.
INPUT OF *receiver* R: m selection bit $c = c_1 || \ldots || c_m$.
COMMON INPUT: a security parameter κ.
ORACLE: *random oracles* H: $[m] \times \{0,1\}^\kappa \to \{0,1\}^l$, $H_1 : [\kappa] \times \{0,1\}^\kappa \to \{0,1\}^m$.
CRYPTOGRAPHIC PRIMITIVE : an ideal $\mathrm{OT}_\kappa^\kappa$ primitive.

(i) Outsourced phase:
(1) R chooses $r \leftarrow \{0,1\}^\kappa$ at random.
(2) $\mathbb{S}$ computes $\kappa \times \kappa$ matrix T at random and sets matrix T' such that $t_i \oplus t_i' = (s_i || \ldots || s_i)$.
(3) For each $j \in [\kappa]$, $\mathbb{S}$ and R invoke $F_{\mathrm{OT}_\kappa^\kappa}$ in the following way:
(a) R acts as *recevier* with input r_j.
(b) S acts as *sender* with input (t^j, t'^j).
(c) R recevies 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) $\mathbb{S}$ sends string s and matrix T to S.
(5) R forms $m \times \kappa$ matrix V at random and sets matrix T' such that $v_i \oplus v_i' = (c_i || \ldots || c_i)$, where $i \in [m]$. For $j \in [\kappa]$, R computes $y_j^0 = v^j \oplus H_1(j, d_j)$ and $y_j^1 = v'^j \oplus H_1(i, d_j \oplus r)$ and then sends messages (y_j^0, y_j^1) to S.
(6) S recovers $m \times \kappa$ matrix E by computing $e^j = y_j^{s_j} \oplus H_1(j, t_j)$, $j \in [\kappa]$. For $i \in [m]$, S computes $z_i^0 = x_i^0 \oplus H(i, e_i)$ and $z_i^1 = x_i^1 \oplus H(i, e_i \oplus s)$ and then sends messages (z_i^0, z_i^1) to R.

(iii) Outputting phase:
(7) For $i \in [m]$, after receiving $(\tilde{z}_i^0, \tilde{z}_i^1)$ to R outputs $\tilde{x}_i = \tilde{z}_i^{c_i} \oplus H(i, v_i)$.

Figure 8: Outsourced oblivious transfer extension protocol.

$$\begin{aligned} \mathbf{d}^j &= \mathbf{t}^j \oplus \left[r_j \cdot \mathbf{s}\right], \\ \mathbf{d}_i &= \mathbf{t}_i \oplus \left[s_i \cdot \mathbf{r}\right], \quad \text{where } i, j \in [\kappa]. \end{aligned} \tag{10}$$

Then, in the responding phase, Step 5, $\mathscr{R}$ essentially computes

$$\begin{aligned} y_j^0 &= \mathbf{v}^j \oplus H_1\left(j, \mathbf{t}_j \oplus \left[s_j \cdot \mathbf{r}\right]\right), \\ y_j^1 &= \mathbf{v}'^j \oplus H_1\left(j, \mathbf{t}_j \oplus \left[\left(s_j \oplus 1\right) \cdot \mathbf{r}\right]\right), \quad \text{where } j \in [\kappa]. \end{aligned} \tag{11}$$

Therefore, in Step 6, for $j \in [\kappa]$, by computing $\mathbf{e}^j = y_j^{s_j} \oplus H_1(j, \mathbf{t}_j)$, $\mathscr{S}$ gets $\mathbf{e}^j = \mathbf{v}^j$ when $s_j = 0$ and gets $\mathbf{e}^j = \mathbf{v}'^j$ when $s_j = 1$. Due to $\mathbf{v}_i \oplus \mathbf{v}_i' = (c_i \| \cdots \| c_i)$, we have $\mathbf{e}^j = \mathbf{v}^j \oplus [s_j \cdot \mathbf{c}]$ and

$$\mathbf{e}_i = \mathbf{v}_i \oplus \left[c_i \cdot \mathbf{s}\right], \quad \text{where } i \in [m], j \in [\kappa]. \tag{12}$$

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

$$z_i^\sigma = x_i^\sigma \oplus H\left(i, \mathbf{v}_i \oplus \left[c_i \oplus \sigma\right] \cdot \mathbf{s}\right). \tag{13}$$

It holds that $H(i, \mathbf{v}_i \oplus [c_i \oplus \sigma] \cdot \mathbf{s}) = H(i, \mathbf{v}_i)$ if and only if $c_i = \sigma$. Concretely, when $c_i = 0$, $\mathscr{R}$ computes

$$\widetilde{x}_i = \widetilde{z}_i^0 \oplus H(i, \mathbf{v}_i) = x_i^0 \oplus H(i, \mathbf{v}_i \oplus [0 \oplus 0] \cdot \mathbf{s}) \oplus H(i, \mathbf{v}_i) = x_i^0. \tag{14}$$

When $c_i = 1$, $\mathcal{R}$ computes

$$\widetilde{x}_i = \widetilde{z}_i^1 \oplus H(i, \mathbf{v}_i) = x_i^1 \oplus H(i, \mathbf{v}_i \oplus [1 \oplus 1] \cdot \mathbf{s}) \oplus H(i, \mathbf{v}_i) = x_i^1. \tag{15}$$

In summary, $\mathcal{R}$ only receives $x_i^{c_i}$ in $\mathbb{O}Tex$ and cannot recover $x_i^{1-c_i}$ by computing $\widetilde{z}_i^{1-c_i} \oplus H(i, \mathbf{v}_i)$ because $\mathcal{R}$ knows nothing about $H(i, \mathbf{v}_i \oplus [0 \oplus 1] \cdot \mathbf{s})$, *i.e.,* $H(i, \mathbf{v}_i \oplus [c_i \oplus (1 - c_i)] \cdot \mathbf{s})$.

This concludes the correctness of $\mathbb{O}Tex$.

We now construct three simulators $Sim_{\mathbb{S}}$, $Sim_{\mathcal{R}}$, and $Sim_{\mathcal{R}}$ for simulating corrupt $\mathcal{S}$, $\mathbb{S}$, and $\mathcal{R}$ such that the produced transcript and the view of real execution are computationally indistinguishable. That is,

$$\begin{aligned} &\{Sim_{\mathbb{S}}(1^\kappa, \mathbf{s}, \perp)\}_{x,\mathbf{s},\mathbf{c},\kappa}{}^c \equiv \{view_{\mathbb{S}}^{\pi}(x, \mathbf{s}, \mathbf{c}, \kappa)\}_{x,\mathbf{s},\mathbf{c},\kappa}, \\ &\{Sim_{\mathcal{S}}(1^\kappa, x, \perp)\}_{x,\mathbf{s},\mathbf{c},\kappa}{}^c \equiv \{view_{\mathcal{S}}^{\pi}(x, \mathbf{s}, \mathbf{c}, \kappa)\}_{x,\mathbf{s},\mathbf{c},\kappa}, \\ &\{Sim_{\mathcal{R}}(1^\kappa, \mathbf{c}, \widetilde{x})\}_{x,\mathbf{s},\mathbf{c},\kappa}{}^c \equiv \{view_{\mathcal{R}}^{\pi}(x, \mathbf{s}, \mathbf{c}, \kappa)\}_{x,\mathbf{s},\mathbf{c},\kappa}, \end{aligned} \tag{16}$$

where x denotes input set $\{(x_i^0, x_i^1)\}$ from $\mathcal{S}$, $\widetilde{x}$ denotes output $\{\widetilde{x}_i\}$ of $\mathcal{R}$, and $\perp$ denotes the null value.

Corrupt $\mathbb{S}$. Given $\mathcal{F}_{\mathrm{OT}}$, it is easy to perfectly simulate the view of the *Server* $\mathbb{S}$ because $\mathbb{S}$ only has input, neither receives messages nor outputs during the execution of the protocol. That is, $\{Sim_{\mathbb{S}}(1^\kappa, \mathbf{s}, \perp)\}_{x,\mathbf{s},\mathbf{c},\kappa}$ is computationally indistinguishable with $\{view_{\mathbb{S}}^{\pi}(x, \mathbf{s}, \mathbf{c}, \kappa)\}_{x,\mathbf{s},\mathbf{c},\kappa}$.

Corrupt $\mathcal{S}$. In protocol, *Sender* $\mathcal{S}$ works only in the *responding phase*, when $\mathcal{S}$ receives messages from $\mathbb{S}$ and $\mathcal{R}$ then sends encrypted inputs to $\mathcal{R}$. The messages obtained by $\mathcal{S}$ are $\{\mathbf{s}, T, (y_j^0, y_j^1)\}$. That is to say, $view_{\mathcal{S}}^{\pi}(x, \mathbf{s}, \mathbf{c}, \kappa)$ consists of input x and messages $\{\mathbf{s}, T, (y_j^0, y_j^1)\}$, where $j \in [\kappa]$. It is trivial for $Sim_{\mathcal{S}}$ to generate a simulation of $\{\mathbf{s}, T\}$ since both are random values from the point of view of $\mathcal{S}$.

Now, $Sim_{\mathcal{S}}$ simulates a simulation of $\{(y_j^0, y_j^1) | j \in [\kappa]\}$ by choosing a κ-length string $\widehat{\mathbf{s}}$, m-length strings $\widehat{\mathbf{r}}_j$, $\kappa \times \kappa$ matrix $\widehat{T}$, and $m \times \kappa$ matrix $\widehat{E}$ at random and computing

$$\begin{aligned} \widehat{y}_j^{\widehat{\mathbf{s}}_j} &= \widehat{\mathbf{e}}^j \oplus H_1\left(j, \widehat{\mathbf{t}}_j\right), \\ \widehat{y}_j^{1-\widehat{\mathbf{s}}_j} &= \widehat{\mathbf{r}}_j. \end{aligned} \tag{17}$$

Let $\{\widehat{\mathbf{s}}, t\widehat{T}n, q(\widehat{y}_j^0, \widehat{y}_j^1)\}$ be the output of $Sim_{\mathcal{S}}(1^\kappa, x, \perp)$. Therefore, we claim that the view generated by $Sim_{\mathcal{S}}$ and the view of corrupt $\mathcal{S}$ in a real protocol are computationally indistinguishable.

Corrupt $\mathcal{R}$. We construct a simulator $Sim_{\mathcal{R}}$ that simulates the view of corrupt $\mathcal{R}$ in the real protocol execution. We first analyze $\mathcal{R}$'s view $view_{\mathcal{R}}^{\pi}(x, \mathbf{s}, \mathbf{c}, \kappa)$ in $\mathbb{O}Tex$. $\mathcal{R}$ obtains matrix D in the outsourced phase and $(\widetilde{z}_i^0, \widetilde{z}_i^1)$ in the outputting phase. Therefore, $view_{\mathcal{R}}^{\pi}(x, \mathbf{s}, \mathbf{c}, \kappa)$ consists of $\mathcal{R}$'s input $\mathbf{c}$ and messages $\{D, (\widetilde{z}_i^0, \widetilde{z}_i^1)\}$. To construct the simulation of the view, $Sim_{\mathcal{R}}$ works as follows.

(i) In the outsourced phase, given the security parameter κ, $\mathcal{R}$'s input $\mathbf{c}$, and the randomness $\mathbf{r}$, $Sim_{\mathcal{R}}$ calls simulation $Sim_{\mathrm{OT}_\kappa^\kappa}$ with input $(\widehat{T}, \widehat{T}', r)$ and gets output $\widehat{D}$. Then, $Sim_{\mathcal{R}}$ appends the output of $Sim_{\mathrm{OT}_\kappa^\kappa}$, *i.e.,* matrix $\widehat{D}$, to its own output.

(ii) In the outputting phase, given the security parameter κ, $\mathcal{R}$'s input $\mathbf{c}$, the randomness V, and output $\widetilde{x}$, $Sim_{\mathcal{R}}$ simulates a simulation of $\{(\widetilde{z}_i^0, \widetilde{z}_i^1) | it \in n[m]\}$ by choosing m-length strings $\widehat{\mathbf{r}}_j$ and computing

$$\begin{aligned} \widehat{z}_i^{\mathbf{c}_i} &= \widetilde{x}_i \oplus H(i, \mathbf{v}_i), \\ \widehat{z}_i^{1-\mathbf{c}_i} &= \widehat{\mathbf{r}}_j. \end{aligned} \tag{18}$$

Then, $Sim_{\mathcal{R}}$ appends $\{(\widehat{z}_i^0, \widehat{z}_i^1) | it \in n(m\}$ to its output.

Combining two phases described above in sequence, we finally claim that the output of $Sim_{\mathcal{R}}$ is computationally indistinguishable from the real execution, *i.e.,* $\{Sim_{\mathcal{R}}(1^\kappa, \mathbf{c}, \widetilde{x})\}_{x,\mathbf{s},\mathbf{c},\kappa}{}^c \equiv \{view_{\mathcal{R}}^{\pi}(x, \mathbf{s}, \mathbf{c}, \kappa)\}_{x,\mathbf{s},\mathbf{c},\kappa}$.

This completes the construction of three simulators: $Sim_{\mathbb{S}}$, $Sim_{\mathcal{R}}$, and $Sim_{\mathcal{R}}$. In summary, $\mathbb{O}Tex$ is secure under semihonest model. □

4.1. Performance Analysis. We now analyze the performance of $\mathbb{O}Tex$ in semihonest model. The complexities of $\mathbb{O}Tex$ are presented in Table 1.

(i) *Computation complexity*. The number of symmetric and asymmetric operation to be executed in $\mathbb{O}Tex$ essentially depends on the size of matrices T, V, and E, respectively. In the outsourced phase, $\mathcal{F}_{\mathrm{OT}_\kappa^\kappa}$ consists of $O(\kappa)$ public-key and $O(\kappa)$ private-key operations, for both $\mathbb{S}$ (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, $\mathcal{S}$ recovers matrix E and encrypts messages (x_i^0, x_i^1), which only consists of $O(m + \kappa)$ symmetric operations. In the outputting phase, $\mathcal{R}$ performs symmetric operations $O(m)$ times to compute its outputs.

(ii) *Communication complexity*. In the outsourced phase, the number of bits transferred between $\mathbb{S}$ and $\mathcal{R}$ is bounded by $O(\kappa^2)$. In the responding phase, $\mathcal{S}$ receives messages from $\mathbb{S}$ 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 $\mathcal{S}$.

(iii) *Round complexity*. The only interaction in $\mathbb{O}Tex$ exists in the outsourced phase between $\mathbb{S}$ and $\mathcal{R}$, *i.e.,* $\mathcal{F}_{\mathrm{OT}_\kappa^\kappa}$. The number of interaction round in $\mathcal{F}_{\mathrm{OT}_\kappa^\kappa}$ is bounded by 1.

Table 1: Complexity of $\mathbb{O}Tex$.

Party	Computation		Communication	Round
	Asymmetric	Symmetric		
Server $\mathbb{S}$	$O(\kappa)$	$O(\kappa)$	$O(\kappa^2)$	1
Sender $\mathcal{S}$	–	$O(m+\kappa)$	$O(ml+\kappa^2)$	–
Receiver $\mathcal{R}$	$O(\kappa)$	$O(m+\kappa)$	$O(ml+\kappa^2)$	1

Table 2: Efficiency comparison.

Protocol	Round	Communication	Asymmetric computation	Security model
[4]	2	$O(ml+m\kappa)$	$O(\kappa)$	Semihonest and malicious
[5]	2	$O((mk/\log n)+\ln\log n)$	$O(k)$	Semihonest
[16]	2	$O(ml+\kappa^2)$	$O(\kappa+\ln)$	Semihonest and one-sided malicious
[30]	2	$O(ml+m\kappa)$	$O(\kappa)$	Semihonest
[31]	3	$O(ml+\log\lvert\mathbb{G}\rvert+\kappa)$	$O(\kappa)$	Malicious
Ours	–	$O(ml+\kappa^2)$	–	Semihonest

Note: the efficiency of *sender* in OT is presented here, who inputs m pairs of l-bit length messages in protocol. Specially, κ is security parameter, $\mathbb{G}$ denotes a group, and $k \approx 2.5\kappa$ in 1-out-of-n OT.

4.2. Efficiency Comparison. Since we focus on the efficiency of the sender $\mathcal{S}$ in OT, we provide comparisons to prior classical protocols from the $\mathcal{S}$'s point of view in Table 2. In $\mathbb{O}Tex$, all asymmetric operations are operated between the receiver $\mathcal{R}$ and the cloud server $\mathbb{S}$, and $\mathcal{S}$ works on the fly and conducts symmetric computations locally.

5. Performance

In this section, we test the performance of $\mathbb{O}Tex$. 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 implement on GitHub: https://github.com/emp-toolkit/emp-ot. We simulated main asymmetric operations in $\mathbb{O}Tex$ on a single machine. We compared $\mathbb{O}Tex$ 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 $\mathbb{O}Tex$ 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 $\mathbb{O}Tex$ 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 $\mathcal{S}$ of $\mathbb{O}Tex$ works offline for most of the time during simulation, and this is one of the advantages of $\mathbb{O}Tex$.

The other advantage of $\mathbb{O}Tex$ found in performance tests is the reduction of communication bottleneck caused by the sender $\mathcal{S}$. 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 OTon setup to the time of protocol finish and then computed average time for each original OT.

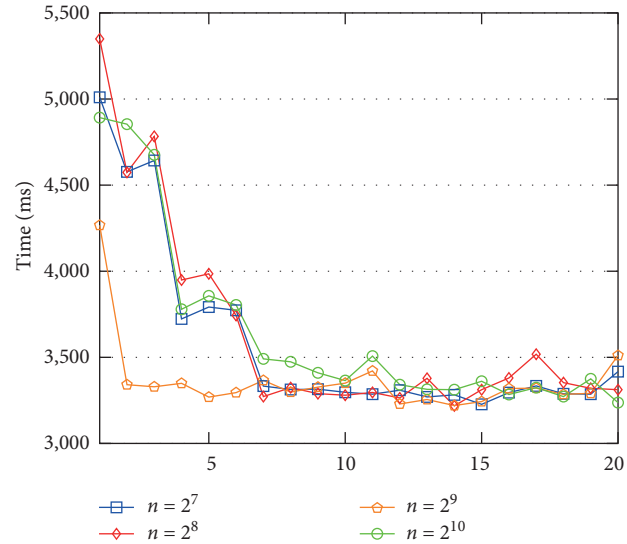

Figure 9: Comparison of running time.

The averaged time is illustrated in Figure 10. We tested $n' = 2^{8+x}$ OT instances for different x values and computed the average running time for one single OT, *i.e.*, OT_n^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 $\mathbb{O}Tex$, however, since the extension process is conducted between $\mathcal{R}$ and $\mathbb{S}$ in the outsourced phase.

6. Applications of $\mathbb{O}Tex$

The $\mathbb{O}Tex$ 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

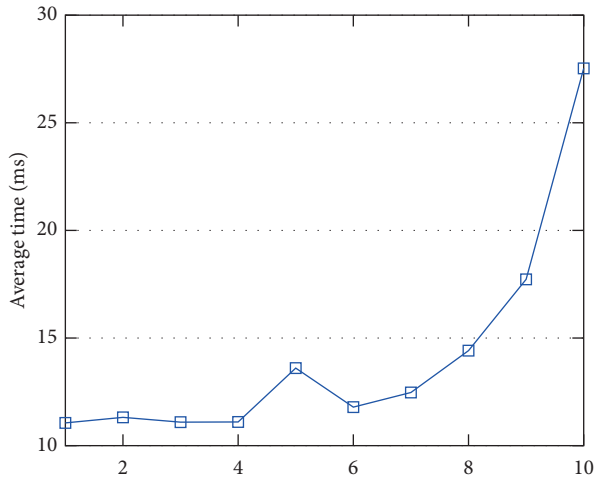

Figure 10: Average time for OT instances in $\mathbb{O}Tex$.

building block, $\mathbb{O}Tex$ 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 $\mathbb{O}Tex$. The resulting protocol can then be simply extended to compute functionality $\mathcal{F}_{\text{PSI}}$ in Figure 11 by applying the $\mathbb{O}Tex$-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 $\{y_i | y_i t \in nYq, hf_{s(}xy_i 7)C \neq ; f_s(x)\}$ with non-negligible probability. We omit concrete security proof here.

6.2. $\mathbb{O}Tex$-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 $\mathbb{O}Tex$. Equation (7) indicates that 1-out-of-n OT extension implies *BaRK-OPRF*; therefore, we focus on the transformation from 1-out-of-n OT extension to $\mathbb{O}Tex$. As a result, $\mathbb{O}Tex$ 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,

Functionality F_{PSI}

Two parties: sender S and receiver R.

(i) Input:
  (a) S inputs $Y = \{y_1, \ldots, y_m\}$.
  (b) R inputs $X = \{x_1, \ldots, x_m\}$.

(ii) Output:
  (a) S has no outputs.
  (b) R outputs $X \cap Y$.

Figure 11: Private set intersection functionality $\mathcal{F}_{\text{PSI}}$.

which implements functionality $\mathcal{F}_{\text{OPRF}}$ in Figure 3. In $\mathbb{O}Tex$, however, we stress that it *always* holds $T_1[i] \oplus T_1'[i] = f_1(\mathbf{r}_2[i])$ and $T_2[j] \oplus T_2'[j] = f_1(\mathbf{r}_3[j])$, 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 $F_{\text{OT}(k_1,k_2)}$ and $F_{\text{OT}(k_2,k_3)}$. That is to say, three parties, $\mathbb{S}$, $\mathcal{S}$, and $\mathcal{R}$, evaluate corporately m instances of 1-out-of-2 OT of l-bit messages where repetition code is used twice in total, each by $\mathbb{S}$ and $\mathcal{R}$, respectively. Now, suppose *Alice* and *Bob* would like to execute m instances of 1-out-of-n OT where *Bob* inputs $\{(y_1^j, y_2^j, \ldots, y_n^j) | j \in [m], |y_i^j| = l\}$ and *Alice* inputs her selection vector $\mathbf{r}_3$, and they need make the following adjustments:

(i) *Bob* organizes the inputs in $\mathbb{O}Tex$ with $m \times l$ matrices $T_3^{(1)}, T_3^{(2)}, \ldots, T_3^{(n)}$, where the j-th row of matrix $T_3^{(i)}$ is y_i^j.

(ii) *Alice* makes her selection vector in $\mathbb{O}Tex$ be $\{\mathbf{r}_3 | jt \in n[m]q, h\mathbf{r}_{3[}j]x \in 7[n]\}$.

Then, the only adaption they need take in $\mathbb{O}Tex$ is that $T_2[j] \oplus T_2'[j] = f_2(\mathbf{r}_3[j])$, where $f_2(\cdot)$ is the linear error correcting code $\mathscr{C}$. To reach the security requirement defined in [13], in $\mathbb{O}Tex$, we need to reset security parameter k rather than κ—concretely $3\kappa < k < 4\kappa$.

Given 1-out-of-n OT protocol designed from $\mathbb{O}Tex$, it is trivial to design $\mathbb{O}Tex$-based OPRF so is to $\mathbb{O}Tex$-based PMT. Again, *Alice* prepares her element $x = \mathbf{r}_3[1]$, while *Bob* holds set $Y = \{y_1^1, y_2^1, \ldots, y_n^1\}$. In brief, if it holds in $\mathbb{O}Tex$ that

$$\begin{aligned} T_1[i] \oplus T_1'[i] &= f_1(\mathbf{r}_2[i]), \\ T_2[j] \oplus T_2'[j] &= \mathscr{C}(\mathbf{r}_3[j]), \end{aligned} \tag{19}$$

where $f_1(\cdot)$ is the repetition code and $\mathscr{C}$ is the pseudorandom code; $\mathbb{O}Tex$ implies the functionality $\mathcal{F}_{\text{OPRF}}$ where the seed s consists of $(V_2, \mathbf{r}_2)$ generated by *Bob* and *Alice* gets $f_s(x)$. Meanwhile, this essentially completes the main construction of $\mathbb{O}Tex$-based PMT protocol. All needed to do next is that *Bob* sends $\{f_s(y_i^1) | i \in [n]\}$ to *Alice* who compares $f_s(x)$ with $\{f_s(y_i^1) | i \in [n]\}$ locally.

6.3. $\mathbb{O}Tex$-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

INPUT OF *Alice*: $\{x_1, \ldots, x_m\}$, $|x_i| = l$.

INPUT OF *Bob*: $\{y_1, \ldots, y_m\}$, $|y_i| = l$.

COMMON INPUT: a security parameter k, pseudorandom code C.

(i) Precomputing phase:

(1) For $i \in m$, *Alice* computes $r_3[i] = C(x_i)$.

(2) For $i, j \in m$, *Bob* computes $T_3^{(i)}[j] = C(y_i)$.

(ii) $\mathbb{O}$Tex phase:

(3) Aided by a sever $\mathbb{S}$, *Alice* and *Bob* execute $\mathbb{O}$Tex with inputs r_3 and $T_3^{(i)}$, respectively.

(4) For $j \in [m]$, *Bob* gets $f_{s_j}(\cdot)$ and *Alice* gets $f_{s_j}(x_i)$ from $\mathbb{O}$Tex.

(iii) Outputting phase:

(5) For each x_i and Y, *Bob* sends $\{f_{s_i}(y_j) \mid i, j \in [m]\}$ to *Alice* who outputs m-length string z, where
- $z_i = 1$, if there exists a j such that $f_{s_i}(x_i) = f_{s_i}(y_j)$,
- $z_i = 0$, otherwise.

(6) *Alice* outputs $\{x_i \mid i \in [m], z_i = 1\}$.

Figure 12: Outsourced private set intersection protocol.

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 $\mathbb{O}Tex$-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: $\{0,1\}^* \longrightarrow [m']$ 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 ($\mathbb{O}Tex$) 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 $\mathbb{O}Tex$ 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.

Conflicts 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 (SFCS 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. Tzeng, "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, "m 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. Tzeng, "Efficient k-out-of-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. Polychroniadou, "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 ot extension," in *Proceedings of the Annual International Cryptology 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.