Unbalanced Private Set Union with Reduced Computation and Communication

Cong Zhang[1], Yu Chen[2,3](✉), Weiran Liu[4], Liqiang Peng[4], Meng Hao[5], Anyu Wang[1,6,7], and Xiaoyun Wang[1,2,3,6,7,8]

[1] Institute for Advanced Study, BNRist, Tsinghua University, Beijing, China
{zhangcong,anyuwang,xiaoyunwang}@tsinghua.edu.cn,
[2] School of Cyber Science and Technology, Shandong University, Qingdao, China
yuchen@sdu.edu.cn
[3] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Qingdao, China
[4] Alibaba Group
{weiran.lwr,plq270998}@alibaba-inc.com,
[5] Singapore Management University
menghao303@gmail.com
[6] Zhongguancun Laboratory, Beijing, China
[7] National Financial Cryptography Research Center, Beijing, China
[8] Shandong Institute of Blockchain, Jinan, China

Abstract. Private set union (PSU) is a cryptographic protocol that allows two parties to compute the union of their sets without revealing anything else. Despite some efficient PSU protocols that have been proposed, they mainly focus on the balanced setting, where the sets held by the parties are of similar size. Recently, Tu et al. (CCS 2023) proposed the first unbalanced PSU protocol which achieves sublinear communication complexity in the size of the larger set. In this paper, we are interested in improving the efficiency of the unbalanced PSU protocol. We find that oblivious key-value store (OKVS) data structure plays an essential role in the most recently proposed PSU constructions and formalize unbalanced PSU as an OKVS decoding process with sublinear communication. Our key insight lies in when OKVS satisfies sparsity property, obtaining the necessary decoding information precisely aligns with the batch private information retrieval (BatchPIR) problem. We give two concrete constructions of unbalanced PSU protocols based on different OKVS encoding strategies. The first is based on oblivious PRF (OPRF) and a newly introduced cryptographic protocol called permuted private equality test, while the second is based on re-randomizable public key encryption. Both our two constructions achieve sublinear communication complexity in the size of the larger set.
We implement our two unbalanced PSU protocols and compare them with the state-of-the-art unbalanced PSU of Tu et al. Experiments show that our protocols achieve a $1.3-5.6\times$ speedup in running time and $2.1-11.8\times$ shrinking in communication cost, depending on set sizes and network environments.

1 Introduction

Private set union (PSU) enables two parties to learn the union of their private sets without revealing anything else about the individual sets. PSU serves as a key tool for multiple applications, including aggregation of network events [BSMD10], improving blacklist accuracy [RMY20], security risk assessments [HLS+16] and universal identifier generation [GMR+21, ZLDL23] etc. The first scalable PSU construction was proposed by Kolesnikov et al. [KRTW19]. Over the last few years, a line of works [GMR+21, JSZ+22, ZCL+23, CZZ+24] have greatly improved the efficiency of PSU. Recent advancements in PSU protocols [ZCL+23, CZZ+24] have demonstrated their computational efficiency and achieved linear communication complexity proportional to the size of the parties' sets.

Classical PSU protocols are typically designed for balanced settings, that is, the sets held by the parties are of similar size. However, in many scenarios, the size of one party's input is much larger than the other party's. A relevant example is blacklist aggregation [KRTW19, JSZ+22, ZCL+23], where two organizations aim to merge individual IP blacklists to mitigate vulnerabilities in their infrastructure. This necessitates privately computing the union of the IP blacklists, as revealing IP addresses in the intersection may leak the detection strategy of certain party. Ramanathan et al. [RMY20] showed that the sets involved in this process are highly unbalanced, e.g., some blacklists include upwards of 500,000 IP addresses, while others contain fewer than 1,000 IP addresses. Leveraging existing PSU protocols in such settings imposes substantial overhead because their communication complexity scales at least linearly with the size of the larger set. This motivates the study of *unbalanced* PSU, where one set is much larger than the other. The design goal is to achieve sublinear communication complexity in the size of the larger set.

Only two existing works have considered PSU in the unbalanced setting. Jia et al. [JSZ$^+$22] proposed an unbalanced PSU protocol based on the shuffle technique. Unfortunately, their protocol fails to achieve the standard security of PSU, which leaks the intersection size to the sender. Furthermore, the communication complexity of their protocol still scales linearly with the size of the larger set. Recently, Tu et al. [TCLZ23] designed the first unbalanced PSU protocol with sublinear communication in the size of the larger set from fully homomorphic encryption (FHE). However, their concrete construction had a large constant term in asymptotic complexity due to the use of bin partition techniques, incurring a significant overhead in implementations. The constant term becomes prominent when the size of the sender's set is relatively small, e.g., the set only contains several hundred elements. In this work, we focus on unbalanced PSU with sublinear communication complexity, aiming to achieve concrete improvements both in communication and computational costs.

1.1 Our Contribution

This paper contributes new unbalanced PSU protocols with reduced computation and communication costs. Our contributions can be summarized as follows.

OKVS decoding with sublinear communication complexity. We observe that nearly all existing PSU protocols [KRTW19, GMR$^+$21, ZCL$^+$23, TCLZ23] explicitly or implicitly employ oblivious key-value store (OKVS) data structure [GPR$^+$21]. OKVS is primarily deployed in multi-query reverse private membership test (mq-RPMT), which is the core sub-protocol of PSU [ZCL$^+$23]. In mq-RPMT, the receiver encodes n key-value pairs (e.g., its set elements and set indicators) using OKVS, which are then decoded by the sender. Subsequently, the receiver tests the set membership relationship by matching the sender's decoding values with the set indicators. Our core observation lies in that the recently proposed efficient OKVS schemes all satisfy *sparsity* [HLP$^+$24]. The decoding process of sparse OKVS could always be improved by using BatchPIR instead of sending the OKVS directly, which achieves sublinear communication complexity with the OKVS size.
Two concrete unbalanced PSU constructions. Adopting different OKVS encoding strategies, we propose two concrete constructions of unbalanced PSU protocols. Our first construction follows the strategy of using oblivious PRF (OPRF) to generate n one-time pads for encrypting the set indicator. To perform value matching, we propose a new protocol called permuted private equality test (p-PEQT), which can be viewed as a generalization of permuted matrix PEQT [TCLZ23]. We provide two constructions of p-PEQT, one based on the Decisional Diffie-Hellman (DDH) assumption, while the other relies on the Permute+Share functionality. Our second construction employs a rerandomizable public key encryption (ReRand-PKE) scheme to encrypt the set indicator, which allows the receiver to match values itself. Both the two concrete protocols achieve sublinear communication in the size of the larger set.
Evaluations. We implement our two unbalanced PSU constructions and compare them with the state-of-the-art unbalanced PSU [TCLZ23]. Our experiments show that our protocols achieve a $1.3-5.6\times$ speedup in running time and $2.1-11.8\times$ shrinking in communication cost, depending on set sizes and network environments. In Figure 1 we display a comparison of running time and communication cost to the state-of-the-art unbalanced PSU [TCLZ23] for small set size $m = 2^4$, large set size $n = 2^{20}$ in low bandwidth setting.

1.2 Overview of Our Techniques

We provide the high-level technical overview of our new unbalanced PSU protocols. For convenience, we denote the parties involved in PSU as the sender $\mathcal{S}$ and the receiver $\mathcal{R}$, and their respective input sets as X and Y with $|X| = m, |Y| = n, m \ll n$.

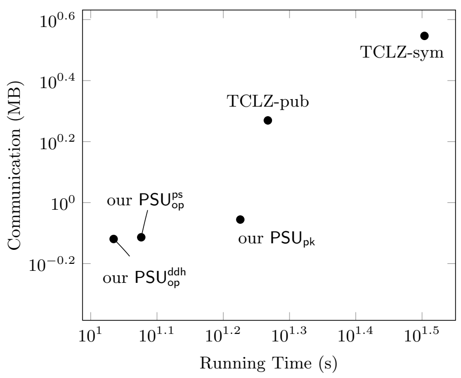

Fig. 1: Running time and communication of our unbalanced PSU protocols ($\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ddh}}$,$\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ps}}$ and $\mathsf{PSU}_{\mathsf{pk}}$) vs. the state of the art [TCLZ23] (TCLZ-pub and TCLZ-sym), for small set size $m = 2^4$, large set size $n = 2^{20}$ in low bandwidth setting (1 Mbps). Both axes are in log-scale.

Communication-efficient OKVS decoding from BatchPIR. An oblivious key-value store (OKVS) scheme [GPR$^+$21] consists of two algorithms, namely, $(\mathsf{Encode}, \mathsf{Decode})$. The Encode algorithm takes a set of key-value pairs $\{(k_i, v_i)\}_{i\in[n]}$ as input, and outputs an abstract data structure D. The Decode algorithm takes D and a key k as input, and outputs a value v. Decode can be called on any key, but if it is called on some k_i that was used to generate D, then the output is the corresponding v_i. The obliviousness stands for that the data structure D reveals nothing about the key set $K = \{k_i\}_{i\in[n]}$, given that all the values v_i are randomly selected. A typical choice of OKVS is polynomial, where encoding involves interpolating all points (k_i, v_i). The encoding complexity of the polynomial is $O(n \log^2 n)$. Recently, several works [GPR$^+$21, RR22, ZCL$^+$23, BPSY23] have made great progress on improving the efficiency and security of OKVS, achieving a linear $O(n)$ encoding complexity.

OKVS plays a crucial role in both balanced and unbalanced PSU constructions [KRTW19, GMR$^+$21, ZCL$^+$23, TCLZ23]. The key step is to have the sender $\mathcal{S}$ compute some decoding values of OKVS D which hold by the receiver $\mathcal{R}$. Due to the obliviousness property of OKVS, D does not leak any information about $\mathcal{R}$'s set. Therefore, a straightforward approach is to have the receiver $\mathcal{R}$ directly send D to the sender $\mathcal{S}$, and let the sender compute the decoding itself. All known PSU protocols in the balanced setting [KRTW19, GMR$^+$21, ZCL$^+$23] follow this simple method, resulting in $O(n)$ communication. However, in the unbalanced setting, where $n \gg m$, the $O(n)$ communication may significantly reduce the efficiency. Only Tu et al.'s protocol [TCLZ23] has achieved sublinear communication complexity. Roughly speaking, their protocol uses polynomials as OKVS instantiation. To decode without directly communicating D, the sender $\mathcal{S}$ encrypts its elements with FHE and sends them to the receiver $\mathcal{R}$. $\mathcal{R}$ utilizes these ciphertexts to homomorphically evaluate the polynomial and returns the ciphertexts to the sender. The compactness of FHE guarantees that the communication in this process is sublinear with n.

Given the fact that polynomial is just an instantiation of OKVS (the encoding complexity is $O(n \log^2 n)$), replacing it with the recently proposed OKVS [GPR$^+$21, BPSY23] directly enhances the encoding efficiency to $O(n)$. However, things are not quite that simple. The decoding process of these efficient OKVS schemes is different from polynomials. Therefore, developing a novel approach to achieve decoding with sublinear communication becomes imperative.

We give our solution as follows. We consider using the OKVS scheme satisfying *sparsity* [HLP$^+$24] as our instantiation, which is satisfied by all recent proposed efficient OKVS schemes [GPR$^+$21, RR22, BPSY23]. In sparse OKVS, the data structure D can be divided into two parts, i.e., $D = D_0||D_1$, where $|D_1| = o(|D_0|)$. We refer to D_0 as the sparse part, and D_1 as the dense part. The decoding process involves summing a constant number (e.g., $2-3$) of positions within the sparse part and an arbitrary number of positions within the dense parts. Directly sending the entire OKVS would introduce significant communication costs in the unbalanced setting due to the $O(n)$ length of the sparse part. Our key observation is that the sender does not actually need to know all the information of sparse part D_0. Instead, it is sufficient to retrieve specific positions in D_0 that correspond to the decoding of its set items. This exactly matches the batch private information retrieval (BatchPIR) [CKGS98] scenario. Leveraging this observation, we can achieve decoding with sublinear communication using the non-trivial BatchPIR scheme [MR23] to retrieve the constant positions in the sparse part. For the dense part D_1, $\mathcal{R}$ can send it to $\mathcal{S}$ directly because its length is already $O(\log n)$. For clarity, we call the above decoding process as *communication-efficient decoding.*

PSU framework. Before describing our construction, we first review the design ideas of existing PSU protocols. We observe that all the recent PSU constructions [JSZ$^+$22, ZCL$^+$23, TCLZ23, CZZ$^+$24] can be divided into two steps. The first step is multi-query reverse private membership test (mq-RPMT) [ZCL$^+$23]. In mq-RPMT, $\mathcal{S}$ inputs its set X and has no output. $\mathcal{R}$ inputs its set Y and receives a vector of indication bits $\vec{b} = (b_1, \dots, b_m)$, where $b_i = 1$ if and only if $x_i \in Y$. Subsequently, the parties engage in a batch of oblivious transfer (OT) protocols. In the i-th OT instance, $\mathcal{S}$ inputs $(x_i, \perp)$, where $\perp$ denotes a dummy item. $\mathcal{R}$ inputs b_i and learns x_i if and only if $x_i \notin Y$. As a result, $\mathcal{R}$ learns $X \setminus Y$ and defines set union as $X \cup Y := (X \setminus Y) \cup Y$.

Since the bottleneck of PSU lies in mq-RPMT, existing PSU protocols primarily focus on using different underlying cryptographic tools to design mq-RPMT. Notably, OKVS (with different instantiations) plays a crucial role in almost all of these constructions. Achieving sublinear communication in OKVS decoding will lead to more efficient PSU constructions in the unbalanced setting. Hereafter, we show two concrete constructions of mq-RPMT utilizing different OKVS encoding strategies, both achieving sublinear communication complexity in n.

mq-RPMT from OPRF and p-PEQT. Our first construction uses oblivious PRF (OPRF) functionality to generate n one-time pads, which are employed to encrypt the set indicator in OKVS values. Additionally, we introduce a new cryptographic protocol named permuted private equality test (p-PEQT) to test whether the sender's decoding results match the receiver's set indicator. We note that our p-PEQT can be viewed as a generalization of permuted matrix PEQT (pm-PEQT) [TCLZ23]. The details of our construction are as follows.

We start with a special case that the sender $\mathcal{S}$ has only one item x in its set X, i.e., $X = \{x\}$. A natural idea is to let the receiver $\mathcal{R}$ pick some random strings r_i and compute the OKVS as $D := \mathsf{Encode}(\{(y_i, r_i)\}_{i\in[n]})$. Then, as we described before, we use the communication-efficient decoding process to render $\mathcal{S}$ to learn the decode result $r' := \mathsf{Decode}(D, x)$ in sublinear communication. After that, the parties test whether $r' = r_j$ for each $j \in [n]$. If $\exists t \in [n]$, s.t. $r' = r_t$, $\mathcal{R}$ knows $x \in Y$. However, this idea encounters two key issues. Firstly, the parties need to perform n equality test, which is inefficient. Secondly, if the receiver $\mathcal{R}$ knows $r' = r_t$, it directly implies $x = y_t$ instead of merely the fact that $x \in Y$, thereby leaking additional information to $\mathcal{R}$. To address these challenges, we consider using the same indicator string to indicate the set membership, that is, $\mathcal{R}$ picks one random string r and computes the OKVS as $D := \mathsf{Encode}(\{(y_i, r)\}_{i\in[n]})$. Then, $\mathcal{S}$ learns $r' := \mathsf{Decode}(D, x)$ and the parties perform a single instance of the equality test. Though this idea works, it ignores an important security issue. The obliviousness property of OKVS mandates that all encoded values must be randomly selected to prevent potential leakage of information about Y. To mitigate this risk, we incorporate an OPRF protocol. Specifically, in OPRF, the sender inputs x and retrieves the PRF value $q = F_k(x)$, while the receiver, without any inputs, obtains a PRF key k. The PRF values could be regarded as one-time pads used to encrypt set indicator r. Subsequently, $\mathcal{R}$ computes the OKVS as $D := \mathsf{Encode}(\{(y_i, r + F_k(y_i))\}_{i\in[n]})$. The parties then engage in a communication-efficient decoding process, allowing $\mathcal{S}$ to learn $\bar{r} = \mathsf{Decode}(D, x)$. Finally, $\mathcal{S}$ computes the decryption $r' := \bar{r} - q$ and tests whether $r' = r$ with $\mathcal{R}$.

Expanding on the previous idea to accommodate the general case of $|X| > 1$, we encounter a challenge with reusing the same OKVS. Consider the scenario where $\mathcal{S}$ has two items x_1, x_2, both belonging to Y. If the same D is utilized for testing both x_1 and x_2, $\mathcal{S}$ will obtain identical decoding results, i.e., $r'_1 = r'_2 = r$, potentially leaking the information that both x_1 and x_2 belong to the intersection. To ensure security, one must execute the aforementioned process for each $x_i \in X$ using a fresh OKVS, resulting in substantial overhead. To overcome this limitation, we propose employing the standard hash-to-bin technique. Specifically, let $\mathcal{S}$ use hash functions $\{h_1, h_2, h_3\}$ to assign its items to m_c bins via cuckoo hashing [PR04], so that each bin has at most one item. $\mathcal{R}$ assigns each of its items y to all of the bins $h_1(y), h_2(y), h_3(y)$. Then, the parties perform the above single-point test on each bin. Though it looks great, the hash-to-bin technique introduces a new security concern. $\mathcal{R}$ may gain additional information about the sender's items, specifically whether the i-th item belongs to the i-th bin, which narrows down the range of x_i. We note that PSU can only allow $\mathcal{R}$ to know if x_i belongs to the entire set Y, rather than the i-th bin. To address this, we introduce a new cryptographic protocol named permuted private equality test (p-PEQT). In this functionality, the sender inputs the vector of decryptions $\vec{r'} := (r'_1, \dots, r'_{m_c})$ and a permutation π over $[m_c]$. The receiver inputs the vector of indicators $\vec{r} := (r_1, \ldots, r_{m_c})$ and receives a bit vector $\vec{b} = (b_1, \ldots, b_{m_c})$, where $b_i = 1$ if and only if $r'_{\pi(i)} = r_{\pi(i)}$. Since $\mathcal{R}$ knows nothing about π, $b_i = 1$ only infers $x_{\pi(i)} \in Y$. We propose two p-PEQT constructions, the first is based on the DDH assumption, while the second is based on the Permute+Share functionality [CGP20]. See Section 3 for more details.

mq-RPMT from ReRand-PKE. Our second construction leverages ReRand-PKE for encrypting the set indicator. The idea is inspired by the PSU protocol of Zhang et al. [ZCL+23] in the balanced setting. Notably, while the communication complexity of their protocol scales linearly with the size of the larger set, i.e., $O(n)$, our protocol achieves sublinear communication complexity. Therefore, our protocol can be seen as an extension of their protocol in the unbalanced setting. The details of our construction are as follows.

Firstly, the receiver $\mathcal{R}$ selects a fixed string s as the set indicator and encrypts it with ReRand-PKE for n times, that is, it computes $s_j := \mathsf{Enc}(pk, s)$ for $j \in [n]$. Then, $\mathcal{R}$ computes the OKVS $D := \mathsf{Encode}(\{(y_j, s_j)\}_{j\in[n]})$. Following this, the sender $\mathcal{S}$ executes the communication-efficient decoding to learn the decoding results, i.e., $s'_i := \mathsf{Decode}(D, x_i), i \in [m]$. Now, the goal is to let $\mathcal{R}$ know whether $\mathsf{Dec}(sk, s'_i) = s$, which signifies whether $x_i \in Y$. However, $\mathcal{S}$ cannot send these s'_i directly to $\mathcal{R}$, since $\mathcal{R}$ can record the relationship between the randomness used for encrypting s_j and y_j. For example, if $\mathcal{R}$ observes $s'_i = s_t$ for some $t \in [n]$, then it deduces $x_i = y_t$ rather than merely $x_i \in Y$. To address this concern, $\mathcal{S}$ re-randomizes each ciphertext s'_i before sending it to $\mathcal{R}$, i.e., $\bar{s_i} := \mathsf{ReRand}(pk, s'_i)$. Subsequently, by testing $\mathsf{Dec}(sk, \bar{s_i}) \overset{?}{=} s$, the receiver $\mathcal{R}$ learns whether $x_i \in Y$.

We highlight a subtle distinction between our two constructions in OKVS encoding. Specifically, the values encoded in this construction consist of ciphertexts generated by the ReRand-PKE scheme, e.g., comprising two elements in a group $\mathbb{G}$ (when instantiated by ElGamal encryption [Gam85]), whereas the values in our first construction are bit strings. Notably, the bit string XOR operation is significantly more efficient than the group operation. Therefore, the encoding efficiency of OKVS in our first construction is much better than that in our second construction. To improve the encoding efficiency, we consider computing the OKVS over $\mathbb{Z}_p$ instead of $\mathbb{G}$, where $|\mathbb{G}| = p$ and p is a prime. This choice is motivated by the fact that addition over $\mathbb{Z}_p$ is considerably more efficient than operations in $\mathbb{G}$. We employ a "pull-down-then-lift" methodology, wherein we initially compute the OKVS over $\mathbb{Z}_p$ and subsequently lift it to the exponent. See Section 4 for more details about this optimization.

1.3 Related Work

In this section, we review previous PSU protocols that are relevant to our work. Most works of PSU protocol only consider balanced settings. Among them, the construction can be divided into two categories based on the underlying cryptographic techniques used. The first category predominantly relies on public-key techniques [KS05, Fri07, HN10, DC17]. The second category mainly relies on symmetric-key techniques in combination with OT [KRTW19, GMR+21, JSZ+22], which is several orders of magnitude faster than the public-key based constructions. Recently, Zhang et al. [ZCL+23] made a breakthrough by proposing the first PSU with linear complexity. Then, Chen et al. [CZZ+24] proposed an improved linear PSU protocol from the DDH assumption. However, when considering the unbalanced setting, all the aforementioned protocols require communication overhead that is at least linearly with the size of large set, resulting in notable performance degradation. For the first time, Jia et al. [JSZ+22] considered the PSU protocol in the unbalanced setting. However, their protocol suffers from two primary drawbacks. Firstly, their protocol still requires communication complexity linear in the larger set size. Secondly, their protocol did not achieve standard security and may leak the intersection size to the sender. Tu et al. [TCLZ23] proposed the first unbalanced PSU protocol with standard security, which also achieved sublinear communication complexity in the size of the larger set. Their main idea is to transform the FHE-based unbalanced private set intersection (PSI) protocol [CLR17, CHLR18, CMdG+21] to the PSU scenarios. They introduced a different polynomial randomization method, and used a newly introduced protocol called permuted matrix private equality test (pm-PEQT) to address potential leakage issues. However, due to the use of partition techniques, their concrete unbalanced PSU protocol had a large constant term in asymptotic complexity.

2 Preliminaries

2.1 Notation

We use κ and λ to denote the computational and statistical security parameters, respectively. We use $[m, n]$ to denote the set $\{m, m+1, \ldots, n\}$ and $[n]$ is shorthand for the case $m = 1$. For a bit string v we let v_i denote the ith bit. We use the abbreviation PPT to denote probabilistic polynomial-time. We say that a function f is negligible in κ if it vanishes faster than the inverse of any polynomial in κ, and write it as $f(\kappa) = \mathsf{negl}(\kappa)$. By $a \leftarrow A$, we denote that a is randomly selected from the set A, $a \leftarrow \mathsf{A}(x)$ denotes that a is the output of the randomized algorithm A on input x, and $a := b$ denotes that a is assigned by b. We denote the concatenation of string x with string y by $x||y$.

2.2 Security Model

We use the standard security definition for two-party computation [Gol04, Lin17] in this work. Similar to most previous protocols for PSU, our work operates in the *semi-honest model*.

Let $\mathcal{F}$ be a functionality between a sender $\mathcal{S}$ with input X and a receiver $\mathcal{R}$ with input Y. Let Π be a two-party protocol for computing $\mathcal{F}$.

Semi-honest Security. Let $\mathsf{view}^{\Pi}_{\mathcal{P}}(X, Y)$ be the views of party $\mathcal{P}$ ($\mathcal{P} \in \{\mathcal{S}, \mathcal{R}\}$) in the protocol, which consists of $\mathcal{P}$'s input, randomness tape, and received messages during the protocol. Let $\mathsf{output}(X, Y)$ be the output of both parties in the protocol.

Definition 1. *A protocol Π is said to securely compute functionality $\mathcal{F}$ against semi-honest $\mathcal{P}$ if for every PPT adversary $\mathcal{A}$ that corrupting $\mathcal{P}$, there exists a PPT simulator $\mathsf{Sim}_{\mathcal{P}}$ such that for all inputs X and Y,*

$$\{\mathsf{view}^{\Pi}_{\mathcal{P}}(X, Y), \mathsf{output}(X, Y)\} \approx_c \{\mathsf{Sim}_{\mathcal{P}}(In(\mathcal{P}), \mathcal{F}(X, Y)), \mathcal{F}(X, Y)\}$$

where $In(\mathcal{P})$ denotes the input of $\mathcal{P}$.

2.3 Oblivious Transfer

Oblivious transfer (OT) [Rab05] is an important cryptographic primitive used in various multiparty computation protocols. The most commonly used variant is the 1-out-of-2 OT. It allows a sender with two inputs (x_0, x_1) and a receiver with a bit $b \in \{0, 1\}$ to engage in a protocol where the receiver learns x_b, and neither party learns any additional information. We give the formal definition of 1-out-of-2 OT functionality in Figure 2.

Parameters: Sender $\mathcal{S}$, Receiver $\mathcal{R}$, message length κ
Functionality:

- Wait for input $b \in \{0, 1\}$ from the receiver $\mathcal{R}$.
- Wait for input (x_0, x_1) from the sender $\mathcal{S}$.
- Give x_b to the receiver $\mathcal{R}$.

Fig. 2: 1-out-of-2 Oblivious Transfer Functionality $\mathcal{F}_{\mathsf{ot}}$

2.4 Oblivious PRF

An oblivious PRF (OPRF) [FIPR05] is a 2-party protocol in which the sender learns (or chooses) a random PRF key k and the receiver learns the PRF output $F_k(q_1), \ldots, F_k(q_n)$ on its inputs $q_1, \ldots, q_n$. We describe the ideal functionality for an OPRF in Figure 3.

Parameters: Sender $\mathcal{S}$, Receiver $\mathcal{R}$, a PRF F.
Functionality:

- Wait for input $\{q_1, \ldots, q_n\}$ from the receiver $\mathcal{R}$.
- Sample a random PRF key k and give it to the sender $\mathcal{S}$.
- Give $\{F_k(q_1), \ldots, F_k(q_n)\}$ to the receiver $\mathcal{R}$.

Fig. 3: Oblivious PRF Functionality $\mathcal{F}_{\mathsf{oprf}}$

Parameters: Sender $\mathcal{S}$, Receiver $\mathcal{R}$, vector length n.
Functionality:

- Wait for input π from the receiver $\mathcal{R}$.
- Wait for input $\vec{x} = (x_1, \ldots, x_n) \in (\{0,1\}^l)^n$ from the sender $\mathcal{S}$.
- Pick random $a_i \leftarrow \{0,1\}^l$ and compute $b_i = a_i \oplus x_{\pi(i)}$ for $i \in [n]$.
- Give $\vec{a} = (a_1, \ldots, a_n)$ to the sender $\mathcal{S}$ and give $\vec{b} = (b_1, \ldots, b_n)$ to the receiver $\mathcal{R}$.

Fig. 4: Permute + Share Functionality $\mathcal{F}_{\mathsf{ps}}$

2.5 Permute + Share

The Permute + Share [MS13, CGP20] functionality works as follows. The sender inputs a vector $\vec{x} = (x_1, \ldots, x_n)$ and the receiver inputs a permutation π over $[n]$. As a result, the parties obtain the additive shares of $\pi(\vec{x}) = (x_{\pi(1)}, \ldots, x_{\pi(n)})$, that is, the sender receives $\vec{a} = (a_1, \ldots, a_n)$ and the receiver receives $\vec{b} = (b_1, \ldots, b_n)$, where $a_i \oplus b_i = x_{\pi(i)}$ for $i \in [n]$. The formal definition of Permute + Share functionality is given in Figure 4.

2.6 Oblivious Key-Value Stores

The Oblivious Key-Value Store (OKVS) [PRTY20, GPR+21, ZCL+23] is a data structure that maps a set of keys to corresponding values. The obliviousness means that when the values are randomly selected, the distribution of the data structure is independent from the key set. The formal definition is as follows:

Definition 2 (Oblivious Key-Value Store). *An OKVS is parameterized by a set $\mathcal{K}$ of keys, a set $\mathcal{V}$ of values, and consists of two algorithms:*

- $\mathsf{Encode}(\{(x_1, y_1), \ldots, (x_n, y_n)\})$*: on input key-value pairs $\{(x_i, y_i)\}_{i \in [n]} \subseteq \mathcal{K} \times \mathcal{V}$, outputs an object D (or, with statistically small probability, an error $\perp$).*
- $\mathsf{Decode}(D, x)$ *: on input D and a key x, outputs a value $y \in \mathcal{V}$.*

Correctness. For all $A \subseteq \mathcal{K} \times \mathcal{V}$ with distinct keys:

$$(x, y) \in A \text{ and } \perp \neq D \leftarrow \mathsf{Encode}(A) \Longrightarrow \mathsf{Decode}(D, x) = y$$

Obliviousness. For all distinct $\{x_1^0, \ldots, x_n^0\}$ and $\{x_1^1, \ldots, x_n^1\}$, if Encode does not output $\perp$ for $\{x_1^0, \ldots, x_n^0\}$ or $\{x_1^1, \ldots, x_n^1\}$, then the following distributions are computationally indistinguishable:

$$\{D | y_i \leftarrow \mathcal{V}, i \in [n], \mathsf{Encode}((x_1^0, y_1), \ldots, (x_n^0, y_n))\}$$
$$\{D | y_i \leftarrow \mathcal{V}, i \in [n], \mathsf{Encode}((x_1^1, y_1), \ldots, (x_n^1, y_n))\}$$

In addition, our protocols also require OKVS to meet the *Randomness*[ZCL+23], *Linearity*[GPR+21] and *Sparsity*[HLP+24].
Randomness. For any $A = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ and for any $x^* \notin \{x_1, \ldots, x_n\}$, the output of $\mathsf{Decode}(D, x^*)$ is statistically indistinguishable to that of uniform distribution over $\mathcal{V}$, where $D \leftarrow \mathsf{Encode}(A)$.

Linearity. An OKVS is *linear* (over a field $\mathbb{F}$) if $\mathcal{V} = \mathbb{F}$ (i.e., "values" are elements of $\mathbb{F}$), the output of Encode is a vector D in $\mathbb{F}^m$, and the Decode function is defined as: $\mathsf{Decode}(D, x) = \langle \mathsf{row}(x), D \rangle := \sum_{j=1}^{m} \mathsf{row}(x)_j D_j$ for some function $\mathsf{row} : \mathcal{K} \to \mathbb{F}^m$. Hence Decode is a linear map from $\mathbb{F}^m$ to $\mathbb{F}$.
Sparsity. An OKVS is *sparse* if the output D of Encode can be structured as $D = D_0 \| D_1$ with $|D_1| = o(|D_0|)$, and for any $x \in \mathcal{K}$, $\mathsf{Decode}(D, x) := \langle \mathsf{spa}(x) \| \mathsf{den}(x), D_0 \| D_1 \rangle$, where two mappings $\mathsf{spa} : \mathcal{K} \to \{0,1\}^{|D_0|}$ outputs a sparse binary vector with a constant weight w and $\mathsf{den} : \mathcal{K} \to \{0,1\}^{|D_1|}$ outputs a dense binary vector. For convenience, let $|D_0| = s, |D_1| = d$, and define $\mathsf{pos}_j : \mathcal{K} \to [s]$ be the function such that $\mathsf{pos}_j(x)$ denotes the j-th position of 1 in $\mathsf{spa}(x)$. Note that $\mathsf{row}(x) = \mathsf{spa}(x) || \mathsf{den}(x)$ for any $x \in \mathcal{K}$.

Due to the extensive use of OKVS in the private set operation (PSO) protocols, many recent works have made significant improvements to the construction of OKVS [GPR+21, RS21, RR22, BPSY23]. And all these efficient OKVS schemes satisfy randomness, linearity and sparsity we defined above.

2.7 Cuckoo Hashing

Cuckoo hashing was introduced by Pagh and Rodler in [PR04]. In this hashing scheme, there are α hash functions $h_1, \ldots, h_\alpha$ used to map n items into $\rho = (1 + \epsilon)n$ bins and a stash, where $\epsilon > 0$ is a constant. We denote the i-th bin as $\mathcal{B}_i$. The Cuckoo hashing can guarantee that there is only one item in each bin, and the approach to avoid collisions is as follows: An element x is inserted into a bin $\mathcal{B}_{h_1(x)}$. Any prior items z of $\mathcal{B}_{h_1(x)}$ are evicted to a new bin $\mathcal{B}_{h_i(z)}$, using h_i to determine the new bin location, where $h_i(z) \neq h_1(x)$ for $i \in [\alpha]$. The procedure is repeated until no more evictions are necessary, or until a threshold number of relocations has been performed. In the latter case, the last element is put in a special stash. According to the empirical analysis in [PSZ18], we can adjust the values of α and ϵ to reduce the stash size to 0 while achieving a hashing failure probability of $2^{-\lambda}$.

2.8 Re-randomizable Public-Key Encryption

A re-randomizable Public-Key Encryption (ReRand-PKE) scheme is a tuple of five algorithms:

- $\mathsf{Setup}(1^\kappa)$: on input the security parameter κ, outputs public parameters pp, which include the description of the plaintext and ciphertext space M, C.
- $\mathsf{KeyGen}(pp)$: on input public parameter pp, outputs a keypair (pk, sk).
- $\mathsf{Enc}(pk, m)$: on input a public key pk and a plaintext $m \in M$, outputs a ciphertext $c \in C$.
- $\mathsf{Dec}(sk, c)$: on input a secret key sk and a ciphertext $c \in C$, outputs a plaintext $m \in M$ or an error symbol $\perp$.
- $\mathsf{ReRand}(pk, c)$: on input a public key pk and a ciphertext $c \in C$, outputs another ciphertext $c' \in C$.

Correctness. For any $pp \leftarrow \mathsf{Setup}(1^\kappa)$, any $(pk, sk) \leftarrow \mathsf{KeyGen}(pp)$, any $m \in M$, any $c \leftarrow \mathsf{Enc}(pk, m)$, and any $c' \leftarrow \mathsf{ReRand}(pk, c)$, it holds that $\mathsf{Dec}(sk, c) = \mathsf{Dec}(sk, c') = m$.
Indistinguishability. For any $pp \leftarrow \mathsf{Setup}(1^\kappa)$, any $(pk, sk) \leftarrow \mathsf{KeyGen}(pp)$, and any $m \in M$, the distribution $c_0 \leftarrow \mathsf{Enc}(pk, m)$ and the distribution $c_1 \leftarrow \mathsf{ReRand}(pk, c_0)$ are identical.
Security. We require the PKE scheme satisfying *single-message multi-ciphertext pseudorandomness* [ZCL+23]. Formally, a PKE scheme is single-message multi-ciphertext pseudorandom if for any PPT $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:

$$\left| \Pr\left[\beta = \beta' : \begin{array}{l} pp \leftarrow \mathsf{Setup}(1^\kappa); \\ (pk, sk) \leftarrow \mathsf{KeyGen}(pp); \\ (m, state) \leftarrow \mathcal{A}_1(pp, pk); \\ \beta \xleftarrow{\text{R}} \{0,1\}; \\ \text{ for } i \in [n] : c^*_{i,0} \leftarrow \mathsf{Enc}(pk, m), c^*_{i,1} \xleftarrow{\text{R}} C; \\ \beta' \leftarrow \mathcal{A}_2(pp, state, \{c^*_{i,\beta}\}_{i \in [n]}) \end{array} \right] - \frac{1}{2} \right|$$

is negligible in κ.

2.9 Batch Private Information Retrieval

In a Batch Private Information Retrieval (BatchPIR) scheme [CGN98, IKOS04, ACLS18, MR23, LLWR24], the client wants to privately retrieve a batch of b entries from the server's dataset D of size n. A BatchPIR scheme consists of three algorithms, all taking the computational security parameter κ as an implicit input:

- $\mathsf{Query}(I) \to (\mathsf{qu}, \mathsf{st})$: on input a set of distinct indexes $I = \{i_1, \dots, i_b\} \in ([n])^b$, outputs a query qu and a private state st including the index set.
- $\mathsf{Answer}(D, \mathsf{qu}) \to \mathsf{ans}$: on input the database D and the query qu, outputs an answer ans.
- $\mathsf{Recover}(\mathsf{st}, \mathsf{ans}) \to \{D_1, \dots, D_b\}$: given the state st and the answer ans, outputs a batch of entries $\{D_1, \dots, D_b\}$.

Correctness. A BatchPIR is correct if for any dataset D and all distinct inputs $I = \{i_1, \dots, i_b\}$, it holds that

$$\mathsf{Recover}(\mathsf{st}, \mathsf{Answer}(D, \mathsf{qu})) = \{D[i_1], \dots, D[i_b]\},$$

where $(\mathsf{st}, \mathsf{qu}) \leftarrow \mathsf{Query}(I)$.
Query privacy. The client's query qu reveals no information about the query indexes. Formally, a BatchPIR scheme satisfies *Query privacy* if, for all PPT adversaries $\mathcal{A}$ and all distinct batch query sets I_1, I_2 with $|I_1| = |I_2|$,

$$\begin{aligned} &\Pr[\mathcal{A}(\mathsf{qu}) = 1 \mid (\mathsf{st}, \mathsf{qu}) \leftarrow \mathsf{Query}(I_1)] \\ -&\Pr[\mathcal{A}(\mathsf{qu}) = 1 \mid (\mathsf{st}, \mathsf{qu}) \leftarrow \mathsf{Query}(I_2)] \le \mathsf{negl}(\kappa). \end{aligned}$$

2.10 Private Set Union

PSU is a special case of secure two-party computation. The ideal functionality for PSU is given in Figure 5.

Parameters: Sender $\mathcal{S}$, Receiver $\mathcal{R}$, set sizes m and n.
Functionality:

- Wait for input $X = \{x_1, \dots, x_m\} \subset \{0,1\}^*$ from the sender $\mathcal{S}$.
- Wait for input $Y = \{y_1, \dots, y_n\} \subset \{0,1\}^*$ from the receiver $\mathcal{R}$.
- Give output $X \cup Y$ to the receiver $\mathcal{R}$.

Fig. 5: Private Set Union Functionality $\mathcal{F}_{\mathsf{psu}}$

3 Unbalanced Private Set Union from OPRF and Permuted PEQT

In this section, we give our first construction of unbalanced PSU protocol based on OPRF and Permuted Private EQuality Test (p-PEQT).

3.1 Permuted Private Equality Test

We give the formal definition of p-PEQT functionality in Figure 6. In this functionality, the sender inputs a vector $\vec{r'} = (r'_1, \dots, r'_n)$ and a permutation π over $[n]$, while the receiver inputs a vector $\vec{r} = (r_1, \dots, r_n)$. The functionality returns $\vec{b} = (b_1, \dots, b_n)$ to the receiver, where $b_i = 1$ if and only if $r_{\pi(i)} = r'_{\pi(i)}$. In other words, the receiver obtains the permuted result indicating whether its inputs match those of the sender.

The p-PEQT could be viewed as a natural generalization of permuted *matrix* PEQT (pm-PEQT) [TCLZ23]. In pm-PEQT, the inputs to be compared by both parties are matrices, and the sender's other input is a matrix permutation (the row-column relationship of any two elements remains unchanged before and after the permutation), while p-PEQT only requires the inputs of both parties to be vectors (note that any $m \times n$ matrix can be represented as a length-mn vector), and there is no limitation for the sender's input permutation. Therefore, the pm-PEQT [TCLZ23] is actually a special case of p-PEQT.

Parameters: Sender $\mathcal{S}$, Receiver $\mathcal{R}$, vector length n
Functionality:

- Wait for input $\vec{r'} = \{r'_1, \ldots, r'_n\} \in (\{0,1\}^l)^n$ and π from the sender $\mathcal{S}$.
- Wait for input $\vec{r} = \{r_1, \ldots, r_n\} \in (\{0,1\}^l)^n$ from the receiver $\mathcal{R}$.
- For $i \in [n]$, set $b_i = 1$ if and only if $r_{\pi(i)} = r'_{\pi(i)}$, otherwise, set $b_i = 0$.
- Give $\vec{b} = (b_1, \ldots, b_n)$ to the receiver $\mathcal{R}$.

Fig. 6: Permuted Private Equality Test Functionality $\mathcal{F}_{\mathsf{p\text{-}peqt}}$

Following a similar idea of pm-PEQT [TCLZ23], we propose two p-PEQT constructions. The first is based on the DDH assumption and the second is based on the Permute+Share functionality. The main difference between our p-PEQT constructions and pm-PEQT lies in the efficiency gained by performing only a single permutation, eliminating the need for sequential row and column permutations. The formal descriptions of our two p-PEQT protocols are given in Figure 7 and Figure 8, respectively.

Theorem 1. *Assuming the DDH problem is hard in group $\mathbb{G}$, the protocol in Figure 7 securely computes $\mathcal{F}_{\mathsf{p\text{-}peqt}}$ against semi-honest adversaries in the random oracle model.*

Theorem 2. *The protocol in Figure 8 securely computes $\mathcal{F}_{\mathsf{p\text{-}peqt}}$ against semi-honest adversaries in the $(\mathcal{F}_{\mathsf{ps}}, \mathcal{F}_{\mathsf{oprf}})$-hybrid model.*

Since our two p-PEQT protocols borrow similar ideas from pm-PEQT [TCLZ23], we do not provide security proof here. For completeness, we give formal proofs of Theorem 1 and 2 in Appendix A.

3.2 Protocol Details

The formal description of our unbalanced PSU protocol from OPRF and permuted PEQT is given in Figure 9.

Correctness. For all $i \in [m_c]$, we consider the following three cases: (1) if $X^*[i] \in Y$, there is an $Y^*[i][\alpha] \in Y, \alpha \in [|Y^*[i]|]$ s.t. $X^*[i] = Y^*[i][\alpha]$. We have $r'_i = \mathsf{Decode}(D_0||D_1, X^*[i]) - q_i = \mathsf{Decode}(D_0||D_1, Y^*[i][\alpha]) - F_k(Y^*[i][\alpha]) = r_i$, according to the correctness of OPRF, OKVS and BatchPIR. Therefore, we have $b_{\pi^{-1}(i)} = 1$ and the receiver learns $\perp$ in the final OT, according to the correctness of p-PEQT and OT. (2) if $X^*[i] \notin Y$ and $X^*[i] \neq d$, we have $r'_i = \mathsf{Decode}(D_0||D_1, X^*[i]) - q_i$, where $q_i = F_k(X^*[i])$ is pseudorandom. By setting $l = \lambda + \log|Y^*[i]|$, a union bound shows probability of $r'_i = r_i$ is negligible $2^{-\lambda}$. As a result, the receiver learns $b_{\pi^{-1}(i)} = 0$ with overwhelming probability and receives $X^*[i]$ in OT. (3) if $X^*[i] \notin Y$ and $X^*[i] = d$, we have r'_i is randomly selected by the sender, which equals r_i with negligible probability. As a result, the receiver learns $b_{\pi^{-1}(i)} = 0$ with overwhelming probability and receives d in OT. Note that the receiver throws dummy item d away from the output. Overall, the receiver learns exactly the item in $Y \setminus X$, and the set union $X \cup Y = (Y \setminus X) \cup Y$.

Theorem 3. *Given a BatchPIR with query privacy and a sparse OKVS scheme, the protocol in Figure 9 securely computes $\mathcal{F}_{\mathsf{psu}}$ against semi-honest adversaries in the $(\mathcal{F}_{\mathsf{oprf}}, \mathcal{F}_{\mathsf{p\text{-}peqt}}, \mathcal{F}_{\mathsf{ot}})$-hybrid model.*

Proof. Due to space limitation, we only sketch here the simulators for the two cases of corrupt $\mathcal{S}$ and corrupt $\mathcal{R}$, and the full proof (via hybrid arguments) is deferred to Appendix B.

Parameters:

- Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$, vector length n.
- A group $\mathbb{G}$ with generator g and order p, and the DDH problem is hard in $\mathbb{G}$.
- Random oracle $H : \{0,1\}^l \to \mathbb{G}$.

Input of $\mathcal{S}$: $\vec{r}' = (r'_1, \dots, r'_n) \in (\{0,1\}^l)^n, \pi$
Input of $\mathcal{R}$: $\vec{r} = (r_1, \dots, r_n) \in (\{0,1\}^l)^n$
Protocol:

1. The receiver $\mathcal{R}$ selects a random $a \leftarrow \mathbb{Z}_p$ and computes $v_i := H(r_i)^a$ for $i \in [n]$. Then, $\mathcal{R}$ sends $\{v_i\}_{i\in[n]}$ to the sender $\mathcal{S}$.
2. $\mathcal{S}$ selects a random $b \leftarrow \mathbb{Z}_p$ and computes $v'_i := H(r'_{\pi(i)})^b, \bar{v}_i = (v_{\pi(i)})^b$ for $i \in [n]$. Then, $\mathcal{S}$ sends $\{v'_i, \bar{v}_i\}_{i\in[n]}$ to the receiver $\mathcal{R}$.
3. For $i \in [n]$, the receiver $\mathcal{R}$ defines $b_i := 1$ if and only if $v'^a_i = \bar{v}_i$, otherwise, $b_i := 0$. Then, $\mathcal{R}$ outputs $\vec{b} := (b_1, \dots, b_n)$.

Fig. 7: Permuted Private Equality Test Protocol $\Pi_{\mathsf{p\text{-}peqt}}$ from DDH Assumption

Parameters:

- Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$, vector length n
- Ideal $\mathcal{F}_{\mathsf{oprf}}$ and $\mathcal{F}_{\mathsf{ps}}$ primitives specified in Figure 3 and Figure 4 respectively.

Input of $\mathcal{S}$: $\vec{r}' = (r'_1, \dots, r'_n) \in (\{0,1\}^l)^n, \pi$
Input of $\mathcal{R}$: $\vec{r} = (r_1, \dots, r_n) \in (\{0,1\}^l)^n$
Protocol:

1. $\mathcal{S}$ and $\mathcal{R}$ invoke the Permute + Share functionality $\mathcal{F}_{\mathsf{ps}}$. The receiver $\mathcal{R}$ acts as the sender in Permute + Share with input $\vec{r}$, and learns $\vec{\alpha}$. The sender $\mathcal{S}$ acts as the receiver in Permute + Share with input π and receives $\vec{\beta}$, where $\alpha_i \oplus \beta_i = r_{\pi(i)}, i \in [n]$.
2. $\mathcal{S}$ and $\mathcal{R}$ invoke the OPRF functionality $\mathcal{F}_{\mathsf{oprf}}$. The receiver $\mathcal{R}$ acts as the receiver in OPRF with input $\{\alpha_i\}_{i\in[n]}$, and learns $\{F_k(\alpha_i)\}_{i\in[n]}$. The sender $\mathcal{S}$ acts as the sender in OPRF with no input, and receives a PRF key k.
3. The sender $\mathcal{S}$ computes $f_i := F_k(r'_{\pi(i)} \oplus \beta_i)$ for $i \in [n]$ and sends $\{f_i\}_{i\in[n]}$ to the receiver $\mathcal{R}$.
4. For $i \in [n]$, the receiver $\mathcal{R}$ defines $b_i := 1$ if and only if $f_i = F_k(\alpha_i)$, otherwise, $b_i := 0$. Then, $\mathcal{R}$ outputs $\vec{b} := (b_1, \dots, b_n)$.

Fig. 8: Permuted Private Equality Test Protocol $\Pi_{\mathsf{p\text{-}peqt}}$ from Permute + Share

Corrupt Sender: The simulator invokes the OPRF receiver's simulator with random values to simulate the sender's view in the OPRF protocol. Then, the simulator picks a random sparse OKVS $D = D_0||D_1$, and executes as an honest receiver with D to simulate the steps that involve OKVS, i.e., the steps 5-8. Finally, the simulator invokes the simulators of p-PEQT and OT to obtain the sender's view in p-PEQT and OT protocols.

Briefly, this simulation is indistinguishable for the following reasons: the one-time pads generated by OPRF are computationally indistinguishable from random values, and then by the obliviousness of OKVS, D is distributed uniformly. The security of underlying OPRF, p-PEQT, and OT protocols also guarantees that the output of the simulator is indistinguishable from the real view.

Corrupt Receiver: The simulator defines the set $Z := X \cup Y \setminus Y$, i.e., the set of elements that X "brings to the union", and uses dummy items to pad Z to m elements. The simulator uses Z as the sender's input to execute protocol honestly. For steps involving BatchPIR, the simulator selects some random indexes to simulate the sender's query. This simulation is indistinguishable from the real view by the security of underlying OPRF, p-PEQT, and OT protocols and the query privacy of the BatchPIR scheme.

Parameters:

- Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$, set size m and n, where $m \ll n$.
- Ideal $\mathcal{F}_{\mathsf{ot}}$, $\mathcal{F}_{\mathsf{oprf}}$ and $\mathcal{F}_{\mathsf{p\text{-}peqt}}$ primitives specified in Figure 2, Figure 3, and Figure 6 respectively.
- A sparse OKVS scheme $(\mathsf{Encode}, \mathsf{Decode})$ with key space $\mathcal{K} = \{0,1\}^*$ and value space $\mathcal{V} = \mathbb{F}$. Let $\mathsf{spa} : \mathcal{K} \to \{0,1\}^s$ and $\mathsf{den} : \mathcal{K} \to \{0,1\}^d$ be the random function used in Encode, and $\mathsf{pos}_j : \mathcal{K} \to [s]$ be the function such that $\mathsf{pos}_j(y)$ denotes the j-th position of 1 in $\mathsf{spa}(y)$. Let w denote the Hamming weight of the sparse part.

Input of $\mathcal{S}$: $X = \{x_1, \ldots, x_m\} \subset \{0,1\}^*$
Input of $\mathcal{R}$: $Y = \{y_1, \ldots, y_n\} \subset \{0,1\}^*$
Protocol:

1. The sender $\mathcal{S}$ inserts set X into the Cuckoo hash table, and fills empty bins with the dummy item d. Let $m_c = (1+\epsilon)m$ denote the length of the Cuckoo hash table. $\mathcal{S}$ denotes the filled Cuckoo hash table as X^* and the item in i-th bin as $X^*[i]$ for $i \in [m_c]$. Let $\sigma : [m] \to [m_c]$ denote the injective function such that $\sigma(1), \ldots, \sigma(m)$ are the non-dummy item bins of X^*.
2. The receiver $\mathcal{R}$ inserts set Y into the simple hash table. Then $\mathcal{R}$ denotes the simple hash table as Y^*, the set of items in i-th bin as $Y^*[i]$ and the α-th item in $Y^*[i]$ as $Y^*[i][\alpha]$ for $i \in [m_c], \alpha \in [|Y^*[i]|]$.
3. The receiver $\mathcal{R}$ selects random $r_i \in \mathbb{F}$ for each bin $i \in [m_c]$. Let $\vec{r} := (r_1, \ldots, r_{m_c})$.
4. $\mathcal{S}$ and $\mathcal{R}$ invoke the OPRF functionality $\mathcal{F}_{\mathsf{oprf}}$. The receiver $\mathcal{R}$ acts as the sender in OPRF with no input, and learns a PRF key k. The sender $\mathcal{S}$ acts as the receiver in OPRF with input $\{X^*[\sigma(j)]\}_{j\in[m]}$ and receives $Q := \{q_1, \ldots, q_m\}$, where $q_j = F_k(X^*[\sigma(j)]), j \in [m]$.
5. The receiver $\mathcal{R}$ defines $\mathcal{P} := \{(Y^*[i][\alpha], r_i + F_k(Y^*[i][\alpha]))\}_{i\in[m_c],\alpha\in[|Y^*[i]|]}$ and computes the sparse OKVS $D_0||D_1 := \mathsf{Encode}(\mathcal{P}) \in \mathbb{F}^s \times \mathbb{F}^d$.
6. The sender $\mathcal{S}$ defines $I := \{\mathsf{pos}_t(X^*[\sigma(j)])\}_{t\in[w],j\in[m]}$. Then, $\mathcal{S}$ computes $(\mathsf{st}, \mathsf{qu}) \leftarrow \mathsf{Query}(I)$ and sends qu to the receiver $\mathcal{R}$.
7. The receiver $\mathcal{R}$ computes $\mathsf{ans} \leftarrow \mathsf{Answer}(D_0, \mathsf{qu})$ and sends (ans, D_1) to the sender $\mathcal{S}$.
8. The sender $\mathcal{S}$ computes $\{D_{t,j}\}_{t\in[w],j\in[m]} := \mathsf{Recover}(\mathsf{st}, \mathsf{ans})$, where $D_{t,j} = D_0[\mathsf{pos}_t(X^*[\sigma(j)])]$. For $j \in [m]$, $\mathcal{S}$ computes $r'_j := \sum_{t\in[w]} D_{t,j} + \langle D_1, \mathsf{den}(X^*[\sigma(j)])\rangle - q_j$. $\mathcal{S}$ also picks random $r'_i \leftarrow \mathbb{F}$ for $i \in [m_c] \setminus \{\sigma(j)\}_{j\in[m]}$. Let $\vec{r}' := (r'_1, \ldots, r'_{m_c})$.
9. The sender $\mathcal{S}$ selects a random permutation π over $[m_c]$. Then, $\mathcal{S}$ and $\mathcal{R}$ invoke the permuted PEQT functionality $\mathcal{F}_{\mathsf{p\text{-}peqt}}$. The receiver $\mathcal{R}$ acts as the receiver with input $\vec{r}$ and receives $\vec{b} = (b_1, \ldots, b_{m_c})$. The sender $\mathcal{S}$ acts as the sender with input $(\vec{r}', \pi)$ and receives nothing. Note that $b_i = 1$ if and only if $r_{\pi(i)} = r'_{\pi(i)}$.
10. The receiver $\mathcal{R}$ initialize set $Z := \{\}$. Then, for $i \in [m_c]$, $\mathcal{S}$ and $\mathcal{R}$ invoke the OT functionality $\mathcal{F}_{\mathsf{ot}}$: $\mathcal{S}$ acts as sender with input $(X^*[\pi(i)], \perp)$ and receives nothing. $\mathcal{R}$ acts as receiver with input b_i and receives z_i. $\mathcal{R}$ sets $Z := Z \cup \{z_i\}$ if $z_i \neq d$.
11. The receiver $\mathcal{R}$ outputs $Y \cup Z$.

Fig. 9: Unbalanced PSU Protocol Π_{psu} from OPRF and p-PEQT

4 Unbanlanced Private Set Union from Re-randomizable PKE

In this section, we give our second construction of unbalanced PSU protocol based on ReRand-PKE. The formal description is given in Figure 9.

Correctness. For all $j \in [m]$, we consider the following two cases: (1) if $x_j \in Y$, there is an $y_i \in Y, i \in [n]$ s.t. $x_j = y_i$. According to the correctness of ReRand-PKE, OKVS and BatchPIR, we have $s'_j = \mathsf{Decode}(D_0||D_1, x_j) = \mathsf{Decode}(D_0||D_1, y_i) = s_i$. Since $s_i = \mathsf{Enc}(pk, 0)$, we have $\mathsf{Dec}(sk, s_i) = 0$, which means $b_j = 1$. So the receiver learns $\perp$ in the final OT. (2) if $x_j \notin Y$, from the random decoding property of OKVS, $s'_j = \mathsf{Decode}(D_0||D_1, x_j)$ is a random ciphertext, resulting in s'_j is not the encryption of 0 with overwhelming probability. Since the size ciphertext space is at least $2^\kappa \geq 2^{\lambda+\log m}$, the union bound guarantees that for all $x_j \notin Y$, the probability that there exists an s'_j s.t. $\mathsf{Dec}(sk, s'_j) = 0$ is negligible. As a result, the receiver learns $b_j = 0$ with overwhelming probability and receives x_j in OT. Overall, the receiver learns exactly the item in $Y \setminus X$, and the set union $X \cup Y = (Y \setminus X) \cup Y$.

Parameters:

- Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$, set size m and n, where $m \ll n$.
- Ideal $\mathcal{F}_{\mathsf{ot}}$ primitive specified in Figure 2.
- A sparse OKVS scheme $(\mathsf{Encode}, \mathsf{Decode})$ with key space $\mathcal{K} = \{0,1\}^*$ and value space $\mathcal{V} = \mathbb{F}$. Let $\mathsf{spa} : \mathcal{K} \to \{0,1\}^s$ and $\mathsf{den} : \mathcal{K} \to \{0,1\}^d$ be the random function used in Encode, and $\mathsf{pos}_j : \mathcal{K} \to [s]$ be the function such that $\mathsf{pos}_j(y)$ denotes the j-th position of 1 in $\mathsf{spa}(y)$. Let w denote the Hamming weight of the sparse part.
- A ReRand-PKE scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReRand})$ that satisfies single-message multi-ciphertext pseudorandomness.

Input of $\mathcal{S}$: $X = \{x_1, \ldots, x_m\} \subset \{0,1\}^*$
Input of $\mathcal{R}$: $Y = \{y_1, \ldots, y_n\} \subset \{0,1\}^*$
Protocol:

1. The receiver $\mathcal{R}$ generates a random key pair $pp \leftarrow \mathsf{Setup}, (pk, sk) \leftarrow \mathsf{KeyGen}(pp)$, a randomness set $R = \{r_1, \ldots, r_n\}$ and computes $s_i := \mathsf{Enc}(pk, 0; r_i)$ for $i \in [n]$.
2. The receiver $\mathcal{R}$ computes the sparse OKVS $D_0 || D_1 := \mathsf{Encode}((y_1, s_1), \ldots, (y_n, s_n)) \in \mathbb{F}^s \times \mathbb{F}^d$.
3. The sender $\mathcal{S}$ defines $I := \{\mathsf{pos}_t(x_j)\}_{t \in [w], j \in [m]}$. Then, $\mathcal{S}$ computes $(\mathsf{st}, \mathsf{qu}) \leftarrow \mathsf{Query}(I)$ and sends qu to the receiver $\mathcal{R}$.
4. The receiver $\mathcal{R}$ computes $\mathsf{ans} \leftarrow \mathsf{Answer}(D_0, \mathsf{qu})$ and sends (ans, pk, D_1) to the sender $\mathcal{S}$.
5. The sender $\mathcal{S}$ computes $\{D_{t,j}\}_{t \in [w], j \in [m]} := \mathsf{Recover}(\mathsf{st}, \mathsf{ans})$, where $D_{t,j} = D_0[\mathsf{pos}_t(x_j)]$. For $j \in [m]$, $\mathcal{S}$ computes $s'_j := \sum_{t \in [w]} D_{t,j} + \langle D_1, \mathsf{den}(x_j) \rangle$.
6. The sender $\mathcal{S}$ selects random $r'_1, \ldots, r'_m$ and computes $\bar{s_j} := \mathsf{ReRand}(pk, s'_j; r'_j)$ for $j \in [m]$.
7. The sender $\mathcal{S}$ sends $\bar{s_1}, \ldots, \bar{s_m}$ to $\mathcal{R}$.
8. The receiver $\mathcal{R}$ sets $b_j = 1$ if and only if $\mathsf{Dec}(sk, \bar{s_j}) = 0$ for $j \in [m]$.
9. The receiver $\mathcal{R}$ initialize set $Z := \{\}$. Then, for $j \in [m]$, $\mathcal{S}$ and $\mathcal{R}$ invoke the OT functionality $\mathcal{F}_{\mathsf{ot}}$: $\mathcal{S}$ acts as sender with input $(x_j, \perp)$ and receives nothing. $\mathcal{R}$ acts as receiver with input b_j. $\mathcal{R}$ receives z_j from OT and sets $Z = Z \cup \{z_j\}$.
10. The receiver $\mathcal{R}$ outputs $Y \cup Z$.

Fig. 10: Unbalanced PSU Protocol Π_{psu} from ReRand-PKE

Theorem 4. *Given a BatchPIR scheme with query privacy, a ReRand-PKE scheme with single-message multi-ciphertext pseudorandomness property and a sparse OKVS scheme, the protocol in Figure 10 securely computes $\mathcal{F}_{\mathsf{psu}}$ against semi-honest adversaries in the $\mathcal{F}_{\mathsf{ot}}$-hybrid model.*

Proof. Due to space limitation, we only sketch here the simulators for the two cases of corrupt $\mathcal{S}$ and corrupt $\mathcal{R}$, and the full proof (via hybrid arguments) is deferred to Appendix C.

Corrupt Sender: The simulator picks a random sparse OKVS $D = D_0 || D_1$, and executes as an honest receiver with D to simulate the steps that involve OKVS, i.e., the steps 1-5. Then, the simulator invokes the simulator of OT to obtain the sender's view in OT protocol.

Briefly, this simulation is indistinguishable for the following reasons: the single-message multi-ciphertext pseudorandomness of the ReRand-PKE scheme ensures that value (ciphertext) is indistinguishable from random, and then by the obliviousness of OKVS, D is distributed uniformly. The security of underlying OT protocol also guarantees that the output of the simulator is indistinguishable from the real view.

Corrupt Receiver: Similar to theorem 3, the simulator defines the set $Z := X \cup Y \setminus Y$, and pads Z to m elements. The simulator uses Z as the sender's input to execute protocol honestly. For steps involving BatchPIR, the simulator selects some random indexes to simulate the sender's query. This simulation is indistinguishable from the real view by the security of the underlying OT protocol and the query privacy of the BatchPIR scheme.

4.1 Optimization

The encoding algorithm of OKVS actually involves solving a system of linear equations in the value space. In the protocol description in Figure 10, the value space is the ciphertext space of the ReRand-PKE scheme, typically corresponding to a certain group $\mathbb{G}$. In our implementation, we use the ElGamal encryption scheme [Gam85] as the ReRand-PKE instantiation. The ciphertext space of the ElGamal encryption scheme is $\mathbb{G}^2$, where DDH assumption holds in $\mathbb{G}$ and the order of $\mathbb{G}$ is a prime p. Solving a linear system in $\mathbb{G}$ incurs a high cost. Therefore, we propose a "pull-down-then-lift" method. Specifically, we first compute the OKVS over $\mathbb{Z}_p$, where $\mathbb{Z}_p$ denotes the finite field of integers modulo p. Subsequently, we lift the computed OKVS from $\mathbb{Z}_p$ to the exponent. This method allows us to mitigate the computational overhead associated with solving linear systems in $\mathbb{G}$, resulting in improved efficiency. Further details are provided below.

The structure of ElGamal ciphertext $c = (g^r, m \cdot pk^r)$, where $g \in \mathbb{G}$ is the generator of group $\mathbb{G}$, $pk = g^{sk}$, and r is a random value in $\mathbb{Z}_p$. Since m is the set indicator and could be set to an arbitrary value, we have $c = (c_1, c_2) = (g^r, g^{sk \cdot r})$ by setting $m = 1$. Our encoding process first computes $d = (d_1, \ldots, d_N) := \mathsf{Encode}(x_i, r_i || sk \cdot r_i)$, and then outputs the OKVS $D = (D_1, \ldots, D_N) := (g^{d_1}, \ldots, g^{d_N})$. However, this approach may raise security concerns. One may want to know if the OKVS generated as above still satisfies obliviousness. The condition for obliviousness is that all values are randomly selected, and in our construction, values are not completely random due to the correlation between r_i and $sk \cdot r_i$. We argue that this construction actually satisfies obliviousness. The key observation is that the lift step is crucial as it can prevent a PPT adversary from "detecting" the correlation in exponential. Specifically, we split d_i in two parts, namely, (d_i^1, d_i^2), where d_i^1 corresponds to the encoding of $\{(x_i, r_i)\}$ and d_i^2 corresponds to the encoding of $\{(x_i, sk \cdot r_i)\}$. We have $d_i^2 = sk \cdot d_i^1$ from the linearity of the encoding algorithm. After lifting, the adversary only sees $pk = g^{sk}, \{g^{d_1^i}, g^{d_2^i} = g^{sk \cdot d_1^i}\}$, which exactly corresponds to a DDH tuple. Therefore, the DDH assumption guarantees generating the OKVS in a "pull-down-then-lift" manner still satisfies obliviousness. We provide a formal proof in Appendix D.

5 Implementation and Performance

We experimentally evaluate our PSU protocols and compare them with the state-of-the-art counterparts [TCLZ23]. For ease of description, we refer to each scheme as follows.

- $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ddh}}$: our OPRF + p-PEQT based construction presented in Figure 9, and the p-PEQT is built from DDH assumption, as shown in Figure 7.
- $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ps}}$: our OPRF + p-PEQT based construction presented in Figure 9, and the p-PEQT is built from Permute + Share functionality, as shown in Figure 8.
- $\mathsf{PSU}_{\mathsf{pk}}$: our ReRand-PKE based construction presented in Figure 10.

5.1 Implementation Details

We implement our protocols and fully re-implement [TCLZ23] based on mpc4j [mpc] . In this way, we can conduct a comprehensive and fair comparison in a unified platform. The complete implementation is merged into mpc4j and freely available now.

The reasons for re-implementing the protocol proposed by Tu et al. [TCLZ23] instead of directly running their implementation on our platform are as follows. Firstly, their implementation did not explore the potential benefits of using preprocessing techniques. The preprocessing technique is to divide protocol execution into two phases, namely the *setup phase* and the *online phase*. The setup phase, which typically includes key distribution, base OT, and input-independent pre-computation, only needs to be executed once. After the setup phase, the parties can then enjoy a faster online phase with their inputs. Preprocessing is a standard technique in secure multiparty computation, especially in PSO protocols [KLS$^+$17, RS21, ZCL$^+$23, KBM23]. We note that in the unbalanced setting, existing works additionally assume that the party with large set can ahead of time knows its input and further perform necessary pre-computations with this knowledge [CHLR18, CMdG$^+$21, HLP$^+$24]. In our re-implementations, we enabled [TCLZ23] to benefit from preprocessing techniques, resulting in a more efficient online phase.

Moreover, the protocols in [TCLZ23] avoid the expensive noise flooding operations by leveraging OPRF to randomize elements in each party's input set. This technique was first introduced in [CHLR18] and has been widely used in unbalanced PSO protocols. However, Tu et al. [TCLZ23] use the OT-based OPRF [KKRT16] in their implementation. As mentioned in [CHLR18], such OPRF has two main limitations: (1) it requires both parties to perform a hash-to-bin operation before OPRF evaluation, hindering the potential for integrating OPRF preprocessing into the setup phase; (2) it requires the sender to pad its simple hash bin with dummy items to prevent the receiver from inferring partial information, which makes the actual number of OPRFs executed by both parties slightly higher than the set size, causing additional overhead. We follow the recommendation of [CMdG+21] and use DH-based OPRF in our re-implementation [JL10]. The sender in such OPRF can select the key and perform arbitrary computations without the knowledge of the input from the receiver, allowing OPRF preprocessing in the setup phase. Additionally, no hash-to-bin operation is required in this OPRF, circumventing unnecessary OPRF evaluations.

Aside from the above improvement, our re-implementation of [TCLZ23] fully complies with their open-source code [APS23] . That is, the underlying components we use are exactly the same as them, with the sole difference being the adoption of DH-OPRF [JL10] for OPRF preprocessing.

The details of the underlying components in our protocol implementations are as follows.

OPRF. Two OPRFs are used in our protocols. The first is the OPRF used in our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ddh}}$ and $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ps}}$ to generate one-time pads (i.e., the step 4 in Figure 9). We opt for the DH-based OPRF [JL10], allowing the receiver to compute the PRF values of its inputs during the setup phase. For better efficiency, we use the FourQ [CL15] to speed up the scalar multiplication of elliptic curve points. The second is the OPRF used in our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ps}}$ to conduct the equality test (i.e., the step 2 in Figure 8). Here, we employ BaRK-OPRF [KKRT16], which facilitates a faster running time during the online phase.

OKVS. We implement the blazing-fast OKVS [RR22] with clustering as our sparse OKVS instantiation, and employ 3 hashing functions to create the sparse OKVS storage. Although the recently proposed RB-OKVS [BPSY23] also satisfies sparsity, its sparse part exhibits a large Hamming weight (typically ≥ 196), which significantly diminishes efficiency. Therefore, we opted not to use RB-OKVS as our instantiation. In $\mathsf{PSU}_{\mathsf{pk}}$, the underlying field of OKVS is instantiated as $\mathbb{F}_p$, while in $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ps}}$ and $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ddh}}$, the underlying field of OKVS is instantiated as $\mathbb{F}_{2^l}$.

ReRand-PKE. We use ElGamal encryption [Gam85] as the ReRand-PKE instantiation. We choose the secp256r1 implementation in OpenSSL [Ope] as the underlying elliptic curve since it provides the most efficient assembly language implementation. We pre-compute lookup tables for the generator g and the public key y in the setup phase to accelerate fixed-point multiplication in the online phase.

BatchPIR. We employ the state-of-the-art vectorized BatchPIR [MR23] as our BatchPIR instantiation. We adopt the Java Native Interface (JNI) technique to invoke the BFV homomorphic encryption [FV12] provided by Microsoft SEAL library v4.0 [SEA22] . We follow the same parameters used in the open-source codes of the vectorized BatchPIR [Vec23] . Specifically, the receiver uses stash-less cuckoo Hashing with 3 hash functions to insert the n entries into $1.2n$ bins. The receiver encodes entries in each bin in three dimensions, places them at suitable slots, and transfers the plaintext polynomial into NTT form. Vectorized BatchPIR also requires both Galois keys and relinearization keys for rotation and multiplication operations. All the above operations are done in the setup phase.

ECC. In addition to DH-based OPRF and ReRand-PKE, our DDH-based p-PEQT also requires ECC (i.e., the p-PEQT construction in Figure 7). Given that addition operations are not needed in this construction, we follow the recommendation by [CZZ+24] and introduce X25519 in Sodium [Sod] for efficient scalar multiplications.

Permute + Share. Similar to the TCLZ protocol [TCLZ23], we implement the Permute + Share functionality based on [MS13].

OT. We follow the default setting in mpc4j by choosing the base OT proposed in [NP01] and the OT extension proposed in [ALSZ13].

5.2 Experimental Setup

We conducted our experiments on a single Intel Core i9-9900K with 3.6GHz and 128GB RAM. All experiments are executed with 8 threads. We set the computational security parameter $\kappa = 128$ and the statistical security parameter $\lambda = 40$. We simulate the network connection using the Linux command tc. To evaluate the efficiency of the protocols in different network settings, we use two simulated network settings, including LAN (10Gbps bandwidth and 0.05ms RTT latency) and WAN (100Mbps, 10Mbps and 1Mbps bandwidth with 80ms RTT latency).

5.3 Performance Evaluation

We compare our protocols with the state-of-the-art unbalanced PSU protocols [TCLZ23], which comprises two protocols: TCLZ-pub and TCLZ-sym. A detailed benchmark for small set sizes $|X| = m \in \{2^4, 2^6, 2^8, 2^{10}\}$ and large set sizes $|Y| = n \in \{2^{18}, 2^{20}, 2^{22}\}$ is presented in Table 1. Additionally, Figure 11 illustrates the variation of communication/running time with small set size/bandwidth.

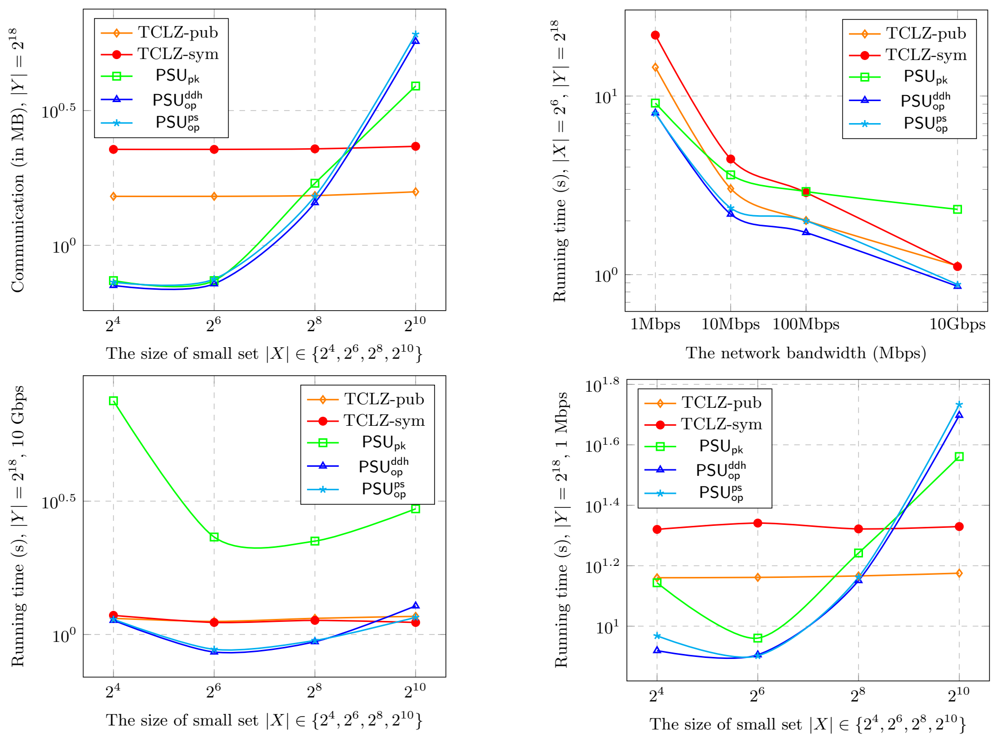

Fig. 11: Communication cost (in MB) and running time (in seconds) comparing our protocols to TCLZ-pub and TCLZ-sym [TCLZ23]. Both x and y-axis are in log scale. The top left figure shows the communication cost increases as the small set size increases. The top right figure shows the runtime decreases as the bandwidth increases. The bottom two figures show the runtime increases as the small set size increases in different bandwidths (i.e., 10 Gbps and 1 Mbps).

Communication comparison. As indicated in Table 1 and the top left figure in Figure 11, our protocols achieve the lowest online communication cost when the small set size is less than 2^8. Compared to the TCLZ protocols, the communication of our constructions is $2.1 \sim 11.8\times$ better. Notably, the communication costs of our protocols increase linearly with the size of small sets, while the communication of TCLZ protocols remains almost unchanged when the small set size is less than 2^{10}. The reason is that the TCLZ protocols use Single Instruction Multiple Data (SIMD) operations to package multiple plaintexts into one ciphertext for concurrent processing. Specifically, the TCLZ protocols employ ciphertexts containing 1638 plaintext slots. Therefore, as long as the number of bins of the sender is less than 1638 (corresponding to a set size of approximately 2^{10}), their communication remains unchanged. As a result, our protocols perform better in the extremely unbalanced setting, showcasing increasing communication efficiency ratios as the gap between the sizes of the two sets widens.

Running time comparison. The experimental results in Table 1 and Figure 11 indicate that our OPRF + p-PEQT based unbalanced PSU protocols $\mathsf{PSU_{op}^{ps}}, \mathsf{PSU_{op}^{ddh}}$ perform better in most settings, and outperform TCLZ protocols. Specifically, the running time of our $\mathsf{PSU_{op}^{ps}}$ and $\mathsf{PSU_{op}^{ddh}}$ protocols surpasses TCLZ protocols by $1.3 \sim 5.6\times$ in different bandwidth setting. For example, for $m = 2^6, n = 2^{18}$ in 1 Mbps bandwidth, our $\mathsf{PSU_{op}^{ddh}}$ requires 7.98 seconds, while TCLZ-pub requires 14.5 seconds, achieving a $1.8\times$ improvement, and TCLZ-sym requires 21.93 seconds, achieving a $2.7\times$ improvement. Our

Param.		Protocol	Comm. (MB)		Running time (s)							
					LAN		100Mbps		10Mbps		1Mbps	
n	m		Setup	Online	Setup	Online	Setup	Online	Setup	Online	Setup	Online
2^{18}	2^4	TCLZ-pub [TCLZ23]	0.76	1.52	2.98	1.15	3.67	1.99	3.93	3.11	9.71	14.46
		TCLZ-sym [TCLZ23]	0.77	2.27	2.98	1.18	4.02	2.48	4.50	4.04	10.25	20.90
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ddh}}$	19.08	0.71	11.72	1.13	15.55	2.44	29.60	2.89	174.53	8.30
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ps}}$	19.09	0.73	12.05	1.14	15.02	2.71	28.79	3.27	173.45	9.29
		Our $\mathsf{PSU}_{\mathsf{pk}}$	19.08	0.74	38.01	7.52	40.16	8.11	52.31	8.32	197.53	13.92
	2^6	TCLZ-pub [TCLZ23]	0.76	1.52	2.84	1.12	3.38	2.01	3.90	3.03	9.59	14.50
		TCLZ-sym [TCLZ23]	0.77	2.27	2.83	1.11	3.72	2.88	4.15	4.44	10.13	21.93
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ddh}}$	19.08	0.72	14.99	0.86	17.19	1.72	31.25	2.18	175.06	8.04
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ps}}$	19.09	0.75	14.00	0.88	17.72	2.00	30.94	2.36	175.34	7.98
		Our $\mathsf{PSU}_{\mathsf{pk}}$	19.08	0.74	34.88	2.32	37.57	2.92	52.14	3.62	196.42	9.13
	2^8	TCLZ-pub [TCLZ23]	0.76	1.53	2.85	1.15	3.48	1.91	3.92	3.08	9.55	14.66
		TCLZ-sym [TCLZ23]	0.77	2.28	2.87	1.13	3.89	2.25	4.11	3.77	9.95	20.98
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ddh}}$	19.08	1.44	21.78	0.94	24.58	1.88	38.30	3.42	183.08	14.15
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ps}}$	19.09	1.52	21.43	0.95	24.41	2.13	38.66	3.13	182.88	14.53
		Our $\mathsf{PSU}_{\mathsf{pk}}$	19.08	1.70	59.15	2.24	61.30	3.07	76.01	4.35	219.35	17.46
	2^{10}	TCLZ-pub [TCLZ23]	0.76	1.58	2.80	1.17	3.43	1.94	3.69	3.17	9.70	14.97
		TCLZ-sym [TCLZ23]	0.77	2.33	2.95	1.11	3.70	2.31	4.33	3.88	9.99	21.36
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ddh}}$	19.08	5.71	22.67	1.28	25.34	3.37	39.50	6.99	184.55	49.83
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ps}}$	19.09	6.07	22.17	1.16	25.39	3.17	39.45	7.43	184.40	54.09
		Our $\mathsf{PSU}_{\mathsf{pk}}$	19.08	3.90	124.79	2.96	128.02	4.21	141.98	7.05	286.85	36.41
2^{20}	2^4	TCLZ-pub [TCLZ23]	0.76	1.86	14.47	1.68	15.27	2.55	15.62	3.90	20.76	18.51
		TCLZ-sym [TCLZ23]	0.77	3.52	14.61	1.63	15.26	3.04	15.81	5.40	21.92	31.90
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ddh}}$	19.08	0.76	42.34	3.07	45.15	4.17	58.63	4.76	205.86	10.83
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ps}}$	19.09	0.77	42.09	3.10	45.77	4.59	59.81	5.26	203.93	11.92
		Our $\mathsf{PSU}_{\mathsf{pk}}$	19.08	0.88	89.39	8.84	94.72	9.40	108.37	10.71	252.10	16.82
	2^6	TCLZ-pub [TCLZ23]	0.76	1.87	14.20	1.61	14.48	2.44	15.16	3.78	21.33	17.95
		TCLZ-sym [TCLZ23]	0.77	3.52	15.23	1.55	15.42	3.63	15.64	6.19	21.37	32.77
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ddh}}$	19.08	1.45	42.53	1.39	46.21	2.24	58.51	3.34	203.39	15.14
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ps}}$	19.09	1.48	42.63	1.34	58.96	2.74	73.43	4.77	216.41	27.73
		Our $\mathsf{PSU}_{\mathsf{pk}}$	19.08	0.89	135.13	8.08	135.17	8.27	146.61	8.94	293.32	15.65
	2^8	TCLZ-pub [TCLZ23]	0.76	1.88	14.66	1.59	15.14	2.41	15.05	3.81	20.70	18.00
		TCLZ-sym [TCLZ23]	0.77	3.53	14.13	1.54	15.13	2.90	15.38	5.41	21.61	32.00
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ddh}}$	19.08	2.86	54.80	1.29	58.14	2.59	73.32	4.56	216.79	26.45
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ps}}$	19.09	2.94	55.18	1.35	58.96	2.74	73.43	4.77	216.41	27.73
		Our $\mathsf{PSU}_{\mathsf{pk}}$	19.08	2.97	138.53	3.19	136.75	3.93	151.06	6.16	299.52	29.09
	2^{10}	TCLZ-pub [TCLZ23]	0.76	1.92	14.42	1.60	15.22	2.45	15.15	3.86	21.41	18.43
		TCLZ-sym [TCLZ23]	0.77	3.58	14.26	1.55	15.21	2.97	16.02	5.48	21.48	32.55
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ddh}}$	19.08	5.76	85.49	1.93	89.58	3.71	102.54	7.90	245.92	50.95
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ps}}$	19.09	6.15	85.76	1.92	91.20	3.82	101.64	7.96	247.21	54.96
		Our $\mathsf{PSU}_{\mathsf{pk}}$	19.08	6.80	232.43	3.65	234.26	5.37	247.47	10.14	431.10	61.93
2^{22}	2^4	TCLZ-pub [TCLZ23]	0.76	3.59	60.60	3.21	60.69	4.51	60.64	6.89	67.03	34.86
		TCLZ-sym [TCLZ23]	0.77	11.13	61.39	3.02	61.74	5.28	63.20	13.31	67.48	97.47
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ddh}}$	19.08	0.94	182.08	7.40	187.18	9.94	202.34	10.08	345.77	17.38
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ps}}$	19.09	0.96	185.43	7.55	190.21	9.34	203.74	10.60	349.34	18.19
		Our $\mathsf{PSU}_{\mathsf{pk}}$	19.08	1.46	302.23	13.27	305.25	14.03	319.06	15.46	465.35	26.58
	2^6	TCLZ-pub [TCLZ23]	0.76	3.59	62.04	3.05	62.10	4.24	61.96	6.75	67.95	34.47
		TCLZ-sym [TCLZ23]	0.77	11.13	61.90	2.98	61.98	5.96	60.80	14.35	67.73	98.07
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ddh}}$	19.08	1.64	221.34	6.03	223.13	7.01	229.80	9.05	373.27	20.91
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ps}}$	19.09	1.66	216.53	6.61	217.72	7.32	235.72	8.51	380.17	21.50
		Our $\mathsf{PSU}_{\mathsf{pk}}$	19.08	2.16	424.83	11.25	417.94	11.98	442.52	13.69	584.50	29.81
	2^8	TCLZ-pub [TCLZ23]	0.76	3.60	60.62	3.26	61.03	4.39	61.71	6.89	66.99	34.31
		TCLZ-sym [TCLZ23]	0.77	11.15	62.10	3.11	61.25	5.24	62.72	13.42	68.35	98.23
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ddh}}$	19.08	5.80	181.34	4.07	181.20	5.79	193.74	9.17	342.64	52.94
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ps}}$	19.09	5.88	184.88	3.93	188.85	6.20	196.29	9.55	340.90	54.68
		Our $\mathsf{PSU}_{\mathsf{pk}}$	19.08	3.55	539.48	10.01	548.18	11.53	556.73	13.76	716.74	40.50
	2^{10}	TCLZ-pub [TCLZ23]	0.76	3.65	60.36	3.07	61.46	4.82	63.69	7.13	67.58	34.65
		TCLZ-sym [TCLZ23]	0.77	11.19	59.93	3.08	63.29	5.36	63.01	13.36	66.54	97.89
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ddh}}$	19.08	10.76	234.33	4.55	238.26	6.51	261.40	14.22	405.45	96.39
		Our $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ps}}$	19.09	11.15	236.02	4.56	239.26	6.89	254.70	14.93	409.62	99.66
		Our $\mathsf{PSU}_{\mathsf{pk}}$	19.08	11.19	551.38	9.28	554.46	13.23	567.37	18.95	706.52	103.62

Table 1: Communication cost (in MB) and running time (in seconds) comparing our protocols to TCLZ-pub and TCLZ-sym [TCLZ23]. The LAN network has 10 Gbps bandwidth and 0.05 ms RTT latency. The best result is marked in green, and the second best result is marked in blue.

ReRand-PKE based construction $\mathsf{PSU}_{\mathsf{pk}}$ exhibits less satisfactory performance due to the relatively long value length of OKVS encoding. In the $\mathsf{PSU}_{\mathsf{pk}}$ protocol, the OKVS values are the ciphertexts of ReRand-PKE, which has a length of 256 bits in our instantiation (i.e., ElGamal encryption). The values of OKVS for protocols $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ps}}$ and $\mathsf{PSU}_{\mathsf{op}}^{\mathsf{ddh}}$ are bit strings, with a length of $\lambda + \log n$, and 64 bits are sufficient in practice.

A counterintuitive phenomenon is that the running time for small set size of 2^4 is actually longer than that of 2^6, as shown in the bottom two figures of Figure 11. This is due to the property of our BatchPIR instantiation, i.e., vectorized BatchPIR [MR23]. The vectorized BatchPIR scheme represents the database as a three-dimensional matrix. During the server-side response process, the second dimension plays a crucial role, involving rotations and ciphertext-ciphertext multiplications. The overall running time is heavily influenced by the size of this dimension. When the sender's set size is 2^4, the number of bins is fewer than that of 2^6. This causes there are more elements in each bin, which leads to a larger matrix size generated by the server and consequently longer running time[9].

Acknowledgements

We are grateful for the helpful comments from the anonymous reviewers. This work is supported by the National Key R&D Program of China (2018YFA0704701, 2020YFA0309705), Shandong Key Research and Development Program (2020ZLYS09), the Major Scientific and Technological Innovation Project of Shandong, China (2019JZZY010133), the Major Program of Guangdong Basic and Applied Research (2019B030302008), Tsinghua University Dushi Program, the National Natural Science Foundation of China (Grant No. 62272269), Taishan Scholar Program of Shandong Province, the National Key Research and Development Program of China (Grant No. 2021YFA1000600), the Major Programs of the National Social Science Foundation of China (Grant No. 22&ZD147), and the National Natural Science Foundation of China (Grant No. 62402272).

References

[ACLS18] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 962–979. IEEE Computer Society, 2018.

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *CCS 2013*, 2013.

[APS23] `https://github.com/real-world-cryprography/APSU`, 2023.

[BPSY23] Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. Near-optimal oblivious key-value stores for efficient psi, PSU and volume-hiding multi-maps. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 301–318. USENIX Association, 2023.

[BSMD10] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas A. Dimitropoulos. SEPIA: privacy-preserving aggregation of multi-domain network events and statistics. In *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*, pages 223–240. USENIX Association, 2010.

[CGN98] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. *IACR Cryptol. ePrint Arch.*, page 3, 1998.

[CGP20] Melissa Chase, Esha Ghosh, and Oxana Poburinnaya. Secret-shared shuffle. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 342–372. Springer, 2020.

[CHLR18] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1223–1237. ACM, 2018.

[9] A similar trend can also be found in the table III of original vectorized BatchPIR paper [MR23].

[CKGS98] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[CL15] Craig Costello and Patrick Longa. FourQ: Four-dimensional decompositions on a Q-curve over the mersenne prime. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 214–235. Springer, 2015.

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017,*, pages 1243–1255. ACM, 2017.

[CMdG$^+$21] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled PSI from homomorphic encryption with reduced computation and communication. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 1135–1150. ACM, 2021.

[CZZ$^+$24] Yu Chen, Min Zhang, Cong Zhang, Minglang Dong, and Weiran Liu. Private set operations from multi-query reverse private membership test. In *Public-Key Cryptography - PKC 2024 - 27th IACR International Conference on Practice and Theory of Public-Key Cryptography,*, Lecture Notes in Computer Science, 2024.

[DC17] Alex Davidson and Carlos Cid. An efficient toolkit for computing private set operations. In *ACISP 2017*, 2017.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324. Springer, 2005.

[Fri07] Keith B. Frikken. Privacy-preserving set union. In *ACNS 2007*, 2007.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.

[Gam85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory*, 31(4):469–472, 1985.

[GMR$^+$21] Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh. Private set operations from oblivious switching. In *PKC 2021*, 2021.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications.* Cambridge University Press, 2004.

[GPR$^+$21] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In *CRYPTO 2021*, 2021.

[HLP$^+$24] Meng Hao, Weiran Liu, Liqiang Peng, Hongwei Li, Cong Zhang, Hanxiao Chen, and Tianwei Zhang. Unbalanced circuit-psi from oblivious key-value retrieval. In *33rd USENIX Security Symposium (To appear)*, 2024.

[HLS$^+$16] Kyle Hogan, Noah Luther, Nabil Schear, Emily Shen, David Stott, Sophia Yakoubov, and Arkady Yerukhimovich. Secure multiparty computation for cooperative cyber risk assessment. In *SecDev 2016*, 2016.

[HN10] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. In *PKC 2010*, 2010.

[IKOS04] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 262–271. ACM, 2004.

[JL10] Stanislaw Jarecki and Xiaomin Liu. Fast secure computation of set intersection. In Juan A. Garay and Roberto De Prisco, editors, *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*, volume 6280 of *Lecture Notes in Computer Science*, pages 418–435. Springer, 2010.

[JSZ$^+$22] Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, Jiajun Du, and Dawu Gu. Shuffle-based private set union: Faster and more secure. In *USENIX Security 22*, 2022.

[KBM23] Florian Kerschbaum, Erik-Oliver Blass, and Rasoul Akhavan Mahdavi. Faster secure comparisons with offline phase for efficient private set intersection. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3,* *2023*. The Internet Society, 2023.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *CCS 2016*, 2016.

[KLS$^+$17] Ágnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas. Private set intersection for unequal set sizes with mobile applications. *Proc. Priv. Enhancing Technol.*, 2017(4):177–197, 2017.

[KRTW19] Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. Scalable private set union from symmetric-key techniques. In *ASIACRYPT*, 2019.

[KS05] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *CRYPTO 2005*, 2005.

[Lin17] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. In Yehuda Lindell, editor, *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer International Publishing, 2017.

[LLWR24] J. Liu, J. Li, D. Wu, and K. Ren. Pirana: Faster multi-query pir via constant-weight codes. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 39–39, Los Alamitos, CA, USA, may 2024. IEEE Computer Society.

[mpc] `https://github.com/alibaba-edu/mpc4j`.

[MR23] Muhammad Haris Mughees and Ling Ren. Vectorized batch private information retrieval. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 437–452. IEEE, 2023.

[MS13] Payman Mohassel and Seyed Saeed Sadeghian. How to hide circuits in MPC an efficient framework for private function evaluation. In *EUROCRYPT 2013*, 2013.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms*, 2001.

[Ope] `https://github.com/openssl/openssl`.

[PR04] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.

[PRTY20] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from paxos: Fast, malicious private set intersection. In *EUROCRYPT 2020*, 2020.

[PSZ18] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.*, 21(2):7:1–7:35, 2018.

[Rab05] Michael O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptol. ePrint Arch.*, 2005:187, 2005.

[RMY20] Sivaramakrishnan Ramanathan, Jelena Mirkovic, and Minlan Yu. BLAG: improving the accuracy of blacklists. In *NDSS*, 2020.

[RR22] Srinivasan Raghuraman and Peter Rindal. Blazing fast PSI from improved OKVS and subfield VOLE. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 2505–2517. ACM, 2022.

[RS21] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: fast OPRF and circuit-psi from vector-ole. In *EUROCRYPT 2021*, 2021.

[SEA22] Microsoft SEAL (release 4.0). `https://github.com/Microsoft/SEAL`, March 2022. Microsoft Research, Redmond, WA.

[Sod] `https://github.com/jedisct1/libsodium`.

[TCLZ23] Binbin Tu, Yu Chen, Qi Liu, and Cong Zhang. Fast unbalanced private set union from fully homomorphic encryption. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, pages 2959–2973. ACM, 2023.

[Vec23] `https://github.com/mhmughees/vectorized_batchpir`, 2023.

[ZCL$^+$23] Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin. Linear private set union from Multi-Query reverse private membership test. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 337–354, Anaheim, CA, August 2023. USENIX Association.

[ZLDL23] Cong Zhang, Weiran Liu, Bolin Ding, and Dongdai Lin. Efficient private multiset ID protocols. In Ding Wang, Moti Yung, Zheli Liu, and Xiaofeng Chen, editors, *Information and Communications Security - 25th International Conference, ICICS 2023, Tianjin, China, November 18-20, 2023, Proceedings*, volume 14252 of *Lecture Notes in Computer Science*, pages 351–369. Springer, 2023.

A Security Proofs of Permuted-PEQT

A.1 Proof of Theorem 1

Below we give the details of the proof of Theorem 1.

Proof. We exhibit simulators $\mathsf{Sim}_{\mathcal{S}}$ and $\mathsf{Sim}_{\mathcal{R}}$ for simulating corrupt $\mathcal{S}$ and $\mathcal{R}$ respectively, and argue the indistinguishability of the produced transcript from the real execution.

Corrupt Sender: $\mathsf{Sim}_{\mathcal{S}}(\vec{r}' = (r'_1, \ldots, r'_n), \pi)$ simulates the view of corrupt semi-honest sender. Note that the simulator only needs to simulate $\{v_i\}_{i\in[n]}$ in step 1, it chooses random group elements $v_i \leftarrow \mathbb{G}, i \in [n]$ to simulate the view.

Now we argue that the view output by $\mathsf{Sim}_{\mathcal{S}}$ is computationally indistinguishable from the real one based on DDH assumption.

Specifically, if there is a distinguisher $\mathcal{D}$ can distinguish the view output by $\mathsf{Sim}_{\mathcal{S}}$ and real view with non-negligible probability, then we can construct a PPT adversary $\mathcal{A}$ to break the DDH assumption. $\mathcal{A}$ works as follows: when $\mathcal{A}$ receives DDH challenge g^x, g^{y_i}, g^{z_i}, where $x, y_i \leftarrow \mathbb{Z}_p$, $\mathcal{A}$ is asked to distinguish if $z_i = x \cdot y_i$ or random values. $\mathcal{A}$ implicitly sets $a = x$, and simulates the view as below. For queries r_i, if r_i is not a component of vector $\vec{r}$, it picks a random group element to assign $H(r_i)$, otherwise, it assigns $H(r_i) = g^{y_i}$. Now $\mathcal{A}$ invokes $\mathcal{D}$ with input $g^{z_i}, i \in [n]$ and outputs $\mathcal{D}$'s output. If $z_i = x \cdot y_i$, $\mathcal{A}$ simulates real view, else, the view corresponds to the simulator's outputs. Therefore, $\mathcal{A}$ can break the DDH assumption with the same advantages as $\mathcal{D}$.

Corrupt Receiver: $\mathsf{Sim}_{\mathcal{R}}(\vec{r} = (r_1, \ldots, r_n))$ simulates the view of corrupt semi-honest $\mathcal{R}$. It executes as follows:

1. In step 1, $\mathsf{Sim}_{\mathcal{R}}$ chooses $a \leftarrow \mathbb{Z}_p$ randomly and computes v_i like an honest receiver.
2. In step 2, $\mathsf{Sim}_{\mathcal{R}}$ chooses random group elements v'_i for $i \in [n]$. $\mathsf{Sim}_{\mathcal{R}}$ also chooses random group elements $\bar{v}_i$ if $b_i = 0$ and sets $\bar{v}_i := v'^{a}_i$ if $b_i = 1$. Then, it appends the $\{v'_i, \bar{v}_i\}_{i\in[n]}$ to the view.

Now we argue that the view output by $\mathsf{Sim}_{\mathcal{R}}$ is indistinguishable from the real one. We formally prove this by a standard hybrid argument method. We define three hybrid transcripts T_0, T_1, T_2 where T_0 is the real view of $\mathcal{R}$, and T_2 is the output of $\mathsf{Sim}_{\mathcal{R}}$.

– Hybrid$_0$. The first hybrid is the real interaction described in Figure 7. Here, an honest $\mathcal{S}$ uses input $\vec{r}', \pi$, and honestly interacts with the corrupt $\mathcal{R}$. Let T_0 denote the real view of $\mathcal{R}$.

– Hybrid$_1$. Let T_1 be the same as T_0, except that for $b_i = 0$, v'_i and $\bar{v}_i$ are replaced by randomly selected group elements. We argue that the view in T_0 and T_1 are computationally indistinguishable based on the DDH assumption. Specifically, if there is a distinguisher $\mathcal{D}$ who can distinguish the T_0 and T_1 with non-negligible probability, then we can construct a PPT adversary $\mathcal{A}$ to break the DDH assumption. $\mathcal{A}$ works as follows: when $\mathcal{A}$ receives DDH challenge g^x, g^{y_i}, $g^{y'_i}$, g^{z_i}, $g^{z'_i}$, where x, y_i, $y'_i \leftarrow \mathbb{Z}_p$, $\mathcal{A}$ is asked to distinguish if $z_i = x \cdot y_i$, $z'_i = x \cdot y'_i$ or random values. $\mathcal{A}$ implicitly sets $b = x$, and simulates the view as below. For queries r_i and r'_i, it assigns $H(r_i) = g^{a^{-1}y_i}$, $H(r'_i) = g^{y'_i}$ (note that $\mathcal{A}$ is a non-uniform adversary for DDH assumption, both $\vec{r}$ and $\vec{r}'$ could be its auxiliary inputs). Now $\mathcal{A}$ invokes $\mathcal{D}$ with input $g^{z_i}, g^{z'_i}, i \in [n]$ and outputs $\mathcal{D}$'s output. If $z_i = x \cdot y_i$, $z'_i = x \cdot y'_i$, $\mathcal{A}$ simulates T_0, else, it simulates T_1. Therefore, $\mathcal{A}$ can break the DDH assumption with the same advantages as $\mathcal{D}$.

– Hybrid$_2$. Let T_2 be the same as T_1, except that for $b_i = 1$, v'_i is replaced by randomly selected group elements and $\bar{v}_i := v'^{a}_i$. We argue that the view in T_1 and T_2 are computationally indistinguishable based on the DDH assumption. Specifically, if there is a distinguisher $\mathcal{D}$ who can distinguish the T_1 and T_2 with non-negligible probability, then we can construct a PPT adversary $\mathcal{A}$ to break the DDH assumption. $\mathcal{A}$ works as follows: when $\mathcal{A}$ receives DDH challenge g^x, g^{y_i}, g^{z_i} where $x, y_i \leftarrow \mathbb{Z}_p$, $\mathcal{A}$ is asked to distinguish if $z_i = x \cdot y_i$ or random values. $\mathcal{A}$ implicitly sets $b = x$, and simulates the view as below. For queries $r_i = r'_i$ it assigns $H(r_i) = H(r'_i) = g^{y_i}$. Now $\mathcal{A}$ invokes $\mathcal{D}$ with input $g^{a \cdot z_i}$, $g^{z_i}, i \in [n]$ and outputs $\mathcal{D}$'s output. If $z_i = x \cdot y_i$, $\mathcal{A}$ simulates T_1, else, it simulates T_2. Therefore, $\mathcal{A}$ can break the DDH assumption with the same advantages as $\mathcal{D}$. The hybrid is exactly the view output by the simulator.

A.2 Proof of Theorem 2

Below we give the details of the proof of Theorem 2.

Proof. We exhibit simulators $\mathsf{Sim}_{\mathcal{S}}$ and $\mathsf{Sim}_{\mathcal{R}}$ for simulating corrupt $\mathcal{S}$ and $\mathcal{R}$ respectively, and argue the indistinguishability of the produced transcript from the real execution.

Corrupt Sender: $\mathsf{Sim}_{\mathcal{S}}(\vec{r'} = (r'_1, \ldots, r'_n), \pi)$ simulates the view of corrupt semi-honest sender. It executes as follows:

1. In step 1, $\mathsf{Sim}_{\mathcal{S}}$ selects random values $\beta_i \leftarrow \{0,1\}^l$ for $i \in [n]$. Then, it invokes Permute + Share receiver's simulator $\mathsf{Sim}^R_{\mathsf{ps}}(\pi, \vec{\beta} = (\beta_1, \ldots, \beta_n))$ and appends the output to the view.
2. In step 2, $\mathsf{Sim}_{\mathcal{S}}$ generates a random PRF key k. Then, it invokes OPRF sender's simulator $\mathsf{Sim}^S_{\mathsf{oprf}}(k)$ and appends the output to the view.

Now we argue that the view output by $\mathsf{Sim}_{\mathcal{S}}$ is indistinguishable from the real one. Since $\mathcal{S}$'s view only involves underlying Permute + Share and OPRF protocols, the correctness of this simulator directly follows the security of underlying protocols.
Corrupt Receiver: $\mathsf{Sim}_{\mathcal{R}}(\vec{r} = (r_1, \ldots, r_n), \vec{b} = (b_1, \ldots, b_n))$ simulates the view of corrupt semi-honest receiver. It executes as follows:

1. In step 1, $\mathsf{Sim}_{\mathcal{R}}$ selects random values $\alpha_i \leftarrow \{0,1\}^l, i \in [n]$ and invokes Permute + Share sender's simulator $\mathsf{Sim}^S_{\mathsf{ps}}(\vec{r}, \vec{\alpha} = (\alpha_1, \ldots, \alpha_n))$ and appends the output to the view.
2. In step 2, $\mathsf{Sim}_{\mathcal{R}}$ selects random values $q_i \leftarrow \{0,1\}^l, i \in [n]$. Then, the simulator $\mathsf{Sim}_{\mathcal{R}}$ invokes OPRF receiver's simulator $\mathsf{Sim}^R_{\mathsf{oprf}}(\{\alpha_i\}_{i\in[n]}, \{q_i\}_{i\in[n]})$ and appends the output to the view.
3. In step 3, for $i \in [n]$, $\mathsf{Sim}_{\mathcal{R}}$ picks random $f_i \leftarrow \{0,1\}^l$ if $b_i = 0$ and sets $f_i = q_i$ if $b_i = 1$. Then, it appends $(f_1, \ldots, f_n)$ to the view.

Now we argue that the view output by $\mathsf{Sim}_{\mathcal{R}}$ is indistinguishable from the real one. We formally prove this by a standard hybrid argument method. We define three hybrid transcripts T_0, T_1, T_2 where T_0 is real view of $\mathcal{R}$, and T_2 is the output of $\mathsf{Sim}_{\mathcal{R}}$.

- Hybrid$_0$. The first hybrid is the real interaction described in Figure 8. Here, an honest $\mathcal{S}$ uses input $\vec{r'}, \pi$, and honestly interacts with the corrupt $\mathcal{R}$. Let T_0 denote the real view of $\mathcal{R}$.
- Hybrid$_1$. Let T_1 be the same as T_0, except that all PRF values $F_k(\cdot)$ are replaced by randomly selected values. This hybrid is computationally indistinguishable from T_0 by the pseudorandomness of the PRF.
- Hybrid$_2$. Let T_2 be the same as T_1, except that the Permute + Share and OPRF execution are replaced by simulator $\mathsf{Sim}^S_{\mathsf{ps}}$, $\mathsf{Sim}^R_{\mathsf{oprf}}$. The security of Permute + Share and OPRF functionality guarantee this view is indistinguishable from T_1. This hybrid is exactly the view output by the simulator.

B Proofs of Theorem 3

Below we give the details of the proof of Theorem 3.

Proof. We exhibit simulators $\mathsf{Sim}_{\mathcal{S}}$ and $\mathsf{Sim}_{\mathcal{R}}$ for simulating corrupt $\mathcal{S}$ and $\mathcal{R}$ respectively, and argue the indistinguishability of the produced transcript from the real execution.

Corrupt Sender: $\mathsf{Sim}_{\mathcal{S}}(X = \{x_1, \ldots, x_m\})$ simulates the view of corrupt semi-honest sender. It executes as follows:

1. In step 1, $\mathsf{Sim}_{\mathcal{S}}$ inserts X into the Cuckoo hash table X^* like an honest sender, and learns the injective function σ.
2. In step 4, $\mathsf{Sim}_{\mathcal{S}}$ selects random values $q_j \leftarrow \mathbb{F}, j \in [m]$. Let $Q = \{q_j\}_{j\in[m]}$. Then, it invokes OPRF receiver's simulator $\mathsf{Sim}^R_{\mathsf{oprf}}(\{X^*[\sigma(j)]\}_{j\in[m]}, Q)$ and appends the output to the view.
3. In step 5-7, $\mathsf{Sim}_{\mathcal{S}}$ samples random $D_0 \| D_1 \leftarrow \mathbb{F}^s \times \mathbb{F}^d$ and executes as an honest sender to obtain $(I, \mathsf{st}, \mathsf{qu})$. Finally, $\mathsf{Sim}_{\mathcal{S}}$ computes $\mathsf{ans} := \mathsf{Answer}(D_0, \mathsf{qu})$ and appends (ans, D_1) to the view.

4. In step 8, $\mathsf{Sim}_{\mathcal{S}}$ computes $r'_j := \mathsf{Decode}(D_0\|D_1, X^*[\sigma(j)])$ for $j \in [m]$ and picks random $r'_i \leftarrow \mathbb{F}$ for $i \in [m_c] \setminus \{\sigma(j)\}_{j\in[m]}$. Let $\vec{r'} = (r'_1, \ldots, r'_{m_c})$
5. In step 9, $\mathsf{Sim}_{\mathcal{S}}$ selects a random permutation π over $[m_c]$. Then, the simulator $\mathsf{Sim}_{\mathcal{S}}$ invokes permuted PEQT sender's simulator $\mathsf{Sim}^S_{\mathsf{p\text{-}peqt}}(\vec{r'}, \pi)$ and appends the output to the view.
6. In step 10, for $i \in [m_c]$, the simulator $\mathsf{Sim}_{\mathcal{S}}$ invokes OT sender's simulator $\mathsf{Sim}^S_{\mathsf{ot}}(X^*[\pi(i)], \perp)$ and appends the output to the view.

Now we argue that the view output by $\mathsf{Sim}_{\mathcal{S}}$ is indistinguishable from the real one. We formally prove this by a standard hybrid argument method. We define four hybrid transcripts T_0, T_1, T_2, T_3 where T_0 is real view of $\mathcal{S}$, and T_3 is the output of $\mathsf{Sim}_{\mathcal{S}}$.

- Hybrid$_0$. The first hybrid is the real interaction described in Figure 9. Here, an honest $\mathcal{R}$ uses input Y, honestly interacts with the corrupt $\mathcal{S}$. Let T_0 denote the real view of $\mathcal{S}$.
- Hybrid$_1$. Let T_1 be the same as T_0, except that all PRF values $F_k(\cdot)$ are replaced by randomly selected values. This hybrid is computationally indistinguishable from T_0 by the pseudorandomness of the PRF.
- Hybrid$_2$. Let T_2 be the same as T_1, except that $D_0\|D_1$ is sampled uniformly from $\mathbb{F}^s \times \mathbb{F}^d$. Note that in the previous hybrid, the values used to compute sparse OKVS are forms of $\{r_i + f_{i,\alpha}\}_{i\in[m_c],\alpha\in[|Y^*[i]|]}$, where $f_{i,\alpha}$'s are truly random values. For each bin $i \in [m_c]$, the sender learns one random value f_i from OPRF[10] or selected it by itself. If $X^*[\sigma(j)] \notin Y^*[j]$, the values $r_i + f_{i,\alpha}$ are all random due to $f_{i,\alpha}$'s are random. If $X^*[\sigma(j)] \in Y^*[j]$, w.l.o.g, assume $f_i = f_{i,t}$ for some $t \in [|Y^*[i]|]$, the value $r_i + f_{i,t}$ is random from the view of sender because r_i is randomly selected by the receiver, and the values $r_i + f_{i,\alpha}$ for $\alpha in[|Y^*[i]|] \setminus \{t\}$ are also ramdom because of the randomness of $f_{i,\alpha}$. As a result, all values $\{r_i + f_{i,\alpha}\}_{i\in[m_c],\alpha\in[|Y^*[i]|]}$ are random from the view of the sender. By the obliviousness property of OKVS, T_1 and T_2 are statistically indistinguishable.
- Hybrid$_3$. Let T_3 be the same as T_2, except that the OPRF, permuted PEQT and OT execution are replaced by simulator $\mathsf{Sim}^R_{\mathsf{oprf}}, \mathsf{Sim}^S_{\mathsf{p\text{-}peqt}}, \mathsf{Sim}^S_{\mathsf{ot}}$. The security of OPRF, permuted PEQT and OT functionality guarantee this view is indistinguishable from T_2. This hybrid is exactly the view output by the simulator.

Corrupt Receiver: $\mathsf{Sim}_{\mathcal{R}}(Y = \{y_1, \ldots, y_n\}, X \cup Y)$ simulates the view of corrupt semi-honest receiver. It executes as follows:

1. In step 1-3, $\mathsf{Sim}_{\mathcal{R}}$ inserts Y into the simple hash table Y^* and picks a random $r_i \leftarrow \{0,1\}^l$ for each bin $i \in [m_c]$ like an honest receiver. Let $\vec{r} := (r_1, \ldots, r_{m_c})$.
2. In step 4, $\mathsf{Sim}_{\mathcal{R}}$ generates a random PRF key k. Then, it invokes OPRF sender's simulator $\mathsf{Sim}^S_{\mathsf{oprf}}(k)$ and appends the output to the view.
3. In step 5, $\mathsf{Sim}_{\mathcal{R}}$ computes sparse OKVS $D_0\|D_1$ as the honest receiver.
4. In step 6, $\mathsf{Sim}_{\mathcal{R}}$ samples random $I := \{i_1, \cdots, i_{mw}\} \leftarrow [s]^{wm}$. Then, $\mathsf{Sim}_{\mathcal{R}}$ computes $(\mathsf{st}, \mathsf{qu}) \leftarrow \mathsf{Query}(I)$ and appends qu to the view.
5. In step 9 and 10, $\mathsf{Sim}_{\mathcal{R}}$ defines the set $Z := X \cup Y \setminus Y$, i.e. the set of elements that X "brings to the union". Now, it uses $\perp$ to pad Z to m elements, then it uses dummy item d to pad Z to m_c elements and permutes these elements randomly. Let $Z = \{z_1, \ldots, z_{m_c}\}$. $\mathsf{Sim}_{\mathcal{R}}$ sets $b_i = 1$ if and only if $z_i = \perp$ and $b_i = 0$ otherwise for $i \in [m_c]$. Let $\vec{b} := (b_1, \ldots, b_{m_c})$. Then, it invokes permuted PEQT receiver's simulator $\mathsf{Sim}^R_{\mathsf{p\text{-}peqt}}(\vec{r}, \vec{b})$, OT receiver's simulator $\mathsf{Sim}^R_{\mathsf{ot}}(b_i, z_i)$ and appends these outputs to the view.

Now we argue that the view output by $\mathsf{Sim}_{\mathcal{R}}$ is indistinguishable from the real one. We formally prove this by a standard hybrid argument method. We define four hybrid transcripts T_0, T_1, T_2 where T_0 is real view of $\mathcal{R}$, and T_2 is the output of $\mathsf{Sim}_{\mathcal{R}}$.

- Hybrid$_0$. The first hybrid is the real interaction described in Figure 9. Here, an honest $\mathcal{S}$ uses input X, honestly interacts with the corrupt $\mathcal{R}$. Let T_0 denote the real view of $\mathcal{R}$.
- Hybrid$_1$. Let T_1 be the same as T_0, except that the query index set I is replaced by uniformly random $i_1, \cdots, i_{wm} \in [s]^{wm}$. This hybrid is computationally indistinguishable from T_0 by the query privacy of the BatchPIR scheme.

[10] Note that here the outputs of OPRF have been replaced with truly random values.

– Hybrid$_2$. Let T_2 be the same as T_1, except that the OPRF, permuted PEQT and OT execution are replaced by simulator $\mathsf{Sim}^S_{\mathsf{oprf}}, \mathsf{Sim}^R_{\mathsf{p\text{-}peqt}}, \mathsf{Sim}^R_{\mathsf{ot}}$. The security of OPRF, permuted PEQT and OT functionality guarantee this view is indistinguishable from T_1. This hybrid is exactly the view output by the simulator.

C Proofs of Theorem 4

Below we give the details of the proof of Theorem 4.

Proof. We exhibit simulators $\mathsf{Sim}_{\mathcal{S}}$ and $\mathsf{Sim}_{\mathcal{R}}$ for simulating corrupt $\mathcal{S}$ and $\mathcal{R}$ respectively, and argue the indistinguishability of the produced transcript from the real execution.

Corrupt Sender: $\mathsf{Sim}_{\mathcal{S}}(X = \{x_1, \dots, x_m\})$ simulates the view of corrupt semi-honest sender. It executes as follows:

1. In step 1-4, $\mathsf{Sim}_{\mathcal{S}}$ generates key pairs $pp \leftarrow \mathsf{Setup}(1^\kappa)$, $(pk, sk) \leftarrow \mathsf{KeyGen}(pp)$ and samples random $D_0 \| D_1 \leftarrow \mathbb{F}^s \times \mathbb{F}^d$. Then, it executes as an honest sender to obtain $(I, \mathsf{st}, \mathsf{qu})$. Finally, $\mathsf{Sim}_{\mathcal{S}}$ computes $\mathsf{ans} := \mathsf{Answer}(D_0, \mathsf{qu})$, and appends (ans, pk, D_1) to the view.
2. In step 9, $\mathsf{Sim}_{\mathcal{S}}$ invokes OT sender's simulator $\mathsf{Sim}^S_{\mathsf{ot}}(x_j, \perp)$ for $j \in [m]$ and appends the output to the view.

Now we argue that the view output by $\mathsf{Sim}_{\mathcal{S}}$ is indistinguishable from the real one. We formally prove this by a standard hybrid argument method. We define four hybrid transcripts T_0, T_1, T_2, T_3 where T_0 is real view of $\mathcal{S}$, and T_3 is the output of $\mathsf{Sim}_{\mathcal{S}}$.

– Hybrid$_0$. The first hybrid is the real interaction described in Figure 9. Here, an honest $\mathcal{R}$ uses input Y, and honestly interacts with the corrupt $\mathcal{S}$. Let T_0 denote the real view of $\mathcal{S}$.
– Hybrid$_1$. Let T_1 be the same as T_0, except that the ciphertexts $\{s_i\}_{i\in[n]}$ are replaced by randomly sampled ciphertexts in the ciphertext space. This hybrid is computationally indistinguishable from T_0 by the single-message multi-ciphertext pseudorandomness of the ReRand-PKE scheme.
– Hybrid$_2$. Let T_2 be the same as T_1, except that $D_0 \| D_1$ is sampled uniformly from $\mathbb{F}^s \times \mathbb{F}^d$. Since the ciphertexts have been replaced with the random ones, the obliviousness property of OKVS guarantees T_1 and T_2 are statistically indistinguishable.
– Hybrid$_3$. Let T_3 be the same as T_2, except that the OT execution is replaced by simulator $\mathsf{Sim}^S_{\mathsf{ot}}$. The security of OT functionality guarantees this view is indistinguishable from T_2. This hybrid is exactly the view output by the simulator.

Corrupt Receiver: $\mathsf{Sim}_{\mathcal{R}}(Y = \{y_1, \dots, y_n\}, X \cup Y)$ simulates the view of corrupt semi-honest receiver. It executes as follows:

1. In step 1-2, $\mathsf{Sim}_{\mathcal{R}}$ generates (pk, sk) and computes $\{s_i\}_{i\in[n]}$, $D_0 || D_1$ as the honest receiver.
2. In step 3, $\mathsf{Sim}_{\mathcal{R}}$ samples random $I := \{i_1, \cdots, i_{mw}\} \leftarrow [s]^{wm}$. Then, $\mathsf{Sim}_{\mathcal{R}}$ computes $(\mathsf{st}, \mathsf{qu}) \leftarrow \mathsf{Query}(I)$ and appends qu to the view.
3. In step 5-8, $\mathsf{Sim}_{\mathcal{R}}$ define the set $Z := X \cup Y \setminus Y$, i.e. the set of elements that X "brings to the union". Next, it uses $\perp$ to pads Z to m elements and permutes these elements randomly. Let $Z = \{z_1, \dots, z_m\}$. For $j \in [m]$, if $z_j = \perp$, $\mathsf{Sim}_{\mathcal{R}}$ sets $b_j = 1$ and $\bar{s_j} := \mathsf{Enc}(pk, 0)$, if $z_j \neq \perp$, $\mathsf{Sim}_{\mathcal{R}}$ sets $b_j = 0$ and $\bar{s_j} := \mathsf{Enc}(pk, \mathsf{Decode}(D_0 || D_1, z_j))$. Then, it appends $\{\bar{s_j}\}_{j\in[m]}$ to the view.
4. In step 9 and 10, for $j \in [m]$, $\mathsf{Sim}_{\mathcal{R}}$ invokes OT receiver's simulator $\mathsf{Sim}^R_{\mathsf{ot}}(b_j, z_j)$ and appends these outputs to the view.

Now we argue that the view output by $\mathsf{Sim}_{\mathcal{R}}$ is indistinguishable from the real one. We formally prove this by a standard hybrid argument method. We define four hybrid transcripts T_0, T_1, T_2, T_3 where T_0 is real view of $\mathcal{R}$, and T_3 is the output of $\mathsf{Sim}_{\mathcal{R}}$.

– Hybrid$_0$. The first hybrid is the real interaction described in Figure 10. Here, an honest $\mathcal{S}$ uses input X, and honestly interacts with the corrupt $\mathcal{R}$. Let T_0 denote the real view of $\mathcal{R}$.
– Hybrid$_1$. Let T_1 be the same as T_0, except that the query index set I is replaced by uniformly random $i_1, \cdots, i_{wm} \in [s]^{wm}$. This hybrid is computationally indistinguishable from T_0 by the query privacy of the BatchPIR scheme.

– Hybrid$_2$. Let T_2 be the same as T_1, except that ciphertexts $\{\bar{s_j}\}$ are computed as the simulator instead of the real protocol. This hybrid is identical to T_1 by the indistinguishability of the ReRand-PKE scheme.
– Hybrid$_3$. Let T_3 be the same as T_2, except that the OT execution is replaced by simulator $\mathsf{Sim}^R_{\mathsf{ot}}$. The security of OT functionality guarantees this view is indistinguishable from T_2. This hybrid is exactly the view output by the simulator.

D Formal Proof of "Pull-down-then-lift" Encoding

As we mentioned in Section 1.2, we use the "pull-down-then-lift" encoding technique in our ReRand-PKE based PSU construction to optimize encoding efficiency. We note that this optimization is deeply bound to the ElGamal encryption scheme. In this section, we formally prove that the OKVS generated in this way still satisfies (a tweaked definition of) obliviousness.

Let $(\mathsf{Encode}, \mathsf{Decode})$ be a standard OKVS scheme, where the value space is $\mathbb{Z}_p$. We describe our new encoding and decoding algorithms $(\mathsf{Encode}^{ptl}, \mathsf{Decode}^{ptl})$ as follows.

– $\mathsf{Encode}^{ptl}(\{x_i\}_{i\in[n]}, (pk, sk))$:
  1. For $i \in [n]$, pick random $r_i \leftarrow \mathbb{Z}_p$.
  2. Compute $d^1 = (d^1_1, \dots, d^1_N) := \mathsf{Encode}(\{(x_i, r_i)\}_{i\in[n]})$ and $d^2 = (d^2_1, \dots, d^2_N) := \mathsf{Encode}(\{(x_i, sk\cdot r_i)\}_{i\in[n]})$, where the random tapes used in two encoding algorithms are the same.
  3. Output $D = ((g^{d^1_1}, g^{d^2_1}), \dots, (g^{d^1_N}, g^{d^2_N}))$
– $\mathsf{Decode}^{ptl}(D, x)$:
  Output $c = (c_1, c_2) := (\prod_{\mathsf{row}(x)_j=1} g^{d^1_j}, \prod_{\mathsf{row}(x)_j=1} g^{d^2_j})$

Note that our new Encode^{ptl} algorithm takes (pk, sk) instead of group elements as inputs because it requires to know the algebraic structure of group elements, i.e., the discrete logarithm r_i for g^{r_i}. Therefore, we prove our new Encode^{ptl} algorithm satisfying the following tweaked obliviousness property.

Theorem 5. *Assume the DDH assumption hold in group $\mathbb{G}$, where order of $\mathbb{G}$ is p and g is a generator. For all distinct $\{x^0_1, \dots, x^0_n\}$ and all distinct $\{x^1_1, \dots, x^1_n\}$, if the Encode algorithm does not output $\perp$ for $\{x^0_1, \dots, x^0_n\}$ or $\{x^1_1, \dots, x^1_n\}$, for any ElGamal key pair (pk, sk), then the following distributions are computationally indistinguishable:*

$$\{D | D \leftarrow \mathsf{Encode}^{ptl}(\{x^0_i\}_{i\in[n]}, (pk, sk))\}$$
$$\{D | D \leftarrow \mathsf{Encode}^{ptl}(\{x^1_i\}_{i\in[n]}, (pk, sk))\}$$

Proof. For any distinct $\{x_1, \dots, x_n\}$, consider the computation process of $\mathsf{Encode}^{ptl}(\{x_i\}_{i\in[n]}, (pk, sk))\}$: Firstly, it picks random $r_i \leftarrow \mathbb{Z}_p, i \in [n]$. Then, it computes the OKVS $d^1 = (d^1_1, \dots, d^1_N) := \mathsf{Encode}(\{(x_i, r_i)\}_{i\in[n]})$ and $d^2 = (d^2_1, \dots, d^2_N) := \mathsf{Encode}(\{(x_i, sk \cdot r_i)\}_{i\in[n]})$. Since the random tapes used in two encoding algorithms are the same, we have $d^2_j = sk \cdot d^1_j$ for $j \in [N]$. Furthermore, since r_i's are randomly selected, the obliviousness of underlining OKVS $(\mathsf{Encode}, \mathsf{Decode})$ guarantees d^1 is indistinguishable to the uniform distribution over $\mathbb{Z}^N_p$. Let $D = ((g^{d^1_1}, g^{d^2_1}), \dots, (g^{d^1_N}, g^{d^2_N})) = \mathsf{Encode}^{ptl}(\{x^0_i\}_{i\in[n]}, (pk, sk))\}$. For $j \in [N]$, we have $(g^{d^1_j}, g^{d^2_j}) = (g^{d^1_j}, g^{sk\cdot d^1_j}) = (g^{d^1_j}, pk^{d^1_j})$, which exactly corresponds to a fresh ElGamal ciphertext of 1. Therefore, the distribution of D can also be written as:

$\mathsf{Dis}(\{x_i\}_{i\in[n]}, (pk, sk))$:

1. For $j \in [N]$, computes $D_j \leftarrow \mathsf{Enc}(pk, 1)$, where Enc is the encryption algorithm of the ElGamal encryption scheme.
2. Output $D = (D_1, \dots, D_N)$

By the single-message multi-ciphertext pseudorandomness of ElGamal encryption scheme, we have the above distribution is indistinguishable to the uniform distribution over ciphertext space, i.e., $\mathbb{G}^2$. Note that the above new description of D is independent from $\{x_i\}_{i\in[n]}$. As a result, two distributions are computationally indistinguishable.