Efficient Private Set Union from Multi-Query Oblivious Membership Transfer

Yewei Guan[1][0009−0007−8823−8580], Hua Guo[1,2][0000−0002−6719−8846], Man Ho Au[3][0000−0003−2068−9530], Jiarong Huo[1][0009−0000−6277−0404], and Renheng Wang[1][0009−0009−7076−9863]

[1] School of Cyber Science and Technology, Beihang University, Beijing, China
{ame_reiori, hguo, huojiarong, wangrh12}@buaa.edu.cn
[2] State Key Laboratory of Complex & Critical Software Environment, Beijing, China
[3] The Hong Kong Polytechnic University, Hong Kong, China
man-ho-allen.au@polyu.edu.hk

Abstract. Private Set Union (PSU) allows two parties to compute the union of their private sets while revealing nothing beyond the union. Existing PSU protocols mainly follow two paradigms: reverse private membership test (RPMT) and invertible Bloom lookup table (IBLT). IBLT-based protocols achieve linear complexity using lightweight symmetric-key operations and offer excellent concrete efficiency. RPMT, on the other hand, provides a more general membership-testing functionality that can serve as a reusable building block for PSU and other private-set computations. However, existing RPMT-based protocols either rely on costly building blocks, such as public-key operations and generic secure two-party computation, or incur super-linear complexity. This raises a natural question: can the generality of the RPMT paradigm be retained while matching the asymptotic and concrete efficiency of IBLT-based PSU?

In this work, we answer this question affirmatively by introducing a lightweight mechanism for multi-query membership testing. Our mechanism yields a concretely efficient multi-query RPMT (mqRPMT) protocol that is useful beyond PSU and may be of independent interest. Building on this protocol, we construct a PSU protocol with linear computation and communication complexity. Our construction avoids expensive public-key operations and generic secure two-party computation, relying instead on efficient symmetric-key primitives.

We extensively evaluate our protocol against state-of-the-art PSU protocols from both the RPMT and IBLT paradigms. Compared with the leading RPMT-based PSU protocol from PKC 2024, our construction is $41.7\times - 48.9\times$ faster in the LAN setting and up to $4.5\times$ faster in WAN settings. Compared with the leading IBLT-based PSU protocol from EUROCRYPT 2026, our protocol achieves competitive and, in several metrics, superior performance: it is $1.9\times - 2.5\times$ faster in LAN settings and reduces communication by up to $1.4\times$. These results demonstrate that the broader applicability of the RPMT abstraction need not come at the cost of asymptotic or concrete efficiency.

Keywords: Private set union · Multi-query reserve private membership test · Oblivious pseudorandom function with shared output.

1 Introduction

Private Set Union (PSU) is a fundamental private set operation that allows two parties, a sender holding a set X and a receiver holding a set Y, to compute $X \cup Y$ without revealing additional information beyond the union. In particular, the protocol should not reveal which elements belong to the intersection $X \cap Y$. PSU has applications in cyber-risk assessment and management [15], IP blacklist and vulnerability-data aggregation [10], private databases supporting full joins [14], and private ID [7]. These applications have motivated substantial recent progress on scalable and concretely efficient PSU protocols.

Most existing PSU protocols [3, 4, 7, 11, 14, 23] follow the reverse private membership test (RPMT) paradigm. At a high level, RPMT allows the receiver to obtain, for each sender element $x_i \in X$, a membership-test bit e_i indicating whether $x_i \in Y$. The receiver can then use these bits as selection bits in oblivious transfer (OT) to obtain exactly the elements in $X \setminus Y$, and locally reconstruct $X \cup Y$. Since OT admits highly efficient constructions [21], the efficiency of RPMT-based PSU is largely determined by the underlying RPMT construction.

The efficiency of RPMT-based PSU therefore depends largely on how these membership-test bits are generated. In the single-query setting, the sender holds one element x, while the receiver holds a set Y, and the protocol securely determines whether $x \in Y$. Such single-query RPMT protocols can be realized efficiently from symmetric-key primitives. However, applying them directly to PSU requires repeating the protocol for every $x_i \in X$, resulting in quadratic complexity. Prior works [3, 11] substantially reduce this overhead using hash-to-bin techniques; however, the resulting protocols still incur an additional logarithmic factor due to the steps needed to hide the correspondence between elements and bins.

To achieve linear complexity, existing works consider multi-query RPMT (mqRPMT), which generates the membership-test bits for all sender elements. Zhang et al. [23] gave two linear-complexity constructions: one based on symmetric-key encryption and generic secure two-party computation, and another based on rerandomizable public-key encryption. Chen et al. [4]'s mqRPMT constructions rely on DDH-like assumptions, using tools such as commutative weak PRFs and permuted OPRFs. These constructions achieve linear computation and communication, but rely either on generic secure computation or on public-key-style algebraic operations. This leaves open the possibility of an mqRPMT-based PSU protocol that achieves linear complexity using only lightweight symmetric-key primitives.

Recently, Piske and Trieu [16] proposed a different approach to PSU based on invertible Bloom lookup tables (IBLTs). Their protocol departs from the RPMT-based framework and achieves linear computation and communication using only lightweight primitives. This result shows that PSU can be made highly practical

outside the RPMT paradigm. However, it does not provide an efficient multi-query RPMT primitive, which remains a useful modular building block for PSU and other private set operations.

Following the standard semi-honest setting of these PSU protocols, we ask: *Can RPMT-based PSU achieve both linear complexity and high concrete efficiency?*

1.1 Technical Overview

We answer the above question affirmatively by constructing an RPMT-based PSU protocol with linear computation and communication using only lightweight symmetric-key primitives. We first revisit the symmetric-key multi-query RPMT construction of Zhang et al. [23], which is the closest predecessor to our work. We then introduce multi-query oblivious membership transfer (multi-query OMT), a new intermediate primitive that leads to an efficient multi-query RPMT protocol. Finally, by combining our multi-query RPMT protocol with standard oblivious transfer, we obtain our PSU protocol.

In what follows, we use $\mathcal{S}$ and $\mathcal{R}$ to denote the sender and the receiver, respectively, with input sets $X = \{x_i\}_{i\in[n]}$ and $Y = \{y_i\}_{i\in[n]}$ of size n.

Review of Zhang et al.'s symmetric-key mqRPMT We review the version of Zhang et al.'s mqRPMT protocol based on symmetric-key encryption and generic secure two-party computation, as it is the closest predecessor to our work. The protocol uses an OKVS as a private lookup table. An OKVS can be viewed as an encoding of key-value pairs such that querying an encoded key returns the associated value, while querying a key outside the encoded set returns a random-looking value.

$\mathcal{R}$ samples a secret value s. For each $y_j \in Y$, it generates a randomized ciphertext $c_j = \mathsf{Enc}(s)$ and encodes the key-value pairs $(H(y_j), c_j)_{j\in[n]}$ into an OKVS D, where H is a hash function, and sends D to $\mathcal{S}$. $\mathcal{S}$ then queries D on each $x_i \in X$ and obtains

$$c_i^* = \mathsf{Decode}(D, H(x_i)).$$

If $x_i \in Y$, then c_i^* is one of the ciphertexts encrypting s; otherwise, by the randomness property of OKVS, c_i^* is random. Thus, membership testing is reduced to checking whether c_i^* decrypts to s.

The subtle point is that neither party can perform this check locally without violating security. If $\mathcal{R}$ gives the decryption key and the secret value s to $\mathcal{S}$, $\mathcal{S}$ can test which decoded values decrypt to s, and hence learn which of its elements belong to Y, i.e., the intersection. Conversely, c_i^* cannot be revealed directly to $\mathcal{R}$. Since c_i^* is the OKVS output obtained from the sender's query x_i, revealing it would allow $\mathcal{R}$ to test or link the sender's queried elements and infer membership information about X. Therefore, Zhang et al. perform the decryption-and-matching step obliviously: for each i, $\mathcal{R}$ learns only whether c_i^* decrypts to s, namely the membership-test bit for x_i, while $\mathcal{S}$ learns nothing

about these bits and c_i^* remains hidden from $\mathcal{R}$. Their realization relies on generic secure two-party computation, which introduces substantial concrete cost.

We avoid this component by changing the architecture of the membership test. Instead of checking whether a hidden value decrypts to a fixed secret s, we reduce membership testing to checking equality between two fresh random values associated with each sender index i.

Our Design We first describe the intermediate functionality that we want to realize. For each sender element x_i, the protocol generates two values: a value t_i held by $\mathcal{S}$, and a value s_i held by $\mathcal{R}$. These values satisfy

$$t_i = s_i \quad \text{if } x_i \in Y,$$

and otherwise t_i is random and independent of s_i. Thus, the membership-test bit for x_i is encoded by the equality relation between t_i and s_i. We formalize this abstraction as multi-query oblivious membership transfer (multi-query OMT), which allows $\mathcal{R}$ to obliviously transfer value s_i to $\mathcal{S}$, who obtains t_i, according to the membership information. Note that the membership-test bit is not revealed in this functionality.

Given multi-query OMT, multi-query RPMT follows directly: $\mathcal{S}$ reveals t_i to $\mathcal{R}$, and $\mathcal{R}$ compares it with s_i to obtain the membership-test bit for x_i. This reveal step is safe because s_i is associated with the sender index i, rather than with any receiver element $y \in Y$, and $\mathcal{R}$ does not learn which sender element corresponds to t_i. Therefore, revealing t_i leaks only the intended membership-test bit.

To reduce communication, we reveal only a short tag of t_i. Specifically, $\mathcal{S}$ sends $H_\ell(t_i)$ instead of t_i, where H_ℓ is modeled as a random oracle that outputs ℓ bits. The receiver computes $H_\ell(s_i)$ locally and compares the two tags. We choose ℓ so that the probability of a false match among all n comparisons is at most $2^{-\lambda}$, where λ is the statistical security parameter.

Finally, by combining the resulting multi-query RPMT protocol with OT, we obtain our RPMT-based PSU protocol. Figure 1 summarizes this construction roadmap.

Realizing multi-query OMT. To realize multi-query OMT efficiently, we introduce another intermediate primitive, called batched masked dual-key OPRF (batched mDK-OPRF). This primitive allows $\mathcal{S}$ to obtain, for each $x_i \in X$, a masked dual-key PRF value

$$x_i' = F_{k,k^*}(x_i) \oplus s_i,$$

while $\mathcal{R}$ obtains the masks $\{s_i\}_{i\in[n]}$ and the PRF values $\{F_{k,k^*}(y_j)\}_{j\in[n]}$ on its own set. Here, F_{k,k^*} is a dual-key PRF, k is held by $\mathcal{S}$, k^* is held by $\mathcal{R}$. One can view x_i' as an encryption of s_i under the one-time pad $F_{k,k^*}(x_i)$. If $\mathcal{S}$ can recover this pad, then it can obtain s_i; otherwise, the resulting value remains random.

Given these values, the rest of the multi-query OMT construction is simple. $\mathcal{R}$ encodes the key-value pairs

$$\{(y_j, F_{k,k^*}(y_j))\}_{j\in[n]}$$

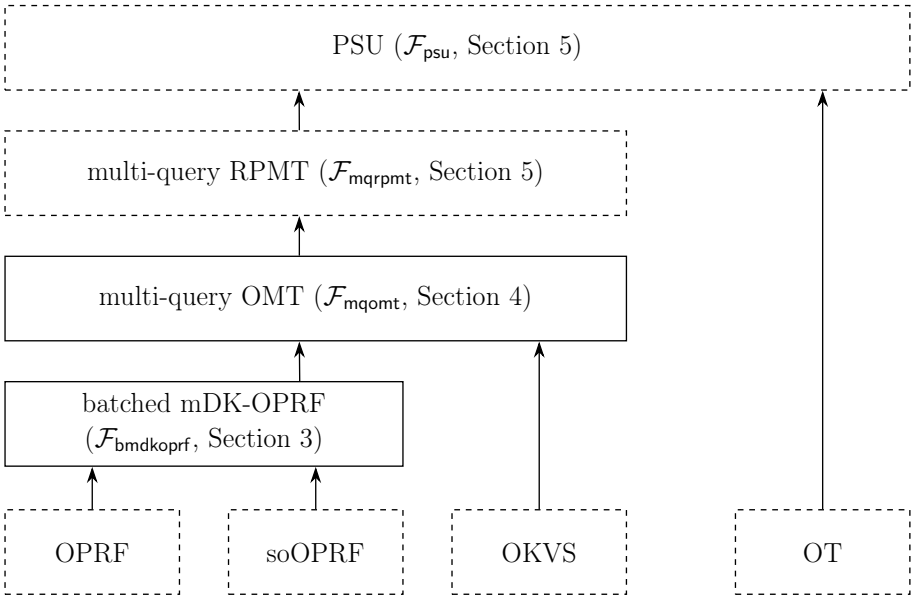


Fig. 1. Summary of our main results. Solid rectangles denote new notions. Dotted rectangles denote notions from previous works.

into an OKVS D and sends D to $\mathcal{S}$. For each x_i, $\mathcal{S}$ computes

$$t_i = x'_i \oplus \mathsf{Decode}(D, x_i).$$

If $x_i \in Y$, then $\mathsf{Decode}(D, x_i) = F_{k,k^*}(x_i)$, and hence $t_i = s_i$. If $x_i \notin Y$, then the randomness property of OKVS ensures that $\mathsf{Decode}(D, x_i)$ is random-looking, so t_i is random and independent of s_i. This realizes the desired multi-query OMT behavior.

Realizing batched mDK-OPRF. It remains to realize batched mDK-OPRF efficiently. We instantiate the dual-key PRF as

$$F_{k,k^*}(x) = \widehat{F}_k(x) \oplus \widetilde{F}_{k^*}(x),$$

where $\widehat{F}$ and $\widetilde{F}$ are secure PRFs. The construction computes the two components separately. First, $\mathcal{S}$ and $\mathcal{R}$ run an OPRF [20] in which $\mathcal{R}$ queries its set Y. As a result, $\mathcal{R}$ obtains $\widehat{F}_k(y_j)$ for each $y_j \in Y$, while $\mathcal{S}$ obtains the key k and can locally compute $\widehat{F}_k(x_i)$ for each $x_i \in X$.

Second, the parties run an OPRF with shared output (soOPRF [1]) on $\mathcal{S}$'s set X. This gives $\mathcal{R}$ the key k^* and a share a_i, and gives $\mathcal{S}$ a share b_i, such that

$$a_i \oplus b_i = \widetilde{F}_{k^*}(x_i).$$

To form the masked value, $\mathcal{R}$ samples s_i and sends $\widetilde{s}_i = s_i \oplus a_i$ to $\mathcal{S}$. Then $\mathcal{S}$ computes

$$\begin{aligned} x'_i &= \widehat{F}_k(x_i) \oplus b_i \oplus \widetilde{s}_i \\ &= \widehat{F}_k(x_i) \oplus \widetilde{F}_{k^*}(x_i) \oplus s_i \\ &= F_{k,k^*}(x_i) \oplus s_i. \end{aligned}$$

Meanwhile, $\mathcal{R}$ computes $F_{k,k^*}(y_j) = \widehat{F}_k(y_j) \oplus \widetilde{F}_{k^*}(y_j)$ locally. Thus, batched mDK-OPRF can be realized from OPRF, soOPRF, and symmetric-key operations.

To reduce communication, $\mathcal{R}$ generates a seed and sends it to $\mathcal{S}$. Both parties locally expand this seed to obtain $\{\widetilde{s}_i\}_{i\in[n]}$. Then $\mathcal{R}$ derives $s_i = \widetilde{s}_i \oplus a_i$. This is equivalent to the unoptimized version in which $\mathcal{R}$ samples s_i and sends $\widetilde{s}_i = s_i \oplus a_i$, except that the $\widetilde{s}_i$ values are generated pseudorandomly from a short seed and need not be sent. Under the security of the pseudorandom generator, the resulting $\{s_i\}_{i\in[n]}$ are indistinguishable from freshly sampled masks.

Complexity The overall protocol has linear computation and communication. The batched mDK-OPRF invokes one OPRF and one soOPRF on sets of size n, and therefore has linear cost using existing efficient instantiations. Multi-query OMT adds one OKVS encoding and n OKVS queries, which are also linear. Multi-query RPMT then requires $\mathcal{S}$ to send one ℓ-bit tag for each sender element, and the final PSU step invokes n OTs. Hence, every layer of the construction has linear complexity. More importantly, all main building blocks in our PSU protocol, including OPRF [20], soOPRF [1], OKVS [18], and OT [21], have efficient instantiations. As a result, our PSU protocol achieves high concrete efficiency.

We compare our protocol with existing works in Table 1. As summarized in Table 1, to the best of our knowledge, this is the first RPMT-based PSU protocol that achieves linear complexity without relying on public-key techniques or generic secure two-party computation.

1.2 Related Works

Early PSU protocols were based on additively homomorphic encryption [5,6,13]. These works laid important foundations, but their reliance on public-key operations makes them less suitable for large-scale deployments. The first scalable PSU protocol was based on reverse private membership test (RPMT) [14], followed by several improvements using hash-to-bin, secure permutation, multi-query RPMT, and related techniques [11]. As discussed above, existing RPMT-based PSU protocols either incur superlinear complexity or rely on public-key-style algebraic operations or generic secure computation.

A recent line of work studies enhanced PSU, motivated by during-execution leakage in protocols based on hash-to-bin techniques [12]. The exact formulation and necessity of this enhanced functionality remain under discussion [17]. Our work focuses on the standard semi-honest PSU functionality considered by the state-of-the-art protocols compared in our evaluation.

Table 1. Comparison of related works and our work on communication and computation complexity. The set size is n. Comm. means the communication complexity, while Comp. means the computational complexity. Tech. means the main techniques used in the construction. ssPMT means that, in their scheme, parties will obtain shares of membership-test bits during the protocol. ○ means the construction relies on public-key techniques. ◐ means the construction relies on generic secure two-party computation. ● means the construction only relies on symmetric-key techniques, without public-key techniques or generic secure two-party computation.

Scheme	Approach	Comm.	Comp.	Tech.
KRTW19 [14]	RPMT	$O(n \log n)$	$O(n \log n \log \log n)$	●
GMR+21 [7]	RPMT	$O(n \log n)$	$O(n \log n)$	●
JSZ+22 [11]	RPMT	$O(n \log n)$	$O(n \log n)$	●
ZCL+23 (PKE) [23]	RPMT	$O(n)$	$O(n)$	○
ZCL+23 (SKE) [23]	RPMT	$O(n)$	$O(n)$	◐
BPSY23 [2]	RPMT	$O(n)$	$O(n)$	○
CZZ+24 [4]	RPMT	$O(n)$	$O(n)$	○
JSZ+24 [12]	ssPMT	$O(n \log n)$	$O(n \log n)$	●
TBZ+25 [22]	ssPMT	$O(n)$	$O(n)$	○
PT26 [16]	IBLT	$O(n)$	$O(n)$	●
Ours	RPMT	$O(n)$	$O(n)$	●

1.3 Our Contributions

In this work, we show that RPMT-based PSU can achieve linear asymptotic complexity and high concrete efficiency simultaneously, using only lightweight symmetric-key primitives. Our main contributions are as follows.

Multi-query oblivious membership transfer (Section 4). We introduce multi-query oblivious membership transfer, an intermediate primitive that encodes membership through equality between sender-held and receiver-held random values. We construct multi-query OMT from our batched masked dual-key OPRF, which can be realized using efficient symmetric-key primitives with linear computation and communication. Building on multi-query OMT, we obtain a concretely efficient multi-query RPMT protocol. Since RPMT is a modular building block for private set operations beyond PSU, we believe this construction may be of independent interest.

RPMT-based private set union (Section 5). By combining our multi-query RPMT protocol with OT, we obtain an RPMT-based PSU protocol with linear computation and communication. This is the first RPMT-based PSU protocol that achieves linear complexity and high concrete efficiency without relying on public-key techniques or generic secure two-party computation.

Evaluation (Section 6). We implement our protocol and evaluate it against the most efficient state-of-the-art PSU protocols under different network settings and input sizes. Compared with the previous leading RPMT-based protocol [4],

our protocol achieves up to 48.9× speedup in the LAN setting and 4.5× speedup in the WAN setting. Compared with the IBLT-based protocol [16], our protocol achieves competitive performance: it is up to 2.5× faster in LAN, 1.4× faster in WAN, and uses up to 1.4× less communication. We will make our implementation publicly available on GitHub.

2 Preliminaries

2.1 Notations

Let κ and λ be the computational and statistical security parameters, respectively. We denote the parties as sender $\mathcal{S}$ and receiver $\mathcal{R}$, and their respective input sets as X and Y with $|X| = |Y| = n$. We use $[n]$ to denote the set $\{1, \ldots, n\}$. We use similar notation for sets, e.g., $X = \{x_1, \ldots, x_n\}$. The notation $a \oplus b$ means bitwise-XOR of a, b. We use $X \approx_c Y$ to denote that X is computationally indistinguishable from Y. $a \leftarrow_R A$ denotes that a is uniformly sampled from distribution A.

2.2 Security Model

Similar to most PSU protocols [4, 11, 14, 23], we focus on the semi-honest model, where adversaries honestly follow the protocol but try to learn additional knowledge from the given protocol execution.

Let Π be a two-party protocol for computing function $f(x_1, x_2)$, where party P_i holds x_i, and $\mathsf{output}(x_1, x_2)$ be the output of both parties. For each party P_i, let $\mathsf{VIEW}_{P_i}(x_1, x_2)$ denote the view of party P_i during an honest execution of protocol Π on inputs x_1 and x_2, which includes P_i's input, random tape, and all messages P_i received in the protocol.

Definition 1. *A two-party protocol Π securely realizes f in the presence of semi-honest adversaries if there exists a simulator Sim such that, for all inputs x_1, x_2 and $i \in \{1, 2\}$:*

$$\{\mathsf{VIEW}_{P_i}(x_1, x_2), \mathsf{output}(x_1, x_2)\} \approx_c \{\mathsf{Sim}(i, x_i, f(x_1, x_2)), f(x_1, x_2)\}.$$

2.3 Private Set Union

The ideal functionality of PSU $\mathcal{F}_{\mathsf{psu}}$ is presented in Figure 2. This functionality considers two parties, a sender $\mathcal{S}$ with private set X and a receiver $\mathcal{R}$ with private set Y. They want to compute the union $X \cup Y$ without any leakage. In our definition, we allow only the receiver obtain the result.

2.4 Oblivious Transfer

A 1-out-of-2 Oblivious Transfer (OT) is a two-party protocol where the sender inputs two strings (m_0, m_1) and the receiver inputs a selection bit b. As a result, the receiver learns m_b and nothing about m_{1-b}, while the sender learns nothing about b. The ideal functionality of OT is presented in Figure 3.

PARAMETERS: Sender $\mathcal{S}$, Receiver $\mathcal{R}$, and the set size n.
FUNCTIONALITY:

- Wait for input $X = \{x_i\}_{i\in[n]} \subset \{0,1\}^*$ from $\mathcal{S}$ and $Y = \{y_i\}_{i\in[n]} \subset \{0,1\}^*$ from $\mathcal{R}$.
- Give the output $X \cup Y$ to $\mathcal{R}$.

Fig. 2. The Ideal Functionality of PSU $\mathcal{F}_{\mathsf{psu}}$

PARAMETERS: Sender $\mathcal{S}$, Receiver $\mathcal{R}$.
FUNCTIONALITY:

- Wait for input $b \in \{0,1\}$ from $\mathcal{R}$.
- Wait for input (m_0, m_1) from $\mathcal{S}$.
- Give m_b to $\mathcal{R}$.

Fig. 3. The Ideal Functionality of 1-out-of-2 OT $\mathcal{F}_{\mathsf{ot}}$

2.5 Oblivious Pseudorandom Function

Oblivious Pseudorandom Function (OPRF) allows the receiver with input x to learn the corresponding PRF value $F_k(x)$ where k is learned by the sender. The ideal functionality of OPRF is presented in Figure 4.

PARAMETERS: Sender $\mathcal{S}$, Receiver $\mathcal{R}$. A PRF scheme F and the set size n.
FUNCTIONALITY:

- Wait for input $X = \{x_i\}_{i\in[n]}$ from $\mathcal{R}$.
- Randomly sample a key k to the PRF and give it to $\mathcal{S}$.
- Give $\{F_k(x_i)\}_{i\in[n]}$ to $\mathcal{R}$.

Fig. 4. The Ideal Functionality of OPRF $\mathcal{F}^F_{\mathsf{oprf}}$

2.6 OPRF with Shared Output

OPRF with Shared Output (soOPRF) is a variant of OPRF, where parties obtain secret-shared PRF values. The ideal functionality of soOPRF is presented in Figure 5.

PARAMETERS: Sender $\mathcal{S}$, Receiver $\mathcal{R}$. A PRF scheme F and the set size n.
FUNCTIONALITY:

- Wait for input $X = \{x_i\}_{i\in[n]}$ from $\mathcal{R}$.
- Randomly sample a key k to the PRF F and give it to $\mathcal{S}$.
- Randomly sample a_i from the range of PRF and compute $b_i = F_k(x_i) \oplus a_i$ for $i \in [n]$.
- Give $\{a_i\}_{i\in[n]}$ to $\mathcal{S}$ and $\{b_i\}_{i\in[n]}$ to $\mathcal{R}$.

Fig. 5. The Ideal Functionality of soOPRF $\mathcal{F}^F_{\mathsf{sooprf}}$

2.7 Oblivious Key-Value Store

An Oblivious Key-Value Store (OKVS) is a data structure to encode a set of key-value points $A = \{(k_i, v_i)\}$ while hiding the keys. A party can query k on the data structure and obtain v_i if $k = k_i$.

Definition 2 (OKVS [8, 18]). *An OKVS scheme consists of two algorithms* Encode *and* Decode*, parameterized by a set $\mathcal{K}$ of keys, a set $\mathcal{V}$ of values, and a set of functions H, with the following syntax:*

- $D/\perp \leftarrow \mathsf{Encode}_H(\{(k_i, v_i)\})$*: On input a set of key-value points $\{(k_i, v_i)\}$, the encode algorithm outputs an object $D \in \mathcal{D}$, or $\perp$ with a negligible probability.*
- $y \leftarrow \mathsf{Decode}_H(D, k)$*: On input D and a key $k \in \mathcal{K}$, the decode algorithm outputs a value $v \in \mathcal{V}$.*

In the rest of the paper, the underlying parameter H is omitted as long as the text remains unambiguous. For abbreviation, we use $\mathsf{Encode}(K, V)$ *to represent* $\mathsf{Encode}(\{(k_i, v_i)\})$ *where K and V have the same size. The scheme satisfies*

- **Correctness:** *For all $A \subseteq \mathcal{K} \times \mathcal{V}$ with distinct keys, if $(k, v) \in A$ and $\perp \neq D \leftarrow \mathsf{Encode}(A)$, then $\mathsf{Decode}(D, k) = v$.*
- **Obliviousness:** *For all distinct $\{k_1^0, \ldots, k_n^0\} \in \mathcal{K}^n$ and $\{k_1^1, \ldots, k_n^1\} \in \mathcal{K}^n$, if* Encode *does not output $\perp$ for $\{k_1^0, \ldots, k_n^0\}$ and $\{k_1^1, \ldots, k_n^1\}$, then the distribution of* $\mathsf{Encode}(\{(k_i^0, v_i)\})$ *is statistically indistinguishable from the distribution of* $\mathsf{Encode}(\{(k_i^1, v_i)\})$*, where $v_1, \ldots, v_n$ are sampled uniformly at random from $\mathcal{V}$.*
- **Randomness:** *For any $K = \{k_1, \ldots, k_n\}, V = \{v_1, \ldots, v_n\}$ and $k^* \notin K$, the output of* $\mathsf{Decode}(D, k^*)$ *is statistically indistinguishable to that of uniform distribution over $\mathcal{V}$, where $D \leftarrow \mathsf{Encode}(K, V)$.*

2.8 Multi-Query Reverse Private Membership Test

Multi-query reverse private membership test (multi-query RPMT) allows two parties, holding sets X and Y, respectively, to determine whether each $x_i \in X$ belongs to Y without any leakage, and the results are delivered to the receiver. The ideal functionality of multi-query RPMT is presented in Figure 6.

PARAMETERS: Sender $\mathcal{S}$, Receiver $\mathcal{R}$, and the set size n.
FUNCTIONALITY:

- Wait for input $X = \{x_i\}_{i\in[n]} \subset \{0,1\}^*$ from $\mathcal{S}$ and $Y = \{y_i\}_{i\in[n]} \subset \{0,1\}^*$ from $\mathcal{R}$.
- For each $i \in [n]$, set $e_i = 1$ if $x_i \in Y$, and $e_i = 0$ otherwise.
- Give $E = \{e_i\}_{i\in[n]}$ to $\mathcal{R}$.

Fig. 6. The Ideal Functionality of Multi-Query RPMT $\mathcal{F}_{\mathsf{mqrpmt}}$

3 Batched Masked Dual-Key Oblivious Pseudorandom Function

We introduce batched masked dual-key OPRF (batched mDK-OPRF), a two-party primitive for batched, masked, and oblivious evaluation of a dual-key PRF. A dual-key PRF is a PRF whose key consists of a pair (k_0, k_1).

The primitive involves two parties, P_0 and P_1, holding input sets $U = \{u_i\}_{i\in[n]}$ and $V = \{v_i\}_{i\in[n]}$, respectively. The dual-key PRF key pair (k_0, k_1) is distributed between the parties: P_0 receives k_0, while P_1 receives k_1. In addition, P_0 obtains the masked PRF evaluations

$$U' = \{F_{k_0,k_1}(u_i) \oplus r_i\}_{i\in[n]},$$

whereas P_1 obtains the corresponding masks

$$R = \{r_i\}_{i\in[n]}$$

and the unmasked PRF evaluations on its input set

$$V' = \{F_{k_0,k_1}(v_i)\}_{i\in[n]}.$$

The ideal functionality of batched mDK-OPRF is presented in Figure 7.

3.1 Protocol Details

Our construction evaluates a dual-key PRF by combining two PRF components, one under each key. We define

$$F_{k_0,k_1}(x) = \widehat{F}_{k_0}(x) \oplus \widetilde{F}_{k_1}(x), \tag{1}$$

where $\widehat{F}$ and $\widetilde{F}$ are PRF families. We evaluate the two components separately using OPRF and soOPRF, respectively.

To evaluate the first component $\widehat{F}_{k_0}$, the parties invoke the OPRF functionality $\mathcal{F}_{\mathsf{oprf}}^{\widehat{F}}$, where P_0 acts as the sender and P_1 inputs V as the receiver. As a

PARAMETERS: Two parties P_0 and P_1. A dual-key PRF scheme F, and the set size n.
FUNCTIONALITY:

- Wait for input $U = \{u_i\}_{i\in[n]}$ from P_0 and $V = \{v_i\}_{i\in[n]}$ from P_1.
- Randomly sample a key pair (k_0, k_1) to the dual-key PRF F.
- Randomly sample r_i from the range of PRF for $i \in [n]$.
- Compute $u'_i = F_{k_0,k_1}(u_i) \oplus r_i$ and $v'_i = F_{k_0,k_1}(v_i)$ for $i \in [n]$.
- Give k_0 and $U' = \{u'_i\}_{i\in[n]}$ to P_0.
- Give $k_1, V' = \{v'_i\}_{i\in[n]}$ and $R = \{r_i\}_{i\in[n]}$ to P_1.

Fig. 7. The Ideal Functionality of Batched mDK-OPRF $\mathcal{F}^{F}_{\mathsf{bmdkoprf}}$

result, P_0 obtains the key k_0 and locally computes $\widehat{u}_i = \widehat{F}_{k_0}(u_i)$ for each $u_i \in U$, while P_1 obtains $\widehat{v}_i = \widehat{F}_{k_0}(v_i)$ for each $v_i \in V$.

To evaluate the second component $\widetilde{F}_{k_1}$, the parties invoke the soOPRF functionality $\mathcal{F}^{\widetilde{F}}_{\mathsf{sooprf}}$, where P_1 acts as the sender and P_0 inputs U as the receiver. As a result, P_1 obtains the key k_1 and shares $\{a_i\}_{i\in[n]}$, while P_0 obtains shares $\{b_i\}_{i\in[n]}$, such that $a_i \oplus b_i = \widetilde{F}_{k_1}(u_i)$ for each $i \in [n]$. Since P_1 holds k_1, it can also locally compute $v'_i = \widehat{v}_i \oplus \widetilde{F}_{k_1}(v_i) = F_{k_0,k_1}(v_i)$ for each $v_i \in V$.

To form the masked evaluations on U, for each $i \in [n]$, P_1 samples r_i uniformly at random and sends $\widetilde{r}_i = a_i \oplus r_i$ to P_0, who then computes $u'_i = \widehat{u}_i \oplus b_i \oplus \widetilde{r}_i = F_{k_0,k_1}(u_i) \oplus r_i$.

To reduce communication, the parties avoid transmitting all values $\{\widetilde{r}_i\}_{i\in[n]}$. Instead, P_1 samples $\mathsf{seed} \leftarrow_R \{0,1\}^\kappa$ and sends it to P_0, after which both parties compute

$$\{\widetilde{r}_i\}_{i\in[n]} = \mathsf{PRG}(\mathsf{seed}).$$

Party P_1 then locally derives $r_i = \widetilde{r}_i \oplus a_i$ for each $i \in [n]$. Under the security of the PRG, the resulting masks are computationally indistinguishable from independently sampled random masks.

The complete batched mDK-OPRF construction is presented in Figure 8.

Introducing an Offline Phase. Following prior work [16, 23], we organize the protocol into offline and online phases. We instantiate the OPRF and soOPRF components using the constructions of [20] and [1], respectively. The input-independent preprocessing, including the VOLE correlations for OPRF, the OT correlations for soOPRF, and the seed seed, is performed offline; only input-dependent computation and communication remain online.

3.2 Security Proof

We first establish the security of the dual-key PRF defined in Equation 1. In our instantiation, both $\widehat{F}_{k_0}$ and $\widetilde{F}_{k_1}$ are secure PRFs, following the security of the

PARAMETERS: Two parties, P_0 and P_1. The set size n, an OPRF functionality $\mathcal{F}_{\mathsf{oprf}}^{\widehat{F}}$, an soOPRF functionality $\mathcal{F}_{\mathsf{sooprf}}^{\widetilde{F}}$, and a PRG PRG.
INPUTS: P_0 inputs $U = \{u_i\}_{i\in[n]}$ while P_1 inputs $V = \{v_i\}_{i\in[n]}$.
PROTOCOL:

1. P_0 and P_1 invoke $\mathcal{F}_{\mathsf{oprf}}^{\widehat{F}}$, where P_1 inputs V as the receiver and P_0 acts as the sender. As a result, P_0 obtains key k_0 while P_1 obtains $\widehat{v}_i = \widehat{F}_{k_0}(v_i)$ for $i \in [n]$.
2. For $i \in [n]$, P_0 computes $\widehat{u}_i = \widehat{F}_{k_0}(u_i)$.
3. P_1 and P_0 invoke $\mathcal{F}_{\mathsf{sooprf}}^{\widetilde{F}}$, where P_1 inputs nothing as the sender and P_0 inputs U as the receiver. As a result, P_1 obtains key k_1, and for $i \in [n]$, P_0 obtains a share b_i while P_1 obtains a share a_i such that $a_i \oplus b_i = \widetilde{F}_{k_1}(u_i)$.
4. P_1 samples $\mathsf{seed} \leftarrow_R \{0,1\}^\kappa$ and sends seed to P_0.
5. Both parties compute $\widetilde{R} = \{\widetilde{r}_i\}_{i\in[n]} = \mathsf{PRG}(\mathsf{seed})$.
6. For $i \in [n]$, P_0 computes $u'_i = \widehat{u}_i \oplus b_i \oplus \widetilde{r}_i$.
7. For $i \in [n]$, P_1 computes $v'_i = \widehat{v}_i \oplus \widetilde{F}_{k_1}(v_i)$ and $r_i = \widetilde{r}_i \oplus a_i$.
8. P_0 outputs k_0 and $U' = \{u'_i\}_{i\in[n]}$, while P_1 outputs k_1, $V' = \{v'_i\}_{i\in[n]}$, and $R = \{r_i\}_{i\in[n]}$.

Fig. 8. Our Batched mDK-OPRF construction $\Pi_{\mathsf{bmdkoprf}}^{F}$

underlying constructions in [20] and [1]. In fact, the following theorem establishes the stronger statement that the security of either component alone is sufficient.

Theorem 1. *Suppose that* $\widehat{F} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$ *and* $\widetilde{F} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$ *are secure PRFs. Then the function*

$$F : \{0,1\}^\kappa \times \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$$

defined in Equation 1 is pseudorandom under the exposure of either one of its keys.

Proof (Proof sketch). Consider first the case in which k_0 is revealed. By the PRF security of $\widetilde{F}$, we may replace $\widetilde{F}_{k_1}$ with a uniformly random function R, with only a negligible change in the adversary's view. This replacement remains valid even given k_0, since k_0 is sampled independently of k_1.

Conditioned on k_0, the function $\widehat{F}_{k_0}$ is fixed. Therefore, the function

$$x \mapsto \widehat{F}_{k_0}(x) \oplus R(x)$$

is itself uniformly random, since XOR with a fixed function is a bijection over the function space. Hence, F_{k_0,k_1} remains pseudorandom even given k_0. The case where k_1 is revealed follows symmetrically from the PRF security of $\widehat{F}$.

We now prove that our construction in Figure 8 securely realizes the functionality of batched mDK-OPRF.

Theorem 2. *In the $\{\mathcal{F}_{\mathsf{oprf}}^{\widehat{F}}, \mathcal{F}_{\mathsf{sooprf}}^{\widetilde{F}}\}$-hybrid model, $\Pi_{\mathsf{bmdkoprf}}^{F}$ securely realizes $\mathcal{F}_{\mathsf{bmdkoprf}}^{F}$ against semi-honest adversaries.*

Proof. We prove correctness and security.

Correctness. For each $i \in [n]$, the OPRF invocation gives P_1 a PRF evaluation $\widehat{v}_i = \widehat{F}_{k_0}(v_i)$. Since P_1 also holds k_1, it computes

$$v'_i = \widehat{v}_i \oplus \widetilde{F}_{k_1}(v_i) = \widehat{F}_{k_0}(v_i) \oplus \widetilde{F}_{k_1}(v_i) = F_{k_0,k_1}(v_i).$$

For each $i \in [n]$, the soOPRF invocation produces shares a_i and b_i such that $a_i \oplus b_i = \widetilde{F}_{k_1}(u_i)$. Moreover, the protocol defines $r_i = \widetilde{r}_i \oplus a_i$, and hence $\widetilde{r}_i = r_i \oplus a_i$. Therefore,

$$\begin{aligned} u'_i &= \widehat{u}_i \oplus b_i \oplus \widetilde{r}_i \\ &= \widehat{F}_{k_0}(u_i) \oplus b_i \oplus a_i \oplus r_i \\ &= \widehat{F}_{k_0}(u_i) \oplus \widetilde{F}_{k_1}(u_i) \oplus r_i \\ &= F_{k_0,k_1}(u_i) \oplus r_i. \end{aligned}$$

Thus, the outputs produced by the protocol satisfy the batched mDK-OPRF functionality.

Security. We construct simulators for the cases where either P_0 or P_1 is corrupted.

Corrupted P_0. The simulator Sim receives the corrupted party's input $U = \{u_i\}_{i\in[n]}$ and its output (k_0, U') from the ideal functionality, where $U' = \{u'_i\}_{i\in[n]}$. It reconstructs the view of P_0 as follows.

Sim provides k_0 as the output received from $\mathcal{F}_{\mathsf{oprf}}^{\widehat{F}}$. It then samples $\mathsf{seed} \leftarrow_R \{0,1\}^{\kappa}$ and computes $\{\widetilde{r}_i\}_{i\in[n]} = \mathsf{PRG}(\mathsf{seed})$. For each $i \in [n]$, the simulator sets

$$b_i = u'_i \oplus \widehat{F}_{k_0}(u_i) \oplus \widetilde{r}_i.$$

It provides $\{b_i\}_{i\in[n]}$ as the output received from $\mathcal{F}_{\mathsf{sooprf}}^{\widetilde{F}}$ and sends seed to P_0 on behalf of P_1.

We first verify consistency with the prescribed ideal output. For each $i \in [n]$, the output computed by P_0 is

$$\begin{aligned} \widehat{F}_{k_0}(u_i) \oplus b_i \oplus \widetilde{r}_i &= \widehat{F}_{k_0}(u_i) \oplus \left(u'_i \oplus \widehat{F}_{k_0}(u_i) \oplus \widetilde{r}_i\right) \oplus \widetilde{r}_i \\ &= u'_i. \end{aligned}$$

Hence, the simulated execution produces exactly the output U' given by the ideal functionality.

We then show that the simulated view has the correct distribution. In the real hybrid execution, seed is sampled uniformly and independently, and the shares $\{b_i\}_{i\in[n]}$ received from the soOPRF functionality satisfy $u'_i = \widehat{F}_{k_0}(u_i) \oplus b_i \oplus \widetilde{r}_i$. Therefore, once U, k_0, U', and seed are fixed, the shares are uniquely determined

by $b_i = u'_i \oplus \widehat{F}_{k_0}(u_i) \oplus \widetilde{r}_i$. Sim samples seed according to the same distribution and determines $\{b_i\}_{i\in[n]}$ using exactly the same relation.

Therefore, conditioned on the input and output of P_0, the joint distribution of $(\mathsf{seed}, \{b_i\}_{i\in[n]})$ in the simulated execution is identical to that in the real hybrid execution. Consequently, the simulated view is identically distributed to the view of the corrupted P_0.

Corrupted P_1. The simulator Sim receives the corrupted party's input $V = \{v_i\}_{i\in[n]}$ and its output (k_1, V', R) from the ideal functionality, where $V' = \{v'_i\}_{i\in[n]}$ and $R = \{r_i\}_{i\in[n]}$.

The simulator reconstructs the view of P_1 as follows. In the OPRF invocation, P_1 submits V and receives $\{\widehat{v}_i\}_{i\in[n]}$. For each $i \in [n]$, Sim sets

$$\widehat{v}_i = v'_i \oplus \widetilde{F}_{k_1}(v_i).$$

In the soOPRF invocation, P_1 receives the key k_1 and shares $\{a_i\}_{i\in[n]}$. To determine these shares, Sim samples $\mathsf{seed} \leftarrow_R \{0,1\}^\kappa$ and computes $\{\widetilde{r}_i\}_{i\in[n]} = \mathsf{PRG}(\mathsf{seed})$. For each $i \in [n]$, it then sets

$$a_i = \widetilde{r}_i \oplus r_i.$$

Sim provides $\{\widehat{v}_i\}_{i\in[n]}$ as the output received from the OPRF invocation, provides $(k_1, \{a_i\}_{i\in[n]})$ as the output received from the soOPRF invocation, and includes seed as the value sampled and sent by P_1 to P_0.

We first verify consistency with the prescribed ideal output. Using the simulated OPRF outputs, P_1 computes, for each $i \in [n]$,

$$\widehat{v}_i \oplus \widetilde{F}_{k_1}(v_i) = \left(v'_i \oplus \widetilde{F}_{k_1}(v_i)\right) \oplus \widetilde{F}_{k_1}(v_i) = v'_i.$$

Similarly, using the simulated soOPRF shares and the sampled seed, P_1 computes

$$\widetilde{r}_i \oplus a_i = \widetilde{r}_i \oplus (\widetilde{r}_i \oplus r_i) = r_i.$$

Hence, the simulated execution produces exactly the output (V', R) given by the ideal functionality.

We then show that the simulated values have the correct distribution. In the real execution, seed is sampled uniformly by P_1. Once R and seed are fixed, the shares $\{a_i\}_{i\in[n]}$ are uniquely determined by $a_i = \widetilde{r}_i \oplus r_i$. Sim samples seed according to the same distribution and determines $\{a_i\}_{i\in[n]}$ using exactly the same relation.

Moreover, once V, k_1, and V' are fixed, the OPRF outputs received by P_1 are uniquely determined by $\widehat{v}_i = v'_i \oplus \widetilde{F}_{k_1}(v_i)$. Indeed, by the definition of the ideal batched mDK-OPRF functionality,

$$\widehat{v}_i = v'_i \oplus \widetilde{F}_{k_1}(v_i) = F_{k_0,k_1}(v_i) \oplus \widetilde{F}_{k_1}(v_i) = \widehat{F}_{k_0}(v_i),$$

which is exactly the value returned by the OPRF functionality in the real hybrid execution.

Therefore, conditioned on the input and output of P_1, the joint distribution of $(\{\widehat{v}_i\}_{i\in[n]}, \{a_i\}_{i\in[n]}, \mathsf{seed})$ in the simulated execution is identical to that in the real execution. Consequently, the simulated view is identically distributed to the view of the corrupted P_1.

3.3 Complexity Analysis

The protocol invokes one batched OPRF and one batched soOPRF, each on n inputs. In addition, each party performs n local evaluations of PRF on its own set, expands one κ-bit seed into n κ-bit strings, and performs $O(n)$ XOR operations. Therefore, the total computational cost is

$$T_{\mathsf{bmdkoprf}}(n) = T_{\mathsf{oprf}}(n) + T_{\mathsf{sooprf}}(n) + O(n).$$

The only communication outside the OPRF and soOPRF invocations is the transmission of the κ-bit seed. Hence, the total communication is

$$C_{\mathsf{bmdkoprf}}(n) = C_{\mathsf{oprf}}(n) + C_{\mathsf{sooprf}}(n) + \kappa.$$

The OPRF and soOPRF invocations are independent and can be executed in parallel. Moreover, the seed is independent of the parties' inputs and can be generated and transmitted in the offline phase.

4 Multi-Query Oblivious Membership Transfer

Multi-query oblivious membership transfer (multi-query OMT) serves as a key building block in our PSU protocol. It allows $\mathcal{R}$ to obliviously transfer values to $\mathcal{S}$ according to the hidden membership-test result.

The primitive takes $X = \{x_i\}_{i\in[n]}$ from $\mathcal{S}$ and Y from $\mathcal{R}$ as inputs. As a result, $\mathcal{R}$ obtains values $S = \{s_i\}_{i\in[n]}$ while $\mathcal{S}$ obtains a value $T = \{t_i\}_{i\in[n]}$. For each $i \in [n]$, these values satisfy $t_i = s_i$ if $x_i \in Y$ and otherwise t_i is random and independent of s_i. The ideal functionality of multi-query OMT is presented in Figure 9.

4.1 Protocol Details

Our construction lets $\mathcal{S}$ first obtain, for each $i \in [n]$, an encryption of s_i and then attempt to decrypt it.

To evaluate the encryption of s_i, parties invoke the batched mDK-OPRF protocol, where $\mathcal{S}$ inputs X as P_0 and $\mathcal{R}$ inputs Y as P_1. As a result, for each $i \in [n]$, $\mathcal{S}$ obtains a masked PRF value $x'_i = F_{k,k^*}(x_i) \oplus s_i$, while $\mathcal{R}$ obtains an unmasked PRF value $y'_i = F_{k,k^*}(y_i)$ and a mask s_i. The masked PRF value can also be viewed as an encryption of s_i under the one-time pad $F_{k,k^*}(x_i)$.

To let $\mathcal{S}$ recover the pad according to the hidden membership-test result, $\mathcal{R}$ sends an OKVS $D = \mathsf{Encode}(Y, Y')$, to $\mathcal{S}$, where $Y' = \{y'_i\}_{i\in[n]}$. Sender $\mathcal{S}$ can then locally compute $t_i = x'_i \oplus \mathsf{Decode}(D, x_i)$ for each $i \in [n]$.

The complete multi-query OMT construction is presented in Figure 10.

PARAMETERS: Sender $\mathcal{S}$, Receiver $\mathcal{R}$. The set size n.
FUNCTIONALITY:

- Wait for input $X = \{x_i\}_{i\in[n]} \subset \{0,1\}^*$ from $\mathcal{S}$.
- Wait for input $Y = \{y_i\}_{i\in[n]} \subset \{0,1\}^*$ from $\mathcal{R}$.
- Randomly sample $s_i \leftarrow_R \{0,1\}^\kappa$ for $i \in [n]$.
- For each $i \in [n]$, set $t_i = s_i$ if $x_i \in Y$, and otherwise, randomly sample $t_i \leftarrow_R \{0,1\}^\kappa$.
- Give $T = \{t_i\}_{i\in[n]}$ to $\mathcal{S}$ and $S = \{s_i\}_{i\in[n]}$ to $\mathcal{R}$.

Fig. 9. The Ideal Functionality of Multi-Query OMT $\mathcal{F}_{\mathsf{mqomt}}$

PARAMETERS: Sender $\mathcal{S}$, Receiver $\mathcal{R}$. The set size n, an OKVS scheme $(\mathsf{Encode}, \mathsf{Decode})$, and a batched mDK-OPRF functionality $\mathcal{F}^F_{\mathsf{bmdkoprf}}$.
INPUTS: $\mathcal{S}$ inputs $X = \{x_i\}_{i\in[n]}$, while $\mathcal{R}$ inputs $Y = \{y_i\}_{i\in[n]}$.
PROTOCOL:

1. $\mathcal{S}$ and $\mathcal{R}$ invoke $\mathcal{F}^F_{\mathsf{bmdkoprf}}$, where $\mathcal{S}$ inputs X as P_0 and $\mathcal{R}$ inputs Y as P_1. As a result, $\mathcal{S}$ obtains k and $X' = \{x'_i\}_{i\in[n]}$ such that $x'_i = F_{k,k^*}(x_i) \oplus s_i$, while $\mathcal{R}$ obtains k^*, $Y' = \{F_{k,k^*}(y_i)\}_{i\in[n]}$, and $S = \{s_i\}_{i\in[n]}$.
2. $\mathcal{R}$ computes an OKVS $D = \mathsf{Encode}(Y, Y')$ and sends D to $\mathcal{S}$.
3. For $i \in [n]$, $\mathcal{S}$ computes $t_i = x'_i \oplus \mathsf{Decode}(D, x_i)$.
4. $\mathcal{S}$ outputs $T = \{t_i\}_{i\in[n]}$, while $\mathcal{R}$ outputs $S = \{s_i\}_{i\in[n]}$.

Fig. 10. Our Multi-Query OMT construction Π_{mqomt}

4.2 Security Proof

Theorem 3. *In the $\mathcal{F}_{\mathsf{bmdkoprf}}$-hybrid model, Π_{mqomt} securely implements $\mathcal{F}_{\mathsf{mqomt}}$ against a semi-honest adversary, provided the OKVS scheme satisfies correctness, obliviousness, and randomness.*

Proof. Our proof includes correctness and security.

Correctness. For each $i \in [n]$, the batched mDK-OPRF invocation gives $\mathcal{S}$ a masked PRF evaluation $x'_i = F_{k,k^*}(x_i) \oplus s_i$. Since the OKVS D is encoded from key-value pairs $\{(y_j, F_{k,k^*}(y_j))\}_{j\in[n]}$, for $x_i \in Y$, the decoded value holds

$$\mathsf{Decode}(D, x_i) = F_{k,k^*}(x_i).$$

Therefore,

$$\begin{aligned} t_i &= x'_i \oplus \mathsf{Decode}(D, x_i) \\ &= F_{k,k^*}(x_i) \oplus s_i \oplus F_{k,k^*}(x_i) \\ &= s_i. \end{aligned}$$

In contrast, for $x_i \notin Y$, the randomness property of OKVS ensures $\mathsf{Decode}(D, x_i)$ is statistically indistinguishable from random, so t_i is random and independent of s_i. Thus, the outputs produced by the protocol satisfy the multi-query OMT functionality.

Security. We construct simulators for the cases where either $\mathcal{S}$ or $\mathcal{R}$ is corrupted.

Corrupted $\mathcal{S}$. The simulator Sim receives the corrupted party's input $X = \{x_i\}_{i\in[n]}$ and its output $T = \{t_i\}_{i\in[n]}$ from the ideal functionality. It reconstructs the view of $\mathcal{S}$ as follows.

Sim samples $k \leftarrow_R \{0,1\}^\kappa$. It chooses a set of distinct dummy keys $\widehat{Y} = \{\widehat{y}_j\}_{j\in[n]}$ from the OKVS key domain and independently samples $\widehat{v}_j \leftarrow_R \{0,1\}^\kappa$ for every $j \in [n]$. It then computes

$$D \leftarrow \mathsf{Encode}\left(\{(\widehat{y}_j, \widehat{v}_j)\}_{j\in[n]}\right).$$

For each $i \in [n]$, the simulator sets

$$x'_i = t_i \oplus \mathsf{Decode}(D, x_i).$$

It provides $(k, \{x'_i\}_{i\in[n]})$ as the output received from $\mathcal{F}^F_{\mathsf{bmdkoprf}}$ and sends D to $\mathcal{S}$ on behalf of $\mathcal{R}$.

We first verify consistency with the prescribed ideal output. Let

$$d_i = \mathsf{Decode}(D, x_i).$$

Using the simulated batched mDK-OPRF outputs and OKVS, $\mathcal{S}$ computes, for each $i \in [n]$,

$$x'_i \oplus d_i = (t_i \oplus d_i) \oplus d_i = t_i.$$

Hence, the simulated execution produces exactly the output T given by the ideal functionality.

It remains to show that the simulated view has the correct distribution. In the real execution, k is sampled uniformly, and $\mathcal{R}$ computes

$$D \leftarrow \mathsf{Encode}\left(\{(y_j, F_{k,k^*}(y_j))\}_{j\in[n]}\right).$$

By Theorem 1, even given k, the values $\{F_{k,k^*}(y_j)\}_{j\in[n]}$ are computationally indistinguishable from independently uniform values. By the obliviousness of the OKVS, an encoding of the actual keys Y with independently uniform values is statistically indistinguishable from an encoding of the dummy keys $\widehat{Y}$ with independently uniform values. Therefore, the pair (k, D) generated by Sim is computationally indistinguishable from the pair in the real execution.

Furthermore, in the real execution, $x'_i = F_{k,k^*}(x_i) \oplus s_i$, where the masks $\{s_i\}_{i\in[n]}$ are independently uniform. Hence, conditioned on (k, D), the values $\{x'_i\}_{i\in[n]}$ are independently uniform. Since $\mathcal{S}$ computes $t_i = x'_i \oplus \mathsf{Decode}(D, x_i)$, the vector T is also independently uniform conditioned on (k, D). Once T and D are fixed, the values $\{x'_i\}_{i\in[n]}$ are uniquely determined by

$$x'_i = t_i \oplus \mathsf{Decode}(D, x_i).$$

In the ideal execution, the simulator receives the corrupted sender's output T from $\mathcal{F}_{\mathsf{mqomt}}$. By the definition of $\mathcal{F}_{\mathsf{mqomt}}$, the marginal distribution of T is independently uniform over $(\{0,1\}^\kappa)^n$. The simulator determines $\{x'_i\}_{i\in[n]}$ using exactly the same relation as in the real execution. Therefore, conditioned on (k, D, T), the distribution of $\{x'_i\}_{i\in[n]}$ in the simulated execution is identical to that in the real execution.

Consequently, the simulated view is computationally indistinguishable from the view of the corrupted $\mathcal{S}$.

Corrupted $\mathcal{R}$. The simulator Sim receives the corrupted party's input $Y = \{y_i\}_{i\in[n]}$ and its output $S = \{s_i\}_{i\in[n]}$ from the ideal functionality. It reconstructs the view of $\mathcal{R}$ as follows.

Sim samples $k^* \leftarrow_R \{0,1\}^\kappa$, and for each $i \in [n]$, independently samples $y'_i \leftarrow_R \{0,1\}^\kappa$. It provides (k^*, Y', S), where $Y' = \{y'_i\}_{i\in[n]}$, as the output received from $\mathcal{F}^F_{\mathsf{bmdkoprf}}$. The OKVS D sent by $\mathcal{R}$ is then computed honestly according to Step 2., namely $D \leftarrow \mathsf{Encode}(Y, Y')$.

We first verify consistency with the prescribed ideal output. The receiver's output in the protocol is S, which is exactly the output received from $\mathcal{F}_{\mathsf{mqomt}}$. Therefore, the simulated execution produces the prescribed ideal output.

We then show that the simulated view has the correct distribution. In the real hybrid execution, $\mathcal{R}$ receives (k^*, Y', S) from $\mathcal{F}^F_{\mathsf{bmdkoprf}}$, where $Y' = \{F_{k,k^*}(y_i)\}_{i\in[n]}$. By Theorem 1, even given k^*, the values $\{F_{k,k^*}(y_i)\}_{i\in[n]}$ are computationally indistinguishable from independently uniform values. Moreover, in $\mathcal{F}^F_{\mathsf{bmdkoprf}}$, the masks S are sampled independently and uniformly, and k^* is sampled uniformly. Hence the tuple (k^*, Y', S) generated by the simulator is computationally indistinguishable from the corresponding tuple in the real hybrid execution. Since D is computed honestly according to Step 2. on (Y, Y') with fresh encoding randomness in both executions, replacing Y' by a computationally indistinguishable vector changes the distribution of D only computationally indistinguishably. Therefore, the receiver's simulated view is computationally indistinguishable from its view in the real hybrid execution.

4.3 Complexity Analysis

Besides one batched mDK-OPRF invocation, $\mathcal{R}$ performs an OKVS encoding of n key-value pairs, while $\mathcal{S}$ performs an OKVS decoding of n keys and $O(n)$ XOR operations. Therefore, the total computational cost is

$$T_{\mathsf{mqomt}}(n) = T_{\mathsf{bmdkoprf}}(n) + O(n).$$

The only communication cost outside the batched mDK-OPRF is the transmission of the OKVS with size $O(n)$. Hence, the total communication is

$$C_{\mathsf{mqomt}}(n) = C_{\mathsf{bmdkoprf}}(n) + O(n).$$

5 Private Set Union

5.1 Our Multi-Query RPMT Construction

By the definition of multi-query OMT, for each $i \in [n]$, $\mathcal{S}$ obtains a value t_i, while $\mathcal{R}$ obtains a value s_i, such that $t_i = s_i$ if $x_i \in Y$, and otherwise t_i is uniformly random and independent of s_i. Therefore, multi-query RPMT can be obtained by having $\mathcal{S}$ reveal t_i to $\mathcal{R}$, who sets $e_i = 1$ if $t_i = s_i$ and $e_i = 0$ otherwise. The revealed value leaks no information beyond the membership-test result: it is equal to the value already held by $\mathcal{R}$ when $x_i \in Y$, and is otherwise an independent random value.

To reduce the communication from $n\kappa$ bits to $n\ell$ bits, $\mathcal{S}$ sends only a truncated hash of each t_i. Specifically, for each $i \in [n]$, $\mathcal{S}$ sends $t'_i = H_\ell(t_i)$ to $\mathcal{R}$, where H_ℓ is modeled as a random oracle that outputs ℓ bits. The receiver computes $s'_i = H_\ell(s_i)$ and sets $e_i = 1$ if $t'_i = s'_i$, and $e_i = 0$ otherwise. The parameter ℓ is chosen so that the probability of any false match is negligible, as analyzed in Section 5.2. The complete construction of our mqRPMT is presented in Figure 11.

PARAMETERS: Sender $\mathcal{S}$, Receiver $\mathcal{R}$. The set size n, a multi-query OMT functionality $\mathcal{F}_{\mathsf{mqomt}}$, and a random oracle $H_\ell : \{0,1\}^* \to \{0,1\}^\ell$.
INPUTS: $\mathcal{S}$ inputs $X = \{x_i\}_{i\in[n]}$, while $\mathcal{R}$ inputs $Y = \{y_i\}_{i\in[n]}$.
PROTOCOL:

1. $\mathcal{S}$ and $\mathcal{R}$ invoke $\mathcal{F}_{\mathsf{mqomt}}$, where $\mathcal{S}$ inputs X as the sender and $\mathcal{R}$ inputs Y as the receiver. As a result, $\mathcal{S}$ obtains $T = \{t_1, \ldots, t_n\}$, while $\mathcal{R}$ obtains $S = \{s_i\}_{i\in[n]}$.
2. For $i \in [n]$, $\mathcal{S}$ computes $t'_i = H_\ell(t_i)$ and sends t'_i to $\mathcal{R}$.
3. For $i \in [n]$, $\mathcal{R}$ computes $s'_i = H_\ell(s_i)$. If $t'_i = s'_i$, then $\mathcal{R}$ sets $e_i = 1$, and otherwise, $e_i = 0$.
4. $\mathcal{R}$ outputs $E = \{e_i\}_{i\in[n]}$.

Fig. 11. Our Multi-Query RPMT construction Π_{mqrpmt}

5.2 Parameter Choice

The hash-then-truncate optimization reduces communication, but may introduce a false match: even when $t_i \neq s_i$, their truncated hash values may be equal. We therefore choose the output length ℓ so that the probability of any false match among the n comparisons is negligible in the statistical security parameter λ.

To this end, we choose the parameter ℓ of the random oracle H_ℓ according to Theorem 4. To ensure the probability of a false match after truncation is at

most $2^{-\lambda}$, it suffices to choose $\ell \geq \lambda + \lceil \log n \rceil$. Specifically, in our instantiation, we choose $\ell = 64$ and $\lambda = 40$, which achieves a probability of $2^{-42} \leq 2^{-\lambda}$ when $n = 2^{22}$.

Theorem 4. *Suppose that $H_\ell : \{0,1\}^* \to \{0,1\}^\kappa$ is a hash function modeled as a random oracle that outputs ℓ bits. Given n pairs of distinct values (x_i, y_i) such that $x_i \neq y_i$, the probability that there exists a false match satisfies*

$$\Pr\left[\exists i \in [n], s.t.\ H_\ell(x_i) = H_\ell(y_i)\right] \leq n \cdot 2^{-\ell}.$$

Proof. Consider any $i \in [n]$. Since H_ℓ is modeled as a random oracle and $x_i \neq y_i$, the values $H_\ell(x_i)$ and $H_\ell(y_i)$ are uniformly independent in $\{0,1\}^\ell$. Therefore,

$$\Pr\left[H_\ell(x_i) = H_\ell(y_i)\right] = 2^{-\ell}.$$

By the union bound, the probability P that there exists a false match among the n pairs (x_i, y_i) satisfies

$$\begin{aligned} P &= \Pr\left[\bigcup_{i\in[n]} \{H_\ell(x_i) = H_\ell(y_i)\}\right] \\ &\leq \sum_{i\in[n]} \Pr\left[H_\ell(x_i) = H_\ell(y_i)\right] \\ &= n \cdot 2^{-\ell}. \end{aligned}$$

5.3 PSU Protocol

Given the multi-query RPMT protocol, PSU can be realized using one OT for each sender element. Specifically, for each $i \in [n]$, $\mathcal{S}$ inputs $(x_i, \perp)$ to the OT, while $\mathcal{R}$ uses the membership-test bit e_i as its selection bit. Thus, $\mathcal{R}$ obtains x_i if $x_i \notin Y$ and obtains $\perp$ otherwise, and can locally compute $X \cup Y = Y \cup \{x_i \mid e_i = 0\}$. The resulting PSU protocol is presented in Figure 12.

We describe the offline/online realization and a communication optimization for the OTs used in the protocol.

The parties generate random OT correlations in the offline phase and convert them into standard OTs in the online phase. Specifically, $\mathcal{S}$ obtains two random messages (s_0, s_1), while $\mathcal{R}$ obtains a random bit r and the corresponding message s_r. In the online phase, $\mathcal{R}$, holding the selection bit b, sends $b \oplus r$ to $\mathcal{S}$, who holds (m_0, m_1). The sender then returns (c_0, c_1), where $c_i = m_i \oplus s_{b\oplus r\oplus i}$ for $i \in \{0,1\}$. Finally, $\mathcal{R}$ recovers m_b by computing $c_b \oplus s_r$.

In each OT used in our PSU protocol, $\mathcal{R}$ only needs to recover x_i when $e_i = 0$; when $e_i = 1$, the intended output is the fixed value $\perp$. Therefore, instead of sending both c_0 and c_1, $\mathcal{S}$ sends only c_0. This halves the sender's online OT payload communication.

PARAMETERS: Sender $\mathcal{S}$, Receiver $\mathcal{R}$. The set size n, a multi-query RPMT functionality $\mathcal{F}_{\mathsf{mqrpmt}}$, and an OT functionality $\mathcal{F}_{\mathsf{ot}}$
INPUTS: $\mathcal{S}$ inputs $X = \{x_i\}_{i\in[n]}$, while $\mathcal{R}$ inputs $Y = \{y_i\}_{i\in[n]}$.
PROTOCOL:

1. $\mathcal{S}$ and $\mathcal{R}$ invoke $\mathcal{F}_{\mathsf{mqrpmt}}$, where $\mathcal{S}$ inputs X as the sender while $\mathcal{R}$ inputs Y as the receiver. As a result, $\mathcal{R}$ obtains $E = \{e_1, \ldots, e_n\}$.
2. $\mathcal{S}$ and $\mathcal{R}$ invoke $\mathcal{F}_{\mathsf{ot}}$. For each $i \in [n]$, $\mathcal{R}$ inputs e_i while $\mathcal{S}$ inputs $(x_i, \perp)$.
3. $\mathcal{R}$ outputs $X \cup Y = Y \cup \{x_i | e_i = 0\}_{i\in[n]}$.

Remark. $\mathcal{S}$ enumerates its input elements in set X in a uniformly random order as $x_1, \ldots, x_n$.

Fig. 12. Our PSU protocol Π_{psu}

5.4 Security Proof

We first prove that our construction in Figure 11 securely realizes the functionality of multi-query RPMT.

Theorem 5. *In the $\mathcal{F}_{\mathsf{mqomt}}$-hybrid model and the random oracle model, Π_{mqrpmt} securely realizes $\mathcal{F}_{\mathsf{mqrpmt}}$ against semi-honest adversaries, with statistical error at most $2^{-\lambda}$.*

Proof. We prove correctness and security.

Correctness. By the correctness of the multi-query OMT functionality, for each $i \in [n]$, $\mathcal{S}$ obtains $t_i = s_i$ if $x_i \in Y$, and otherwise t_i is uniformly random and independent of s_i. Hence, if $x_i \in Y$, then $t'_i = s'_i$ and $\mathcal{R}$ sets $e_i = 1$. If $x_i \notin Y$, then $\mathcal{R}$ sets $e_i = 0$ unless a false match occurs. By the parameter choice in Section 5.2, the protocol is correct except with probability at most $2^{-\lambda}$.

Security. We construct simulators for the cases where either $\mathcal{S}$ or $\mathcal{R}$ is corrupted.

Corrupted $\mathcal{S}$. The simulator Sim receives the corrupted party's input $X = \{x_i\}_{i\in[n]}$. Since $\mathcal{S}$ receives no output from $\mathcal{F}_{\mathsf{mqrpmt}}$, the simulator is not given an ideal output.

For each $i \in [n]$, Sim independently samples $t_i \leftarrow_R \{0,1\}^\kappa$ and provides $T = \{t_i\}_{i\in[n]}$ as the output received from $\mathcal{F}_{\mathsf{mqomt}}$. The values $\{t'_i\}_{i\in[n]}$ sent by $\mathcal{S}$ are then computed according to Step 2. of the protocol.

In the real hybrid execution, each t_i received by $\mathcal{S}$ is uniformly random, regardless of whether $x_i \in Y$. Specifically, if $x_i \in Y$, then $t_i = s_i$ for a uniformly random s_i; otherwise, t_i is sampled uniformly and independently by $\mathcal{F}_{\mathsf{mqomt}}$. Therefore, the simulated view is identically distributed to the view of the corrupted $\mathcal{S}$ in the real hybrid execution.

Corrupted $\mathcal{R}$. The simulator Sim receives the corrupted party's input $Y = \{y_i\}_{i\in[n]}$ and its ideal output $E = \{e_i\}_{i\in[n]}$ from $\mathcal{F}_{\mathsf{mqrpmt}}$. It reconstructs the view of $\mathcal{R}$ as follows.

For each $i \in [n]$, Sim independently samples $s_i \leftarrow_R \{0,1\}^\kappa$ and provides $S = \{s_i\}_{i\in[n]}$ as the output received from $\mathcal{F}_{\mathsf{mqomt}}$. It then computes $s'_i = H_\ell(s_i)$. To simulate the message received from $\mathcal{S}$, for each $i \in [n]$, Sim sets $t'_i = s'_i$ if $e_i = 1$, and otherwise samples $t'_i \leftarrow_R \{0,1\}^\ell \setminus \{s'_i\}$.

We first verify consistency with the prescribed ideal output. For each $i \in [n]$, if $e_i = 1$, then $t'_i = s'_i$, and hence $\mathcal{R}$ outputs 1. If $e_i = 0$, then $t'_i \neq s'_i$, and hence $\mathcal{R}$ outputs 0. Therefore, the simulated execution produces exactly the output E given by the ideal functionality.

It remains to show that the simulated view has the correct distribution. In the real hybrid execution, the values $\{s_i\}_{i\in[n]}$ received from $\mathcal{F}_{\mathsf{mqomt}}$ are uniformly random and independent. If $x_i \in Y$, then $t_i = s_i$, and therefore $t'_i = s'_i$. If $x_i \notin Y$, then t_i is uniformly random and independent of s_i. Conditioned on the absence of a false match, t'_i is uniformly distributed over $\{0,1\}^\ell \setminus \{s'_i\}$.

Thus, conditioned on the absence of a false match, the joint distribution of $(\{s_i\}_{i\in[n]}, \{t'_i\}_{i\in[n]})$ in the simulated execution is identical to that in the real hybrid execution. By Section 5.2, the probability of a false match is at most $2^{-\lambda}$. Consequently, the simulated view is statistically indistinguishable from the view of the corrupted $\mathcal{R}$, with statistical distance at most $2^{-\lambda}$.

We then prove that our PSU protocol in Figure 12 securely realizes the functionality of PSU.

Theorem 6. *In the $(\mathcal{F}_{\mathsf{mqrpmt}}, \mathcal{F}_{\mathsf{ot}})$-hybrid model, Π_{psu} securely realizes $\mathcal{F}_{\mathsf{psu}}$ against semi-honest adversaries.*

Proof. We prove correctness and security.

Correctness. By the correctness of the multi-query RPMT functionality, $\mathcal{R}$ obtains $e_i = 1$ if $x_i \in Y$, and otherwise obtains $e_i = 0$. By the correctness of the OT functionality, for each $i \in [n]$, $\mathcal{R}$ obtains x_i if $e_i = 0$, and obtains $\perp$ otherwise. Therefore, $\mathcal{R}$ can compute

$$X \cup Y = Y \cup \{x_i \mid e_i = 0\}.$$

Hence, the protocol satisfies correctness.

Security. We construct simulators for the cases where either $\mathcal{S}$ or $\mathcal{R}$ is corrupted.

Corrupted $\mathcal{S}$. The simulator Sim receives the corrupted party's input X. Since $\mathcal{S}$ receives no output from $\mathcal{F}_{\mathsf{psu}}$, the simulator is not given an ideal output.

Sim samples a uniformly random enumeration $x_1, \ldots, x_n$ of the elements in X, as in the real protocol. It provides this enumeration as $\mathcal{S}$'s input to $\mathcal{F}_{\mathsf{mqrpmt}}$ and provides $(x_i, \perp)$ as $\mathcal{S}$'s input to the i-th invocation of $\mathcal{F}_{\mathsf{ot}}$.

Neither $\mathcal{F}_{\mathsf{mqrpmt}}$ nor $\mathcal{F}_{\mathsf{ot}}$ gives any output to $\mathcal{S}$. Therefore, Sim samples the same randomness and reconstructs exactly the same view as in the real hybrid execution. Consequently, the simulated view is identically distributed to the view of the corrupted $\mathcal{S}$.

Corrupted $\mathcal{R}$. Sim receives the corrupted party's input Y and its ideal output $Z = X \cup Y$ from $\mathcal{F}_{\mathsf{psu}}$. It reconstructs the view of $\mathcal{R}$ as follows. Sim computes

$D = Z \setminus Y = X \setminus Y$ and lets $w = |D|$. It uniformly selects a subset $I_0 \subseteq [n]$ of size w and defines

$$e'_i = \begin{cases} 0, & \text{if } i \in I_0, \\ 1, & \text{otherwise.} \end{cases}$$

It provides $E' = \{e'_i\}_{i\in[n]}$ as the output of $\mathcal{F}_{\mathsf{mqrpmt}}$.

Sim then chooses a uniformly random bijection $\phi : I_0 \to D$. For each $i \in [n]$, it provides

$$q'_i = \begin{cases} \phi(i), & \text{if } i \in I_0, \\ \perp, & \text{otherwise} \end{cases}$$

as the output received from the i-th invocation of $\mathcal{F}_{\mathsf{ot}}$.

We first verify consistency with the prescribed ideal output. Using the simulated multi-query RPMT and OT outputs, $\mathcal{R}$ computes

$$Y \cup \{q'_i \mid e'_i = 0\} = Y \cup D = Z.$$

Hence, the simulated execution produces exactly the union given by the ideal functionality.

It remains to show that the simulated view has the correct distribution. In the real hybrid execution, $\mathcal{S}$ enumerates the elements of X in a uniformly random order. Therefore, conditioned on Y and Z, the positions corresponding to the w elements in $X \setminus Y$ form a uniformly random subset of $[n]$ of size w. Moreover, the elements of $X \setminus Y$ appear in these positions in a uniformly random order. At all remaining positions, the membership-test bit is 1 and the corresponding OT output is $\perp$.

The simulator generates E' and the OT outputs according to exactly the same distribution. Therefore, conditioned on the input and output of $\mathcal{R}$, the simulated view is identically distributed to its view in the real hybrid execution.

5.5 Complexity Analysis

Multi-query RPMT The multi-query RPMT protocol invokes one multi-query OMT on n inputs. Then, each party performs n hash and truncation operations, while $\mathcal{R}$ additionally performs n local comparisons. Therefore, the total computational cost is

$$T_{\mathsf{mqrpmt}}(n) = T_{\mathsf{mqomt}}(n) + O(n).$$

The only communication outside the multi-query OMT is the transmission of n ℓ-bit tags. Thus, the total communication is

$$C_{\mathsf{mqrpmt}}(n) = C_{\mathsf{mqomt}}(n) + n\ell.$$

PSU Besides the multi-query RPMT, PSU protocol invokes one random OT of size n. Each party then performs $O(n)$ XOR operations to convert random OT correlations into standard OTs. Hence, the total computational cost is

$$T_{\mathsf{psu}}(n) = T_{\mathsf{mqrpmt}}(n) + T_{\mathsf{rot}}(n) + O(n).$$

The additional communication outside the multi-query RPMT is random OT and the transmission of n messages c_0. We suppose that c_0 and each input element have the same bit length ℓ'. Hence, the total communication is

$$C_{\mathsf{psu}}(n) = C_{\mathsf{mqrpmt}}(n) + C_{\mathsf{rot}}(n) + n\ell'.$$

6 Experiment

In this section, we evaluate our PSU protocol and compare its communication cost and running time with those of state-of-the-art PSU protocols [4, 16, 22, 23] under different network settings.

6.1 Implementation Details

We implement our protocol in C++, and the source code is available at `https://anonymous.4open.science/r/OMT-PSU-E941/`. We instantiate the functionalities used in our constructions with OPRF from [20], soOPRF from [1], OKVS from [18], and OT from [21]. We implement OPRF using VOLE in libOTe[4], instantiating the code family with Expand-Convolute codes [19]. The implementation of OT is also provided in libOTe. We use the implementation of soOPRF in secure-join[5]. Finally, we use the OKVS implementation in volepsi[6] with field $\mathbb{F} = GF(2^{128})$.

6.2 Experimental Setup

The benchmark machine is equipped with an Intel(R) Xeon(R) CPU E5-2696 v4 @ 2.20GHz and 256GB RAM. We simulate the network connection using the Linux command tc. Similar to previous works [4, 16], in the LAN setting, the bandwidth is set to 10 Gbps with 0.2 ms RTT. For the WAN setting, we consider WAN1 with 200 Mbps and 80 ms RTT, WAN2 with 50 Mbps and 80 ms RTT, and WAN3 with 10 Mbps and 80 ms RTT.

Our evaluations are performed with a statistical security parameter $\lambda = 40$ and a computational security parameter $\kappa = 128$. The elements of the input sets are generated randomly as 128-bit strings. All results are averaged over five runs. The communication cost refers to the total amount of data sent and received by the receiver.

[4] `https://github.com/osu-crypto/libOTe`
[5] `https://github.com/ladnir/secure-join`
[6] `https://github.com/ladnir/volepsi`

6.3 Comparison with Existing Protocols

We compare our protocol with state-of-the-art PSU protocols using their open-source implementations. Most PSU protocols are based on RPMT, and we compare with the state-of-the-art RPMT-based protocols [23][7] and [4][8]. The work of [23] provides two PSU protocols based on public-key and symmetric-key techniques, respectively. We choose the more efficient PKE-PSU protocol for comparison. In addition, the protocol of [4] is based on elliptic-curve cryptography. In our comparison, we choose its Curve25519-based protocol. For PSU with enhanced security, referred to as enhanced PSU (ePSU), we compare with the state-of-the-art ePSU protocol [22][9]. We also compare with the recent IBLT-based protocol [16][10]. Since the public implementation of [16] supports only 64-bit elements, we modify our implementation to represent each set element as a 64-bit string to ensure a fair comparison. We mark the protocols using 64-bit elements with ★.

We evaluate these protocols for set sizes $n \in \{2^{14}, 2^{16}, 2^{20}, 2^{22}\}$. The comparison results are presented in Table 2. Since our protocol and several prior protocols [16, 23] include an offline phase, we report the running time and communication cost of both the offline and online phases in Table 2. To compare with protocols without an offline phase, we also report the total (offline + online) running time and communication cost. Similar to [16], our protocol uses SoftSpokenOT [21] to generate random OT correlations in the offline phase. We use different parameter settings for SoftSpokenOT to optimize its performance in the LAN and WAN settings. Therefore, we report two offline-phase communication costs in Table 2.

As shown in Table 2, compared with the RPMT-based protocols and the ePSU protocol, our protocol achieves the best online running time in all settings except WAN3, where it achieves the second-best performance. Specifically, in the LAN setting, the online phase of our protocol achieves a 41.66×–48.89× speedup, while its total running time is also the lowest, corresponding to a 3.54×–6.61× speedup. In the WAN setting, the online phase of our protocol achieves a 1.36×–4.51× speedup in WAN1 and WAN2. For large set sizes $n \in \{2^{20}, 2^{22}\}$, our protocol also achieves the best total running time in WAN1 and WAN2, with a speedup of 1.18×–2.56×.

Compared with the IBLT-based protocol [16], our PSU protocol achieves comparable performance. It reduces the online communication cost by a factor of 1.01×–1.26× and the total communication cost by a factor of 1.16×–1.44×. The improvement is more significant for smaller set sizes, where the IBLT has a larger expansion factor. In terms of running time, our online phase is 1.86×–2.49× faster in the LAN setting. In the WAN settings, it is up to 1.43× faster, although it is 1.25× slower in the least favorable case. Considering the total running time,

[7] `https://github.com/alibaba-edu/mpc4j/tree/main/mpc4j-s2pc-pso/src/main/java/edu/alibaba/mpc4j/s2pc/pso/psu/zcl23`

[8] `https://github.com/yuchen1024/Kunlun/tree/master/mpc/pso`

[9] `https://doi.org/10.5281/zenodo.14725816`

[10] `https://github.com/asu-crypto/IBLT-based-PSU`

Table 2. The running time (in seconds) and communication cost (in MB) comparisons between our protocol and other PSU protocols. Cells with − imply that the scheme does not have this phase. The best result among PMT-based PSU is marked in blue. The best result between PT26 [16] and ours is marked in orange. In our scheme and PT26 [16], we set the field size as 3 for LAN and 8 for WAN.

Size	Scheme	Running Time (seconds)												Comm. (MB)		
		LAN			WAN1			WAN2			WAN3					
		Offline	Online	Total	Offline	Online	Total	Offline	Online	Total	Offline	Online	Total	Offline	Online	Total
2^{14}	ZCL+23 [23]	5.541	3.603	9.144	5.778	3.962	9.740	5.942	4.136	10.08	5.780	5.638	11.42	0.004	3.053	3.057
	CZZ+24 [4]	–	1.657	1.657	–	2.296	2.296	–	2.414	2.414	–	3.482	3.482	–	1.644	1.644
	TBZ+25 [22]	–	3.905	3.905	–	5.615	5.615	–	5.938	5.938	–	8.487	8.487	–	4.760	4.760
	Ours	0.434	0.034	0.467	2.063	1.237	3.299	2.046	1.779	3.825	4.408	4.976	9.384	3.259 3.207	5.011	8.270 8.218
	PT26★ [16]	0.176	0.087	0.263	1.844	0.984	2.828	2.300	1.790	4.090	3.725	5.842	9.566	5.565 3.007	6.156	11.72 9.163
	Ours★	0.433	0.035	0.468	2.065	1.232	3.296	2.048	1.758	3.806	4.403	4.871	9.274	3.259 3.207	4.886	8.145 8.093
2^{16}	ZCL+23 [23]	5.489	10.90	16.39	5.630	11.66	17.29	5.678	12.55	18.23	5.793	17.80	23.59	0.004	12.20	12.21
	CZZ+24 [4]	–	6.160	6.160	–	6.447	6.447	–	7.386	7.386	–	11.64	11.64	–	6.482	6.482
	TBZ+25 [22]	–	11.97	11.97	–	13.98	13.98	–	14.42	14.42	–	25.18	25.18	–	16.75	16.75
	Ours	1.203	0.126	1.329	2.856	2.089	4.944	3.328	4.610	7.938	9.442	18.02	27.47	9.366 9.155	20.20	29.57 29.35
	PT26★ [16]	0.276	0.248	0.524	2.581	2.285	4.866	3.171	4.857	8.028	7.949	19.47	27.41	17.11 7.771	21.80	38.91 29.57
	Ours★	1.091	0.133	1.224	2.897	2.067	4.964	3.347	4.488	7.836	9.442	17.60	27.05	9.366 9.155	19.70	29.07 28.85
2^{20}	ZCL+23 [23]	5.454	179.7	185.2	5.538	181.9	187.4	5.534	198.9	204.4	5.545	281.1	286.6	0.004	195.2	195.2
	CZZ+24 [4]	–	96.30	96.30	–	84.05	84.05	–	96.66	96.66	–	167.8	167.8	–	103.2	103.2
	TBZ+25 [22]	–	156.5	156.5	–	153.2	153.2	–	176.7	176.7	–	359.3	359.3	–	263.0	263.0
	Ours	12.56	2.183	14.74	15.49	18.80	34.29	24.44	56.79	81.23	109.0	276.5	385.5	130.8 127.4	323.6	454.4 451.0
	PT26★ [16]	1.953	4.689	6.643	8.811	26.38	35.19	18.58	65.15	83.73	77.18	279.1	356.3	199.8 88.64	319.9	519.7 408.6
	Ours★	12.58	2.184	14.76	15.59	18.40	33.98	24.38	55.49	79.87	109.3	269.5	378.7	130.8 127.4	315.6	446.4 443.0
2^{22}	ZCL+23 [23]	5.458	765.1	770.6	5.536	792.4	797.9	5.522	846.8	852.3	5.529	1185	1191	0.004	780.6	780.6
	CZZ+24 [4]	–	388.6	388.6	–	331.4	331.4	–	381.7	381.7	–	690.9	690.9	–	412.8	412.8
	TBZ+25 [22]	–	635.4	635.4	–	594.2	594.2	–	713.7	713.7	–	1440	1440	–	1063	1063
	Ours	49.49	9.328	58.82	55.93	73.42	129.4	89.80	225.0	314.8	424.2	1101	1525	519.3 505.8	1295	1814 1801
	PT26★ [16]	8.617	23.31	31.93	28.84	99.10	127.9	67.85	247.5	315.4	303.7	1106	1410	797.3 352.0	1281	2078 1633
	Ours★	49.56	9.383	58.95	55.87	72.23	128.1	90.55	220.0	310.6	423.7	1073	1497	519.3 505.8	1263	1782 1769

our protocol achieves comparable performance in WAN2 and WAN3, ranging from 1.06× slower to 1.07× faster.

7 Conclusion

In this work, we focus on constructing an efficient RPMT-based PSU protocol in the semi-honest model. We propose a new functionality called multi-query OMT,

which allows the receiver to obliviously transfer values to the sender according to the membership information. The obtained values can then be revealed directly and compared in the clear to derive the membership-test bits. We realize this functionality using our batched mDK-OPRF, a novel functionality that enables the batched, masked, and oblivious evaluation of a dual-key PRF. These two functionalities are built entirely from symmetric-key primitives without using hash-to-bin techniques. This results in an RPMT-based PSU protocol that combines linear complexity with high concrete efficiency.

We find that multi-query OMT can also be leveraged in other variants of PSU, such as PSU with enhanced security [12]. Specifically, we may replace the comparison in the clear with a secure comparison protocol. We leave this direction to future work.

Claim: Independent and concurrent work. This work was completed and submitted for peer review in June 2026, before [9] became publicly available on July 13, 2026. We learned of [9] only after its release, and the two works were therefore developed independently.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Alamati, N., Policharla, G.V., Raghuraman, S., Rindal, P.: Improved alternating-moduli prfs and post-quantum signatures. In: Annual International Cryptology Conference. pp. 274–308. Springer (2024)
2. Bienstock, A., Patel, S., Seo, J.Y., Yeo, K.: Near-optimal oblivious key-value stores for efficient psi, psu and volume-hiding multi-maps. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 301–318 (2023)
3. Chandran, G.R., Schneider, T., Stillger, M., Weinert, C.: Concretely efficient private set union via circuit-based psi. In: Proceedings of the 20th ACM Asia Conference on Computer and Communications Security. pp. 149–162 (2025)
4. Chen, Y., Zhang, M., Zhang, C., Dong, M., Liu, W.: Private set operations from multi-query reverse private membership test. In: IACR international conference on public-key cryptography. pp. 387–416. Springer (2024)
5. Davidson, A., Cid, C.: An efficient toolkit for computing private set operations. In: Australasian Conference on Information Security and Privacy. pp. 261–278. Springer (2017)
6. Frikken, K.: Privacy-preserving set union. In: International Conference on Applied Cryptography and Network Security. pp. 237–252. Springer (2007)
7. Garimella, G., Mohassel, P., Rosulek, M., Sadeghian, S., Singh, J.: Private set operations from oblivious switching. In: IACR international conference on public-key cryptography. pp. 591–617. Springer (2021)
8. Garimella, G., Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Oblivious key-value stores and amplification for private set intersection. In: Annual International Cryptology Conference. pp. 395–425. Springer (2021)
9. Hao, M., Wang, G., Yang, X., Xing, P., Chen, H., Zhang, T., Xue, H., Yang, G., Li, H., Deng, R.H.: Enhanced private set union from secret-shared private membership test. In: USENIX Security'26 (2026)

10. Hogan, K., Luther, N., Schear, N., Shen, E., Stott, D., Yakoubov, S., Yerukhimovich, A.: Secure multiparty computation for cooperative cyber risk assessment. In: 2016 IEEE Cybersecurity Development (SecDev). pp. 75–76. IEEE (2016)
11. Jia, Y., Sun, S.F., Zhou, H.S., Du, J., Gu, D.: Shuffle-based private set union: Faster and more secure. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 2947–2964 (2022)
12. Jia, Y., Sun, S.F., Zhou, H.S., Gu, D.: Scalable private set union, with stronger security. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 6471–6488 (2024)
13. Kissner, L., Song, D.: Privacy-preserving set operations. In: Annual International Cryptology Conference. pp. 241–257. Springer (2005)
14. Kolesnikov, V., Rosulek, M., Trieu, N., Wang, X.: Scalable private set union from symmetric-key techniques. In: International conference on the theory and application of cryptology and information security. pp. 636–666. Springer (2019)
15. Lenstra, A., Voss, T.: Information security risk assessment, aggregation, and mitigation. In: Australasian Conference on Information Security and Privacy. pp. 391–401. Springer (2004)
16. Piske, L., Trieu, N.: Is psi really faster than psu? achieving efficient psu with invertible bloom filters. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 438–467. Springer (2026)
17. Pu, S., Gao, J., Trieu, N.: Malicious private set union with two-sided output. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 408–437. Springer (2026)
18. Raghuraman, S., Rindal, P.: Blazing fast psi from improved okvs and subfield vole. In: Proceedings of the 2022 ACM SIGSAC conference on computer and communications security. pp. 2505–2517 (2022)
19. Raghuraman, S., Rindal, P., Tanguy, T.: Expand-convolute codes for pseudorandom correlation generators from lpn. In: Annual International Cryptology Conference. pp. 602–632. Springer (2023)
20. Rindal, P., Schoppmann, P.: Vole-psi: fast oprf and circuit-psi from vector-ole. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 901–930. Springer (2021)
21. Roy, L.: Softspokenot: Quieter ot extension from small-field silent vole in the minicrypt model. In: Annual international cryptology conference. pp. 657–687. Springer (2022)
22. Tu, B., Bai, Y., Zhang, C., Cao, Y., Chen, Y.: Fast enhanced private set union in the balanced and unbalanced scenarios. In: Proceedings of the 34th USENIX Conference on Security Symposium. SEC '25, USENIX Association, USA (2025)
23. Zhang, C., Chen, Y., Liu, W., Zhang, M., Lin, D.: Linear private set union from multi-query reverse private membership test. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 337–354 (2023)