Enhanced Private Set Union from Secret-shared Private Membership Test

Meng Hao[1], Guodong Wang[2], Xinpeng Yang[3], Pengzhi Xing[2], Hanxiao Chen[2,✉], Tianwei Zhang[3], Haiyang Xue[1], Guomin Yang[1], Hongwei Li[2,✉], Robert H. Deng[1]
[1]*Singapore Management University*
[2]*University of Electronic Science and Technology of China*
[3]*Nanyang Technological University*

Abstract

Private set union (PSU) enables two parties to compute the union of their sets without revealing additional information. Jia et al. (USENIX Security 2024) point out that most scalable PSU protocols suffer from *during-execution leakage*, whereby membership information is revealed to the receiver before the protocol completes, and consequently introduce the functionality of *enhanced PSU* to mitigate this issue. While several recent works propose protocols for enhanced PSU, existing solutions either incur high computational overhead due to reliance on computation-intensive public-key primitives, or suffer from large communication costs arising from generic secure computation.

In this work, we present a modular framework for constructing enhanced PSU based on the multi-query secret-shared private membership test (ssPMT) protocol (ASIACRYPT 2023) and a new primitive called secret-shared oblivious transfer with default (ssOTd). Our framework eliminates during-execution leakage by ensuring that all intermediate information remains secret-shared between the parties throughout the protocol. At its core, we present two efficient ssPMT constructions relying primarily on lightweight symmetric-key primitives. We further design a customized, communication-efficient ssOTd protocol to optimize the overhead of enhanced PSU. As a byproduct, we obtain a computation-efficient construction of multi-query reverse PMT (mqRPMT) that avoids the computation-heavy public-key primitives used in the state-of-the-art approaches.

We implement two enhanced PSU protocols, denoted as ePSU-fast and ePSU-low, which are optimized for computational efficiency and communication efficiency, respectively, making them suitable for different network settings. Extensive evaluations show that ePSU-low simultaneously achieves lower computation and communication overhead than the state-of-the-art protocols by Jia et al. (USENIX Security 2024) and Tu et al. (USENIX Security 2025). Moreover, the ePSU-fast protocol further reduces computation costs compared to ePSU-low, at the expense of increased communication.

1 Introduction

Private set union (PSU) [26] enables a sender $\mathcal{S}$ holding a set X and a receiver $\mathcal{R}$ holding a set Y to privately compute the union $X \cup Y$, which is revealed to the receiver, while revealing no additional information beyond the output. PSU is a fundamental primitive with many practical applications. Representative examples include privacy-preserving IP and domain blacklist aggregation among Internet service providers and security vendors [21, 41], cyber risk assessment and management [29], and private database operations such as supporting a full join across distributed databases owned by mutually distrustful parties [28, 50]. In these applications, parties need to combine datasets while hiding the intersected elements.

In recent years, a number of works [9, 11, 17, 25, 28, 50] and references therein have advanced the development of PSU. Notably, Zhang et al. [50] propose a general and modular framework for constructing PSU protocols. Their framework consists of two main components: multi-query reverse private membership test (mqRPMT) and oblivious transfer (OT). Figure 3 depicts this framework. Specifically, the two parties first invoke mqRPMT, where the sender $\mathcal{S}$ inputs its set X, and the receiver $\mathcal{R}$ inputs its set Y and learns the membership bits $b_i := 1\{x_i \in Y\}$ for each $x_i \in X$. Then, for each element $x_i \in X$, the parties invoke an OT instance, where $\mathcal{S}$ inputs two messages $(x_i, \perp)$ and $\mathcal{R}$ inputs the choice bit b_i, obtaining x_i if and only if $b_i = 0$, i.e., when $x_i \notin Y$. Most recent PSU protocols [9, 11, 17, 25, 50] fall within this framework, differing mainly in their concrete instantiations of the mqRPMT component. Section 1.2 presents a detailed comparison of existing PSU constructions.

However, Jia et al. [23] point out that the previous mqRPMT-based PSU framework suffers from *during-execution leakage*. In particular, mqRPMT reveals membership information (including the intersection size) to the receiver before the protocol execution completes. Such leakage could lead to serious practical consequences. As shown by Jia et al., the recent work [19] demonstrates concrete attacks on protocols that leak intersection sizes, successfully

extracting sensitive information (e.g., COVID-19 status and personal interests) from real-world datasets. In the context of mqRPMT-based PSU, an adversarial receiver can exploit during-execution leakage by aborting the protocol once membership information is obtained (e.g., due to network disruptions or server failures), and then re-running the protocol with modified inputs. By aggregating intersection sizes across multiple executions, the receiver can infer partial information about the intersection. We refer to [23] for a comprehensive analysis of this leakage and illustrative attack examples.

To address this during-execution leakage, Jia et al. [23] define a new PSU functionality with enhanced security, referred to as *enhanced PSU*. It captures the stronger security guarantee that the receiver learns no information prior to obtaining the final output $X \cup Y$. They further present a corresponding enhanced PSU protocol that is primarily built from symmetric-key primitives. In a subsequent work, Tu et al. [44] show that the construction of Jia et al. incurs substantial communication overhead, mainly due to the use of the generic permute+share protocol [10] with super-linear asymptotic complexity. To reduce this overhead, Tu et al. [44] propose a new enhanced PSU construction based on the DDH assumption, achieving linear complexity in both computation and communication. Nevertheless, this improvement comes at the cost of relying on expensive public-key operations, which result in higher computational overhead compared to the symmetric-key-based construction of Jia et al.

In summary, existing enhanced PSU protocols either incur substantial computational overhead due to reliance on expensive public-key cryptographic primitives, or suffer from high communication costs stemming from communication-intensive generic secure computation. Motivated by these limitations, we ask the following question:

Can we design enhanced PSU protocols that simultaneously achieve low computation and low communication overhead?

1.1 Our Contribution

In this work, we answer the above question affirmatively by proposing a modular framework for constructing enhanced PSU and two instantiations primarily from symmetric-key primitives. We summarize our contributions as follows.

A Modular Framework for Enhanced PSU. To eliminate during-execution leakage, we propose a modular framework for enhanced PSU that ensures all intermediate information remains secret-shared between the parties throughout the protocol. Our framework is built upon multi-query secret-shared PMT (ssPMT) [32] and a new primitive called secret-shared oblivious transfer with default (ssOTd). The ssPMT functionality enables private membership tests while keeping the results secret-shared, and the proposed ssOTd protocol further allows the receiver to efficiently obtain the final PSU output with secret-shared membership information.

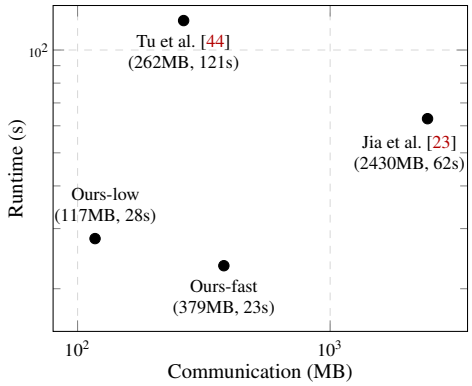


Figure 1: Communication and runtime comparisons between our two constructions (ePSU-low and ePSU-fast) and the state-of-the-art protocols of Jia et al. [23] and Tu et al. [44] for a set size of 2^{20}. Both axes are shown on a logarithmic scale.

Two Efficient Constructions of ssPMT. We present two efficient ssPMT constructions based primarily on symmetric-key primitives. The first construction is built from secret-shared oblivious PRF (OPRF) [1, 46], while the second leverages a new primitive called secret-shared bit permutation. These two constructions offer complementary tradeoffs between computational and communication efficiency, making them suitable for different network settings. Both constructions significantly outperform the prior ssPMT construction [32]. It is worth noting that building upon our first ssPMT construction, we realize the first linear-complexity enhanced PSU protocol from symmetric-key primitives.

Computation-efficient mqRPMT from SKE. As a byproduct, we obtain computation-efficient mqRPMT protocols derived from our ssPMT constructions, eliminating the reliance on public-key primitives such as DDH used in the state-of-the-art work [11]. From a theoretical perspective, our approach shifts mqRPMT from computation-intensive public-key operations to lightweight symmetric-key primitives. The evaluation demonstrates that our mqRPMT protocols achieve lower running time even in bandwidth-limited scenarios.

Extensive Experimental Evaluation. We implement two enhanced PSU protocols, denoted as ePSU-fast and ePSU-low, optimized for computational efficiency and communication efficiency, respectively. Our experimental results show that ePSU-low consistently and substantially outperforms state-of-the-art protocols across all evaluated settings. Specifically, it achieves a $14.25 \sim 20.72\times$ reduction in communication and a $1.44 \sim 17.47\times$ reduction in runtime compared to Jia et al. [23], and a $2.06 \sim 2.24\times$ reduction in communication and a $1.46 \sim 4.35\times$ reduction in runtime compared to Tu et al. [44]. Moreover, ePSU-fast further improves computational efficiency over ePSU-low, at the cost of increased communication overhead. Figure 1 provides an overview comparison with the state-of-the-art protocols.

Table 1: Asymptotic communication (Comm.) and computation (Comp.) complexities, security against during-execution leakage (Sec.), and computational assumptions (Assum.) of two-party PSU protocols. Here, n denotes the set size, and ske (resp. pke) refers to symmetric (resp. public) key primitives.

Protocols	Comm.	Comp.	Sec.	Assum.
[28]	$O(n\log n)$	$O(n\log n\log\log n)$	×	ske
[17]	$O(n\log n)$	$O(n\log n)$	×	ske
[25]	$O(n\log n)$	$O(n\log n)$	×	ske
[9]	$O(n\log n)$	$O(n\log n)$	×	ske
[50]	$O(n)$	$O(n)$	×	ske/pke
[11]	$O(n)$	$O(n)$	×	pke
[23]	$O(n\log n)$	$O(n\log n)$	√	ske
[44]	$O(n)$	$O(n)$	√	pke
Ours-low	$O(n\log n)$	$O(n\log n)$	√	ske
Ours-fast	$O(n)$	$O(n)$	√	ske

1.2 Related Work

We focus on recent works on standard PSU and PSU against during-execution leakage that are most relevant to our approach, and omit other variants such as unbalanced PSU [45, 49] and multi-party PSU [14–16, 32]. Table 1 summarizes a comparison of existing protocols.

Early progress of PSU was made by Kolesnikov et al. [28]. They introduce the notion of reverse private membership test (RPMT) and demonstrate that PSU protocols can be built from RPMT based on symmetric key primitives, which significantly outperform earlier constructions based on additively homomorphic encryption. Subsequently, Garimella et al. [17] present a permuted characteristic functionality and propose a unified framework for private set operations based on it. Concurrently, Jia et al. [25] employ permutation techniques to construct a generalized RPMT and derive two corresponding PSU protocols. While both the works of Garimella et al. and Jia et al. mainly rely on symmetric key primitives, these works introduce super-linear $O(n\log n)$ communication and computational overhead, mainly due to the use of generic permute+share protocols [10].

Recently, Zhang et al. [50] present the first linear-complexity PSU protocols. They extend RPMT to the multi-query setting, formalized as multi-query RPMT (mqRPMT), and present a general PSU framework from mqRPMT and OT. Their mqRPMT functionality is instantiated with either symmetric-key or public-key encryption. Subsequently, Chen et al. [11] present optimized mqRPMT constructions using commutative weak PRFs and permuted oblivious PRFs. Both constructions rely on DDH-based public-key assumptions and achieve linear computational and communication complexity, which outperform previous PSU protocols. More recently, Chandran et al. [9] instantiate the mq-RPMT functionality with circuit-based PSI protocols, which can be constructed from mainly symmetric-key techniques. Nevertheless, their PSU protocol incurs super-linear asymptotic complexity, resulting in slightly better runtime but approximately twice the communication cost compared to Chen et al. [11].

Despite these advances, Jia et al. [23] observe that the above (mq)RPMT-based PSU constructions reveal membership information to the receiver before the protocol completes, leading to a specific during-execution leakage pattern. To address this issue, they propose PSU against this during-execution leakage pattern, dubbed as enhanced PSU. Their protocol is designed based on a new functionality of equality conditional random generation (ECRG), along with oblivious programmable PRFs and permuted+share techniques. However, the generic permuted+share protocol they used incurs a super-linear $O(n\log n)$ communication and computational complexity. Especially, the large communication cost becomes the dominant performance bottleneck for larger datasets. Subsequently, to mitigate this cost, Tu et al. [44] present communication-efficient protocols with $O(n)$ complexity utilizing permuted equality-conditional random generation (pECRG). Nevertheless, their approach employs public-key primitives, imposing a heavy computational burden compared to symmetric-key operations.

In contrast to these works, we introduce a modular framework that is as lean as the mqRPMT-based PSU paradigm, while preventing the during-execution leakage inherent in previous PSU protocols. We present two efficient instantiations primarily based on symmetric-key primitives, denoted as ePSU-fast and ePSU-low, which are optimized for computational efficiency and communication efficiency, respectively. Notably, ePSU-fast is the first PSU protocol against during-execution leakage built from symmetric-key primitives that achieves linear computational and communication complexity. On the other hand, ePSU-low incurs super-linear asymptotic complexity due to a permutation operation; however, our customized protocol confines this cost to a small fraction of the overall overhead, as it operates only on bit inputs.

Concurrent Work. A concurrent work on PSU against during-execution leakage [31] optimizes the protocols of [44], but it still relies on computationally intensive public-key techniques. In contrast, our protocols primarily utilize lightweight symmetric-key primitives. Furthermore, our protocols achieve $2.15 \sim 2.68\times$ lower communication overhead than this concurrent work. We also note that [31] studies the leakage in Cuckoo-hashing-based PSU. While a similar leakage analysis was initially presented in [30], Remark 1 of [30] clarifies that this leakage only arises under its specifically defined *non-leaky model*, rather than the standard semi-honest model adopted by our protocols.

Additionally, another concurrent work on PSU [38] revisits the enhanced PSU functionality defined in [23], arguing that the functionality of enhanced PSU is equivalent to that of standard PSU. It provides a reduction claiming that any protocol realizing standard PSU functionality can also realize enhanced PSU functionality, and vice versa. Subsequently, a

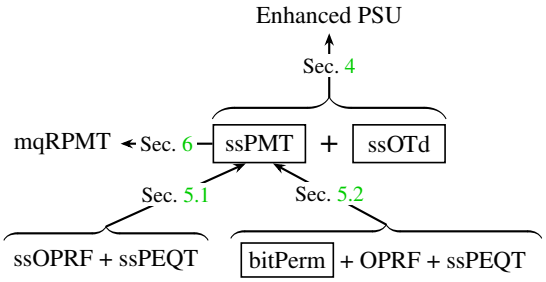


Figure 2: High-level overview of our techniques. The rectangles consist of our new protocols in this work.

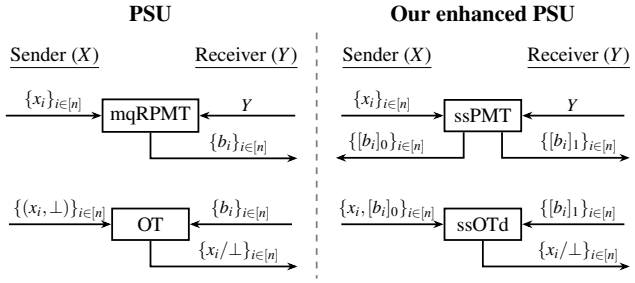


Figure 3: Our enhanced PSU framework, compared with the standard PSU framework.

recent update to [23] (in its ePrint version [24], Appendix B) clarifies that there are some subtle differences between its definition of enhanced PSU functionality and that of standard PSU. We would like to emphasize that, irrespective of these definitional nuances in prior works, a central goal of this work is to prevent such during-execution leakage in real-world PSU-based applications, as our protocols ensure that all intermediate information remains securely secret-shared between the parties throughout the entire execution.

2 Technical Overview

In this section, we provide a high-level overview of our techniques and summarize them in Figure 2.

2.1 General Framework of Enhanced PSU

General Framework. We present a modular and general framework for constructing enhanced PSU from multi-query secret-shared private membership test (ssPMT) [32] and a new primitive called secret-shared oblivious transfer with default (ssOTd). Figure 3 illustrates our enhanced PSU framework in comparison with the standard PSU framework based on mqRPMT and OT. Our framework inherits the simplicity of the standard PSU paradigm, while strengthening it by enforcing that all intermediate information remains secret-shared between the two parties. As a result, the receiver learns the output only upon protocol completion, thereby preventing during-execution leakage.

Our framework consists of two phases. In the first phase, the parties execute ssPMT, where the sender $\mathcal{S}$ inputs a set $X = \{x_i\}_{i\in[n]}$, while the receiver $\mathcal{R}$ inputs a set $Y = \{y_i\}_{i\in[n]}$. The protocol outputs $\{[b_i]_0\}_{i\in[n]}$ to $\mathcal{S}$ and $\{[b_i]_1\}_{i\in[n]}$ to $\mathcal{R}$, where $b_i := 1\{x_i \in Y\}$ denotes the membership bit. Unlike multi-query (reverse) PMT, ssPMT produces the membership bits in a secret-shared form, so that neither party learns b_i in the clear. The work [32] proposes an ssPMT construction based on generic GMW-style secure computation for SKE decryption and comparison circuits, which incurs substantial computational and communication overhead, as confirmed by our experimental evaluation. To address this inefficiency, in Section 2.2, we introduce two efficient ssPMT constructions.

In the second phase, the parties take as input the secret-shared membership bits produced by ssPMT, while the sender additionally inputs the set X. They execute the customized ssOTd protocol, which enables the receiver to obtain exactly those elements in X that are not contained in its set Y. Below, we describe our ssOTd protocol in detail.

Functionality and Construction of ssOTd. We first introduce our ssOTd functionality. The sender $\mathcal{S}$ inputs a *single* message x, and both parties input secret shares of the choice bit b. Then, the functionality returns x to the receiver $\mathcal{R}$ if $b = 0$, otherwise, a public default output $\perp$.

A naive solution proceeds as follows using the random OT (ROT) correlation. Assume that the two parties have an ROT instance, where $\mathcal{S}$ holds (m_0, m_1) and $\mathcal{R}$ holds $[b]_1$ together with $m_{[b]_1}$. First, $\mathcal{S}$ sends two masked messages to $\mathcal{R}$: if $[b]_0 = 0$, it sends $t_0 := x \oplus m_0$ and $t_1 := \perp \oplus m_1$; otherwise, it sends $t_0 := \perp \oplus m_0$ and $t_1 := x \oplus m_1$. Then, $\mathcal{R}$ computes the output as $t_{[b]_1} \oplus m_{[b]_1}$. This protocol can be viewed as a simplified variant of the secret-shared OT protocol [37], which additionally supports that the outputs remain secret-shared between the two parties. In the ROT-hybrid model, the communication overhead of this protocol is 2ℓ bits, where ℓ denotes the message length in bits.

We propose a customized and more efficient protocol with only $(\ell+1)$ bits of communication. The key observation is that the choice bit is inevitably revealed to $\mathcal{R}$ upon output, since the default value is public. Consequently, in our protocol, $\mathcal{S}$ sends only a single masked message $t := x \oplus m_{[b]_0}$ together with its share of the choice bit $[b]_0$. $\mathcal{R}$ outputs $t \oplus m_{[b]_1}$ if $[b]_0 = [b]_1$, and outputs $\perp$ otherwise. As a result, our protocol achieves approximately a $2\times$ reduction in communication cost compared to the naive protocol.

2.2 Efficient Constructions of ssPMT

We present two efficient constructions of ssPMT (ssPMT-fast and ssPMT-low), which offer a tradeoff between computation and communication overhead. ssPMT-fast is optimized for

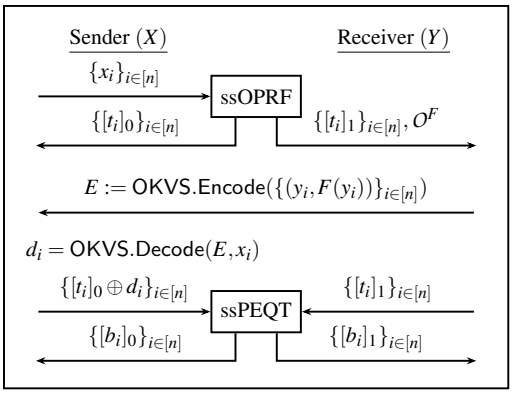


Figure 4: Construction of ssPMT-fast from secret-shared OPRF.

computation efficiency, while ssPMT-low is optimized for low communication overhead.

ssPMT-fast from Secret-shared OPRF. To realize ssPMT, our primary objective is to determine whether $x_i \in Y$ for each $x_i \in X$, while maintaining the membership information private to both parties. Recall that in the standard PMT protocol, to determine whether $x_i \in Y$, the parties engage in an OPRF protocol[1], where the receiver $\mathcal{R}$ obtains the PRF function and the sender $\mathcal{S}$ obtains $\mathsf{PRF}(x_i)$. Then, $\mathcal{R}$ sends to $\mathcal{S}$ the Y's PRF evaluation $\{\mathsf{PRF}(y_i)\}_{y_i\in Y}$, denoted as $\mathsf{PRF}(Y)$. Finally, $\mathcal{S}$ obtains the output by locally determining whether $\mathsf{PRF}(x_i) \in \mathsf{PRF}(Y)$. However, this standard solution directly leaks the membership information to $\mathcal{S}$.

To address this issue, our main idea is to enable $\mathcal{S}$ to determine $x_i \in Y$ in the secret-shared manner. Specifically, instead of outputting the PRF evaluation $\mathsf{PRF}(x_i)$ to $\mathcal{S}$, we employ secret-shared OPRF [1, 46], which returns its secret shares $[\mathsf{PRF}(x_i)]_0, [\mathsf{PRF}(x_i)]_1$ to $\mathcal{S}$ and $\mathcal{R}$, respectively. This raises another problem: how to determine whether $\mathsf{PRF}(x_i) \in \mathsf{PRF}(Y)$ given that $\mathsf{PRF}(x_i)$ is secret-shared between the two parties and $\mathsf{PRF}(Y)$ is held by $\mathcal{R}$. A naive solution requires $|Y|$ invocations to the secret-shared private equality test (ssPEQT) protocol to compare $\mathsf{PRF}(x_i)$ with every item in $\mathsf{PRF}(Y)$, which will incur prohibitively large overhead. We solve this problem by letting $\mathcal{R}$ encode the pairs $(Y, \mathsf{PRF}(Y))$ into an oblivious key-value store (OKVS) [18] and transmit this encoding to $\mathcal{S}$. The properties of OKVS ensure that $\mathcal{S}$ will decode to $\mathsf{PRF}(x_i)$ if $x_i \in Y$ and a random value otherwise, while the encoding hides the information of Y given that $\mathsf{PRF}(Y)$ are pseudo-random. Therefore, for each $x_i \in X$, if $x_i \in Y$ holds, the value decoded from the OKVS encoding exactly matches the secret-shared PRF evaluation $\mathsf{PRF}(x_i)$. We require only one invocation of ssPEQT for each x_i, and hence ssPMT-fast achieves linear complexity. The high-level idea of ssPMT-fast is shown in Figure 4.

ssPMT-low from Bit Permutation. Similar to our first construction, the starting point of ssPMT-low is to transform the problem of determining whether $x_i \in Y$ into a single ssPEQT. To this end, we utilize hashing-to-bins techniques that are widely used in general private set operations [9, 23, 35, 36, 44]. Specifically, the sender $\mathcal{S}$ uses three hash functions h_1, h_2, h_3 to assign $x_1, \ldots, x_n$ into a hash table T_x with $m = (1+\varepsilon)n$ bins via Cuckoo hashing [33], ensuring that each bin accommodates at most one element, where ε is a small constant. At the same time, the receiver $\mathcal{R}$ assigns each $y_i \in Y$ to all bins of a size-m hash table T_y determined by $h_1(y_i), h_2(y_i), h_3(y_i)$, where each bin of T_y may include multiple elements. Due to the same hash functions, it ensures that for any $x_i \in X$ mapped to the b-th bin of T_x, if there is a $y_j \in Y$ with $y_j = x_i$, then y_j must lie in the b-th bin of T_y. Then, both parties perform OPPRF [27] for each bin $b \in [m]$: $\mathcal{R}$, as the OPPRF sender, programs all items in $T_y[b]$ to the same random value r_b; $\mathcal{S}$, as the OPPRF receiver, inputs $T_x[b]$ and learns the PRF evaluation r'_b. It holds that the sender's r'_b would be equal to r_b when $T_x[b] \in T_y[b]$.

To obtain secret-shared membership information, a straightforward solution is that both parties invoke ssPEQT with inputs r'_b and r_b, respectively, and learn the secret-shared membership indication bit. Looking ahead, in the PSU protocol, after obtaining the union, the receiver implicitly learns the result of $1\{r'_b = r_b\}$. However, this will leak additional information to the receiver about the intersection [23, 44]. For example, assuming the receiver does not learn the item in the b-th bin of T_x, then the receiver knows the items in $T_y[b]$ containing an intersected item with non-negligible probability. To address this issue, previous enhanced PSU works [23, 44] employ different techniques to randomly permute the pairs $\{(r_i, r'_i)\}_{i\in[m]}$ before ssPEQT while maintaining their equality relations. In particular, Jia et al. [23] employ a general-purpose permute+share protocol [10], which incurs substantial communication overhead, while Tu et al. [44] design a customized permutation based on computation-intensive DDH assumptions, resulting in high computational cost.

In contrast to prior works, we apply the random permutation *after* executing ssPEQT. Our key observation is that secret-shared permutation protocols inherently incur costs proportional to the input length. After ssPEQT, however, the data to be permuted consists of a single bit per item, which substantially reduces the overhead. Motivated by this observation, we design a specialized secret-shared bit permutation protocol (bitPerm), constructed from classical permutation networks [4, 47] and lightweight OT. Similar to prior secret-shared permutation protocols, our bitPerm incurs $O(n \log n)$ communication and computation complexity. Nevertheless, as demonstrated in our evaluation, bitPerm contributes only a small fraction of the total overhead of the enhanced PSU protocol due to its small hidden constant. Besides, Cuckoo hashing reorders the items, which can be modeled by a permutation π. An additional benefit of bitPerm is that it can restore the original ordering by setting the input permutation as π.

[1] We reverse the sender and receiver roles for aligning with our protocol.

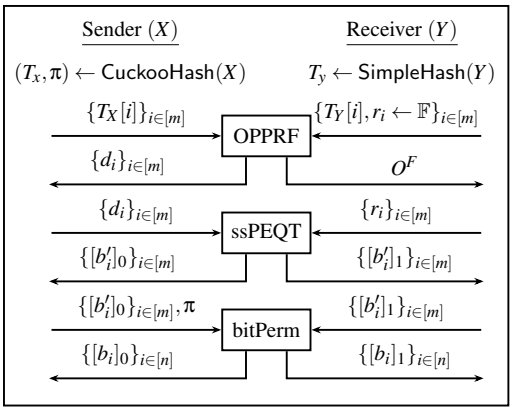


Figure 5: Construction of ssPMT-low from bit permutation. In Cuckoo hashing, $m := (1+\varepsilon)n$ denotes the number of bins, and π is an injective function $\pi : [n] \to [m]$ such that x_i is mapped to the $\pi(i)$-th bin of T_x.

We refer the readers to Section 5.2 for a detailed explanation. The high-level idea of ssPMT-low is illustrated in Figure 5.

2.3 Computation-efficient mqRPMT

As a byproduct, we obtain a computation-efficient multi-query reverse PMT (mqRPMT) protocol that relies on lightweight OT and OPRF, rather than public-key primitives (specifically, the DDH assumption) used in the state of the art [11]. From a theoretical perspective, our work shifts the construction of mqRPMT from computation-intensive public-key operations to lightweight symmetric-key primitives. Our mqRPMT protocol can be straightforwardly derived from ssPMT by revealing the secret-shared membership results to the receiver. We defer the detailed protocol description and optimizations to Section 6. mqRPMT is not only applicable to standard PSU, but also serves as a core component in other private set operations, such as private set intersection cardinality (PSI-card) [11, 12] and private set intersection cardinality and sum (PSI-card-sum) [17, 22].

3 Preliminaries

Notations. We denote by κ and λ the computational and statistical security parameters, respectively. The expressions $X \approx_c Y$ and $X \approx_s Y$ indicate that the distributions of X and Y are computationally and statistically indistinguishable, respectively. For a positive integer n, let $[n] := \{1, \ldots, n\}$. The notation $s \leftarrow S$ means that s is sampled uniformly at random from the set S. We use $[x]_0$ and $[x]_1$ to denote the secret shares of x held by the sender and the receiver, respectively. The indicator function $1\{\mathsf{event}\}$ equals 1 if the event occurs and 0 otherwise. Finally, O^F denotes oracle access to function F.

Functionality $\mathcal{F}_{\mathsf{ePSU}}$

Parameters: Sender $\mathcal{S}$ and receiver $\mathcal{R}$. Set size n.
Protocol:
1. Wait for input $X = \{x_i\}_{i\in[n]} \in \mathbb{U}^n$ from $\mathcal{S}$ and $Y = \{y_i\}_{i\in[n]} \in \mathbb{U}^n$ from $\mathcal{R}$.
2. Upon receiving (response) from the adversary, output (finished) to $\mathcal{S}$ and $Z := X \cup Y$ to $\mathcal{R}$.

Figure 6: Functionality of enhanced PSU. The highlighted region distinguishes the functionality $\mathcal{F}_{\mathsf{ePSU}}$ from standard PSU, and removing it degrades $\mathcal{F}_{\mathsf{ePSU}}$ to standard PSU.

3.1 Threat Model

We use the universal composability (UC) framework [8] to prove security in the presence of a semi-honest, static adversary. We say that a protocol Π *UC-realizes* an ideal functionality $\mathcal{F}$ if for any *real-world* adversary $\mathcal{A}$, there exists an *ideal-world* adversary Sim such that for any environment $\mathcal{Z}$ with an arbitrary auxiliary input z, the output distribution of $\mathcal{Z}$ in the real-world execution interacting with $\mathcal{A}$ and parties running protocol Π is indistinguishable from the output distribution of $\mathcal{Z}$ in the ideal-world execution interacting with Sim and $\mathcal{F}$. We defer the formal definition to Appendix A.

3.2 Enhanced Private Set Union

To address the during-execution leakage in previous PSU protocols [11, 17, 25, 28, 50], Jia et al. [23] introduce a new functionality of enhanced PSU. We recall the functionality in Figure 6, where we also include the standard PSU functionality for comparison. The main difference from the standard PSU functionality is that enhanced PSU additionally sends an output (finished) to the sender $\mathcal{S}$ once the computation is completed.

We explain that the enhanced PSU functionality $\mathcal{F}_{\mathsf{ePSU}}$ captures the security of during-execution leakage, namely, the receiver cannot obtain any information before obtaining the output $X \cup Y$. Generally, we assume that a real-world adversary corrupting the receiver receives some non-trivial leakage message $\mathsf{Leak}(X \cup Y)$ (e.g., the intersection cardinality $|X \cap Y|$ in most existing constructions [11, 25, 28, 50]), before obtaining the protocol output $X \cup Y$. To prove the protocol's security, the simulator in the ideal world has only two strategies: (1) send Y simulating the receiver to $\mathcal{F}_{\mathsf{ePSU}}$ immediately, and (2) not send Y to $\mathcal{F}_{\mathsf{ePSU}}$ immediately.

Following the first strategy, the simulator can receive $X \cup Y$ from $\mathcal{F}_{\mathsf{ePSU}}$ and simulate the message $\mathsf{Leak}(X \cup Y)$. In the ideal world, the sender will forward (finished) received from $\mathcal{F}_{\mathsf{ePSU}}$ to the environment; however, in the real world, the sender will not output (finished) to the environment, since the execution has not been completed. Therefore, the environment can distinguish the two worlds. On the other hand, following

Functionality $\mathcal{F}_{\mathsf{bitPerm}}$

Parameters: Sender $\mathcal{S}$ and receiver $\mathcal{R}$. Input size m and output size $n \leq m$.

Protocol:

1. Wait for input an injective function $\pi : [n] \to [m]$ and $\{[b_i]_0\}_{i\in[m]} \in \mathbb{F}_2^m$ from $\mathcal{S}$ and $\{[b_i]_1\}_{i\in[m]} \in \mathbb{F}_2^m$ from $\mathcal{R}$.
2. Reconstruct $b_i := [b_i]_0 \oplus [b_i]_1 \in \mathbb{F}_2$ for $i \in [m]$, and compute $\{z_i\}_{i\in[n]}$ such that $z_i := b_{\pi(i)}$ for $i \in [n]$.
3. Sample $\{[z_i]_0\}_{i\in[n]}, \{[z_i]_1\}_{i\in[n]} \leftarrow \mathbb{F}_2^n$ such that $[z_i]_0 \oplus [z_i]_1 = z_i$.
4. Output $\{[z_i]_0\}_{i\in[n]}$ to $\mathcal{S}$ and $\{[z_i]_1\}_{i\in[n]}$ to $\mathcal{R}$.

Figure 7: Functionality of secret-shared bit permutation.

the second strategy, the simulator does not know $X \cup Y$ and hence cannot guess the correct $\mathsf{Leak}(X \cup Y)$. Therefore, the environment can still distinguish the two worlds.

3.3 Secret-shared Bit Permutation

Secret-shared permutation enables two parties to privately permute a vector in a secret-shared manner. Specifically, the sender $\mathcal{S}$ and the receiver $\mathcal{R}$ hold secret shares of a vector $\{b_i\}_{i\in[n]}$, and $\mathcal{S}$ additionally inputs a permutation π over $[n]$. After execution, the parties obtain secret shares of a vector $\{z_i\}_{i\in[n]}$ such that $z_i := b_{\pi(i)}$.

In Figure 7, we introduce a customized functionality of secret-shared *bit* permutation, denoted as bitPerm. This functionality has two key features: (1) the input consists of bit vectors, and (2) the permutation π is generalized to an injective function, allowing the input vector to be longer than the output vector. As shown in [17], the latter can be implemented by first permuting the input vector so that the desired elements appear at the front, and then truncating the output vector accordingly for both parties.

Following recent network-based permutation works [10, 17, 20], we present an efficient protocol of bitPerm in Figure 8, constructed from oblivious transfer (OT) and permutation networks such as the Beneš network [4] and the Waksman network [47]. The high-level idea is to permute a random vector sampled by the receiver and use the resulting random correlation as a mask to permute the actual input vector. Compared to prior protocols, ours introduces two differences. First, in Step 1, we program a network for an input injective function, rather than a standard permutation as seen in prior works. This is required in ssPMT-low to restore the expanded and reordered set back to its original sequence. Second, our protocol operates exclusively on bit vectors throughout the entire execution, specializing the arbitrary-length inputs supported by previous works. This focus is crucial for efficiency, as the cost of permutation protocols scales proportionally with input length. Due to the similarity of the constructions, we omit the security proof and refer to [10, 17, 20] for detailed security analysis.

Protocol Π_{bitPerm}

Parameters: Sender $\mathcal{S}$ and receiver $\mathcal{R}$. Input size m and output size $n \leq m$. The number w of switch gates.

Input: $\mathcal{S}$ inputs $\{[b_i]_0\}_{i\in[m]} \in \mathbb{F}_2^m$ and an injective function $\pi : [n] \to [m]$, and $\mathcal{R}$ inputs $\{[b_i]_1\}_{i\in[m]} \in \mathbb{F}_2^m$.

Protocol:

1. $\mathcal{S}$ programs a permutation network for the injective function π, in which the number of switch gates is w. $\mathcal{S}$ obtains $\{c_i\}_{i\in[w]} \in \mathbb{F}_2^w$, where each c_i denotes the control signal of the i-th switch gate in a topological order.
2. $\mathcal{S}$ and $\mathcal{R}$ invoke functionality $\mathcal{F}_{\mathsf{ROT}}$, where $\mathcal{R}$ as the sender learns $\{r_{i,0}, r_{i,1}\}_{i\in[w]} \in (\mathbb{F}_2 \times \mathbb{F}_2)^w$, and $\mathcal{S}$ as the receiver inputs $\{c_i\}_{i\in[w]} \in \mathbb{F}_2^w$ and learns $\{r_{i,c_i}\}_{i\in[w]} \in \mathbb{F}_2^w$.
3. For $i \in [m]$, $\mathcal{R}$ samples $s_i \leftarrow \mathbb{F}_2$ and sets it as the i-th input of the network.
4. For i-th switch gate in a topological order, let α_i, β_i denote the values of two input wires and u_i, v_i denote the values of two output wires. The parties execute as follows.
   (a) $\mathcal{R}$ sets $[u_i]_1 := [\alpha_i]_1 \oplus r_{i,0} \in \mathbb{F}_2$ and $[v_i]_1 := [\alpha_i]_1 \oplus [\beta_i]_1 \oplus [u_i]_1 \in \mathbb{F}_2$.
   (b) $\mathcal{R}$ computes $q_i := ([\alpha_i]_1 \oplus [\beta_i]_1) \oplus r_{i,0} \oplus r_{i,1} \in \mathbb{F}_2$ and sends q_i to $\mathcal{S}$ in parallel with all gates.
   (c) $\mathcal{S}$ sets $[u_i]_0 := [\alpha_i]_0 \oplus (q_i \cdot c_i \oplus r_{i,c_i}) \oplus ([\alpha_i]_0 \oplus [\beta_i]_0) \cdot c_i \in \mathbb{F}_2$ and $[v_i]_0 := [\alpha_i]_0 \oplus [\beta_i]_0 \oplus [u_i]_0 \in \mathbb{F}_2$.
5. Let t_i (secret-shared between $\mathcal{S}$ and $\mathcal{R}$) be the i-th output of the network for $i \in [n]$, satisfying $t_i = s_{\pi(i)}$.
6. For $i \in [m]$, $\mathcal{R}$ computes $d_i := [b_i]_1 \oplus s_i \in \mathbb{F}_2$. $\mathcal{R}$ sends $\{d_i\}_{i\in[m]}$ to $\mathcal{S}$ in parallel with the previous communication.
7. For $i \in [n]$, $\mathcal{S}$ computes $g_i := d_{\pi(i)} \oplus [b_{\pi(i)}]_0 \in \mathbb{F}_2$ and sets $[z_i]_0 := g_i \oplus [t_i]_0 \in \mathbb{F}_2$, and $\mathcal{R}$ sets $[z_i]_1 := [t_i]_1 \in \mathbb{F}_2$.
8. $\mathcal{S}$ outputs $\{[z_i]_0\}_{i\in[n]}$ and $\mathcal{R}$ outputs $\{[z_i]_1\}_{i\in[n]}$.

Figure 8: Protocol of secret-shared bit permutation.

Given that m is the length of the input vector and w denotes the number of switch gates of the permutation network for length-m input vectors with $w = O(m \log m)$, the communication cost of the protocol is $w + m$ bits in a single round, except for the communication-efficient ROT. Therefore, the complexity of bitPerm is $O(m \log m)$. It is worth noting that the recent work [34] presents a linear-complexity secret-shared permutation protocol based on secret-shared OPRF. However, its concrete communication cost includes a factor of κ, which makes it unsuitable for our bit inputs.

3.4 Other Building Blocks

Random Oblivious Transfer. Random oblivious transfer (ROT) [3] is a fundamental primitive in secure computation, where the sender provides no input and receives two random messages m_0 and m_1, while the receiver inputs a choice bit c and receives the corresponding message m_c. The functionality $\mathcal{F}_{\mathsf{ROT}}$ is given in Figure 9. Recent silent OT extension tech-

Functionality $\mathcal{F}_{\mathsf{ROT}}$

Parameters: Sender $\mathcal{S}$ and receiver $\mathcal{R}$. Input size n and message bit-length ℓ.

Protocol:

1. Wait for input $\{c_i\}_{i\in[n]} \in \mathbb{F}_2^n$ from $\mathcal{R}$.
2. Sample $\{m_{i,0}, m_{i,1}\}_{i\in[n]} \leftarrow (\mathbb{F}_{2^\ell} \times \mathbb{F}_{2^\ell})^n$.
3. Output $\{m_{i,0}, m_{i,1}\}_{i\in[n]}$ to $\mathcal{S}$ and $\{m_{i,c_i}\}_{i\in[n]}$ to $\mathcal{R}$.

Figure 9: Functionality of random oblivious transfer.

Functionality $\mathcal{F}_{(\mathsf{ss})\mathsf{PEQT}}$

Parameters: Sender $\mathcal{S}$ and receiver $\mathcal{R}$. Input size n and bit-length ℓ.

Protocol:

1. Wait for input $\{x_i\}_{i\in[n]} \in \mathbb{F}_{2^\ell}^n$ from $\mathcal{S}$ and $\{y_i\}_{i\in[n]} \in \mathbb{F}_{2^\ell}^n$ from $\mathcal{R}$.
2. For $i \in [n]$, compute $b_i := 1\{x_i = y_i\} \in \mathbb{F}_2$.
3. PEQT: Output $\{b_i\}_{i\in[n]}$ to $\mathcal{R}$.
4. ssPEQT: For $i \in [n]$, sample $[b_i]_0, [b_i]_1 \leftarrow \mathbb{F}_2$ such that $[b_i]_0 \oplus [b_i]_1 = b_i$. Output $\{[b_i]_0\}_{i\in[n]}$ to $\mathcal{S}$ and $\{[b_i]_1\}_{i\in[n]}$ to $\mathcal{R}$.

Figure 10: Functionality of (secret-shared) private equality test.

niques [7,40,48] enable the generation of n random choice-bit ROT relations with compelling computational overhead and sublinear communication to n.

(Secret-shared) Private Equality Test. Private equality test (PEQT) enables two parties, each holding a private input (denoted by x and y), to determine whether $x = y$ to the receiver, without revealing any additional information about their inputs. We also consider a secret-shared variant, denoted by ssPEQT, in which the equality result is secret-shared between the two parties. The corresponding ideal functionalities are depicted in Figure 10. PEQT can be efficiently instantiated using OPRF [39], while ssPEQT can be realized from OT [43].

Secret-Shared Oblivious Pseudorandom Function. Oblivious pseudorandom function (OPRF) enables two parties to jointly evaluate a pseudorandom function while preserving input privacy. In a standard multi-query OPRF, the sender $\mathcal{S}$ holds no input and implicitly obtains a random function F, while the receiver $\mathcal{R}$ inputs a set of queries $\{q_i\}_{i\in[n]}$ and obtains the corresponding evaluations $\{F(q_i)\}_{i\in[n]}$. We employ a secret-shared variant of OPRF [1,46], denoted by ssOPRF. In ssOPRF, the function evaluations are not revealed in the clear to the receiver. Instead, the two parties obtain additive secret shares of each value $F(q_i)$. As a result, neither party learns any information about the PRF outputs beyond their respective shares. The ideal functionality $\mathcal{F}_{\mathsf{ssOPRF}}$ is depicted in Figure 11. The recent work [1] presents efficient constructions of ssOPRF instantiated from MPC-friendly alternating-moduli PRF, which was first proposed by Boneh et al. [6] and later optimized in [1,2,13].

Functionality $\mathcal{F}_{\mathsf{ssOPRF}}$

Parameters: Sender $\mathcal{S}$ and receiver $\mathcal{R}$. Input size n and output bit-length ℓ.

Protocol:

1. Wait for input $\{q_i\}_{i\in[n]} \in \mathbb{F}^n$ from $\mathcal{R}$.
2. Sample a random function $F : \mathbb{F} \to \mathbb{F}_{2^\ell}$. For $i \in [n]$, compute $z_i := F(q_i) \in \mathbb{F}_{2^\ell}$ and sample $[z_i]_0, [z_i]_1 \leftarrow \mathbb{F}_{2^\ell}$ such that $[z_i]_0 \oplus [z_i]_1 = z_i$.
3. Output $(\{[z_i]_0\}_{i\in[n]}, O^F)$ to $\mathcal{S}$ and $\{[z_i]_1\}_{i\in[n]}$ to $\mathcal{R}$.

Figure 11: Functionality of secret-shared multi-query oblivious PRF.

Functionality $\mathcal{F}_{\mathsf{OPPRF}}$

Parameters: Sender $\mathcal{S}$ and receiver $\mathcal{R}$. Input size m,n and output bit-length ℓ.

Protocol:

1. Wait for input $L = \{(x_j, v_j)\}_{j\in[m]} \in (\mathbb{F} \times \mathbb{F}_{2^\ell})^m$ from $\mathcal{S}$ and $\{q_i\}_{i\in[n]} \in \mathbb{F}^n$ from $\mathcal{R}$.
2. Sample a random function $F : \mathbb{F} \to \mathbb{F}_{2^\ell}$ such that $F(x) = v$ for $(x, v) \in L$. For $i \in [n]$, compute $z_i := F(q_i) \in \mathbb{F}_{2^\ell}$.
3. Output O^F to $\mathcal{S}$ and $\{z_i\}_{i\in[n]}$ to $\mathcal{R}$.

Figure 12: Functionality of multi-query oblivious programmable PRF.

Oblivious Programmable Pseudorandom Function. Oblivious programmable pseudorandom function (OPPRF), introduced by Kolesnikov et al. [27], extends the OPRF functionality by allowing the sender $\mathcal{S}$ to program the pseudorandom function F at a designated set of inputs. Specifically, for each sender-chosen pair (x_j, v_j), the functionality enforces that $F(x_j) = v_j$, while for all non-programmed inputs, the function outputs are distributed uniformly at random. The ideal functionality of OPPRF is shown in Figure 12. State-of-the-art OPPRF constructions [39,43] are realized by combining OPRF with OKVS.

Oblivious Key-Value Store. An oblivious key-value store (OKVS) [18] is a data structure defined by two algorithms. The encoding algorithm Encode takes as input a set of key-value pairs and outputs an encoded data structure, while the decoding algorithm Decode takes as input a key and the encoded data structure and returns the corresponding value. The obliviousness property guarantees that, when the OKVS encodes random values, the resulting data structure is statistically independent of the encoded key set. Recent works [5, 18, 39] propose efficient OKVS constructions with linear-time encoding and compact encoding expansion rate. We defer the definition and properties of OKVS to Appendix B.

Hashing to Bins. The hashing-to-bins technique was intro-

Functionality $\mathcal{F}_{\mathsf{ssPMT}}$

Parameters: Sender $\mathcal{S}$ and receiver $\mathcal{R}$. Set size n.

Protocol:

1. Wait for input $X = \{x_i\}_{i\in[n]} \in \mathbb{U}^n$ from $\mathcal{S}$ and $Y = \{y_i\}_{i\in[n]} \in \mathbb{U}^n$ from $\mathcal{R}$.
2. For $i \in [n]$, compute $b_i := 1\{x_i \in Y\} \in \mathbb{F}_2$ and sample $[b_i]_0, [b_i]_1 \leftarrow \mathbb{F}_2$ such that $[b_i]_0 \oplus [b_i]_1 = b_i$.
3. Output $\{[b_i]_0\}_{i\in[n]}$ to $\mathcal{S}$ and $\{[b_i]_1\}_{i\in[n]}$ to $\mathcal{R}$.

Figure 13: Functionality of multi-query secret-shared private membership test.

duced by Pinkas et al. [35] and has since become a standard tool in private set operations. At a high level, the sender $\mathcal{S}$ uses three hash functions $h_1, h_2, h_3 : \{0,1\}^* \to [m]$ to map its items $X = \{x_i\}_{i\in[n]}$ into a hash table T_x with $m := (1+\varepsilon)n$ bins via Cuckoo hashing [33], where ε is a small constant. The Cuckoo hashing procedure ensures that each bin contains at most one item, and for every $x_i \in X$, there exists an index $\tau \in [3]$ such that $T_x[h_\tau(x_i)] = x_i \| \tau$. Following the recent works [23, 44], we choose parameters such that the failure probability of Cuckoo hashing is negligible. On the other hand, the receiver $\mathcal{R}$ uses the same hash functions h_1, h_2, h_3 to map its items $Y = \{y_i\}_{i\in[n]}$ into a hash table T_y with m bins using simple hashing. Specifically, for each $y_i \in Y$, the receiver inserts the tagged items $(y_i\|1), (y_i\|2), (y_i\|3)$ into the bins $T_y[h_1(y_i)], T_y[h_2(y_i)], T_y[h_3(y_i)]$, respectively. As a result, if there exists an item $y_j \in Y$ such that $x_i = y_j$ for some $x_i \in X$, then there exists an index $\tau \in [3]$ for which $T_x[h_\tau(x_i)] \in T_y[h_\tau(y_j)]$.

4 Modular Framework for Enhanced PSU

We first introduce two key building blocks and then present our general framework for enhanced PSU from them.

4.1 Building Blocks

Secret-shared Private Membership Test [32]. Figure 13 reviews the functionality of secret-shared private membership test (ssPMT) proposed in the work [32]. In Section 5, we present two efficient constructions of ssPMT. This functionality enables the two parties to obtain secret-shared membership information, which is essential for preventing during-execution leakage and, consequently, for realizing enhanced PSU. To achieve the enhanced security in PSU, the ssPMT functionality itself must also be secure against during-execution leakage. This requirement is naturally satisfied, as the functionality outputs only secret-shared values that do not reveal any additional information to either party.

Secret-shared Oblivious Transfer with Default. In Figure 14, we present a new functionality of secret-shared OT

Functionality $\mathcal{F}_{\mathsf{ssOTd}}$

Parameters: Sender $\mathcal{S}$ and receiver $\mathcal{R}$. Input size n, bit-length ℓ, and public default output $\perp$.

Protocol:

1. Wait for input $\{x_i\}_{i\in[n]} \in \mathbb{F}_{2^\ell}^n$ and $\{[b_i]_0\}_{i\in[n]} \in \mathbb{F}_2^n$ from $\mathcal{S}$ and $\{[b_i]_1\}_{i\in[n]} \in \mathbb{F}_2^n$ from $\mathcal{R}$.
2. For $i \in [n]$, compute $z_i := x_i \in \mathbb{F}_{2^\ell}$ if $[b_i]_0 \oplus [b_i]_1 = 0$ and $z_i := \perp$ otherwise.
3. Upon receiving (response) from the adversary, output (finished) to $\mathcal{S}$ and $\{z_i\}_{i\in[n]}$ to $\mathcal{R}$.

Figure 14: Functionality of secret-shared oblivious transfer with default.

Protocol Π_{ssOTd}

Parameters: Sender $\mathcal{S}$ and receiver $\mathcal{R}$. Input size n, bit-length ℓ, and public default output $\perp$.

Input: $\mathcal{S}$ inputs $\{x_i\}_{i\in[n]} \in \mathbb{F}_{2^\ell}^n$ and $\{[b_i]_0\}_{i\in[n]} \in \mathbb{F}_2^n$, and $\mathcal{R}$ inputs $\{[b_i]_1\}_{i\in[n]} \in \mathbb{F}_2^n$.

Protocol:

1. $\mathcal{S}$ and $\mathcal{R}$ invoke functionality $\mathcal{F}_{\mathsf{ROT}}$, where $\mathcal{S}$ learns $\{m_{i,0}, m_{i,1}\}_{i\in[n]} \in (\mathbb{F}_{2^\ell} \times \mathbb{F}_{2^\ell})^n$, and $\mathcal{R}$ inputs $\{[b_i]_1\}_{i\in[n]} \in \mathbb{F}_2^n$ and learns $\{m_{i,[b_i]_1}\}_{i\in[n]} \in \mathbb{F}_{2^\ell}^n$.
2. For $i \in [n]$, $\mathcal{S}$ computes $t_i := x_i \oplus m_{i,[b_i]_0} \in \mathbb{F}_{2^\ell}$. $\mathcal{S}$ sends $\{t_i, [b_i]_0\}_{i\in[n]}$ to $\mathcal{R}$.
3. For $i \in [n]$, $\mathcal{R}$ computes $b_i := [b_i]_0 \oplus [b_i]_1 \in \mathbb{F}_2$, and sets $z_i := t_i \oplus m_{i,[b_i]_1} \in \mathbb{F}_{2^\ell}$ if $b_i = 0$ and $z_i := \perp$ otherwise.
4. $\mathcal{S}$ outputs (finished) and $\mathcal{R}$ outputs $\{z_i\}_{i\in[n]}$.

Figure 15: Protocol of secret-shared oblivious transfer with default.

with default (ssOTd). The protocol is presented in Figure 15. We note that ssOTd is required to provide security against during-execution leakage. Accordingly, similar to the enhanced PSU functionality, ssOTd outputs (finished) to the sender, which ensures that the receiver cannot obtain any information before the protocol completes. The security is provided in Theorem 1, and we defer the proof to Appendix E.1.

Theorem 1. *The protocol Π_{ssOTd} in Figure 15 UC-realizes the functionality $\mathcal{F}_{\mathsf{ssOTd}}$ in Figure 14 in $\mathcal{F}_{\mathsf{ROT}}$-hybrid model against static, semi-honest adversaries.*

4.2 Modular Enhanced PSU Framework

With the above two building blocks, we provide a general framework for constructing enhanced PSU in Figure 16. Our protocol is lean and modular, which benefits the security analysis and component optimizations. To prevent during-execution leakage of the enhanced PSU protocol, we require that every functionality invoked by the protocol be secure against such leakage. As shown in Section 4.1, the ssPMT

Protocol Π_{ePSU}

Parameters: Sender $\mathcal{S}$ and receiver $\mathcal{R}$. Set size n.
Input: $\mathcal{S}$ inputs $X = \{x_i\}_{i\in[n]} \in \mathbb{U}^n$ and $\mathcal{R}$ inputs $Y = \{y_i\}_{i\in[n]} \in \mathbb{U}^n$.
Protocol:

1. $\mathcal{S}$ randomly permutes the input set X.
2. $\mathcal{S}$ and $\mathcal{R}$ invoke functionality $\mathcal{F}_{\mathsf{ssPMT}}$, where $\mathcal{S}$ inputs X and $\mathcal{R}$ inputs Y. $\mathcal{S}$ learns $\{[b_i]_0\}_{i\in[n]} \in \mathbb{F}_2^n$, and $\mathcal{R}$ learns $\{[b_i]_1\}_{i\in[n]} \in \mathbb{F}_2^n$, where $b_i := 1\{x_i \in Y\} \in \mathbb{F}_2$.
3. $\mathcal{S}$ and $\mathcal{R}$ invoke functionality $\mathcal{F}_{\mathsf{ssOTd}}$, where $\mathcal{S}$ inputs $(\{x_i, [b_i]_0\}_{i\in[n]})$, and $\mathcal{R}$ inputs $\{[b_i]_1\}_{i\in[n]}$. $\mathcal{S}$ learns (finished) and $\mathcal{R}$ learns $\{z_i\}_{i\in[n]} \in \mathbb{U}^n$, where $z_i := x_i$ if $b_i = 0$ and $z_i := \perp$ otherwise.
4. $\mathcal{S}$ outputs (finished) and $\mathcal{R}$ outputs $Z := Y \cup \{z_i \mid z_i \neq \perp \text{ for } i \in [n]\}$.

Figure 16: Protocol of enhanced PSU.

functionality outputs only secret shares of the membership information and therefore inherently prevents during-execution leakage; the ssOTd functionality is explicitly designed to address the during-execution leakage.

We emphasize that the random permutation in the first step is required to prove the protocol's security. If we omit this permutation step, in the real protocol, the environment and the real-world adversary $\mathcal{A}$ that corrupts the receiver will know the positions of $\perp$ and $X \setminus Y$ in the output $\{z_i\}_{i\in[n]}$. However, the ideal-world adversary Sim does not have the information of X, therefore, it can not correctly simulate the correct ordering of $\{z_i\}_{i\in[n]}$ in the ideal world. We give the security in Theorem 2 and defer the proof to Appendix E.2.

Theorem 2. *The protocol Π_{ePSU} in Figure 16 UC-realizes the functionality $\mathcal{F}_{\mathsf{ePSU}}$ in Figure 6 in $(\mathcal{F}_{\mathsf{ssPMT}}, \mathcal{F}_{\mathsf{ssOTd}})$-hybrid model against static, semi-honest adversaries.*

5 Efficient Constructions of Secret-shared PMT

In this section, we present two efficient constructions of secret-shared PMT, which dominate the overall overhead of the enhanced PSU protocol. The two constructions offer a trade-off between computational and communication efficiency. Section 5.1 presents our computation-efficient construction, denoted ssPMT-fast, while Section 5.2 presents our communication-efficient construction, denoted ssPMT-low.

5.1 Construction from Shared-output OPRF

Our first construction ssPMT-fast is based on secret-shared OPRF. We describe the detailed protocol in Figure 17. The core idea is that for each sender item x_i, two parties compute secret shares of the PRF evaluation $t_i := F(x_i)$. Then, the receiver computes and sends the encoding $E :=$ $\mathsf{OKVS.Encode}(\{(y, F(y))\}_{y\in Y})$ for receiver set Y. Finally, the sender computes $d_i := \mathsf{OKVS.Decode}(E, x_i)$ and both parties check whether $t_i = d_i$. This check is equivalent to checking $x_i \in Y$, because OKVS ensures that if $x_i \in Y$, $d_i = F(x_i) = t_i$; otherwise, d_i is random and thus $t_i \neq d_i$ with overwhelming probability. Our protocol actually checks whether $[t_i]_0 \oplus d_i = [t_i]_1$ in the secret-shared manner, which is equivalent to testing whether $t_i = d_i$ since $t_i = [t_i]_0 \oplus [t_i]_1$.

Protocol $\Pi_{\mathsf{ssPMT\text{-}fast}}$

Parameters: Sender $\mathcal{S}$ and receiver $\mathcal{R}$. Set size n, OKVS encoding size m and bit-length $\ell := \lambda + \log n$.
Input: $\mathcal{S}$ inputs $X = \{x_i\}_{i\in[n]} \in \mathbb{U}^n$, and $\mathcal{R}$ inputs $Y = \{y_i\}_{i\in[n]} \in \mathbb{U}^n$.
Protocol:

1. $\mathcal{S}$ and $\mathcal{R}$ invoke functionality $\mathcal{F}_{\mathsf{ssOPRF}}$, where $\mathcal{S}$ as the receiver inputs X and receives $\{[t_i]_0\}_{i\in[n]} \in \mathbb{F}_{2^\ell}^n$, and $\mathcal{R}$ as the sender receives $\{[t_i]_1\}_{i\in[n]} \in \mathbb{F}_{2^\ell}^n$ and O^F, where $F : \mathbb{U} \to \mathbb{F}_{2^\ell}$.
2. $\mathcal{R}$ computes $f_i := F(y_i) \in \mathbb{F}_{2^\ell}$ for $i \in [n]$, and computes encoding $E := \mathsf{OKVS.Encode}(L) \in \mathbb{F}_{2^\ell}^m$, where $L := \{(y_i, f_i)\}_{i\in[n]} \in (\mathbb{U} \times \mathbb{F}_{2^\ell})^n$. $\mathcal{R}$ sends E to $\mathcal{S}$.
3. For $i \in [n]$, $\mathcal{S}$ computes $d_i := \mathsf{OKVS.Decode}(E, x_i) \in \mathbb{F}_{2^\ell}$ and sets $[t_i]_0 := [t_i]_0 \oplus d_i \in \mathbb{F}_{2^\ell}$.
4. $\mathcal{S}$ and $\mathcal{R}$ invoke functionality $\mathcal{F}_{\mathsf{ssPEQT}}$, where $\mathcal{S}$ inputs $\{[t_i]_0\}_{i\in[n]}$ and $\mathcal{R}$ inputs $\{[t_i]_1\}_{i\in[n]}$. $\mathcal{S}$ learns $\{[b_i]_0\}_{i\in[n]} \in \mathbb{F}_2^n$ and $\mathcal{R}$ learns $\{[b_i]_1\}_{i\in[n]} \in \mathbb{F}_2^n$.
5. $\mathcal{S}$ outputs $\{[b_i]_0\}_{i\in[n]}$ and $\mathcal{R}$ outputs $\{[b_i]_1\}_{i\in[n]}$.

Figure 17: Protocol of multi-query secret-shared private membership test from secret-shared OPRF.

The asymptotic computational and communication complexities of this construction are both $O(n)$, since the underlying secret-shared OPRF and secret-shared PEQT functionalities can be realized with linear complexity.

We state the protocol's security in the following and defer the proof to Appendix E.3

Theorem 3. *The protocol $\Pi_{\mathsf{ssPMT\text{-}fast}}$ in Figure 17 UC-realizes the functionality $\mathcal{F}_{\mathsf{ssPMT}}$ in Figure 13 in $(\mathcal{F}_{\mathsf{ssOPRF}}, \mathcal{F}_{\mathsf{ssPEQT}})$-hybrid model against static, semi-honest adversaries.*

5.2 Construction from Bit Permutation

In Figure 18, we present our second construction ssPMT-low from the customized bit permutation, OPPRF, and ssPEQT. Since the design intuition has already been presented in Section 2.2, we mainly focus on the permutation used in Step 6. As mentioned earlier, revealing the order of elements in the sender's Cuckoo hashing table would leak the sender's set privacy. Therefore, the sender must hide this order information. Instead of applying a random permutation, bitPerm just needs to take as input a customized injective function π in-

Protocol $\Pi_{\mathsf{ssPMT\text{-}low}}$

Parameters: Sender $\mathcal{S}$ and receiver $\mathcal{R}$. Set size n and bit-length $\ell := \lambda + \log m$. Cuckoo hashing parameters: hash functions h_1, h_2, h_3 and the number of bins m.
Input: $\mathcal{S}$ inputs $X = \{x_i\}_{i\in[n]} \in \mathbb{U}^n$, and $\mathcal{R}$ inputs $Y = \{y_i\}_{i\in[n]} \in \mathbb{U}^n$.
Protocol:

1. $\mathcal{S}$ maps X into a Cuckoo hash table T_x of m bins such that for any x_i, there exists $\tau \in [3]$ satisfying $T_x[h_\tau(x_i)] = x_i \| \tau$. $\mathcal{S}$ defines an injective function $\pi : [n] \to [m]$ such that x_i is mapped to the $\pi(i)$-th bin of T_x.
2. $\mathcal{R}$ maps Y into a simple hash table T_y of m bins, such that for any y_i and all $\tau \in [3]$, it holds that $y_i \| \tau \in T_y[h_\tau(y_i)]$.
3. $\mathcal{R}$ samples $\{r_i\}_{i\in[m]} \leftarrow \mathbb{F}_{2^\ell}^m$. For $i \in [m]$, $\mathcal{R}$ defines $L_i := \{(y\|\tau, r_i) | (y\|\tau) \in T_y[i]\}$, i.e., L_i is the list of pairs $(y\|\tau, r_i)$ for all entries $(y\|\tau)$ in $T_y[i]$.
4. $\mathcal{S}$ and $\mathcal{R}$ invoke functionality $\mathcal{F}_{\mathsf{OPPRF}}$, where $\mathcal{R}$ as the sender inputs $\{L_i\}_{i\in[m]}$ and receives O^F, while $\mathcal{S}$ as the receiver inputs T_x and learns $\{d_i\}_{i\in[m]} \in \mathbb{F}_{2^\ell}^m$.
5. $\mathcal{S}$ and $\mathcal{R}$ invoke functionality $\mathcal{F}_{\mathsf{ssPEQT}}$, where $\mathcal{S}$ inputs $\{d_i\}_{i\in[m]}$ and $\mathcal{R}$ inputs $\{r_i\}_{i\in[m]}$. $\mathcal{S}$ learns $\{[b'_i]_0\}_{i\in[m]} \in \mathbb{F}_2^m$ and $\mathcal{R}$ learns $\{[b'_i]_1\}_{i\in[m]} \in \mathbb{F}_2^m$.
6. $\mathcal{S}$ and $\mathcal{R}$ invoke functionality $\mathcal{F}_{\mathsf{bitPerm}}$, where $\mathcal{S}$ inputs $(\pi, \{[b'_i]_0\}_{i\in[m]})$ and $\mathcal{R}$ inputs $\{[b'_i]_1\}_{i\in[m]}$. $\mathcal{S}$ learns $\{[b_i]_0\}_{i\in[n]} \in \mathbb{F}_2^n$, and $\mathcal{R}$ learns $\{[b_i]_1\}_{i\in[n]} \in \mathbb{F}_2^n$, where $b_i = b'_{\pi(i)}$ for $i \in [n]$.
7. $\mathcal{S}$ outputs $(\{[b_i]_0\}_{i\in[n]})$ and $\mathcal{R}$ outputs $\{[b_i]_1\}_{i\in[n]}$.

Figure 18: Protocol of multi-query permuted secret-shared private membership test from bit permutation.

duced by Cuckoo hashing. This restores the Cuckoo table into the original order of the sender's set. Moreover, the injective function $\pi : [n] \to [m]$ is the sender's private input for the bitPerm functionality and remains private to the receiver when executing bitPerm. Therefore, this design avoids the leakage from Cuckoo hashing.

The asymptotic computational and communication complexities of this construction are both $O(n \log n)$. The underlying OPPRF and secret-shared PEQT functionalities can be realized with $O(n)$ complexity. However, the bit permutation protocol incurs $O(n \log n)$ overhead. Since these operations are over single bits, the hidden constant is small, and this step contributes to only a minor portion of the total overhead. This is further confirmed in our evaluation.

We state the protocol's security in the following and defer the proof to Appendix E.4.

Theorem 4. *The protocol $\Pi_{\mathsf{ssPMT\text{-}low}}$ in Figure 18 UC-realizes the functionality $\mathcal{F}_{\mathsf{ssPMT}}$ in Figure 13 in $(\mathcal{F}_{\mathsf{OPPRF}}, \mathcal{F}_{\mathsf{ssPEQT}}, \mathcal{F}_{\mathsf{bitPerm}})$-hybrid model against static, semi-honest adversaries.*

Functionality $\mathcal{F}_{\mathsf{mqRPMT}}$

Parameters: Sender $\mathcal{S}$ and receiver $\mathcal{R}$. Set size n.
Protocol:

1. Wait for input $X = \{x_i\}_{i\in[n]} \in \mathbb{U}^n$ from $\mathcal{S}$ and $Y = \{y_i\}_{i\in[n]} \in \mathbb{U}^n$ from $\mathcal{R}$.
2. For $i \in [n]$, compute $b_i := 1\{x_i \in Y\}$.
3. Output $b_1, \ldots, b_n$ to $\mathcal{R}$.

Figure 19: Functionality of multi-query reverse private membership test.

Protocol Π_{mqRPMT}

Parameters: Sender $\mathcal{S}$ and receiver $\mathcal{R}$. Set size n.
Input: $\mathcal{S}$ inputs $X = \{x_i\}_{i\in[n]} \in \mathbb{U}^n$, and $\mathcal{R}$ inputs $Y = \{y_i\}_{i\in[n]} \in \mathbb{U}^n$.
Protocol:

1. $\mathcal{S}$ and $\mathcal{R}$ invoke functionality $\mathcal{F}_{\mathsf{ssPMT}}$, where $\mathcal{S}$ inputs X and $\mathcal{R}$ inputs Y. $\mathcal{S}$ learns $\{[b_i]_0\}_{i\in[n]} \in \mathbb{F}_2^n$, and $\mathcal{R}$ learns $\{[b_i]_1\}_{i\in[n]} \in \mathbb{F}_2^n$.
2. $\mathcal{S}$ sends $\{[b_i]_0\}_{i\in[n]}$ to $\mathcal{R}$. For $i \in [n]$, $\mathcal{R}$ computes $b_i := [b_i]_0 \oplus [b_i]_1$.
3. $\mathcal{R}$ outputs $\{b_i\}_{i\in[n]}$.

Figure 20: Protocol of multi-query reverse private membership test.

6 Extension to Multi-query Reverse PMT

We present the functionality of multi-query reverse PMT (mqRPMT) in Figure 19. It is a core component of private set operations such as intersection cardinality (PSI-card) [11, 12] and intersection cardinality and sum (PSI-card-sum) [17, 22]. We give our mqRPMT protocol in Figure 20, which is straightforwardly constructed from our ssPMT functionality. We omit the security proof, since its security depends directly on that of ssPMT. In Figure 23 of Appendix C, we further present an optimized mqRPMT protocol from secret-shared OPRF.

Note that the state-of-the-art mqRPMT protocol [11] is constructed from public-key primitives, especially the DDH assumption, which introduces large computation overhead. Our protocols only utilize lightweight symmetric-key operations and largely improve the computational efficiency.

7 Implementation and Performance

7.1 Experimental Setup

We implement our protocols in C++, and the source code is available at https://github.com/CryptMatrix/ePSU-from-ssPMT and https://doi.org/10.5281/zenodo.20411092. All experiments are conducted on a server equipped with two Intel Xeon Platinum 8368Q processors with 1 TB of RAM. For fair comparison, we

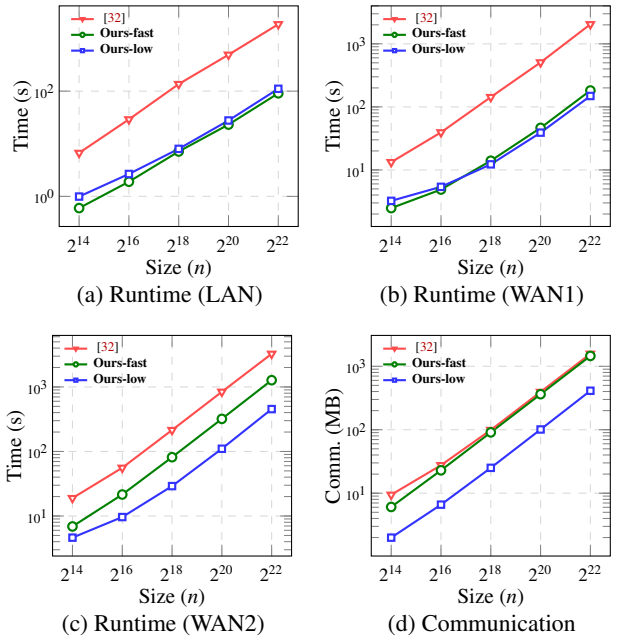


Figure 21: Communication and runtime comparison of ssPMT between the protocol [32] and our two constructions.

re-run the open-sourced implementations of all baseline protocols under the same experimental setup, and execute both our protocols and the baselines using a single thread. Each experiment is repeated ten times, and we report the average results. Network conditions are emulated using the Linux `tc` command. Following recent works [23, 44], we consider three network settings: a LAN setting with 10 Gbps bandwidth and 0.02 ms RTT, a WAN1 setting with 100 Mbps bandwidth and 80 ms RTT, and a WAN2 setting with 10 Mbps bandwidth and 80 ms RTT. Same as these works, we set the computational security parameter to $\kappa = 128$ and the statistical security parameter to $\lambda = 40$, and use 3 hash functions and no stash for Cuckoo hashing. For the implementation details, we adopt silent OT from the libOTe library [42]. For OPRF, OKVS, and ssPEQT, we use the implementations from the Vole-PSI library [39]. Our ssOPRF implementations are built upon the MPC-friendly PRF proposed in [1]. We set the bit-length of items to 128 for all our experiments, which is the same as or larger than that of baselines [9, 11, 23, 44, 50].

7.2 Performance of Our Secret-shared PMT

We compare the performance of our ssPMT protocols (ssPMT-fast and ssPMT-low) with the state-of-the-art construction [32]. The results are shown in Figure 21.

Specifically, both of our protocols significantly outperform the protocol of [32] in terms of running time. This is mainly because [32] relies on generic secure computation to evaluate SKE decryption and comparison circuits, which incurs substantial computational overhead. In particular, ssPMT-fast achieves an $11.19 \sim 21.25\times$ speedup in the LAN setting and a $2.56 \sim 11.23\times$ speedup in the two WAN settings. Similarly, ssPMT-low improves the running time by $6.75 \sim 17.76\times$ in the LAN setting and by $4.11 \sim 13.73\times$ in the two WAN settings. In terms of communication, ssPMT-fast reduces the communication cost by a factor of $1.07 \sim 1.55\times$ compared to [32], while ssPMT-low achieves a more substantial reduction of $3.82 \sim 4.73\times$.

7.3 Performance of Our Enhanced PSU

We compare the performance of our two enhanced PSU protocols to that of the state-of-the-art protocols proposed by Jia et al. [23] and Tu et al. [44]. The results are presented in Table 2. We also compare our enhanced PSU with standard PSU protocols that suffer from during-execution leakage and defer the results to Figure 24 in Appendix D.

Communication Comparisons. Our ePSU-low protocol achieves the lowest communication cost across all evaluated set sizes, ranging from 2^{14} to 2^{22}. Specifically, ePSU-low reduces communication by a factor of $14.25 \sim 20.72\times$ compared to Jia et al. [23], and by $2.06 \sim 2.24\times$ compared to Tu et al. [44]. The primary source of these savings lies in our secret-shared bit permutation. Although this building block incurs super-linear communication overhead, its hidden constant is small, and it significantly reduces the overhead of permutation employed in prior works [23, 44].

On the other hand, our ePSU-fast protocol incurs higher communication overhead than ePSU-low. This is mainly because the underlying secret-shared OPRF protocol [1] requires higher communication than standard OPRF constructions. Improving the communication efficiency of secret-shared OPRF is an interesting direction for future work. Nevertheless, ePSU-fast still achieves a $5.15 \sim 6.41\times$ reduction in communication compared to Jia et al. [23], albeit with a $1.34 \sim 1.45\times$ increase relative to Tu et al. [44].

Runtime Comparisons. In the LAN setting, our ePSU-fast protocol achieves the lowest runtime across all evaluated set sizes. Specifically, it improves the running time of the symmetric-key-based protocol of Jia et al. [23] by $1.77 \sim 3.25\times$, and that of the public-key-based protocol of Tu et al. [44] by $4.34 \sim 5.31\times$. Moreover, although ePSU-low incurs a modest runtime increase compared to ePSU-fast, it consistently achieves the second-best performance in the same LAN setting. Concretely, ePSU-low reduces runtime by $1.44 \sim 2.24\times$ relative to Jia et al. [23], and by $3.03 \sim 4.35\times$ relative to Tu et al. [44]. Overall, both of our constructions consistently outperform existing enhanced PSU protocols in terms of computational cost. This improvement stems from avoiding computation-intensive generic permutation protocols used by Jia et al. [23] and the public-key operations employed by Tu et al. [44].

Table 2: Comparison of communication and runtime among enhanced PSU protocols by Jia et al. [23], Tu et al. [44], and our two constructions. – denotes an execution error. The best result is highlighted in blue.

Size	Protocol	Comm. (MB)	Runtime (seconds)		
			LAN	WAN1	WAN2
2^{14}	[23]	32.921	2.131	9.887	33.555
	[44]	4.760	3.201	5.176	7.921
	Ours-fast	6.381	0.654	2.827	7.247
	Ours-low	2.309	1.055	3.539	5.014
2^{16}	[23]	137.419	3.959	20.904	122.511
	[44]	16.751	9.348	12.095	23.107
	Ours-fast	24.020	1.956	5.285	21.969
	Ours-low	7.685	2.747	5.886	10.146
2^{18}	[23]	577.731	12.854	67.060	504.519
	[44]	65.422	31.421	38.092	83.531
	Ours-fast	95.042	7.237	14.597	84.493
	Ours-low	29.241	8.108	13.104	31.215
2^{20}	[23]	2430.470	62.939	284.218	2140.660
	[44]	262.961	121.961	143.623	328.189
	Ours-fast	379.558	23.358	48.525	333.525
	Ours-low	117.268	28.044	41.108	122.468
2^{22}	[23]	–	–	–	–
	[44]	1062.792	488.979	575.539	1320.168
	Ours-fast	1520.828	92.238	189.289	1327.803
	Ours-low	476.630	112.253	156.541	508.717

In the two WAN settings, our ePSU-low protocol achieves the lowest runtime across almost all evaluated set sizes, except for two small-set cases in the moderate-bandwidth WAN1 setting. This behavior is expected, as ePSU-low incurs the lowest communication overhead and is therefore particularly well-suited for bandwidth-limited WAN environments. Specifically, in both WAN settings, ePSU-low improves the running time of Jia et al. [23] by $2.79 \sim 17.47\times$, and that of Tu et al. [44] by $1.46 \sim 3.67\times$. The performance gap compared to the LAN setting arises because, in bandwidth-constrained WAN environments, communication cost dominates the overall runtime. On the other hand, our ePSU-fast protocol achieves runtime reductions of $3.49 \sim 6.41\times$ relative to Jia et al. [23] in both WAN settings. Compared to Tu et al. [44], ePSU-fast improves runtime by $1.83 \sim 3.04\times$ in the WAN1 setting, and achieves comparable performance in the more bandwidth-limited WAN2 setting.

Performance Breakdown. We show the performance breakdown of our two enhanced PSU constructions in Table 3. The results are evaluated in the LAN setting. The overhead of both enhanced PSU constructions is dominated by the ssPMT protocol, while the overhead of ssOTd is marginal. Notably, in ePSU-low, thanks to our communication-optimized ssOTd, its communication accounts for only about 15% of the total overhead. Below, we analyze the overhead breakdown of the two ssPMT constructions.

For ssPMT-low, the communication cost is dominated by OPPRF and ssPEQT, whereas the bitPerm protocol contributes to only about 6% of the total communication overhead.

Table 3: Performance breakdown of our two enhanced PSU constructions.

Protocol	Sub-protocol		Comm. (MB)		Runtime (s)	
Ours-fast (size 2^{16})	ssPMT	ssOPRF+OKVS	22.933	20.926	1.897	1.398
		ssPEQT		2.007		0.531
	ssOTd		1.087		0.078	
	Total ePSU		24.02		1.956	
Ours-fast (size 2^{20})	ssPMT	ssOPRF+OKVS	363.216	331.311	23.134	16.517
		ssPEQT		31.905		6.970
	ssOTd		16.342		0.324	
	Total ePSU		379.558		23.358	
Ours-low (size 2^{16})	ssPMT	OPPRF	6.598	3.674	2.664	0.244
		ssPEQT		2.492		1.659
		bitPerm		0.432		0.788
	ssOTd		1.087		0.086	
	Total ePSU		7.685		2.747	
Ours-low (size 2^{20})	ssPMT	OPPRF	100.926	54.167	27.685	2.805
		ssPEQT		39.896		14.585
		bitPerm		6.866		10.742
	ssOTd		16.342		0.335	
	Total ePSU		117.268		28.044	

Similarly, the runtime of ssPMT-low is largely dominated by OPPRF and ssPEQT. This is because bitPerm operates on single-bit values and thus incurs a small constant factor. Hence, unlike prior works [23, 44], the permutation, although the only super-linear component, has a minor impact on the overall concrete overhead. For ssPMT-fast, the communication cost is dominated by ssOPRF. In contrast to ssPMT-low, ssPEQT contributes only marginally to the overall overhead. A similar trend is observed in the runtime breakdown, where ssOPRF is the primary performance bottleneck.

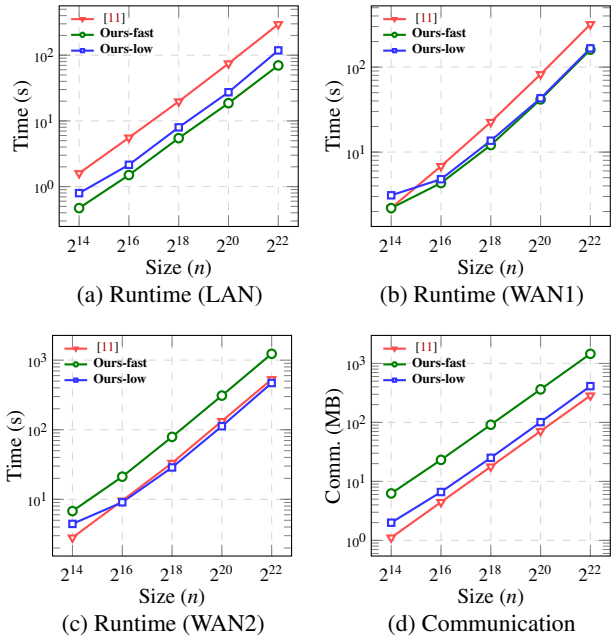
7.4 Performance of Our mqRPMT

We compare our two mqRPMT constructions built from ssPMT, denoted as mqRPMT-low and mqRPMT-fast, with the state-of-the-art protocol[2] proposed by Chen et al. [11]. The results are shown in Figure 22.

In the LAN setting, the running time of both protocols significantly outperforms that of the protocol of [11], since we rely on symmetric-key operations rather than the computation-intensive public-key primitives used in [11]. Specifically, mqRPMT-fast (resp., mqRPMT-low) achieves a $3.39 \sim 4.23\times$ (resp., $2.01 \sim 2.73\times$) computation improvement over Chen et al. [11].

In the WAN setting, while incurring larger communication costs compared to [11], our two protocols still improve the overall running time for set sizes ranging from 2^{16} to 2^{22}. Concretely, mqRPMT-low introduces about a $1.41 \sim 1.48\times$ increase in communication; however, it improves the running time by $1.42 \sim 1.92\times$ in the medium-bandwidth WAN1 setting and achieves slightly better running time even in the extremely low-bandwidth WAN2 setting. mqRPMT-fast further

[2] Although there exist mqRPMT constructions based on symmetric-key operations [9, 50], we do not include them as baselines since the protocol of [11] still outperforms them.



(a) Runtime (LAN) (b) Runtime (WAN1)

(c) Runtime (WAN2) (d) Communication

Figure 22: Communication and runtime comparison of mqRPMT between the protocol [11] and our constructions.

increases the communication overhead and therefore achieves a $1.57 \sim 1.99\times$ improvement only in the WAN1 setting.

8 Conclusion

In this work, we present a modular framework for constructing enhanced PSU based on the multi-query secret-shared private membership test (ssPMT) protocol and a new primitive called secret-shared oblivious transfer with default (ssOTd). Our framework can be viewed as a secret-shared variant of the standard PSU paradigm based on mqRPMT and OT, and it fundamentally prevents during-execution leakage by keeping all intermediate information secret-shared. We further propose two efficient ssPMT constructions relying primarily on lightweight symmetric-key primitives. Building on these components, our enhanced PSU protocols significantly outperform previous protocols.

Acknowledgments

We would like to express our deepest gratitude for the invaluable help provided by our shepherd as well as all the reviewers for their constructive comments. This work is supported by the Lee Kong Chian Chair Professorship, Singapore Management University, and the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grants (Proposal ID: 24-SIS-SMU-052). This work is also supported by the Nanyang Technological University Centre in Computational Technologies for Finance (NTU-CCTF), the National Research Foundation, Singapore, and Cyber Security Agency of Singapore under its National Cybersecurity R&D Programme and CyberSG R&D Cyber Research Programme Office. This work is also supported by the National Natural Science Foundation of China under Grant 62502079, the China Postdoctoral Science Foundation under Grant BX20240053, and the Sichuan Natural Science Foundation 2026NSFSC1463. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NTU-CCTF, National Research Foundation, Singapore, Cyber Security Agency of Singapore as well as CyberSG R&D Programme Office, Singapore.

Ethical Considerations

We consider direct stakeholders (those who implement or deploy our enhanced PSU protocols) and indirect stakeholders (those who use systems built from our enhanced PSU protocols).

Direct stakeholders include research teams and organizations that integrate our enhanced PSU protocols into privacy-preserving computation platforms.

Research Teams. Our work does not involve human subjects, physical experiments, or the collection of real-world sensitive data. The study is confined to protocol design, formal security analysis, implementation, and experimental evaluation using synthetic datasets. While we open-source our codes to foster reproducible research, these implementations are research prototypes and should not be treated as production-ready libraries without further system-level hardening.

Privacy Computation Organizations. Organizations may adopt our enhanced PSU protocols to protect sensitive information. However, deployers must recognize the limitations of our semi-honest, static adversary model, where our protocols are secure only when the parties follow the protocol specification. Real-world deployment introduces the risk of malicious behavior beyond our security model, such as arbitrary modification of protocol execution. Appropriate real-world defenses, such as access controls and upgrading to malicious-secure protocols, are essential before operational deployment.

Indirect stakeholders include end users whose data is processed by systems built on our protocols, as well as regulators and auditors responsible for oversight.

End Users. End users benefit from stronger privacy guarantees enabled by enhanced PSU. However, there are inherent deployment risks: because the protocol reveals the union of the datasets, the output itself may still allow for sensitive inferences about individuals, especially if datasets are highly correlated. We emphasize that our protocols should only be deployed within systems that explicitly account for this output leakage and faithfully adhere to standard cryptographic engineering best practices.

Regulators and Auditors. Regulators and auditors may leverage our formal definitions and security proofs to evaluate the compliance and privacy guarantees of deployed systems. Our modular framework supports transparent reasoning about what information is preserved or revealed. This clarity facilitates responsible deployment in regulated settings and aids auditors in ensuring that proper safeguards against out-of-model attacks are properly implemented.

Open Science

Our artifacts include the source code, benchmarking scripts, and detailed operation documentation. We have consolidated these research artifacts at https://github.com/CryptMatrix/ePSU-from-ssPMT and https://doi.org/10.5281/zenodo.20411092.

References

[1] Navid Alamati, Guru-Vamsi Policharla, Srinivasan Raghuraman, and Peter Rindal. Improved alternating-moduli prfs and post-quantum signatures. In *International Cryptology Conference*, pages 274–308, 2024.

[2] Martin R Albrecht, Alex Davidson, Amit Deo, and Daniel Gardham. Crypto dark matter on the torus: Oblivious prfs from shallow prfs and tfhe. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 447–476, 2024.

[3] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM Conference on Computer and Communications Security*, pages 535–548, 2013.

[4] Václad E Beneš. Optimal rearrangeable multistage connecting networks. *Bell system technical journal*, 43(4):1641–1656, 1964.

[5] 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 *USENIX Security Symposium*, pages 301–318, 2023.

[6] Dan Boneh, Yuval Ishai, Alain Passelègue, Amit Sahai, and David J Wu. Exploring crypto dark matter: New simple prf candidates and their applications. In *Theory of Cryptography Conference*, pages 699–729, 2018.

[7] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round ot extension and silent non-interactive secure computation. In *ACM Conference on Computer and Communications Security*, pages 291–308, 2019.

[8] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001.

[9] Gowri R Chandran, Thomas Schneider, Maximilian Stillger, and Christian Weinert. Concretely efficient private set union via circuit-based psi. In *ACM Asia Conference on Computer and Communications Security*, pages 149–162, 2025.

[10] Melissa Chase, Esha Ghosh, and Oxana Poburinnaya. Secret-shared shuffle. In *International conference on the theory and application of cryptology and information security*, pages 342–372, 2020.

[11] Yu Chen, Min Zhang, Cong Zhang, Minglang Dong, and Weiran Liu. Private set operations from multi-query reverse private membership test. In *International Workshop on Practice and Theory in Public Key Cryptography*, pages 387–416, 2024.

[12] Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Fast and private computation of cardinality of set intersection and union. In *International Conference on Cryptology and Network Security*, pages 218–231, 2012.

[13] Itai Dinur, Steven Goldfeder, Tzipora Halevi, Yuval Ishai, Mahimna Kelkar, Vivek Sharma, and Greg Zaverucha. Mpc-friendly symmetric cryptography from alternating moduli: Candidates, protocols, and applications. In *International Cryptology Conference*, pages 517–547, 2021.

[14] Minglang Dong, Cong Zhang, Yujie Bai, and Yu Chen. Efficient multi-party private set union without non-collusion assumptions. In *USENIX Security Symposium*, pages 2005–2024, 2025.

[15] Jiahui Gao, Son Nguyen, Marina Blanton, and Ni Trieu. Pulse: Parallel private set union for large-scale entities. In *ACM Conference on Computer and Communications Security*, pages 1784–1798, 2025.

[16] Jiahui Gao, Son Nguyen, and Ni Trieu. Toward a practical multi-party private set union. *Proceedings on Privacy Enhancing Technologies*, 4:622–635, 2024.

[17] Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh. Private set operations from oblivious switching. In *International Workshop on Practice and Theory in Public Key Cryptography*, pages 591–617, 2021.

[18] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In *International Cryptology Conference*, pages 395–425, 2021.

[19] Xiaojie Guo, Ye Han, Zheli Liu, Ding Wang, Yan Jia, and Jin Li. Birds of a feather flock together: How set bias helps to deanonymize you via revealed intersection sizes. In *USENIX Security Symposium*, pages 1487–1504, 2022.

[20] Feng Han, Xiao Lan, Weiran Liu, Lei Zhang, Hao Ren, Lin Qu, and Yuan Hong. Concretely efficient correlated oblivious permutation. Cryptology ePrint Archive, Paper 2025/449. URL: https://eprint.iacr.org/2025/449.

[21] Kyle Hogan, Noah Luther, Nabil Schear, Emily Shen, David Stott, Sophia Yakoubov, and Arkady Yerukhimovich. Secure multiparty computation for cooperative cyber risk assessment. In *IEEE Cybersecurity Development*, pages 75–76, 2016.

[22] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. On deploying secure computing: Private intersection-sum-with-cardinality. In *IEEE European Symposium on Security and Privacy*, pages 370–389, 2020.

[23] Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, and Dawu Gu. Scalable private set union, with stronger security. In *USENIX Security Symposium*, 2024.

[24] Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, and Dawu Gu. Scalable private set union, with stronger security. Cryptology ePrint Archive, Paper 2024/922, 2024. URL: https://eprint.iacr.org/archive/2024/922/20260422:031332.

[25] Yanxue Jia, Shifeng Sun, Hong-Sheng Zhou, Jiajun Du, and Dawu Gu. Shuffle-based private set union: Faster and more secure. In *USENIX Security Symposium*, pages 2947–2964, 2022.

[26] Lea Kissner and Dawn Song. Privacy-preserving set operations. In *International Cryptology Conference*, pages 241–257, 2005.

[27] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *ACM Conference on Computer and Communications Security*, pages 1257–1272, 2017.

[28] Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. Scalable private set union from symmetric-key techniques. In *Annual International Conference on the Theory and Application of Cryptology and Information Security*, pages 636–666, 2019.

[29] Arjen Lenstra and Tim Voss. Information security risk assessment, aggregation, and mitigation. In *Australasian Conference on Information Security and Privacy*, pages 391–401, 2004.

[30] Keyang Liu, Xingxin Li, and Tsuyoshi Takagi. Review the cuckoo hash-based unbalanced private set union: Leakage, fix, and optimization. In *European Symposium on Research in Computer Security*, pages 331–352, 2024.

[31] Qiang Liu, JaeYoung Bae, and JoonWoo Lee. Leakage-free enhanced private set union for balanced and unbalanced scenarios. Cryptology ePrint Archive, Paper 2025/2087, 2025.

[32] Xiang Liu and Ying Gao. Scalable multi-party private set union from multi-query secret-shared private membership test. In *International conference on the theory and application of cryptology and information security*, pages 237–271, 2023.

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

[34] Stanislav Peceny, Srinivasan Raghuraman, Peter Rindal, and Harshal Shah. Efficient permutation correlations and batched random access for two-party computation. In *International Workshop on Practice and Theory in Public Key Cryptography*, pages 76–109, 2025.

[35] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security Symposium*, pages 515–530, 2015.

[36] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based psi with linear communication. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 122–153, 2019.

[37] Lucas Piske, Jeroen van de Graaf, Anderson CA Nascimento, and Ni Trieu. Shared ot and its applications. *IACR Communications in Cryptology*, 2(2), 2025.

[38] Sihang Pu, Jiahui Gao, and Ni Trieu. Malicious private set union with two-sided output. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 408–437, 2026.

[39] Srinivasan Raghuraman and Peter Rindal. Blazing fast psi from improved okvs and subfield vole. In *ACM Conference on Computer and Communications Security*, pages 2505–2517, 2022.

[40] Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. Expand-convolute codes for pseudorandom correlation generators from lpn. In *International Cryptology Conference*, pages 602–632, 2023.

[41] Sivaramakrishnan Ramanathan, Jelena Mirkovic, and Minlan Yu. Blag: Improving the accuracy of blacklists. In *Network and Distributed System Security Symposium*, 2020.

[42] Peter Rindal and Lawrence Roy. libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe.

[43] Peter Rindal and Phillipp Schoppmann. Vole-psi: fast oprf and circuit-psi from vector-ole. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 901–930, 2021.

[44] Binbin Tu, Yujie Bai, Cong Zhang, Yang Cao, and Yu Chen. Fast enhanced private set union in the balanced and unbalanced scenarios. In *USENIX Security Symposium*, pages 3437–3456, 2025.

[45] Binbin Tu, Yu Chen, Qi Liu, and Cong Zhang. Fast unbalanced private set union from fully homomorphic encryption. In *ACM Conference on Computer and Communications Security*, pages 2959–2973, 2023.

[46] Aron van Baarsen and Marc Stevens. Amortizing circuit-psi in the multiple sender/receiver setting. *IACR Communications in Cryptology*, 1(3), 2024.

[47] Abraham Waksman. A permutation network. *Journal of the ACM (JACM)*, 15(1):159–163, 1968.

[48] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated ot with small communication. In *ACM Conference on Computer and Communications Security*, pages 1607–1626, 2020.

[49] Cong Zhang, Yu Chen, Weiran Liu, Liqiang Peng, Meng Hao, Anyu Wang, and Xiaoyun Wang. Unbalanced private set union with reduced computation and communication. In *ACM Conference on Computer and Communications Security*, pages 1434–1447, 2024.

[50] Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin. Linear private set union from multi-query reverse private membership test. In *USENIX Security Symposium*, pages 337–354, 2023.

A Threat Model

We give a formal definition of the UC framework. Let $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(1^\kappa,x,y,z)$ denote the output of environment $\mathcal{Z}$ when interacting with adversary $\mathcal{A}$ and two parties running protocol Π, on security parameter κ, inputs x,y for two parties, auxiliary input z for environment $\mathcal{Z}$. Let $\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}(1^\kappa,x,y,z)$ denote the output of environment $\mathcal{Z}$ after interacting in the ideal world with adversary Sim and ideal functionality $\mathcal{F}$, on security parameter κ, inputs x,y for two parties, auxiliary input z for environment $\mathcal{Z}$. Then, we have:

Definition 1. *Let $\mathcal{F}$ be a two-party functionality and Π be a protocol. We say that protocol Π UC-realizes functionality $\mathcal{F}$, if for any real-world adversary $\mathcal{A}$, there exists an ideal-world adversary Sim such that for any environment $\mathcal{Z}$,*

$$\left\{\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}(1^\kappa,x,y,z)\right\} \approx_c \left\{\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(1^\kappa,x,y,z)\right\}.$$

We prove the security of our protocol in the $\mathcal{G}$-hybrid model, where the parties execute a protocol with real messages and also have access to a sub-functionality $\mathcal{G}$. Supposing the protocol Π invokes sub-functionality $\mathcal{G}$, we say protocol Π UC-realizes functionality $\mathcal{F}$ in the $\mathcal{G}$-hybrid model.

B Definition of Oblivious Key-Value Store

We give the formal definition of OKVS as follows.

Definition 2 (Oblivious Key-Value Store [18])**.** *An oblivious key-value store (OKVS) is parameterized by a key space $\mathcal{K}$, a value space $\mathcal{V}$, and the statistical security parameter λ, and consists of two algorithms:*

- Encode*: on input a set of key-value pairs $L \in (\mathcal{K}\times\mathcal{V})^n$, outputs a vector $D \in \mathcal{V}^m$ or a failure indicator $\perp$ with probability bounded by $1/2^\lambda$.*
- Decode*: on input a vector $D \in \mathcal{V}^m$ and a key $k \in \mathcal{K}$, outputs a value $v \in \mathcal{V}$.*

Correctness: For all $L \in (\mathcal{K}\times\mathcal{V})^n$ with distinct keys for which $D \leftarrow \mathsf{Encode}(L)$ and $D \neq \perp$, it holds that $\forall (k,v) \in L$, $\mathsf{Decode}(D,k) = v$.

Obliviousness: For any distinct $\{k_1,\ldots,k_n\} \in \mathcal{K}^n$ and $\{k'_1,\ldots,k'_n\} \in \mathcal{K}^n$, Encode does not output $\perp$ on $\{k_1,\ldots,k_n\}$ and $\{k'_1,\ldots,k'_n\}$, and then the following distributions are statistically indistinguishable

$$\{\mathsf{Encode}(\{(k_1,v_1),\ldots,(k_n,v_n)\}) \mid v_i \leftarrow \mathcal{V}, i \in [n]\} \approx_s$$
$$\{\mathsf{Encode}(\{(k'_1,v_1),\ldots,(k'_n,v_n)\}) \mid v_i \leftarrow \mathcal{V}, i \in [n]\}.$$

Double obliviousness: For any distinct $\{k_1,\ldots,k_n\} \in \mathcal{K}^n$ such that Encode does not output $\perp$ on $\{k_1,\ldots,k_n\}$, then $\{\mathsf{Encode}(\{(k_1,v_1),\ldots,(k_n,v_n)\}) \mid v_i \leftarrow \mathcal{V}, i \in [n]\}$ is statistically indistinguishable from uniform distribution over $\mathcal{V}^m$.

Random decoding: For any $L := \{(k_i,v_i)_{i\in[n]}\} \in (\mathcal{K}\times\mathcal{V})^n$ with distinct keys such that $D \leftarrow \mathsf{Encode}(L)$ and $D \neq \perp$, it holds that for any $k \notin \{k_i\}_{i\in[n]}$, $\mathsf{Decode}(D,k)$ is statistically indistinguishable from uniform distribution over $\mathcal{V}$.

Protocol $\Pi_{\mathsf{mqRPMT\text{-}fast}}$

Parameters: Sender $\mathcal{S}$ and receiver $\mathcal{R}$. Set size n.
Input: $\mathcal{S}$ inputs $X = \{x_i\}_{i\in[n]}$ and $\mathcal{R}$ inputs $Y = \{y_i\}_{i\in[n]}$.
Protocol:

1. $\mathcal{S}$ and $\mathcal{R}$ invoke functionality $\mathcal{F}_{\mathsf{ssOPRF}}$, where $\mathcal{S}$ as the receiver inputs $\{x_i\}_{i\in[n]}$. $\mathcal{S}$ receives $\{[t_i]_0\}_{i\in[n]}$ and $\mathcal{R}$ as the sender receives $\{[t_i]_1\}_{i\in[n]}$ and O^F.
2. $\mathcal{R}$ computes $f_i := F(y_i)$ for $i \in [n]$, and computes the OKVS encoding $E := \mathsf{OKVS.Encode}(L)$, where $L := \{(y_i, f_i)\}_{i\in[n]}$. $\mathcal{R}$ sends E to $\mathcal{S}$.
3. For $i \in [n]$, $\mathcal{S}$ computes $d_i := \mathsf{OKVS.Decode}(E, x_i)$ and sets $[t_i]_0 := [t_i]_0 \oplus d_i$.
4. For $i \in [n]$, $\mathcal{S}$ and $\mathcal{R}$ invoke functionality $\mathcal{F}_{\mathsf{PEQT}}$, where $\mathcal{S}$ inputs $\{[t_i]_0\}_{i\in[n]}$ and $\mathcal{R}$ inputs $\{[t_i]_1\}_{i\in[n]}$. $\mathcal{R}$ learns $\{b_i\}_{i\in[n]}$.
5. $\mathcal{R}$ outputs $\{b_i\}_{i\in[n]}$.

Figure 23: Protocol of multi-query reverse private membership test from secret-shared OPRF.

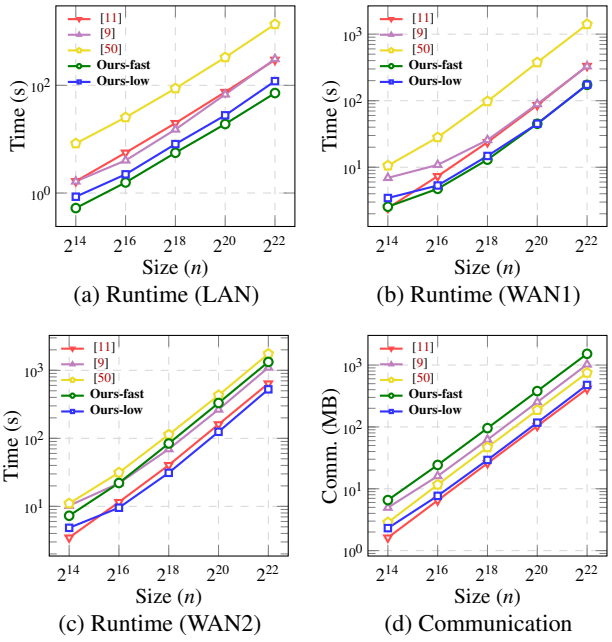


Figure 24: Performance comparison between our enhanced PSU constructions and PSU [9, 11, 50] suffering from during-execution leakage

C Computation-efficient mqRPMT from secret-shared OPRF

In Figure 23, we present an optimized mqRPMT protocol from secret-shared OPRF, rather than black-box invocation of ssOPRF-based ssPMT. The main observation is that in mqRPMT, the output is revealed to the receiver. Therefore, in our secret-shared OPRF-based ssPMT, we can invoke PEQT, instead of secret-shared PEQT. PEQT can be instantiated from OPRF, but secret-shared PEQT needs to execute the GMW protocol for the evaluation of the equality circuit. The security proof is similar to that of ssPMT-fast, except that ssPEQT is replaced with PEQT; hence, we omit it here.

D Comparison with PSU Suffering from During-execution Leakage

In Figure 24, we compare our enhanced PSU protocols with PSU [9, 11, 50] that suffer from during-execution leakage.

We first compare with the first linear-complexity PSU protocol [50]. In the LAN setting, ePSU-fast (resp., ePSU-low) achieves a $15.65 \sim 18.93\times$ (resp., $9.75 \sim 11.90\times$) running-time improvement over [50]. In two WAN settings, ePSU-fast and ePSU-low improve the running time by $1.30 \sim 8.34\times$ and $2.28 \sim 8.34\times$, respectively. For communication, ePSU-low reduces the cost by $1.23 \sim 1.60\times$ compared with [50].

We further compare our protocols with the symmetric-key-based protocol [9]. In the LAN setting, ePSU-fast (resp., ePSU-low) achieves a $2.57 \sim 4.29\times$ (resp., $1.81 \sim 2.58\times$) running-time improvement over [9]. In the WAN1 setting, ePSU-fast and ePSU-low improve the running time by $1.90 \sim 2.71\times$ and $1.74 \sim 2.04\times$, respectively. In the extremely low-bandwidth WAN2 setting, ePSU-low achieves a $2.09 \sim 2.29\times$ speedup. For communication, ePSU-low reduces the communication cost by $2.09 \sim 2.15\times$ compared with [9], whereas ePSU-fast uses about $1.33 \sim 1.53\times$ more communication.

Finally, we compare with the state-of-the-art public-key-based construction [11]. Specifically, in LAN setting, ePSU-fast (resp., ePSU-low) achieves a $3.14 \sim 4.12\times$ (resp., $1.93 \sim 2.71\times$) computation improvement over [11]. In the WAN1 setting, our protocols improve the performance for set sizes from 2^{16} to 2^{22}. ePSU-low introduces about a $1.13 \sim 1.18\times$ increase in communication; however, it improves the running time by $1.37 \sim 1.91\times$ in the medium-bandwidth WAN1 setting and achieves slightly better running time even in the extremely low-bandwidth WAN2 setting. ePSU-fast further increases the communication overhead and therefore achieves a $1.54 \sim 1.91\times$ improvement only in the WAN1 setting.

E Security Proof

E.1 Proof of Theorem 1

Proof. Let $\mathcal{A}$ be a static, semi-honest adversary that interacts with parties in the real-world protocol. We construct an adversary Sim for the ideal world such that no environment $\mathcal{Z}$ can distinguish with non-negligible probability whether it is interacting with $\mathcal{A}$ and the above protocol or with Sim in the ideal process for $\mathcal{F}_{\mathsf{ssOTd}}$. Specifically, simulator Sim starts by invoking a copy of $\mathcal{A}$ and running a simulated interaction of $\mathcal{A}$ with $\mathcal{Z}$ and parties running the protocol. We always implicitly

assume that Sim passes all communication between $\mathcal{A}$ and $\mathcal{Z}$. Below, we construct simulators to prove that the simulated execution is indistinguishable from the real-world protocol execution.

Corrupted sender. Sim proceeds as follows:

1. Sim randomly samples $\{m_{i,0}, m_{i,1}\}_{i\in[n]} \in (\mathbb{F}_{2^\ell} \times \mathbb{F}_{2^\ell})^n$, and simulates $\mathcal{A}$ receiving $(\{m_{i,0}, m_{i,1}\}_{i\in[n]})$ from $\mathcal{F}_{\mathsf{ROT}}$.
2. Sim receives $\{t_i, [b_i]_0\}_{i\in[n]}$ that $\mathcal{A}$ would send to the receiver.
3. Sim externally sends $\mathcal{S}$'s input X to $\mathcal{F}_{\mathsf{ssOTd}}$ and receives back the output (finished).

The joint view of $\mathcal{Z}$ and $\mathcal{A}$ in the real-world protocol is indistinguishable from the joint view of $\mathcal{Z}$ and $\mathcal{A}$ within Sim in the ideal process, since in both the ideal simulation and the real execution, the only message received by $\mathcal{A}$ is random $\{m_{i,0}, m_{i,1}\}_{i\in[n]}$.

Corrupted receiver. Sim proceeds as follows:

1. Sim receives $\{[b_i]_1\}_{i\in[n]}$ that $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{ROT}}$. Sim randomly samples $\{m_{i,[b_i]_1}\}_{i\in[n]} \leftarrow \mathbb{F}_{2^\ell}^n$ and simulates $\mathcal{A}$ receiving $\{m_{i,[b_i]_1}\}_{i\in[n]}$ from $\mathcal{F}_{\mathsf{ROT}}$.
2. Sim externally sends $\mathcal{R}$'s input $\{[b_i]_1\}_{i\in[n]}$ to $\mathcal{F}_{\mathsf{ssOTd}}$ and receives back the output $\{z_i\}_{i\in[n]}$.
3. For $i \in [n]$, Sim sets $[b_i]_0 := [b_i]_1$ and $t_i := z_i \oplus m_{i,[b_i]_1}$ if $z_i \neq \perp$, and sets $[b_i]_0 := 1 \oplus [b_i]_1$ and randomly samples $t_i \leftarrow \mathbb{F}_{2^\ell}$ otherwise. Sim simulates $\mathcal{A}$ receiving $\{t_i, [b_i]_0\}_{i\in[n]}$ from $\mathcal{S}$.

We demonstrate that the joint view of $\mathcal{Z}$ and $\mathcal{A}$ in the real-world protocol is indistinguishable from the joint view of $\mathcal{Z}$ and $\mathcal{A}$ within Sim in the ideal process. In both the ideal simulation and the real execution, the message received by $\mathcal{A}$ consists of n uniformly random values $\{m_{i,[b_i]_1}\}_{i\in[n]}$ from functionality $\mathcal{F}_{\mathsf{ROT}}$. Moreover, $\{t_i, [b_i]_0\}_{i\in[n]}$ is received by $\mathcal{A}$ in both worlds after invoking functionality $\mathcal{F}_{\mathsf{ssOTd}}$ with the same distribution. This is because $[b_i]_0$ is determined by $[b_i]_1$ and z_i, and $t_i = x_i \oplus m_{i,[b_i]_0}$ is uniformly random if $[b_i]_0 = 1 \oplus [b_i]_1$. Note that the simulator can query the ideal functionality and obtain the protocol output $\{z_i\}_{i\in[n]}$ only after it (acting as the sender) has received all protocol messages from the adversary. This ordering guarantees that the adversary learns the output information only upon protocol completion. Consequently, our protocol avoids the during-execution leakage. This completes the proof.

□

E.2 Proof of Theorem 2

Proof. We construct simulators to prove that the simulated execution is indistinguishable from the real-world protocol execution.

Corrupted sender. Sim proceeds as follows:

1. Sim receives X that $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{ssPMT}}$. Sim randomly samples $\{[b_i]_0\}_{i\in[n]} \leftarrow \mathbb{F}_2^n$ and simulates $\mathcal{A}$ receiving $\{[b_i]_0\}_{i\in[n]}$ from $\mathcal{F}_{\mathsf{ssPMT}}$.
2. Sim receives $(\{x_i, [b_i]_0\}_{i\in[n]})$ that $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{ssOTd}}$. Sim simulates $\mathcal{A}$ receiving (finished) from $\mathcal{F}_{\mathsf{ssOTd}}$.
3. Sim externally sends $\mathcal{S}$'s input X to $\mathcal{F}_{\mathsf{ePSU}}$ and receives back the output (finished).

We demonstrate that the joint view of $\mathcal{Z}$ and $\mathcal{A}$ in the real-world protocol is indistinguishable from the joint view of $\mathcal{Z}$ and $\mathcal{A}$ within Sim in the ideal process. This is because in both the ideal simulation and the real execution, the message received by $\mathcal{A}$ consists of n uniformly random bits from functionality $\mathcal{F}_{\mathsf{ssPMT}}$. Moreover, the same (finished) is received by $\mathcal{A}$ in both worlds after invoking functionality $\mathcal{F}_{\mathsf{ssOTd}}$. Besides, the outputs of $\mathcal{S}$ and $\mathcal{R}$ are the same in both worlds.

Corrupted receiver. Sim proceeds as follows:

1. Sim receives Y that $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{ssPMT}}$. Sim randomly samples $\{[b_i]_1\}_{i\in[n]} \leftarrow \mathbb{F}_2^n$ and simulates $\mathcal{A}$ receiving $\{[b_i]_1\}_{i\in[n]}$ from $\mathcal{F}_{\mathsf{ssPMT}}$.
2. Sim receives $\{[b_i]_1\}_{i\in[n]}$ that $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{ssOTd}}$.
3. Sim externally sends $\mathcal{R}$'s input Y to $\mathcal{F}_{\mathsf{ePSU}}$ and receives back the output Z.
4. Sim sets $Z' := Z \setminus Y$ and uses $\perp$ to pad Z' to n elements. Sim sends randomly shuffled Z' to $\mathcal{A}$.

We demonstrate that the joint view of $\mathcal{Z}$ and $\mathcal{A}$ in the real-world protocol is indistinguishable from the joint view of $\mathcal{Z}$ and $\mathcal{A}$ within Sim in the ideal process. In both the ideal simulation and the real execution, the message received by $\mathcal{A}$ consists of n uniformly random bits from functionality $\mathcal{F}_{\mathsf{ssPMT}}$. Moreover, $\{z_i\}_{i\in[n]}$ is received by $\mathcal{A}$ in both worlds after invoking functionality $\mathcal{F}_{\mathsf{ssOTd}}$ with the same distribution since $\{z_i\}_{i\in[n]}$ is computed according to the protocol output and randomly permuted. Note that the simulator can query the ideal functionality and obtain the protocol output Z only after it (acting as the sender) has received all protocol messages from the adversary. This ordering guarantees that the adversary learns the output information only upon protocol completion. Consequently, our protocol avoids the during-execution leakage. This completes the proof.

□

E.3 Proof of Theorem 3

Proof. We first demonstrate that in the real-world protocol, false positives may arise due to the random decoding property of OKVS, which should be avoided for indistinguishability. For any element $x_i \notin Y$, the decoded value d_i obtained from OKVS is uniformly random. Consequently, the probability that d_i is equal to the corresponding OPRF evaluation t_i is $2^{-\ell}$, where ℓ denotes the bit length of the OKVS output. We set $\ell := \lambda + \log n$. By the union bound over all n elements, the probability that any false positive occurs is at most $n \cdot 2^{-\ell} = 2^{-\lambda}$, which is negligible. We then construct simulators to prove that the simulated execution is indistinguishable from the real-world protocol execution.

Corrupted sender. Sim proceeds as follows:

1. Sim receives X that $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{ssOPRF}}$. Sim randomly samples $\{[t_i]_0\}_{i\in[n]} \leftarrow \mathbb{F}_{2^\ell}^n$ and simulates $\mathcal{A}$ receiving $\{[t_i]_0\}_{i\in[n]}$ from $\mathcal{F}_{\mathsf{ssOPRF}}$.
2. Sim randomly samples $E \leftarrow \mathbb{F}_{2^\ell}^m$ and simulates $\mathcal{A}$ receiving E from $\mathcal{R}$.
3. Sim receives $\{[\hat{t}_i]_0\}_{i\in[n]}$ that $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{ssPEQT}}$.
4. Sim externally sends $\mathcal{S}$'s input X to $\mathcal{F}_{\mathsf{ssPMT}}$ and receives back the output $\{[b_i]_0\}_{i\in[n]}$.
5. Sim simulates $\mathcal{A}$ receiving $\{[b_i]_0\}_{i\in[n]}$ from $\mathcal{F}_{\mathsf{ssPEQT}}$.

We demonstrate that the joint view of $\mathcal{Z}$ and $\mathcal{A}$ in the real-world protocol is indistinguishable from the joint view of $\mathcal{Z}$ and $\mathcal{A}$ within Sim in the ideal process. In both the ideal simulation and the real execution, the message received by $\mathcal{A}$ consists of n uniformly random values $\{[t_i]_0\}_{i\in[n]}$ from functionality $\mathcal{F}_{\mathsf{ssOPRF}}$ and an OKVS encoding E. The main difference is that E is computed from the OKVS encoding with the input $y_i, F(y_i)$ in the real-world execution, while E is randomly sampled in the simulation execution. We argue that by the double obliviousness property of OKVS, in the real-world protocol E is still indistinguishable from a uniform distribution, as the encoded values $F(y_i)$ are uniformly random and F is uniform by the property of ssOPRF. Moreover, $\{[b_i]_0\}_{i\in[n]}$ is received by $\mathcal{A}$ in both worlds after invoking functionality $\mathcal{F}_{\mathsf{ssPEQT}}$ with the same distribution since $\{[b_i]_0\}_{i\in[n]}$ is computed according to the output of functionality.

Corrupted receiver. Sim proceeds as follows:

1. Sim samples $\{[t_i]_1\}_{i\in[n]} \leftarrow \mathbb{F}_{2^\ell}^n$ and a random function $F : \mathbb{U} \to \mathbb{F}_{2^\ell}$. Sim simulates $\mathcal{A}$ receiving $(\mathcal{O}^F, \{[t_i]_1\}_{i\in[n]})$ from $\mathcal{F}_{\mathsf{ssOPRF}}$.
2. Sim receives E that $\mathcal{A}$ would send to $\mathcal{S}$.
3. Sim receives $\{[t_i]_1\}_{i\in[n]}$ that $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{ssPEQT}}$.
4. Sim externally sends $\mathcal{R}$'s input Y to $\mathcal{F}_{\mathsf{ssPMT}}$ and receives back the output $\{[b_i]_1\}_{i\in[n]}$.
5. Sim simulates $\mathcal{A}$ receiving $\{[b_i]_1\}_{i\in[n]}$ from $\mathcal{F}_{\mathsf{ssPEQT}}$.

We demonstrate that the joint view of $\mathcal{Z}$ and $\mathcal{A}$ in the real-world protocol is indistinguishable from the joint view of $\mathcal{Z}$ and $\mathcal{A}$ within Sim in the ideal process. In both the ideal simulation and the real execution, the message received by $\mathcal{A}$ consists of n uniformly random values $\{[t_i]_1\}_{i\in[n]}$ from functionality $\mathcal{F}_{\mathsf{ssOPRF}}$ and a random function $\mathcal{O}^F$. Moreover, $\{[b_i]_1\}_{i\in[n]}$ is received by $\mathcal{A}$ in both worlds after invoking functionality $\mathcal{F}_{\mathsf{ssPEQT}}$ with the same distribution since $\{[b_i]_1\}_{i\in[n]}$ is computed according to the output of functionality. This completes the proof.

□

E.4 Proof of Theorem 4

Proof. Similar to the analysis in the proof of Theorem 3, we set the bit length of OKVS outputs to $\ell := \lambda + \log m$, which ensures negligible false positives. We then construct simulators to prove that the simulated execution is indistinguishable from the real-world protocol execution.

Corrupted sender. Sim proceeds as follows:

1. Sim receives T_x that $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{OPPRF}}$. Sim randomly samples $\{d_i\}_{i\in[m]} \leftarrow \mathbb{F}_{2^\ell}^n$ and simulates $\mathcal{A}$ receiving $\{d_i\}_{i\in[m]}$ from $\mathcal{F}_{\mathsf{OPPRF}}$.
2. Sim receives $\{d_i\}_{i\in[m]}$ that $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{ssPEQT}}$. Sim randomly samples $\{[b'_i]_0\}_{i\in[m]} \in \mathbb{F}_2^m$ and simulates $\mathcal{A}$ receiving $\{[b'_i]_0\}_{i\in[m]}$ from $\mathcal{F}_{\mathsf{ssPEQT}}$.
3. Sim receives $(\pi, \{[b'_i]_0\}_{i\in[m]})$ $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{bitPerm}}$.
4. Sim externally sends $\mathcal{S}$'s input X to $\mathcal{F}_{\mathsf{ssPMT}}$ and receives back the output $\{[b_i]_0\}_{i\in[n]}$.
5. Sim simulates $\mathcal{A}$ receiving $\{[b_i]_0\}_{i\in[n]}$ from $\mathcal{F}_{\mathsf{bitPerm}}$.

We demonstrate that the joint view of $\mathcal{Z}$ and $\mathcal{A}$ in the real-world protocol is indistinguishable from the joint view of $\mathcal{Z}$ and $\mathcal{A}$ within Sim in the ideal process. The main difference is that in the real-world protocol, $\{d_i\}_{i\in[m]}$ are outputs from the OPPRF functionality, while in the simulated execution, $\{d_i\}_{i\in[m]}$ are randomly sampled. We argue that they are indistinguishable in both the ideal simulation and the real execution. This is because in the real-world protocol, for each bin, the sender queries only once and obtains an OPPRF evaluation. If the elements in that bin include this query, the sender receives a random r_i, otherwise, the sender learns a uniformly random value. In both cases, d_i is uniformly random, which is the same distribution as that of the simulated world. Besides, in both the ideal simulation and the real execution, the messages $\{[b'_i]_0\}_{i\in[m]}$ received by $\mathcal{A}$ are n uniformly random bits with identical distributions. Moreover, $\{[b_i]_0\}_{i\in[n]}$ is received by $\mathcal{A}$ in both worlds after invoking functionality $\mathcal{F}_{\mathsf{ssPEQT}}$ with the same distribution since $\{[b_i]_0\}_{i\in[n]}$ is computed according to the output of functionality.

Corrupted receiver. Sim proceeds as follows:

1. Sim receives key-value pairs $L = \{L_i\}_{i\in[m]}$ that $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{OPPRF}}$. Sim samples a random function $\mathcal{O}^F$, programs it on the points L, and sends $\mathcal{O}^F$ to $\mathcal{A}$.
2. Sim receives $\{r_i\}_{i\in[m]}$ that $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{ssPEQT}}$. Sim randomly samples $\{[b'_i]_1\}_{i\in[m]} \in \mathbb{F}_2^m$ and simulates $\mathcal{A}$ receiving $\{[b'_i]_1\}_{i\in[m]}$ from $\mathcal{F}_{\mathsf{ssPEQT}}$.
3. Sim receives $\{[b'_i]_1\}_{i\in[m]}$ that $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{bitPerm}}$.
4. Sim externally sends $\mathcal{R}$'s input Y to $\mathcal{F}_{\mathsf{ssPMT}}$ and receives back the output $\{[b_i]_1\}_{i\in[n]}$.
5. Sim simulates $\mathcal{A}$ receiving $\{[b_i]_1\}_{i\in[n]}$ from $\mathcal{F}_{\mathsf{bitPerm}}$.

We demonstrate that the joint view of $\mathcal{Z}$ and $\mathcal{A}$ in the real-world protocol is indistinguishable from the joint view of $\mathcal{Z}$ and $\mathcal{A}$ within Sim in the ideal process. In both the ideal simulation and the real execution, the message received by $\mathcal{A}$ consists of a random function $\mathcal{O}^F$ from $\mathcal{F}_{\mathsf{OPPRF}}$ programmed on $\{L_i\}_{i\in[m]}$ and n uniformly random values $\{[b'_i]_1\}_{i\in[m]}$ from functionality $\mathcal{F}_{\mathsf{ssPEQT}}$. Moreover, $\{[b_i]_1\}_{i\in[n]}$ is received by $\mathcal{A}$ in both worlds after invoking functionality $\mathcal{F}_{\mathsf{ssPEQT}}$ with the same distribution since $\{[b_i]_1\}_{i\in[n]}$ is computed according to the output of functionality. This completes the proof.

□