Malicious Private Set Union with Two-Sided Output

Sihang Pu[1], Jiahui Gao[2], and Ni Trieu[2]

[1] East China Normal University sihang.pu@gmail.com
[2] Arizona State University [jgao76,nitrieu]@asu.edu

Abstract. Private Set Union (PSU) allows two parties to compute the union of their private sets without revealing any additional information—in particular, it hides their common elements (the intersection). Although recent years have seen significant progress under the semi-honest model, resulting in several efficient two-party PSU protocols, notable gaps remain: (1) some prior works model the semi-honest PSU functionality inaccurately, and (2) practical and scalable maliciously secure protocols are still lacking, except when relying on heavy generic techniques (e.g., FHE, GMW, or general purpose NIZK).

In this paper, we address these issues directly and summarize our contributions as follows:

1. We revisit the formal definition of PSU, covering both the standard one-sided functionality (where only one party receives the output) and the two-sided variant (where both parties receive the output), refuting several flawed claims from prior work, and show that the notion of "during-execution leakage" was not well-defined in the literature, since the "enhanced" functionality is actually *equivalent* to the standard one.
2. We show how one of the fastest semi-honest protocols can be strengthened against malicious senders with a simple ad-hoc modification, while preserving its efficiency and simplicity.
3. As our main result, we present the first practical, concretely efficient, and maliciously secure two-sided PSU protocol, achieving at least a quadratic improvement over prior work. Along the way, we also resolve the challenge of assuring honest behavior for the hash-to-curve function *in the PSU context*—a task generally regarded as impractical due to the non-algebraic nature of the hash function.
4. We implement both protocols and compare them with existing schemes. Our experiments demonstrate that our maliciously secure protocols are only $1.1-2.4\times$ slower than the most efficient *semi-honest* protocols in the literature.

Table of Contents

Malicious Private Set Union with Two-Sided Output . . . 1
Sihang Pu, Jiahui Gao, and Ni Trieu
1 Introduction . . . 3
1.1 Our Contributions . . . 4
1.2 Performance Highlights . . . 4
1.3 Related Work . . . 5
2 Technical Overview . . . 5
2.1 Definition of Private Set Union . . . 6
2.2 Recap: Semi-Honest Protocol . . . 6
2.3 Warmup: PSU with Malicious Sender . . . 7
2.4 Main Result: PSU with Two-Sided Output . . . 8
3 Preliminaries . . . 12
3.1 Computational Assumptions . . . 13
3.2 The Map-to-Point Function . . . 13
3.3 Arguments of Knowledge . . . 14
4 Revisiting the PSU Functionality . . . 14
4.1 PSU Functionality for Semi-Honest Adversaries . . . 14
4.2 PSU Functionality for Malicious Adversaries . . . 17
5 Warmup: PSU with Malicious Sender . . . 18
5.1 Protocol Description . . . 18
5.2 Efficiency Analysis . . . 19
5.3 Security Analysis . . . 20
6 Main Protocol with Two-Sided Malicious Security . . . 20
6.1 Protocol Description . . . 21
6.2 Efficiency Analysis . . . 22
6.3 Security Analysis . . . 22
6.4 The Adapted Shuffling Argument . . . 22
6.5 The Batched Argument for DDH Relation . . . 24
7 Implementation and Performance . . . 24
7.1 Concrete Performance of Our Protocols . . . 26
A More Related Works . . . 31
B Additional Preliminaries . . . 32
B.1 Universal Composability . . . 32
B.2 Idealized Models . . . 32
C Trivial Instantiation of Standard PSU Functionality . . . 32
D The Enhanced PSU Functionality from JSZG24 . . . 33
E Missing Proofs . . . 34
E.1 Proof of Theorem 3 . . . 34
F PSU with Malicious Sender Realizing the Standard One-Sided Functionality . . . 39

1 Introduction

Private set union (PSU) is a cryptographic protocol that enables two parties to compute the union of their private sets while leaking no additional information. A subtlety here is that even the set difference is revealed as part of the output, the intersection remains hidden.

PSU has a wide range of important applications. For instance, it supports privacy-preserving identity management systems, enabling private-ID functionality [BKM+20]. It can facilitate joint cyber risk assessment and management by allowing organizations to collaboratively construct and share IP blacklists or vulnerability data without disclosing sensitive details [HLS+16]. In private database systems, PSU enables full joins across multiple data sources while keeping the underlying records private [KRTW19]. It also plays a role in association rule learning [KC04], where it helps securely identify patterns across distributed datasets, and in joint graph computations [BS05], where parties can compute properties of a graph without exposing their individual components. In the domain of network security, PSU allows for the aggregation of multi-domain network events [BSMD10], making collective traffic analysis possible without revealing proprietary information.

Given these diverse applications, the design of efficient, scalable, and secure PSU protocols has become a critical research focus. In recent years, notable progress has been made in the two-party setting: Kolesnikov, Rosulek, Trieu, and Wang [KRTW19] introduced a practical oblivious transfer (OT)-based PSU framework secure against semi-honest adversaries, which has since been optimized and extended by numerous follow-up works [GMR+21,JSZ+22,ZCL+23,BPSY23,JSZG24], leading to significant efficiency gains.

Nevertheless, despite this progress, two fundamental limitations remain in the existing literature, which we outline below.

Semi-Honest Setting. Jia et al. [JSZG24] observe that the standard PSU functionality might leak the intersection to the receiver, because who could infer it from the received union and its own input. They further argue that this leakage might be critical if the receiver aborts early, and they denote it "during-execution leakage." To mitigate this, they propose an "enhanced" functionality for stronger security. We find these claims unconvincing, and the enhanced functionality comes with additional overhead. This motivates us to ask:

Does standard functionality actually allow such leakage? More broadly, what is the proper definition for PSU?

Malicious Setting. While the semi-honest setting has seen rapid progress, only two works, to the best of our knowledge, have attempted to address malicious security.

The first, by Hazay and Nissim [HN12], observes that it is trivial to make PSU secure against malicious adversaries by simply having the sender transmit its items to the receiver in the clear. Although this may sound counterintuitive, it reflects the fact that a malicious receiver is very powerful in the ideal world: by submitting random inputs to the functionality, it can effectively extract the sender's items. To account for this, they put forward the functionality with *two-sided* output for malicious security (see Section 2 for an intuitive explanation).

In the same work, they also proposed a protocol with *quadratic* (computational and communication) overhead $O(\kappa \cdot (n\ell + N))$ in the security parameter κ, where $\ell = \kappa$ is the item length and n, N the set size. While feasible in some sense, this approach is impractical—and, more seriously, *insecure* due to flaws in their bucketing technique: precisely speaking, after applying the bucketing technique, the receiver can tell which buckets contain items lying in the set difference and which do not, as shown in [HN12, Page 27]. Buckets that do not contain those items still

have a non-negligible chance of holding intersections. For detailed attacks against such bucketing, see [JSZ+22]. [3]

A later work [GHLL22] considered only the standard one-sided functionality with malicious senders, using fully homomorphic encryption (FHE). Thus, the following important question remains open:

Is there a practical and highly scalable PSU protocol with two-sided output that is secure against malicious adversaries?

Regarding the second question above, it may appear natural to assume that a malicious protocol with two-sided output is far more costly than its semi-honest counterpart. For example, in the case of private set intersection (PSI)—a closely related problem to PSU—the fastest known malicious PSI protocols with two-sided output [MPR+20, YLS+24] still require at least $9-110$ *hours* of computation (with setup) for sets of size 2^{20}, whereas semi-honest PSI can be completed in under a second (with setup) [RR22]. The gap should be even more striking for PSU, which is generally considered harder than PSI because it must also hide the intersection. As a result, even the best *semi-honest* PSU protocols today still take $76-127$ seconds for sets of size 2^{20} [ZCL+23, CZZ+24, TBZ+25].

However, we will show, perhaps surprisingly, that a malicious two-sided PSU protocol could be much simpler and substantially more efficient than two-sided PSI, taking only $184-192$ seconds of computation for sets of size 2^{20}. This is just $1.6-2.4\times$ slower than the state-of-the-art semi-honest protocols.

1.1 Our Contributions

In this work, we address the questions above through the following contributions.

- In Section 4, we revisit the PSU functionality, covering both the standard $\mathcal{F}_{\mathsf{PSU}}$ with one-sided output, and the two-sided output version $\mathcal{F}_{\mathsf{PSU}_2}$. Along the way, we refute several flawed claims from prior work, and show that the notion of "during-execution leakage" was not well-defined in the literature—because the so-called "enhanced" functionality is actually *equivalent* to the standard one (Theorem 1).
- In Section 5, we boost the security of a semi-honest instantiation of $\mathcal{F}_{\mathsf{PSU}}$ against malicious senders with an *ad-hoc fix*, while retaining its simplicity and efficiency.
- In Section 6, as our main result, we introduce the first practical and efficient protocol for $\mathcal{F}_{\mathsf{PSU}_2}$ in the malicious setting. Our protocol achieves *linear* complexity in the security parameter, significantly improving over the quadratic overhead in existing work [HN12], by leveraging new ideas in non-standard ways.
- In Section 7, we implement both protocols and show that they are only $1.1-2.4\times$ slower than the state-of-the-art *semi-honest* protocols. We highlight the performance results in Section 1.2.

1.2 Performance Highlights

We briefly report the concrete performance of our protocols in Table 1, whereas detailed comparison and benchmark are presented in Section 7. Note that [ZCL+23] is the SKE variant and [CZZ+24] is the Curve25519 variant, as these two variants are the best performing ones. There is a recent

[3] Note that the same authors later withdrew this attack in a follow-up work [JSZG24], by stating that the same leakage could also be derived from the functionality itself—a claim we refute in Section 4.

Table 1: Performance comparison for protocols with set size 2^{20} over a LAN network. Here, `2M` denotes malicious security for a **two-sided** output functionality, `1M` denotes malicious security for a **one-sided** output functionality, and `SH` denotes semi-honest security for one-sided functionality. All protocols use 128-bit items, except [TBZ+25], which only supports 64-bit items. The symbol "–" indicates values not provided in their implementation.

Protocols	Runtime (ms)			Communication (KB)		
	Offline	Online	Total	Offline	Online	Total
[ZCL+23]*, `SH`	2,377,272	**76,170**	2,452,996	78,647	435,977	514,624
[CZZ+24], `SH`	24	108,682	**108,706**	0	**105,682**	**105,682**
[TBZ+25], `SH`	–	127,739	–	–	262,961	–
[GHLL22],`1M`	6,180	310,682	316,862	5	233,636	233,641
Ours, `1M`	24	**137,200**	**137,224**	0	**151,310**	**151,310**
Ours, `2M`	8,155	**184,421**	**192,576**	0	327,680	**327,680**
[MPR+20] PSI, `2M`	≈ 0	35,583,000	35,583,000	≈ 0	436,719	436,719
[YLS+24] PSI, `2M`	$> 10^8$	2,020,400	$> 10^8$	30,459,555	**287,300**	30,746,855

* According to [BPSY23], the advancement of OKVS improves its efficiency by $6 \sim 8\%$, but it still yields prohibitive *non-reusable* offline overhead, making it impractical for large sets.

work [CSSW25] improving the efficiency of [CZZ+24] by $\sim$ 10%, but their code has not been available yet for comparison.

Since no practical (or implementable) malicious PSU protocol with two-sided output exists in the literature, we compare instead against state-of-the-art two-sided malicious *PSI* protocols. For PSU, all runtime numbers are collected by running their open-sourced code on the same server (2.0 GHz CPU, 24 GB RAM, with SIMD and AES-NI enabled, single-threaded). For PSI, we report the numbers directly from their papers, given the huge performance gap.

1.3 Related Work

To give a comprehensive overview, we survey existing two-party PSU protocols under the semi-honest model in Appendix A. We trace the evolution of semi-honest 2-party PSU protocols over the past few years [GMR+21, JSZ+22, ZCL+23, BPSY23, CZZ+24, CSSW25, JSZG24], all of which build on the OT-based framework introduced in [KRTW19].

In contrast, only two protocols address malicious security, namely [HN12] and [GHLL22], both of which are already discussed in this section.

2 Technical Overview

In this section, we provide a high-level overview of our constructions. To set the stage, we start by recalling the functionality in Section 2.1. We next show how a semi-honest protocol (Section 2.2) can be extended to handle a malicious sender through an ad-hoc modification (Section 2.3), and conclude with our main two-sided output protocol built on new techniques (Section 2.4).

2.1 Definition of Private Set Union

We begin by recalling the standard PSU functionality with one-sided output below. The functionality $\mathcal{F}_{\mathsf{PSU}}$ takes as input a set W from the sender P_1 and a set V from the receiver P_0, and returns the union $V \cup W$ to the receiver.

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

- Wait for a set $V = \{v_1, \dots\} \subset \{0,1\}^\ell$ from party P_0.
- Wait for a set $W = \{w_1, \dots\} \subset \{0,1\}^\ell$ from party P_1.
- If $|V| \neq n$ or $|W| \neq N$, send $\perp$ to party P_0 and halt.
- Send $V \cup W$ to party P_0 and halt.

Since the difference $W \setminus V$ is inherently revealed as part of the output, the only information hidden from P_0 is the intersection $V \cap W$ within its own set V. In other words, P_0 should not learn whether any particular element $v_i \in V$ belongs to the intersection.

In their recent work [JSZG24, Section 3.2], Jia et al. claim that P_0 can select some "suitable" hash function to map both $W \setminus V$ and V into buckets, and then apply an analysis from prior works [JSZ+22] to find the intersection. This claim, however, is flawed: since the intersection $V \cap W$ is *independent* of the difference $W \setminus V$, it should remain **perfectly hidden** within V, even when P_0 knows both V and $W \setminus V$. A thorough discussion regarding $\mathcal{F}_{\mathsf{PSU}}$ and its variants is provided in Section 4.

2.2 Recap: Semi-Honest Protocol

Our starting point is a semi-honest protocol proposed by Chen et al. [ZCL+23, CZZ+24] which is one of the most efficient PSU protocols in the literature (as shown in Table 1). Assuming a cyclic group $\mathbb{G} = \langle g \rangle$ with prime order p, parties P_0 and P_1 hold V and W as in Section 2.1, then the protocol works as follows:

Semi-Honest PSU

`Step 1` P_0 sends blinded elements $\left\{x_i := H(v_i)^k\right\}_{i\in[n]}$ to P_1, where $k \xleftarrow{\$} \mathbb{Z}_p^*$, and H is a hash function producing uniformly random group elements in $\mathbb{G}$ under the idealized assumption.

`Step 2` P_1 sends $\{y_j := H(w_j)^s\}_{j\in[N]}$ to P_0 where $s \xleftarrow{\$} \mathbb{Z}_p^*$.

`Step 3` P_1 returns a shuffled list $\left\{e_{\pi(i)} := x_i^s\right\}_{i\in[n]}$ to P_0 following a random permutation $\pi : [n] \mapsto [n]$.

`Step 4` P_0 can unblind each $e_{i\in[n]}$ to get a sequence of bits $b_{j\in[N]}$ indicating whether $y_j \in \left\{e_i^{1/k}\right\}_{i\in[n]}$.

So far, this is exactly a shuffled OPRF [CHL22] built from Blinded DH exponentiation [Cha82] or Hashed Diffie-Hellman [NPR99] (hereafter denoted by HashDH), and the entire transcripts are pseudorandom under the generalized DDH assumption. Since the permutation is hidden, the only information P_0 learned is whether $w_{j\in[N]} \in V$ since

$$y_j \in \left\{e_i^{1/k}\right\}_{i\in[n]} \iff H(w_j)^s \in \left\{H(v_{\pi^{-1}(i)})^s\right\}_{i\in[n]},$$

which also implies the cardinality of the intersection $|V \cap W|$. To complete a PSU protocol, add the last step:

Semi-Honest PSU (cont.)

`Step 5` P_0 and P_1 input b_j and $(w_j, \perp)$ to an instance of OT functionality $\mathcal{F}_{\mathsf{OT}}$ for each $j \in [N]$, which returns w_j to P_0 if $b_j = 0$ and $\perp$ if $b_j = 1$. P_0 outputs $V \cup \{w_j\}_{b_j=0}$.

This protocol is simple, clean, secure against semi-honest adversaries, and enjoys strict linear communication and computational complexity.

Towards Malicious Security. To enhance the above protocol with malicious security, it's natural to consider using maliciously secure building blocks. However, a challenge arises that we still need to "bind" these blocks together, otherwise, a malicious receiver could *arbitrarily* choose its OT selection bits (i.e., b_j in `Step 5` above) even the shuffled OPRF and OT are both secure against malicious adversaries. It is unclear how to do this if we treat them as black boxes.

Perhaps counter-intuitively, the "binding" is not necessary for malicious security since Hazay and Lindell [HL10] showed that it's trivial to secure against a malicious receiver: if the simulator $\mathcal{S}$ samples a random set of size $|V|$ then sends it to $\mathcal{F}_{\mathsf{PSU}}$, $\mathcal{S}$ would get honest sender's set W from $\mathcal{F}_{\mathsf{PSU}}$ when the item universe is super-polynomially large since the intersection is empty with overwhelming probability. It further implies that a trivial protocol (that is, P_1 directly sends W to P_0) realizing $\mathcal{F}_{\mathsf{PSU}}$ against malicious adversaries, remains *insecure* against semi-honest adversaries (as the intersection should be hidden for semi-honest receivers). Please refer to Appendix C for more detail.

Given that the fully malicious security for $\mathcal{F}_{\mathsf{PSU}}$ is trivial, there are two viable options to upgrade a semi-honest protocol to anything non-trivial: either we consider the security against malicious sender and *semi-honest* receiver, or we consider the highly non-trivial functionality with two-sided output where *both* parties get the union $V \cup W$. We discuss the former in Section 2.3 as a warmup and the latter in Section 2.4.

2.3 Warmup: PSU with Malicious Sender

Intuitively, a malicious sender might arbitrarily choose OT messages in such a way that the honest receiver is unable to retrieve the intended item in $W \setminus V$. To mitigate such adversarial behavior, the simulator $\mathcal{S}$ must extract P_1's effective input during simulation. Without this extraction, $\mathcal{S}$ cannot appropriately instruct the ideal functionality $\mathcal{F}_{\mathsf{PSU}}$ to produce a consistent output for P_0.

If the shuffled OPRF described earlier were secure against malicious adversaries, $\mathcal{S}$ would be able to extract the input straightforwardly. However, the hash function H disrupts the underlying algebraic structure, making it challenging to achieve malicious security. While it is possible to make HashDH verifiable and UC-compatible, as shown in 2HashDH [JKK14], its verifiability for malicious security inherently conflicts with the shuffling property essential to the PSU protocol. Other maliciously secure shuffled OPRFs [MPR+20] built from significantly more expensive techniques which we will discuss in the next section.

We present a very simple ad-hoc fix to adapt the protocol in Section 2.2 to handle malicious senders. Instead of relying on malicious OPRF, we simply modify the `Step 5` to:

Sender-Malicoius PSU

`Step 5′` P_0 and P_1 input b_j and $(w_j \,\|\, \phi_j, \perp)$ to $\mathcal{F}_{\mathsf{OT}}$ for each $j \in [N]$, where ϕ_j is a (zero knowledge) argument of knowledge (AoK) to prove P_1 knows some $\alpha \in \mathbb{Z}_p$ such that $H(w_j)^\alpha = y_j$. Finally, P_0 outputs $V \cup \{w_j\}_{b_j=0 \,\wedge\, \phi_j \text{ is valid}}$.

Here y_j serves as a commitment of w_j in some sense and P_1 opens it during the OT phase. To see why this works, consider the following cases when ϕ_j is accepted:

- If $w_j \in V$: no matter whether $b_j = 0$, P_0 always outputs w_j;
- If $w_j \notin V$: P_0 only outputs w_j when $b_j = 0$. Recall $b_j = 0$ indicates $y_j \notin \{e_i^{1/k}\}$, which implies $H(w_j)^{k\cdot\alpha} \notin \{e_i\}_{i\in[n]}$ where $\{e_i\}$ are (adversarially) chosen by P_1 in `Step 3`. However, given H producing uniformly random group elements, $k \xleftarrow{\$} \mathbb{Z}_p$ and $w_j \notin V$, $H(w_j)^k$ remains pseudorandom to P_1 according to DDH. The adversary P_1 can find $\alpha, e_1, \ldots, e_n$ such that $H(w_j)^k \in \{e_i^{1/\alpha}\}$ with only negligible probability. Therefore, P_0 outputs w_j with overwhelming probability.

When ϕ_j is valid, the honest receiver P_0 behaves exactly as if interacting with the functionality $\mathcal{F}_{\mathsf{PSU}}$ Since P_1 receives no output and the only transcript (namely, $\{x_i\}$) remains pseudorandom, the adapted protocol appears to enjoy the security against malicious senders. In this view, the effective input of P_1 is precisely the set $\{w_j \mid b_j = 0 \wedge \phi_j \text{ is valid}\}$.

Subtle Issue and the Relaxation. There is a subtle issue that needs to be addressed: how should invalid AoK be handled? Note that P_0 cannot abort upon detecting an invalid AoK, as the adversary could exploit this behavior to determine whether $w_j \in V$ by *intentionally* attaching an invalid $w_j \,\|\, \phi_j$: P_0 would only abort if $w_j \notin V$, while it would receive $\perp$ and not abort otherwise. Consequently, P_0 must ignore any invalid AoK, even if it knows P_1 is cheating.

However, this introduces a complication during simulation. The simulator $\mathcal{S}$ relies on valid w_j as P_1's effective input, but the number of valid w_j might not match the expected size of W. Additionally, $\mathcal{S}$ cannot simply pad the set to resolve this mismatch as it does not know V. To address this, we consider a relaxation of the functionality $\mathcal{F}_{\mathsf{PSU}}$ [4] such that it no longer enforces a strict check on the sender's set size—mirroring malicious PSI protocols in the literature, which also allows a relaxed input size.

It is worth noting that the standard $\mathcal{F}_{\mathsf{PSU}}$ can also be realized *without* such relaxation by following a similar approach as what we show below. For completeness, we provide a detailed protocol realizing the standard $\mathcal{F}_{\mathsf{PSU}}$ in Appendix F.

2.4 Main Result: PSU with Two-Sided Output

One-sided functionality for PSU is too limited in a malicious setting, as it gives the receiver too much power. The receiver could potentially learn far more information than it should *without being detected* by the honest sender. To mitigate this limitation and rule out trivial protocols, we adopt the two-sided PSU functionality $\mathcal{F}_{\mathsf{PSU}_2}$ proposed in [HN12] (shown below), which ensures that the honest party can detect whether the malicious party behaves in an unusual way. For example, P_0 may find it suspicious if P_1 inputs a large set that has no intersection with P_0's set.

Functionality $\mathcal{F}_{\mathsf{PSU}_2}$

- Wait for a set $V = \{v_1, \ldots\} \subset \{0,1\}^\ell$ from party P_0.
- Wait for a set $W = \{w_1, \ldots\} \subset \{0,1\}^\ell$ from party P_1.
- If $|V| \neq n$ or $|W| \neq N$, send $\perp$ to both parties and halt.
- Send $V \cup W$ to the ideal adversary $\mathcal{S}$.
- Upon receiving a signal $\{\mathsf{deliver}, \mathsf{abort}\}$ from $\mathcal{S}$, do the following and halt:

[4] Formally defined in Figure 2 (*without highlighted code*) of Section 4.

- deliver: send $V \cup W$ to *honest* parties;
- abort: send $\perp$ to *honest* parties;

Achieving two-sided functionality $\mathcal{F}_{\mathsf{PSU}_2}$ is generally more challenging because it requires addressing the presence of a malicious receiver and tackling the "binding" challenge described in Section 2.2. Furthermore, the ad-hoc fix described in Section 2.3 no longer works: we have to enforce a check on the set size for both parties, as both receive the union.

Potential Building Blocks. The first approach that comes to mind is to leverage the maliciously secure shuffled and distributed OPRF (sDOPRF) developed by Miao, Patel, Raykova, Seth, and Yung [MPR+20], where both parties learn a sequence of indication bits $\{b_j\}$ of the other party's set (similar to what happens in `Step 4`).

Unfortunately, while this seems like a natural fit, the malicious sDOPRF has a significant downside: it's expensive in both computation and communication. The protocol relies on obliviously evaluating Dodis-Yampolskiy PRF [DY05], which requires Paillier [JL09, CDGS19] or Camenisch-Shoup encryptions [CS03, MPR+20]. Both of them are far less efficient than DH-based approaches. In fact, the sDOPRF proposed in [MPR+20] is *at least 195* times slower than the semi-honest shuffled HashDH. Another drawback of this protocol is that it relies on stronger assumptions than plain DDH, specifically the q-Diffie-Hellman Inversion assumption [MSK02, BB04] and the Strong RSA assumption [FO97].

Recently, there is a follow-up [YLS+24] replacing Camenisch-Shoup used in [MPR+20] with SPDZ [DPSZ12]. Thus it still depends on stronger assumptions (q-DHI and lattices) than DDH. While [YLS+24] is "somewhat" more efficient than [MPR+20] in the online phase—reducing runtime from 9 hours to 36 minutes—it requires super heavy and interactive offline computation and communication (29 GB of bandwidth and 110 hours of runtime for sets of size 2^{20}, as we shown in Table 1).

Upon closer examination, we find that malicious sDOPRF is overkill for PSU. [5] We will show that it is sufficient to realize $\mathcal{F}_{\mathsf{PSU}_2}$ against malicious adversaries using a *semi-honest* shuffled HashDH, incorporated into a carefully and innovatively designed protocol.

Strawman Protocol as a First Attempt. Instead of using OT, we modify the shuffled HashDH (described in Section 2.2) such that each party directly returns the *unblinded* elements that are in the set difference:

Malicious PSU (draft version)

`Step 1''` P_0 computes $K := g^k, \{x_i := H(v_i)^k\}_{i\in[n]}$, and P_1 computes $S := g^s, \{y_j := H(w_j)^s\}_{j\in[N]}$.
P_0 and P_1 exchange $\sigma_0 \leftarrow H_{\mathsf{com}}(K, \{x_i\})$ and $\sigma_1 \leftarrow H_{\mathsf{com}}(S, \{y_j\})$, where H_{com} is another hash function.

`Step 2''` P_0 and P_1 exchange $K, \{x_i\}$ and $S, \{y_j\}$ to check if σ_0 or σ_1 is consistent. [a]

`Step 3''` P_0 sends $\left\{u_{\pi_0(j)} := y_j^k\right\}_{j\in[N]}$ to P_1 following π_0, and
P_1 sends $\left\{e_{\pi_1(i)} := x_i^s\right\}_{i\in[n]}$ to P_0 following π_1.

`Step 4''` P_0 sends $\left\{e'_i \mid e'_i = e_i^{1/k} \wedge e'_{i\in[n]} \notin \{y_j\}_{j\in[N]}\right\}$ to P_1, and
P_1 sends $\left\{u'_j \mid u'_j = u_j^{1/s} \wedge u'_{j\in[N]} \notin \{x_i\}_{i\in[n]}\right\}$ to P_0.

[5] For example, malicious sDOPRF implies two-sided PSI with cardinality (i.e., both parties only learn the intersection cardinality). This, however, is *not* implied by two-sided PSU.

`Step 5″` P_0 and P_1 computes $\{(u'_j)^{1/k}\}$ and $\{(e'_i)^{1/s}\}$, respectively.

[a] Precisely speaking, the parties should abort if they receive any $x_i = g$ or $y_i = g$ during the protocol, where $\mathbb{G} = \langle g \rangle$. For simplicity, we omit this standard check here and in the following sections.

For the moment, let's ignore the fact that this protocol does not produce output yet. It is easy to verify that P_0 gets $H(w_j)$ for $w_j \notin V$ when both parties are honest because

$$(u'_j)^{1/k} = \left(y_{j'}^{k/s}\right)^{1/k} = H(w_{j'}), \text{where } j' := \pi_0^{-1}(j),$$

and P_1 only sends back u'_j such that

$$u'_j \notin \{x_i\}_{i\in[n]} \iff u'_j = H(w_{j'})^k \notin \left\{H(v_i)^k\right\}_{i\in[n]} \implies w_{j'} \notin V.$$

The second implication holds with overwhelming probability since $k \neq 0$ and H outputs uniformly random group elements. Then, it is clear that the above protocol is secure against semi-honest adversaries: the transcripts are either pseudorandom or correspond only to the set difference, revealing no information about the intersected items.

Without loss of generality, we assume P_1 is corrupted by the adversary $\mathcal{A}$ as the protocol is symmetric from both sides and each party has the same code. To get malicious security, we first note that the `Step 1″` and `Step 2″` committing each party's secret scalar *before* seeing any transcripts to prevent "rushing" adversaries, and it allows $\mathcal{S}$ to extract $\mathcal{A}$'s effective input.

Remember that $\mathcal{A}$ might learn the intersection cardinality after `Step 3″`, so the simulator $\mathcal{S}$ should extract and input a consistent set to $\mathcal{F}_{\mathsf{PSU}_2}$ at the very beginning. That is, $\mathcal{S}$ should extract the input *before* simulating $\{x_i\}$. Suppose both H and H_{com} are random oracles simulated by $\mathcal{S}$. The simulator $\mathcal{S}$ programs $H(w_j)$ to g^r with $r \xleftarrow{\$} \mathbb{Z}_p^*$. By checking query histories of H_{com} and H, $\mathcal{S}$ is able to find $S, \{y_j\}$ and w_j, respectively. Then $\mathcal{S}$ could know whether w_j is an effective input of $\mathcal{A}$ by checking $y_j \stackrel{?}{=} S^r$.

How to Reveal Correct Items? We want P_0 to output $\{w_j \notin V\}$ instead of $\{H(w_j)\}_{w_j\notin V}$. One might consider attaching a ciphertext ct_j to each y_j in `Step 1″`, where ct_j encrypts w_j using a pseudorandom key $H(w_j)^r$ with a random r, and modifying `Step 4″` to send $(u'_j)^r$ instead of u'_j. However, achieving malicious security requires attaching proof to convince P_0 that ct_j is valid. This introduces the same issue discussed in Section 2.3, where $\mathcal{A}$ can deliberately attach an *invalid* proof to observe whether P_0 aborts. We also cannot attach ct_j *only* for $w_j \notin V$ in `Step 4″` since the permutation π_0 is hidden from P_1. Moreover, proving the validity of ct_j for every $w_j \in W$ is infeasible because H is not algebraic (unless we resort to generic approaches).

If H was not a hash function, but an injective one, this issue would disappear, as there would be no need for ct_j or its corresponding proof: P_0 could simply retrieve w_j by inverting $H(w_j)$. One of our key insights is that we can make H an **invertible** mapping instead of a compressing hash, by using group $\mathbb{G}$ in a non-black-box way. Specifically, when instantiating $\mathbb{G}$ over elliptic curves and selecting $q \approx 2^\gamma$, we define $H := \mathsf{M2P} \circ \Pi$, where $\Pi : \{0,1\}^\gamma \mapsto \{0,1\}^\gamma$ is an ideal permutation and $\mathsf{M2P} : \mathbb{F}_q \mapsto \mathbb{G}$ is an invertible mapping from field elements to curve points with an unknown exponent. Assuming $\mathsf{M2P}$ is injective for the moment, we obtain its inverse $H^{-1} := \Pi^{-1} \circ \mathsf{M2P}^{-1}$ where $\mathsf{M2P}^{-1} : \mathbb{G} \mapsto \mathbb{F}_q \cup \perp$. Please refer to Section 3.2 for more detail.

Concretely, Elligator 2 [BHKL13] and its compatible curves can instantiate $\mathsf{M2P}$ as described in [AHH21]. To the best of our knowledge, this is the first work to employ Elligator in a non-standard way: we first apply the direct map and then use the inverse map to **recover** the encoded

item. In contrast, prior uses of Elligator either relied solely on the direct map $\mathsf{M2P}$ to "hash" into groups [AHH21], or only applied the inverse map $\mathsf{M2P}^{-1}$ in a rejection-sampling procedure to sample random bitstrings [MRR20, RT21].

It is important to note that in our case, H cannot be modeled as a random oracle, as its output distribution is far from uniform over $\mathbb{G}$. Nevertheless, we can still extract effective inputs from $\mathcal{A}$ by programming $\Pi(w_j)$ to $\mathsf{M2P}^{-1}(g^r)$ instead of programming on H. Another issue arises when H is no longer a random oracle: does DDH still hold for the tuple $\{g, g^s, H(w), H(w)^s\}$? Fortunately, a DDH-equivalent variant holds in this case when $\mathsf{M2P}$ is nearly injective [6] as shown in Lemma 1 of Section 3.

How to Prevent Dishonest Hashing? It is currently far away from the desired malicious security as P_0 cannot catch any dishonest behavior of $\mathcal{A}$ (i.e., corrupted P_1). For instance, assuming $n = N$, let us consider the following misbehaviors:

- $\mathcal{A}$ might set $\{y_j := g^{r_j}\}_{j\in[N]}$ in `Step 1''`, and send $\{e_i := K^{r_i}\}_{i\in[n]}$ in `Step 3''`. Such behavior cannot be caught by P_0 as $y_j^k = e_j$ for every $j \in [N]$, which means P_0 has nothing to send in `Step 4''` and it expects nothing to receive from P_1. Thus there are no further communication from either party until the protocol ends and P_0 would just output V.
- Even it executes `Step 3''` honestly, $\mathcal{A}$ is always able to cheat in `Step 1''`. For example, $\mathcal{A}$ could set $\{y_j := H(w_j)^{r_j}\}_{j\in[N]}$ with different scalars, then $\mathcal{A}$ can easily learn whether $w_j \in V$ after `Step 3''` by unblinding $u_{\pi_0(j)}$ with r_j to see whether

$$u_{\pi_0(j)}^{1/r_j} \in \{x_i\}_{i\in[n]} \iff y_j^{k/r_j} \in \{x_i\}_{i\in[n]} \iff H(w_j)^k \in \left\{H(v_i)^k\right\}_{i\in[n]}$$

  holds, because each r_j is bound to w_j.

The first issue can be addressed using the cut-and-choose paradigm, which allows P_0 to verify that $\mathcal{A}$ generates each e_i honestly. However, this approach is typically λ times more expensive, where λ is the statistical security parameter.

The second issue is more critical: directly proving that $\mathcal{A}$ executes `Step 1''` honestly is hard (or impractical) because of the non-algebraic nature of H.

Our second key idea is therefore to **postpone this check** (on `Step 1''`) until the end of the protocol (i.e., lazy-check), while ensuring that *all other steps* are executed honestly beforehand. We present the final protocol below.

Malicious PSU (final version, informal)

`Step 1*` P_0 computes $K := g^K, \left\{x_i := H(v_i \,\|\, 0^{\gamma-\ell})^k\right\}_{i\in[n]}$, and
P_1 computes $S := g^s, \left\{y_j := H(w_j \,\|\, 0^{\gamma-\ell})^s\right\}_{j\in[N]}$. They exchange $\sigma_0 \leftarrow H_{\mathsf{com}}(K, \{x_i\})$ and $\sigma_1 \leftarrow H_{\mathsf{com}}(S, \{y_j\})$.

`Step 2*` P_0 and P_1 exchange $K, \{x_i\}$ and $S, \{y_j\}$. They **abort** if σ_1 or σ_0 is invalid.

`Step 3*` P_0 returns $\left\{u_{\pi_0(j)} := y_j^k\right\}_{j\in[N]}$ with $\chi_0 \leftarrow \mathsf{AoK}_{\mathcal{R}_{\pi_0}}$ to P_1, and
P_1 returns $\left\{e_{\pi_1(i)} := x_i^s\right\}_{i\in[n]}$ with $\chi_1 \leftarrow \mathsf{AoK}_{\mathcal{R}_{\pi_1}}$ to P_0.

`Step 4*` They **abort** if χ_1 or χ_0 is invalid. P_0 computes

$$I := \left\{i \in [n] \mid e_i \notin \{y_j^k\}_{j\in[N]}\right\}, J' := \left\{\pi_0(j) \in [N] \mid y_j^k \notin \{e_i\}_{i\in[n]}\right\}$$

[6] Here, nearly injective means it has at most $c_{\max}$ preimages for each mapped point, where $c_{\max}$ is a small constant.

and sends $\left\{e'_i := e_i^{1/k}\right\}_{i\in I}$ with $\phi_0 \leftarrow \mathsf{AoK}^{I}_{\mathcal{R}_{\mathsf{DDH}}}$ to P_1.
Similarly, P_1 computes

$$J := \left\{j \in [N] \mid u_j \notin \{x_i^s\}_{i\in[n]}\right\}, I' := \left\{\pi_1(i) \in [n] \mid x_i^s \notin \{u_j\}_{j\in[N]}\right\}$$

and sends $\left\{u'_j := u_j^{1/s}\right\}_{j\in J}$ with $\phi_1 \leftarrow \mathsf{AoK}^{J}_{\mathcal{R}_{\mathsf{DDH}}}$ to P_0.

`Step 5*` P_0 **aborts** if ϕ_1 is invalid, $J \neq J'$, or any $w'_j \leftarrow H^{-1}\left((u'_j)^{1/k}\right)$ for $j \in J$ does not have $\gamma-\ell$ trailing zeros; otherwise P_0 outputs $V \cup \{w'_j\}$. P_1 carries out a similar check and derives the corresponding output.

In `Step 3*`, instead of following cut-n-choose, P_1 attaches a specialized zero-knowledge argument of knowledge AoK to prove $\{e_i\}_{i\in[n]}$ is a permutated list of blinded x_i:

$$\mathsf{AoK}_{\mathcal{R}_\pi}\left\{s \in \mathbb{Z}_p, \pi : [n] \mapsto [n] \mid \forall\, i \in [n],\, g^s = S \wedge x_i^s = e_{\pi(i)}\right\}.$$

The concrete construction of this AoK is described in Section 6.4 and is adapted from a shuffling argument [Gro10].

In `Step 4*`, P_1 incorporates another batched AoK to demonstrate that every sent u'_j are unblinded u_j:

$$\mathsf{AoK}^{J}_{\mathcal{R}_{\mathsf{DDH}}}\left\{s \in \mathbb{Z}_p \mid \forall\, j \in J,\, u'_j = u_j^{1/s} \wedge g = S^{1/s}\right\},$$

where the set $J \subset [N]$ denotes the set of indices corresponding to the set difference. The concrete construction of this AoK is given in Section 6.5. Once it is verified, P_0 is convinced that

$$\left(u'_{\pi_0(j)}\right)^{1/k} = u_{\pi_0(j)}^{1/(k\cdot s)} = y_j^{1/s}.$$

After the above two AoK are verified, we can address the dishonest hashing by *restricting* the domain size of items in `Step 1*`. If $\mathcal{A}$ generates y_j dishonestly in `Step 1*`, i.e., $y_j \neq H(\cdot)^s$, then P_0 would realize that $y_j \notin \{e_i^{1/k}\}_{i\in[n]}$ for any e_i received from $\mathcal{A}$. This follows from the fact that each $e_i^{1/k} = H(v_{\pi_1^{-1}(i)})^s$, given that $\mathcal{A}$ is required to execute `Step 3*` honestly.

This implies that $\pi_0(j) \in J'$, as shown in `Step 4*`. Consequently, P_0 should be able to compute $(u'_{\pi_0(j)})^{1/k} = y_j^{1/s}$ in `Step 5*`; otherwise, it aborts if $j \notin J$. Moreover, $H^{-1}(y_j^{1/s})$ produces a uniformly random element $w'_j \in \{0,1\}^\gamma$, since $H^{-1} := \Pi^{-1} \circ \mathsf{M2P}^{-1}$, where Π is modeled as an ideal permutation over $\{0,1\}^\gamma$, and $\mathsf{M2P}$ outputs group elements with *unknown* exponent.

By choosing the field size $\gamma \geq 2\kappa$, item length ℓ so that $\gamma - \ell \geq \kappa$ (where κ is the security parameter), and encoding each w_j as $H(w_j \,\|\, 0^{\gamma-\ell})$, the probability that $w'_j \in \{0,1\}^\ell \,\|\, 0^{\gamma-\ell}$ is negligible. In other words, if $\mathcal{A}$ misbehaves in `Step 1*`, P_0 would catch this by revealing invalid items (i.e., items outside the domain) and abort.

3 Preliminaries

In this work, the computational and statistical security parameters are denoted by κ, λ, respectively. We use [.] notation to refer to a set. For example, $[m]$ implies the set $\{1, \ldots, m\}$. Additionally, we use $[i, j]$ to denote the set $\{i, \ldots, j\}$. We denote the concatenation of two bit-strings x and y by $x \,\|\, y$.

For additional preliminaries—including the UC framework and the idealized functionalities for the random oracle $\mathcal{F}_{\mathsf{RO}}$ and ideal permutations $\mathcal{F}_{\mathsf{IP}}$—we refer the reader to Appendix B.

3.1 Computational Assumptions

Definition 1 (Generalized Decisional Diffie-Hellman). *k-DDH states that given $\{\mathbb{G}, p, g, g^s, g^{r_1}, \ldots, g^{r_k}\}$, $g^{sr_1}, \ldots, g^{sr_k}$ are pseudorandom.*

Apart from the standard DDH, we state the non-uniform variant of the DDH assumption below and show it is equivalent to DDH as implicitly implied by [AHH21, Theorem 3.6]

Definition 2 ($\mathcal{D}_\mathbb{G}$-DDH). *Given $\{\mathbb{G}, p, g, g^s, D_1, \ldots, D_k\}$ and a distribution $\mathcal{D}_\mathbb{G}$ over $\mathbb{G}$, $\{D_1^s, \ldots, D_k^s\}$ are pseudorandom, where $D_{i\in[k]} \xleftarrow{\$} \mathcal{D}_\mathbb{G}$.*

Lemma 1 ($\mathcal{D}_\mathbb{G}$-DDH $\Leftrightarrow$ DDH [AHH21]). *Let $\mathbb{G}$ be a cyclic group of order p, $1_\mathbb{G}$ the identity element, and $\mathcal{D}_\mathbb{G}$ a probability distribution on $\mathbb{G}$. If there exists a PPT rejection sampler RS for $(\mathbb{G}, \mathcal{D}_\mathbb{G})$ with acceptance rate $\frac{1}{c}$, then the probability of PPT adversaries against $\mathcal{D}_\mathbb{G}$-DDH and DDH of solving the respectively other problem differs by at most $k\mathcal{D}_\mathbb{G}(1_\mathbb{G}) + \frac{1}{p}$ and solving k-DDH with the help of a $\mathcal{D}_\mathbb{G}$-DDH adversary requires at most kc executions of RS on average.*

Proof (sketch). Given an adversary $\mathcal{A}$ against $\mathcal{D}_\mathbb{G}$-DDH, the reduction $\mathcal{R}$ can solve DDH instance (g, A, B, C) by invoking the rejection sampler RS on B^r for randomized $r \xleftarrow{\$} \mathbb{Z}_p^*$. Suppose B^r passes the rejection sampling, then $\mathcal{R}$ feeds $\mathcal{A}$ with (g, A, B^r, C^r) and outputs what $\mathcal{A}$ outputs. For k-DDH, $\mathcal{R}$ just repeats this process k times.

The other direction is trivial since DDH is self-reducible. □

3.2 The Map-to-Point Function

We recall some curve-specific and invertible mapping $\mathsf{M2P} : \mathbb{F}_q \mapsto \mathbb{G}$ and $\mathsf{M2P}^{-1} : \mathbb{G} \mapsto \mathbb{F}_q \cup \perp$. $\mathsf{M2P}$ maps a field element to a curve point with an unknown exponent. As investigated in [AHH21], we do not model $\mathsf{M2P} \circ H$ entirely as a random oracle since the output of $\mathsf{M2P}$ is far from uniformly random.

Given $\frac{|\mathbb{F}_q|}{|\mathbb{G}|}$ is a constant and $\mathsf{M2P}$ has at most $c_{\max}$ preimages for each point, we recall the following lemmas from [AHH21].

Lemma 2 (Existence of Rejection Sampler [AHH21]). *There is a PPT rejection sampler for $(\mathbb{G}, \mathcal{D}_\mathbb{G})$ with average acceptance rate $\frac{|\mathbb{F}_q|}{|\mathbb{G}| \cdot c_{\max}}$. Therefore, Lemma 1 holds if $c_{\max} = \mathsf{poly}(\kappa)$.*

Lemma 3 (Uniform Inverse Map [AHH21]). *For any uniformly sampled $g \xleftarrow{\$} \mathbb{G}$, $\mathsf{M2P}^{-1}(g)$ outputs results $r \neq \perp$ with probability $p \geq \frac{|\mathbb{F}_q|}{|\mathbb{G}| \cdot c_{\max}}$ and the distribution of outputs $r \neq \perp$ is uniform in $\mathbb{F}_q$.*

Concrete Instantiation. In this work, we choose a concrete instantiation of $\mathsf{M2P}$ and $\mathsf{M2P}^{-1}$ as the direct map and the inverse map of Elligator 2 [BHKL13, Vai], which has $c_{\max} = 2$. Note that the potential instantiation (i.e., elliptic curve) might have a small co-factor $h_c > 1$, then in this case we consider $|\mathbb{G}_p| = |\mathbb{G}|/h_c$ and $c'_{\max} = c_{\max} \cdot h_c$. In detail, the direct map is nearly injective that every $r \in \mathbb{F}_q$ maps to a different curve point, except that r and $-r$ map to the same point. The inverse map outputs a random preimage of the given curve point by flipping the sign by outputting $\mathsf{M2P}^{-1}(R) \xleftarrow{\$} \{r, -r\}$ such that $\mathsf{M2P}(\pm r) = R \in \mathbb{G}$.

PARAMETERS: A prover $\mathcal{P}$ and a verifier $\mathcal{V}$.

FUNCTIONALITY:

- Upon receiving input $(\mathcal{R}, x, w)$ from $\mathcal{P}$:
  - Return $(\mathcal{R}, x, 1)$ to $\mathcal{V}$ if $\mathcal{R}(x, w) = 1$, and return $(\mathcal{R}, x, 0)$ otherwise.

Fig. 1: Ideal Functionality $\mathcal{F}_{\mathsf{AoK}}$

3.3 Arguments of Knowledge

We recall a useful lemma below, which is used extensively in zero-knowledge arguments (and proofs).

Definition 3 (Schwartz-Zippel Lemma). *Let $P[X_1, \ldots, X_n]$ be a non-zero multivariate polynomial of degree d over $\mathbb{F}_q$, then the probability of $P(r_1, \ldots, r_n) = 0$ for randomly chosen $r_1, \ldots, r_n \xleftarrow{\$} \mathbb{F}_q$ is at most $\frac{d}{q-1}$.*

In this work, we usually write (zero knowledge) arguments of knowledge (AoK) as

$$\mathsf{AoK}\{w \mid \mathcal{R}(x, w) = 1\}$$

where x is the (publicly known) statement, w is the witness, and $\mathcal{R}$ is the relation we want to prove. We recall the ideal functionality for AoK in Figure 1. In practice (and in our implementations), we use the Fiat-Shamir heuristic [FS87] to compress an interactive public-coin protocol into non-interactive zero-knowledge (NIZK) in the random oracle model.

Regarding the UC-security, we assume the existence of *straight-line extractor* for any AoK used in this work and we assume they UC-realize the $\mathcal{F}_{\mathsf{AoK}}$ in the proof of our main protocols. There are multiple ways to make AoK UC-compatible and with different trade-offs which we outline below:

- Relying on the idealized group model [Sho97, Mau05]. In this case, the simulator $\mathcal{S}$ is able to extract the randomness and the witness.
- Relying on UC-commitments as shown in [CF01, CLOS02, HM04].
- Using Fischlin transformation [Fis05] for Σ-protocols, which is about ~ 20 times slower than Fiat-Shamir as evaluated in [CL24].

4 Revisiting the PSU Functionality

We revisit the private set union functionality (recalled in Figure 2) by considering both semi-honest and malicious adversaries.

4.1 PSU Functionality for Semi-Honest Adversaries

In a very recent work by Jia et al. [JSZG24] (denoted JSZG24), the authors also revisit PSU functionality and protocols against semi-honest adversaries by making the following claims:

1. In [JSZG24, Section 3.2], they show that, given the union and its own set, the receiver can *"... learn that items belonging to $X \cap Y$ with high probability"*; they treat these shared items as leakage under the standard functionality $\mathcal{F}_{\mathsf{PSU}}$ (Figure 2).

PARAMETERS: Set sizes n and N, item length ℓ, and two parties: P_0 and P_1

FUNCTIONALITY:

- Wait for a set $V = \{v_1, \dots\} \subset \{0,1\}^\ell$ from party P_0.
- Wait for a set $W = \{w_1, \dots\} \subset \{0,1\}^\ell$ from party P_1.
- If $|V| \neq n$ or $|W| > N$ or $|W| < N$, send $\perp$ to party P_0 then halt .
- Send $V \cup W$ to party P_0 and halt.

Fig. 2: Ideal Functionality of Private Set Union $\mathcal{F}_{\mathsf{PSU}}$. We also consider a relaxed functionality in Section 5 by *ignoring* the highlighted code.

  - As a consequence of the above claim, the authors argue the bucketing attack from [JSZ+22] is *ineffective* when applied to [KRTW19], since the same leakage can already be inferred from the union.
2. In [JSZG24, Section 3.3], they introduce what they call "during-execution leakage," capturing the fact that the receiver may abort before the protocol completes; they also recall an attack by Guo et al. [GHL+22] to justify the practical relevance of such leakage.
3. In [JSZG24, Section 4], they put forward an "enhanced PSU functionality" aimed at preventing such during-execution leakage and providing stronger security.

In this section, however, we argue that the above claims are inaccurate and provide the following corrections. Specifically, we show that (1) the functionality (and the union it reveals) does not introduce any meaningful leakage, so the bucketing attack [JSZ+22] remains *effective* against [KRTW19]; (2) neither during-execution leakage nor the attack of Guo et al. [GHL+22] aligns with the standard semi-honest security model; and (3) the proposed "enhanced PSU functionality" neither captures during-execution behavior nor is it well defined.

No Leakage from the Standard Functionality. Here we view "leakage derived from the union" as equivalent to "leakage from the functionality," since the PSU functionality $\mathcal{F}_{\mathsf{PSU}}$ provides no other interface. Any information an adversary obtains must therefore come from the functionality's output, namely the union.

Recall that the authors of [JSZG24] claim that, given access to a standard PSU functionality, a receiver can infer information about the intersection as follows. After receiving the union $X \cup Y$ from $\mathcal{F}_{\mathsf{PSU}}$, it forms the two sets $X \setminus Y$ and Y (where X and Y denote the sender's and receiver's inputs), and hashes them into two tables T_0 and T_1, respectively, using an "appropriate" hash function h. They claim that if some bin is empty in T_0 but occupied in T_1 (e.g., $\exists, i, y_j$ such that $h(y_j) = i$, $T_0[i] = \varnothing$, and $T_1[i] = y_j$), then one can conclude with good probability that $y_j \in X \cap Y$. The intuition is that, when hashing X into a table with $|X| \approx |T_0|$, empty bins should be rare; thus an empty bin $T_0[i]$ suggests that some element $x \in X$ that is missing from $X \setminus Y$ (i.e., some $x \in X \cap Y$) should have mapped to i-th bin. If additionally x and y_j collide in that bin (as $T_1[i] = y_j$), there is a non-negligible probability to infer $x = y_j$. This is essentially the bucketing attack of [JSZ+22], originally used against an earlier PSU protocol of [KRTW19] (denoted KRTW19).

However, that attack against KRTW19 relies on an implicit assumption that **does not hold** in the ideal world of PSU functionalities: namely, that the sender and receiver agree on the *same* randomly sampled hash function h. The bucketing attack is probabilistic, with its success probabilities taken over the sampling $h \xleftarrow{\$} \mathcal{H}$. In the ideal world, the receiver only observes $X \setminus Y$ and Y, and no hash function is sampled by anyone; any probabilities the receiver computes (e.g., the

probability of an empty bin, or the probability that $y_j \in X \cap Y$) come solely from its *own* choice of $h \xleftarrow{\$} \mathcal{H}$ and therefore do not reveal information about the intersection. Hence, the attack is invalid in the ideal world.

In fact, given that the intersection $X \cap Y$ of two sets is indeed *independent* of its difference $X \setminus Y$, the intersection should be **perfectly hidden** in Y even with the knowledge of $X \setminus Y$ and Y. In other words, when $X \setminus Y$ and Y are fixed, the intersection $X \cap Y$ still has the full entropy and can be chosen independently from elements in Y. As a simple example, consider the sender holds $X := \{x, 4\}$ where $x \in [3]$ and receiver holds $Y := \{1, 2, 3\}$. In this case, the adversary (receiver) $\mathcal{A}$ obtains $X \cup Y = \{1, 2, 3, 4\}$, which reveals *no information* about x except $x \in [3]$. It is clear that $\mathcal{A}$ can only guess x with probability no greater than $1/3$, and information-theoretically cannot do any better.

Therefore, the standard functionality $\mathcal{F}_{\mathsf{PSU}}$ is well-defined for semi-honest adversaries and yields no leakage beyond the output.

During-Execution Leakage Violates Semi-Honest Behavior. In [JSZG24, Section 3.3], the authors further observed that the protocol in [KRTW19] could let the receiver learn shared items *during* protocol execution (i.e., *before* an abort), rather than only after the protocol completes. They refer to this kind of leakage as "during-execution leakage."

Moreover, to motivate the practical influence of such notion, they observe that such during-execution leakage can be leveraged by a realistic attack [GHL+22] to reveal more information [7].

However, we would like to point out that the attack from [GHL+22] relies on a malicious receiver adaptively choosing its input set across many sessions, and therefore does not apply to semi-honest adversaries. Furthermore, the early-abort of a receiver already violates the standard semi-honest behavior of parties.

Remark 1. We stress that the early-abort only lets the receiver learn its *own output* (e.g., part of the union items, the cardinality of the intersection, etc.) before the protocol completes. This is fundamentally *different* from the type of leakage discussed above, and learning the output before termination should not be considered a security issue in the semi-honest setting.

As a direction for future work, one could explore hybrid security models that explicitly account for during-execution behavior and sit between the semi-honest and malicious settings.

The Enhanced Functionality is Not Well-Defined. The authors propose an "enhanced functionality" with stronger security (recalled in Figure 3) to formally model such during-execution leakage in [JSZG24, Section 4]. For clarity, we simplify the description by omitting unnecessary notions (e.g., session id, states of each party).

We show that this enhanced PSU functionality is actually *equivalent to* the standard functionality (i.e., there is no non-trivial difference) in either *semi-honest* or *malicious* settings. Our reduction works in the $\mathcal{F}_{\mathsf{PSU}}$-hybrid setting within UC model, with black-box access to $\mathcal{F}_{\mathsf{PSU}}$.

We provide detailed discussion and the proof of Theorem 1 in Appendix D.

Theorem 1. *The standard functionality $\mathcal{F}_{\mathsf{PSU}}$ defined in Figure 2 is equivalent to the "enhanced" functionality $\mathcal{F}_{e\mathsf{PSU}}$ (recalled in Figure 3) within the UC framework, against both semi-honest and malicious adversaries. That is, any protocol realizing $\mathcal{F}_{\mathsf{PSU}}$ can also realize $\mathcal{F}_{e\mathsf{PSU}}$, and vice versa.*

[7] This is a heuristic attack with *non-negligible* correctness error, and according to their experiments, they can recover 30% $\sim$ 80% elements of the intersection in 70 $\sim$ 135 protocol invocations, when the set size is ranging from 512 $\sim$ 2048.

PARAMETERS: Set sizes n and N, item length ℓ, and two parties: P_0 and P_1

FUNCTIONALITY:

- Wait for a set $V = \{v_i, \dots\} \subset \{0,1\}^\ell$ from party P_0.
- Wait for a set $W = \{w_i, \dots\} \subset \{0,1\}^\ell$ from party P_1.
- If $|V| \neq n$ or $|W| \neq N$, send $\perp$ to party P_0 then halt.
- Upon receiving $\langle\mathsf{Response}\rangle$ from $\mathcal{S}$, send $Z := V \cup W$ to P_0.
- Send $\langle\mathsf{Finished}\rangle$ to P_1.

Fig. 3: The "enhanced" PSU Functionality $\mathcal{F}_{e\mathsf{PSU}}$ from [JSZG24], with modifications highlighted in blue to show the differences from the standard $\mathcal{F}_{\mathsf{PSU}}$ functionality (Figure 2).

Loosely speaking (as a proof sketch), a sender can simply pass its input to $\mathcal{F}_{\mathsf{PSU}}$ and then immediately output $\langle\mathsf{Finished}\rangle$, which suffices to realize $\mathcal{F}_{e\mathsf{PSU}}$. Note that $\langle\mathsf{Finished}\rangle$ has to be sender's output and cannot be any internal tape, otherwise not observable, since the environment only sees honest party's output. Importantly, sender can even send $\langle\mathsf{Finished}\rangle$ at any step since parties are interactive Turing machines in UC.

More broadly, when adversaries are semi-honest (which is the setting in [JSZG24]), we do not believe there is any meaningful difference between standalone and UC model, including for PSU. This is a folklore observation. For example, Maji, Prabhakaran, and Rosulek note on [MPR10, p.6] that *"... we point out that the traditional (standalone) security definition is equivalent to the UC security definition, since the simulation required by semi-honest security does not, and need not, extract the inputs of the corrupt players ..."*.

4.2 PSU Functionality for Malicious Adversaries

While the standard functionality is well-defined in semi-honest settings as discussed above, the malicious setting is a bit tricky. For instance, a malicious receiver might leverage the knowledge of the intersection cardinality and arbitrarily modify its input to extract additional information about the intersection, by using the attack mentioned in [GHL+22]. Though the cardinality is implied by the union of sets (i.e., it is part of the output of the functionality), the malicious receiver could *abort* the protocol as soon as it learns the cardinality.

However, as pointed out in [HL10, Example 2], PSU functionality against malicious adversaries can be trivially realized by communicating their sets in the plain, since a malicious receiver can extract the sender's entire input by querying the functionality with randomized elements. For completeness, we provide a detailed discussion in Appendix C. This renders the above attack *less powerful* for the PSU context: a rational adversary might prefer not to abort the protocol so it can completely learn the sender's set.

Semi-honest Receiver and Malicious Sender. As discussed above and shown in [HL10], a protocol realizing the standard functionality (Figure 2) secure against malicious adversaries does not imply security against semi-honest adversaries. Therefore, the best we can hope for this functionality is the security against a *semi-honest* receiver and malicious sender (since the security against a malicious receiver is trivial).

Two-sided Output. To rule out trivial instantiations of PSU functionalities against malicious adversaries, Hazay and Lindell suggested the functionality with two-sided output [HL10]. This also makes sense in practice: if a malicious party P_0 inputs a randomized set to learn the entire set of the other party P_1, P_1 would be suspicious about P_0's behavior since the intersection is empty; whereas in standard functionality, the sender (P_1) is oblivious about the receiver's (P_0) behavior.

PARAMETERS: Set sizes n and N, item length ℓ, and two parties: P_0 and P_1.

FUNCTIONALITY:

- Wait for a set $V = \{v_1, \dots\} \subset \{0,1\}^\ell$ from party P_0.
- Wait for a set $W = \{w_1, \dots\} \subset \{0,1\}^\ell$ from party P_1.
- If $|V| \neq n$ or $|W| \neq N$, send $\perp$ to *both* parties and halt.
- Send $V \cup W$ to the ideal adversary $\mathcal{S}$.
- Upon receiving a signal $\{\mathsf{deliver}, \mathsf{abort}\}$ from $\mathcal{S}$, do the following and halt:
  - deliver: send $V \cup W$ to *honest* parties;
  - abort: send $\perp$ to *honest* parties;

Fig. 4: Ideal Functionality of Two-Sided Private Set Union PSU_2.

In Figure 4, we formally describe the PSU functionality with two-sided output, where the corrupted party may deviate from its input, may *abort* the procedure at any time, and may decide the time of message delivery (i.e., could be rushing).

5 Warmup: PSU with Malicious Sender

In this section, we present a simple PSU protocol designed for a semi-honest receiver and a malicious sender, which is particularly suited for one-sided input PSU scenarios. For a detailed discussion on the protocol addressing both malicious senders and receivers, we refer the reader to Section 4.2 and Appendix C.

5.1 Protocol Description

Our protocol builds on the common framework of existing two-party semi-honest PSU protocols (see Section 2.3), with the key difference in the OT phase: the receiver is now able to prevent a malicious sender from injecting invalid inputs during OT execution. To achieve this, we require an AoK for each OT message, guaranteeing that the sender uses consistent inputs throughout the protocol. In particular, the AoK ensures that P_1 knows the exponent k such that $H(w_i)^k = y_i$. The full construction of our protocol is given in Figure 5.

Note that the OT execution can be optimized to reduce approximately half of the communication cost for this step. This is possible because $\mathcal{P}_0$ discards the OT message when $b_i = 1$. Recall that OT typically consists of two steps:

- Random OT: The sender obtain two random messages (r_0, r_1), while the receiver obtains r_b, where b is the receiver's choice bit.
- OT Correction: The sender sends $(r_0 \oplus m_0, r_1 \oplus m_1)$ to the receiver, allowing the receiver to unmask them and obtain m_b, where (m_0, m_1) are the sender's OT messages.

PARAMETERS:

- Set sizes n and N, item-length ℓ, security parameter κ, statistical parameter λ, a cyclic group $\mathbb{G} := \langle g \rangle$ with prime order p, and two parties: P_0 holding $V = \{v_1, \ldots, v_n\} \subset \{0,1\}^\ell$, and P_1 holding $W = \{w_1, \ldots, w_N\} \subset \{0,1\}^\ell$.
- An oblivious transfer and a non-interactive zero-knowledge AoK.
- Two one-way hash functions $H : \{0,1\}^\ell \to \mathbb{G}$, and $H_{com} : \{0,1\}^\star \to \{0,1\}^\gamma$.

PROTOCOL:

I. **Shuffle OPRF:**
1. Each P_i chooses a random private exponent k_i in the group $\mathbb{G}$.
2. For each $i \in [n]$, P_0 computes $x_i = H(v_i)^{k_0}$, and sends a set $\{x_i\}_{i\in[n]}$ to P_1. Similarly, for each $i \in [N]$, P_1 computes $y_i = H(w_i)^{k_1}$, and sends a set $\{y_i\}_{i\in[N]}$.
3. P_1 chooses a permutation function $\pi : [n] \to [n]$. P_1 computes a set $E := \{e_i := x_{\pi(i)}^{k_1}\}_{i\in[n]}$, and sends it to P_0.

II. **OT Phase:**
1. P_0 prepare a binary vector $\mathbf{b}$ of length N. For each $i \in [N]$, P_0 computes $y_i' = y_i^{k_0}$, sets $b_i = 0$ if $y_i' \notin E$, and $b_i = 1$ if $y_i' \in E$.
2. For each $i \in [N]$, P_0 and P_1 send b_i and $\{w_i \,\|\, \phi_i, \perp\}$, respectively, to an OT instance, where $\phi_i = \mathsf{AoK}\{k \mid H(w_i)^k = y_i\}$ is an argument of knowledge [a] AoK to prove that P_1 knows some exponent k such that $H(w_i)^k = y_i$.
3. P_0 sets $R = \varnothing$. For any $i \in [N] \wedge b_i = 0$, P_0 receives $w_i' \,\|\, \phi_i'$ from each OT instance. P_0 **discards** w_i' if ϕ_i' is invalid; otherwise, sets $R = R \cup \{w_i'\}$.
4. P_0 outputs $V \cup R$.

[a] For example, $\phi = (c, z)$ where $c = H_{com}\left(H(w_i) \,\|\, y_i \,\|\, H(w_i)^r\right)$ and $z = c \cdot k_1 + r$

Fig. 5: Our PSU Protocol for Semi-Honest Receiver and Malicious Sender

In our case, the receiver never needs to unmask $r_1 \oplus m_1$ when $m_1 = (\perp, \perp)$. As a result, the sender can skip sending the second message in the correction phase. This reduces the communication cost by nearly half, as the random OT phase incurs a fixed communication cost derived from the base OT.

Another optimization is that P_1 can truncate the PRF values $x_{\pi(i)}^{k_1}$, which have a length equivalent to that of an elliptic curve point (e.g., approximately 2κ), to $\lceil \lambda + \log(nN) \rceil$ before sending X to the receiver. Similarly, the receiver can perform the same truncation on $y_i^{k_0}$ before executing the set membership check.

5.2 Efficiency Analysis

Our protocol has communication and computation costs that scale linearly with the number of parties, $O(n+N)$. Specifically, the communication cost is given by $2\kappa(n+N) + n\lceil \lambda + \log(nN) \rceil + 3\kappa^2 + (|\phi|\kappa + \ell)N$, where the first two terms arise from the shuffle OPRF phase, assuming the length of an elliptic curve point is 2κ; the third term corresponds to the base OT; and the last term accounts for the OT extension as ℓN and AoK as $\phi\kappa N$, where $\phi = 3$. When considering a typical parameter set where $\lambda = 40, \kappa = 128, \ell = 128$ and assuming $n = N$ for an easier analysis, the communication cost is $(936 + 2\log(n))n + 49152$ bits.

Regarding the computational cost, our protocol requires $2(n+N)+3\kappa^2$ scalar multiplications, $n+N$ hash-to-point evaluations, along with the additional cost of computing the AoK which costs $3N$ scalar multiplications in total. We present the concrete performance results of our PSU protocol in Section 7.

5.3 Security Analysis

The security of our one-side PSU protocol is presented in the following theorem.

Theorem 2. *The protocol shown in Figure 5 realizes the relaxed functionality defined in Figure 2 (*without *the highlighted code) in the standalone model by modeling H, H_{com} as random oracles, against the malicious sender and semi-honest receiver.*

Proof. We begin by proving the protocol's security against a semi-honest receiver, followed by addressing the case of a malicious sender.

The receiver P_0 is corrupted. Since P_1 has no output and the adversary $\mathcal{A}$ is semi-honest, the simulator $\mathcal{S}$ only needs to simulate the transcripts. The simulation follows the same approach as in the semi-honest PSU protocol, with one additional step: $\mathcal{S}$ samples ϕ_i for each $w_i \in W$ using the zero-knowledge simulator of AoK.

The sender P_1 is corrupted. We first examine the transcripts. The simulator $\mathcal{S}$ can generate the first message from P_0 (which is also the only transcript) by sampling $\{R_i\}_{i\in[n]}$ where $R_i \leftarrow \mathbb{G}$. By the Decisional Diffie-Hellman (DDH) assumption, this is indistinguishable from the real protocol execution.

Now, let's analyze the final output. $\mathcal{S}$ must provide an input to $\mathcal{F}_{\mathsf{PSU}}$ that ensures the honest party P_0 receives the correct union. To do this, $\mathcal{S}$ observes the OT input of P_1, denoted as $\{w'_i \,\|\, \phi'_i, \alpha_i\}_{i\in[N]}$, and sends $\{w'_i\}_{i\in[N]}$ to $\mathcal{F}_{\mathsf{PSU}}$ (without the highlighted code) to complete the simulation for every valid ϕ_i.

We now argue that this approach correctly simulates the final output. Consider the cases where ϕ'_i is accepted, meaning there exists some $k \in \mathbb{Z}_p^*$ such that $H(w'_i)^k = y_i$. In this case:

- If $w'_i \in V$, then P_0 always outputs w_j.
- If $w'_i \notin V$, then P_0 only outputs w'_i when $y_i^{k_0} \notin E$, which implies $H(w'_i)^{k_0 \cdot k} \notin E$, where $E = \{e_i\}$ is chosen by P_1 in Step I.3. Since H produces uniformly random group elements and k_0 is sampled from $\mathbb{Z}_p^*$, the value $H(w'_i)^{k_0}$ remains pseudorandom to P_1 (by the DDH assumption) as long as $H(w'_i)^{k_0} \notin \{x_i\}_{i\in[n]}$. The adversary $\mathcal{A}$ can only find values $k, e_1, \ldots, e_n$ such that, $H(w'_i)^{k_0} \in \{e_i^{1/k}\}$ with negligible probability. As a result, P_0 outputs w'_i with overwhelming probability.

For inputs $\{w'_i \,\|\, \phi'_i\}$ where ϕ'_i is invalid, P_0 never outputs them unless $w'_i \in V$, and $\mathcal{S}$ does not include them in its input to $\mathcal{F}_{\mathsf{PSU}}$. Therefore, the final output of P_0 remains identical in both the simulated and real executions. □

6 Main Protocol with Two-Sided Malicious Security

We provide a very efficient PSU protocol with two-sided malicious security in Figure 6 where we only describe the behavior for P_0 since the protocol is *symmetric* from both sides. Note that we discard the *explicit transformation* between $\mathbb{F}_q$ and $\{0,1\}^\gamma$ for brevity since we always consider a $q \approx 2^\gamma$.

6.1 Protocol Description

Loosely speaking, we continue to use the semi-honest shuffled HashDH protocol. However, unlike existing frameworks that attach an OT phase to retrieve the set difference, we have each party directly return the "unblinded" elements in the difference. Then, we apply an "invertible" hash function to reveal the correct items. Please refer to Section 2.4 for high-level intuitions of the construction, and refer to Sections 6.4 and 6.5 for detailed construction of the corresponding zero-knowledge AoK.

PARAMETERS:

- Set sizes n and N, item-length ℓ, security parameter κ, statistical parameter λ, a cyclic group $\mathbb{G} := \langle g \rangle$ with prime order p, and two parties: P_0 holding $V = \{v_1, \ldots, v_n\} \subset \{0,1\}^\ell$, and P_1 holding $W = \{w_1, \ldots, w_N\} \subset \{0,1\}^\ell$.
- An invertible map-to-point function $\mathsf{M2P} : \mathbb{F}_q \mapsto \mathbb{G}$ with $c_{\max}$ preimages described in Section 3.2, a hash function $H_{\mathsf{com}} : \{0,1\}^\star \mapsto \{0,1\}^\gamma$, and an ideal permutation $\Pi : \{0,1\}^\gamma \mapsto \{0,1\}^\gamma$.
- AoK ideal functionalities for relations $\mathcal{R}_\pi$ (Section 6.4) and $\mathcal{R}_{\mathsf{DDH}}$ (Section 6.5)

PROTOCOL:

I. **Offline Setup:**
   1. P_0 samples random private exponents $k_0 \xleftarrow{\$} \mathbb{Z}_p^*$, and computes $h_0 = g^{k_0}$.

II. **Exchanging Committed Inputs (1st & 2nd Round):**
   1. For each $i \in [n]$, P_0 pads v_i with $\gamma - \ell$ zeros and computes $x_i = \mathsf{M2P}\left(\Pi(v_i \,\|\, 0^{\gamma-\ell})\right)^{k_0}$. P_0 generates a commitment $\sigma_0 \leftarrow H_{\mathsf{com}}(h_0, x_1 \ldots x_n)$.
   3. P_0 sends σ_0 to P_1, and receives σ_1 from P_1.
   4. P_0 sends h_0 and the set $\{x_i\}_{i\in[n]}$ to P_1, while receiving $h_1, \{y_i\}_{i\in[N]}$ from P_1. P_0 **aborts** if $H_{\mathsf{com}}(h_1, y_1 \ldots y_N) \neq \sigma_1$.

III. **Shuffling Argument (3rd Round):**
   1. P_0 computes $\{u_i = y_{\pi_0(i)}^{k_0}\}_{i\in[N]}$ and an adapted shuffling argument for $\mathcal{R}_\pi$: $\chi_0 = \mathsf{AoK}\{\pi_0, k_0 \mid u_i = y_{\pi_0(i)}^{k_0}\}$, where $\pi_0 : [N] \mapsto [N]$ is a random permutation.
   2. P_0 sends $\{u_i\}_{i\in[N]}, \chi_0$ to P_1, and receives $\{e_i\}_{i\in[n]}, \chi_1$ from P_1.
   3. P_0 verifies that χ_1 is valid given $\{e_i\}_{i\in[n]}, \{x_i\}_{i\in[n]}, h_1$, and **aborts** otherwise.

IV. **Final Response (4th Round):**
   1. P_0 initializes a set $E' := \{(i, e_i^{1/k_0})\}_{i\in[n]}, A := \{(i, y_i)\}_{i\in[N]}$. For each $i \in [n]$, if $e_i^{1/k_0} = y_{i'}$ for some $i' \in [N]$, P_0 updates $E' = E' \setminus (i, e_i^{1/k_0})$ and $A = A \setminus (i', y_{i'})$. P_0 generates an argument for $\mathcal{R}_{\mathsf{DDH}}$: $\phi_0 = \mathsf{AoK}\{k_0 \mid \forall (i, e_i') \in E',\ g^{k_0} = h_0 \wedge e_i' = e_i^{1/k_0}\}$.
   2. P_0 sends E', ϕ_0 to P_1 and receives U', ϕ_1 from P_1. P_0 **aborts** if ϕ_1 is invalid, or there exists any $(i, \cdot) \in A$ such that $\left(\pi_0^{-1}(i), \cdot\right) \notin U'$.
   3. P_0 initializes a set $T := \varnothing$. For each $(i, u_i') \in U'$, find $\{r_i^{(1)}, \ldots, r_i^{(c_{\max})}\}$ such that $\mathsf{M2P}(r_i^{(j)}) = (u_i')^{1/k_0}$ for any $j \in [c_{\max}]$. Then P_0 updates $T = T \cup t_i$ if there is some $j \in [c_{\max}]$ such that $\Pi^{-1}(r_i^{(j)}) = t_i \,\|\, 0^{\gamma-\ell}$; otherwise, P_0 **aborts**.
   4. P_0 outputs $V \cup T$.

Fig. 6: Our PSU_2 Protocol with Two-Sided Malicious Security

6.2 Efficiency Analysis

The protocol presented in Figure 6 has strict linear communication and computational complexity $O(n+N)$. Similar to the efficiency analysis of our PSU protocol in Section 5, we consider a typical parameter set where $\lambda = 40, \kappa = 128, \ell = 128, \gamma = 2\kappa, q = 2^{255} - 19, c_{\max} = 2$ and assuming $n = N$ for an easier analysis. In this way, each party needs to send less than $(2n + 1 + |\chi| + |\phi|) \cdot 2\kappa$ bits, where $|\chi| = 3n + 9$ (Section 6.4) and $|\phi| = 2$. Therefore, our total communication bandwidth is $32 \cdot 2 \cdot (5n + 12)$ Bytes.

Regarding the computation, each party needs $4n - 2|V \cap W|$ scalar multiplication over $\mathbb{G}$, $2n - |V \cap W|$ calls for $\mathsf{M2P}$ or $\mathsf{M2P}^{-1}$ (which costs less than a fixed-base scalar multiplication [AFQ+14]), and additional computation costs for χ, ϕ which takes $9n - 3|V \cap W|$ multiscalar multiplication in total (shown in Section 6.4 and Section 6.5). Please refer to Section 7 for more detail.

Co-factor Issues. Concretely, when the underlying elliptic curve has a cofactor $h_c > 1$ (i.e., the elliptic curve group decomposes into h_c prime-order subgroups), there can be up to $c_{\max} \cdot h_c$ curve points corresponding to a single encoded item. In particular, for each mapped primer-order subgroup element $g_i \in \mathbb{G}$, there are h_c candidate curve points distributed across the cosets. To recover the real item t_i, we enumerate these curve points, apply $\mathsf{M2P}^{-1}$, and then perform at most $c_{\max}$ applications of Π^{-1}, until we find a unique (with overwhelming probability) t_i satisfying $\mathsf{M2P}(\Pi(t_i \,\|\, 0)) = g_i$. For Curve25519, we have $c_{\max} = 2$ and $h_c = 8$.

On average, this process requires about $h_c/2$ evaluations of $\mathsf{M2P}^{-1}$ and $c_{\max} \cdot h_c/2$ evaluations of Π^{-1} per encoded item. Importantly, these overheads remain marginal compared to the dominant cost of scalar multiplications.

6.3 Security Analysis

Theorem 3. *Asssuming H_{com} and Π instantiate $\mathcal{F}_{\mathsf{RO}}$ (Figure 9) and $\mathcal{F}_{\mathsf{IP}}$ (Figure 10) respectively, the protocol presented in Figure 6 UC-realizes the two-sided PSU functionality $\mathcal{F}_{\mathsf{PSU}_2}$ (Figure 4) in the $\{\mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{IP}}, \mathcal{F}_{\mathsf{AoK}}\}$-hybrid model, against malicious adversaries with $q \geq 2^{\gamma} \cdot (1 - 2^{-\lambda})$, $\gamma \geq \kappa + \max(\ell, \log(nN \cdot c_{\max}))$, and $\ell \geq \lambda + \log(nN)$.*

Proof. Please refer to Appendix E.1 for more detail.

6.4 The Adapted Shuffling Argument

In this section, we present an adapted shuffling argument in Figure 7 to prove

$$\mathsf{AoK}_{\mathcal{R}_\pi}\left\{s \in \mathbb{Z}_p, \pi : [n] \mapsto [n] \mid \forall i \in [n]\, g^s = h \wedge g_i^s = h_{\pi(i)}\right\}$$

for a statement $\{g, h, g_1, \ldots, g_n, h_1, \ldots, h_n \in \mathbb{G}\}$ where $\forall i \in [n]$, $g_i \xleftarrow{\$} \mathbb{G}$. It is built on the argument of shuffling homomorphic ciphertexts from [Gro10, Sect. 4].

Zero-knowledge. The zero-knowledge is almost the same as in [Gro10, Sect. 4]. The simulator $\mathcal{S}$ samples $x_1, \ldots, x_n, v, v' \xleftarrow{\$} \mathbb{Z}_p$, and $c_\pi = \mathsf{com}(0)$, $c_d = \mathsf{com}(0)$, $c_z = \mathsf{com}(0)$. Then $\mathcal{S}$ sets $G_d = \prod_{i=1}^n h_i^{z_i} \cdot \prod_{i=1}^n g_i^{-x_i}$, $g_u = g^v h^{-\Delta}$, and $c_u = g^{v'} \cdot \mathsf{com}(x_1, \ldots, x_n; 0)^{-\Delta} \cdot c_z^{-v} c_d^{-\Delta}$. Finally, $\mathcal{S}$ simulates ψ following the shuffling argument of known content.

Soundness. Similar to [Gro10, Sect. 4], the extractor $\mathcal{E}$ can extract some permutation π' from ψ, and can extract s and $sr_z + r_d$ from $\{g, h, g_u, v, \Delta\}$ and $\{g, c_z^v c_d^\Delta / (c_z^u \mathsf{com}(\boldsymbol{x})), c_u, v', \Delta\}$ as they

PARAMETER: Statement $\{g, g_1, \ldots, g_n \xleftarrow{\$} \mathbb{G}, h, h_1, \ldots, h_n \in \mathbb{G}\}$ and witness $\{s \in \mathbb{Z}_p, \pi : [n] \mapsto [n]\}$, such that $g^s = h \wedge \forall\, i \in [n],\ g_i^s = h_{\pi(i)}$. The (batched) Pedersen commitment $\mathsf{com}(m_1, \ldots, m_n; r) = g^r \prod_{i=1}^n f_i^{m_i}$ with uniformly random $f_i \xleftarrow{\$} \mathbb{G}$. A prover $\mathcal{P}$ and a verifier $\mathcal{V}$.

PROTOCOL:

1. $\mathcal{P}$ samples $d_1, \ldots, d_n, r_d, r_\pi, r_z \xleftarrow{\$} \mathbb{Z}_p$, computes $c_d = \mathsf{com}(d_1, \ldots, d_n; r_d)$, $c_\pi = \mathsf{com}(\pi(1), \ldots, \pi(n); r_\pi)$, and $G_d = \prod_{i=1}^n g_i^{d_i}$.
2. $\mathcal{P}$ sends c_π, c_d, G_d to $\mathcal{V}$.
3. $\mathcal{V}$ sends $z_1, \ldots, z_n \xleftarrow{\$} \mathbb{Z}_p$ to $\mathcal{P}$.
4. $\mathcal{P}$ computes $x_i = s \cdot z_{\pi(i)} + d_i$. $c_z = \mathsf{com}(z_{\pi(1)}, \ldots, z_{\pi(n)}; r_z)$, and samples $u, u' \xleftarrow{\$} \mathbb{Z}_p$.
5. $\mathcal{P}$ sends $x_1, \ldots, x_n, c_z, g_u = g^u, c_u = c_z^{-u} \cdot g^{u'}$ to $\mathcal{V}$.
6. $\mathcal{V}$ sends $\Delta \xleftarrow{\$} \{0,1\}^\gamma$ to $\mathcal{P}$.
7. $\mathcal{P}$ sends $v = \Delta \cdot s + u, v' = \Delta \cdot (s r_z + r_d) + u'$, and an argument (from [Gro10, Sec. 3]):
$$\psi = \mathsf{AoK}\{\pi, r_\rho \mid c_z c_\pi^\Delta = \mathsf{com}(z_{\pi(1)} + \Delta\pi(1), \ldots, z_{\pi(n)} + \Delta\pi(n); r_\rho)\}.$$
8. $\mathcal{V}$ checks if ψ is valid, and the following equations hold
$$\begin{cases} g^{v'} = \dfrac{c_z^v\, c_d^\Delta}{\mathsf{com}(x_1, \ldots, x_n; 0)^\Delta} \cdot c_u, \\ g^v = h^\Delta \cdot g_u, \\ \prod_{i=1}^n g_i^{x_i} = G_d \cdot \prod_{i=1}^n h_i^{z_i}. \end{cases}$$

Fig. 7: Our Adapted Shuffling Argument.

are AoK for DDH relations. Given the argument is accepted and $g_i \xleftarrow{\$} \mathbb{G}$, assuming $g_i = g^{r_i}$ with $r_i \xleftarrow{\$} \mathbb{Z}_p^*$, we have
$$g^{\sum_{i=1}^n r_i d_i} g^{\sum_{i=1}^n s \cdot r_i \cdot z_{\pi(i)}} = g^{\sum_{i=1}^n r_i x_i}$$
and $x_i = s z_{\pi'(i)} + d_i$. By rearranging the terms, there is
$$\sum_{i=1}^n s r_i (z_{\pi'(i)} - z_{\pi(i)}) = 0.$$
According to the Schwartz-Zippel lemma (Definition 3), except with probability $1/p$, this equation equals zero only when $\pi' = \pi$.

Efficiency of the Argument. This is a 5-move interactive public-coin protocol, so we can compress it into a NIZK in the random oracle model by using Fiat-Shami transformation which is more practical. On the other hand, we can also get a NIZK in UC model by following the standard approaches as mentioned in Section 3.3.

The commitments c_d, c_π and part of the computation of ψ for $\mathcal{P}$ can be done at the offline stage (without knowing the statement). Therefore, at the online stage, $\mathcal{P}$ needs $3n$ multiscalar multiplications over $\mathbb{G}$; by applying batch verification [Gro10, Sect. 6], $\mathcal{V}$ can verify all of 3 commitments (where two of them are from ψ) in a single batch, therefore $\mathcal{V}$ needs $3n + 14$ multiscalar multiplications over $\mathbb{G}$. The proof size is $(3n + 9) \cdot 2\kappa$ bits in total.

6.5 The Batched Argument for DDH Relation

For completeness, we present a batched version of AoK for DDH relations [CP93] in Figure 8 to prove

$$\mathsf{AoK}_{\mathcal{R}_{\mathsf{DDH}}}\{s \in \mathbb{Z}_p \mid \forall\, i \in [n]\, g_i^s = h_i\}$$

for a statement $\{g_1, \ldots, g_n, h_1, \ldots, h_n \in \mathbb{G}\}$.

PARAMETER: Statement $\{g_1, \ldots, g_n \in \mathbb{G}, h_1, \ldots, h_n \in \mathbb{G}\}$ and witness $s \in \mathbb{Z}_p$, such that $\forall\, i \in [n]$, $g_i^s = h_i$. A prover $\mathcal{P}$ and a verifier $\mathcal{V}$.

PROTOCOL:

1. $\mathcal{V}$ sends $a_1, \ldots, a_n \xleftarrow{\$} \mathbb{Z}_p$ to $\mathcal{P}$.
2. $\mathcal{P}$ samples $r \xleftarrow{\$} \mathbb{Z}_p$, computes $R = \prod_{i=1}^n g_i^{a_i \cdot r}$, and sends R to $\mathcal{V}$.
3. $\mathcal{V}$ sends $\Delta \in \{0,1\}^\gamma$ to $\mathcal{P}$.
4. $\mathcal{P}$ sends $z = \Delta \cdot s + r$ to $\mathcal{V}$.
5. $\mathcal{V}$ checks whether $R \cdot \prod_{i=1}^n h_i^{a_i \cdot \Delta} = \prod_{i=1}^n g_i^{a_i \cdot z}$

Fig. 8: Batched Argument for DDH Relation.

Zero-knowledge. The perfect zero-knowledge is straightforward by simulating the transcript as $z \xleftarrow{\$} \mathbb{Z}_p, \Delta \xleftarrow{\$} \{0,1\}^\gamma$ and $R := \prod_{i=1}^n h_i^{-a_i \cdot \Delta} g_i^{a_i \cdot z}$.

Soundness. The extractor $\mathcal{E}$ can extract the witness in a standard way $s' = \frac{z - z'}{\Delta - \Delta'}$. Assuming $h_i = g_i^{s_i}$ and $g_i = g^{x_i}$, $\mathcal{V}$ accepts two transcripts with the same R implies $\sum_{i=1}^n x_i \cdot a_i (\frac{z - z'}{\Delta - \Delta'} - s_i) = 0$. According to the Schwartz-Zippel lemma (Definition 3), except with probability $1/p$, this equation equals zero only when $\forall i \in [n]$, $s_i = s'$ and $x_i \neq 0$.

Efficiency of the Argument. After applying Fiat-Shamir transformation, the proof size is $2 \cdot 2\kappa$ bits as it only contains Δ and z. Regarding the computation, $\mathcal{P}$ needs n multiscalar multiplications, and $\mathcal{V}$ needs $2n$ multiscalar multiplications.

7 Implementation and Performance

We implement both of our protocols and compare them thoroughly with the state-of-the-art semi-honest PSU protocols in this section. We also provide a micro-benchmark for our two-sided protocol in Section 7.1 for better understanding of its efficiency. Our implementation will be made publicly available [8].

Environment. We ran all experiments on an Ubuntu server equipped with a 8-core (4th Gen) SapphireRapids Intel Xeon "Golden Cove" processor at 2.0 GHz and 24 GB of RAM. The system had AVX2 and AVX-512 (SIMD) instructions enabled, with AES-NI support, and all tests were performed single-threaded. To evaluate the performance of our protocol, we test it under different network conditions by configuring varying latencies and bandwidths using the Linux `tc` command.

[8] https://github.com/sihangpu/Malicious-Private-Set-Union

Table 2: Running time and communication cost of our malicious PSU protocols, compared to the state-of-the-art semi-honest and one-sided malicious protocols. The last line describes the blowup factor of our malicious protocols compared to the `baseline` protocols: [ZCL+23] for the online time and [CZZ+24] for the total time/communication.

Set Size	Protocol	Runtime (ms): Online Time (left); Total Time (right)								Comm. (KB)
		LAN (10 Gbps)		1 Gbps		500 Mbps		100 Mbps		
2^{14}	[ZCL+23] SH	**1,427**	59,766	**1,290**	75,355	**1,428**	76,322	**1,537**	79,576	6,974
	[CZZ+24] SH	1,667	**1,667**	1,886	**1,887**	1,963	**1,963**	2,019	**2,019**	**1,657**
	[GHLL22] 1M	4,750	10,867	5,294	11,888	5,469	12,120	5,829	13,063	3,876
	Ours 1M	2,384	2,384	2,403	2,403	2,455	2,455	2,565	2,565	1,934
	Ours 2M	2,828	2,941	3,023	3,140	3,156	3,269	3,500	3,622	5,120
2^{16}	[ZCL+23] SH	**4,670**	190,179	**5,172**	242,411	**5,419**	249,435	**6,312**	250,393	27,807
	[CZZ+24] SH	6,735	**6,736**	7,607	**7,607**	7,973	**7,973**	8,365	**8,366**	**6,610**
	[GHLL22] 1M	18,353	24,470	20,646	27,426	21,657	28,336	22,923	29,974	15,507
	Ours 1M	8,850	8,851	9,149	9,150	9,349	9,350	10,278	10,279	7,750
	Ours 2M	11,303	11,750	12,383	13,019	13,548	14,076	14,205	14,717	20,480
2^{20}	[ZCL+23] SH	**76,170**	2,452,996	**86,260**	3,317,616	**88,546**	3,393,033	**108,863**	3,744,297	435,977
	[CZZ+24] SH	108,682	**108,706**	114,681	**114,710**	126,332	**126,362**	141,871	**141,913**	**105,682**
	[GHLL22] 1M	310,682	316,862	352,524	359,202	355,151	361,689	378,264	385,392	233,641
	Ours 1M	137,200	137,224	146,158	146,200	149,349	149,382	165,975	166,023	124,934
	Ours 2M	184,421	192,576	200,508	209,485	207,202	216,115	230,034	239,125	327,680
Blowup	Ours 1M	1.89×	1.26×	1.73×	1.18×	1.68×	1.18×	1.52×	1.16×	1.18×
	Ours 2M	2.42×	1.77×	2.32×	1.82×	2.34×	1.71×	2.11×	1.68×	3.10×

To simulate a LAN network, we set the bandwidth to 10Gbps with a round-trip latency of 1ms, and for a WAN network, we set the bandwidth ranging from 100Mbps to 1Gbps with 80ms latency.

Parameters. Unless particularly specified, we use statistical parameter $\lambda = 40$, security parameter $\kappa = 128$, and item length $\ell = 128$ as common choices.

Implementation. Our implementation is written in Rust with around 2000 lines of code, where we use the library `curve25519-dalek` to implement the prime order group $\mathbb{G}$ (in Edwards form), and follow the explicit formula on `https://elligator.org/formulas` to implement the compatible Elligator 2 as M2P. Note that due to Elligator 2 maps field elements to Montgomery points, we additionally rely on the conversion between Montgomery and Edwards point from `curve25519-dalek`. We implement the ideal permutation by applying Luby-Rackoff [LR88] over AES-128 with a fixed key. We use collision resistant hash function Blake2 to instantiate the random oracle (in our protocols and corresponding zero-knowledge AoK with Fiat-Shamir transformation). For our one-sided protocol, we additionally use maliciously secure OT extensions following Keller-Orsini-Scholl [KOS15] from `ocelot`. [9]

[9] `https://galoisinc.github.io/swanky/ocelot`

Table 3: Micro-benchmark results (in milliseconds, ms) for our two-sided PSU protocol with sets of size 2^{16} in the LAN network environment. The last row reports the percentage of the total runtime spent on each operation.

Offline	**1st & 2nd Rounds**		**Shuffling Argument**		**Final Response**			**Point Compression**	**Total**
	M2P	Blind (x_i)	Shuffle (u_i)	AoK (χ_0)	Unblind (E', u'_i)	AoK (ϕ_0)	$\mathsf{M2P}^{-1}$		
447	602	1,212	1,212	864	2,424	432	2,444	1,611	11,750
3.8%	5.1%	10.3%	10.3%	7.3%	20.6%	3.6%	20.8%	13.7%	

7.1 Concrete Performance of Our Protocols

We present the comparison table in Table 2 and report a micro-benchmark of our two-sided protocol in Table 3. For the `baselines` of semi-honest one-sided protocols, we choose the SKE variant from [ZCL+23] and the Curve25519 variant from [CZZ+24], as they achieve the lowest online time and the lowest total time, respectively (see Table 1). For malicious security, the protocol of [GHLL22] (we choose their fastest variant M4) is the only implementable PSU construction against malicious adversaries available in the literature, which also adopts the same relaxed functionality as our one-sided protocol in Section 5.

Comparison with `baselines`. Our malicious one-sided protocol (`1M`) is only $1.52-1.89\times$ slower in online time than the fastest semi-honest protocol [ZCL+23], while requiring just $1.18\times$ more communication than the most compact one [CZZ+24]. In terms of total runtime, it is only $\mathbf{1.16-1.26}\times$ slower. Similarly, our two-sided protocol (`2M`) is only $2.11-2.42\times$ slower in the online phase than [ZCL+23], with $3.10\times$ more communication than [CZZ+24], and incurs just $\mathbf{1.68-1.77}\times$ slowdown in total runtime.

Comparison with [GHLL22]. Relative to the only existing sender-malicious PSU protocol [GHLL22], our one-sided construction is $\mathbf{2.31-5.09}\times$ faster overall and yields 1.87–$2.00\times$ less communication.

Benchmark. As shown in Table 3, our two-sided protocol spends 41.2% of its runtime in shuffled Diffie-Hellman (blinding/unblinding group elements), 25.9% in direct and inverse mappings between group elements and real items, and only 10.9% in zero-knowledge arguments. Performance can be further improved by about 15% if we forgo point compression, at the cost of higher communication.

Acknowledgments.

The second and third authors were partially supported by NSF award #2451972, and ARPA-H SP4701-23-C-0074.

Sihang Pu gratefully acknowledges partial support from the ERC under grant OBELiSC (101115790).

References

AFQ+14. Diego F. Aranha, Pierre-Alain Fouque, Chen Qian, Mehdi Tibouchi, and Jean-Christophe Zapalowicz. Binary elligator squared. In Antoine Joux and Amr M. Youssef, editors, *SAC 2014*, volume 8781 of *LNCS*, pages 20–37. Springer, Cham, August 2014.

AHH21. Michel Abdalla, Björn Haase, and Julia Hesse. Security analysis of CPace. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 711–741. Springer, Cham, December 2021.

BB04. Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 56–73. Springer, Berlin, Heidelberg, May 2004.

BHKL13. Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 967–980. ACM Press, November 2013.

BKM+20. Prasad Buddhavarapu, Andrew Knox, Payman Mohassel, Shubho Sengupta, Erik Taubeneck, and Vlad Vlaskin. Private matching for compute. Cryptology ePrint Archive, Paper 2020/599, 2020. https://eprint.iacr.org/2020/599.

BPSY23. Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. Near-optimal oblivious key-value stores for efficient PSI, PSU and volume-hiding multi-maps. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 301–318. USENIX Association, August 2023.

BS05. Justin Brickell and Vitaly Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 236–252. Springer, Berlin, Heidelberg, December 2005.

BSMD10. Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. SEPIA: Privacy-Preserving aggregation of Multi-Domain network events and statistics. In *19th USENIX Security Symposium (USENIX Security 10)*, Washington, DC, August 2010. USENIX Association.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

CDGS19. Jan Camenisch, Angelo De Caro, Esha Ghosh, and Alessandro Sorniotti. Oblivious PRF on committed vector inputs and application to deduplication of encrypted data. In Ian Goldberg and Tyler Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 337–356. Springer, Cham, February 2019.

CF01. Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Berlin, Heidelberg, August 2001.

Cha82. David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982.

CHL22. Sílvia Casacuberta, Julia Hesse, and Anja Lehmann. SoK: Oblivious pseudorandom functions. In *2022 IEEE European Symposium on Security and Privacy*, pages 625–646. IEEE Computer Society Press, June 2022.

CL24. Yi-Hsiu Chen and Yehuda Lindell. Optimizing and implementing fischlin's transform for UC-secure zero-knowledge. Cryptology ePrint Archive, Paper 2024/526, 2024.

CLOS02. Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

CP93. David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer, Berlin, Heidelberg, August 1993.

CS03. Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 126–144. Springer, Berlin, Heidelberg, August 2003.

CSSW25. Gowri R Chandran, Thomas Schneider, Maximilian Stillger, and Christian Weinert. Concretely efficient private set union via circuit-based PSI. ACM AsiaCCS 2025, 2025.

CZZ+24. Yu Chen, Min Zhang, Cong Zhang, Minglang Dong, and Weiran Liu. Private set operations from multi-query reverse private membership test. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part II*, volume 14603 of *LNCS*, pages 387–416. Springer, Cham, April 2024.

DCZB24. Minglang Dong, Yu Chen, Cong Zhang, and Yujie Bai. Breaking free: Efficient multi-party private set union without non-collusion assumptions. Cryptology ePrint Archive, Paper 2024/1146, 2024. https://eprint.iacr.org/2024/1146.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Berlin, Heidelberg, August 2012.

DY05. Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 416–431. Springer, Berlin, Heidelberg, January 2005.

Fis05. Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Berlin, Heidelberg, August 2005.

FO97. Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Burton S. Kaliski, Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 16–30. Springer, Berlin, Heidelberg, August 1997.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987.

GHL+22. 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 Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 1487–1504. USENIX Association, August 2022.

GHLL22. Dov Gordon, Carmit Hazay, Phi Hung Le, and Mingyu Liang. More efficient (reusable) private set union. Cryptology ePrint Archive, Paper 2022/713, 2022.

GMR+21. Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh. Private set operations from oblivious switching. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 591–617. Springer, Cham, May 2021.

GNT24. Jiahui Gao, Son Nguyen, and Ni Trieu. Toward a practical multi-party private set union. In *The 24th Privacy Enhancing Technologies Symposium (PoPETs)*, pages 622–635, 2024.

GPR+21. Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 395–425, Virtual Event, August 2021. Springer, Cham.

Gro10. Jens Groth. A verifiable secret shuffle of homomorphic encryptions. *Journal of Cryptology*, 23(4):546–579, October 2010.

HL10. Carmit Hazay and Yehuda Lindell. A note on the relation between the definitions of security for semi-honest and malicious adversaries. Cryptology ePrint Archive, Report 2010/551, 2010.

HLS+16. Kyle Hogan, Noah Luther, Nabil Schear, Emily Shen, David Stott, Sophia Yakoubov, and Arkady Yerukhimovich. Secure multiparty computation for cooperative cyber risk assessment. In *2016 IEEE Cybersecurity Development (SecDev)*, pages 75–76, 2016.

HM04. Dennis Hofheinz and Jörn Müller-Quade. Universally composable commitments using random oracles. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 58–76. Springer, Berlin, Heidelberg, February 2004.

HN12. Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. *Journal of Cryptology*, 25(3):383–433, July 2012.

JKK14. Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 233–253. Springer, Berlin, Heidelberg, December 2014.

JL09. Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 577–594. Springer, Berlin, Heidelberg, March 2009.

JSZ+22. Yanxue Jia, Shifeng Sun, Hong-Sheng Zhou, Jiajun Du, and Dawu Gu. Shuffle-based private set union: Faster and more secure. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 2947–2964. USENIX Association, August 2022.

JSZG24. Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, and Dawu Gu. Scalable private set union, with stronger security. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024.

KC04. M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1026–1037, 2004.

KOS15. Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 724–741. Springer, Berlin, Heidelberg, August 2015.

KRTW19. Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. Scalable private set union from symmetric-key techniques. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part II*, volume 11922 of *LNCS*, pages 636–666. Springer, Cham, December 2019.

LG23. Xiang Liu and Ying Gao. Scalable multi-party private set union from multi-query secret-shared private membership test. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part I*, volume 14438 of *LNCS*, pages 237–271. Springer, Singapore, December 2023.

LR88. Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2), 1988.

Mau05. Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 1–12. Springer, Berlin, Heidelberg, December 2005.

MPR10. Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. A zero-one law for cryptographic complexity with respect to computational UC security. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 595–612. Springer, Berlin, Heidelberg, August 2010.

MPR+20. Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. Two-sided malicious security for private intersection-sum with cardinality. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 3–33. Springer, Cham, August 2020.

MRR20. Ian McQuoid, Mike Rosulek, and Lawrence Roy. Minimal symmetric PAKE and 1-out-of-N OT from programmable-once public functions. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 425–442. ACM Press, November 2020.

MSK02. Shigeo Mitsunari, Ryuichi Sakai, and Masao Kasahara. A new traitor tracing. *IEICE Transactions*, E85-A(2):481–484, February 2002.

NPR99. Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and KDCs. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 327–346. Springer, Berlin, Heidelberg, May 1999.

RR22. Srinivasan Raghuraman and Peter Rindal. Blazing fast PSI from improved OKVS and subfield VOLE. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2505–2517. ACM Press, November 2022.

RT21. Mike Rosulek and Ni Trieu. Compact and malicious private set intersection for small sets. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1166–1181. ACM Press, November 2021.

Sho97. Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Berlin, Heidelberg, May 1997.

TBZ+25. Binbin Tu, Yujie Bai, Cong Zhang, Yang Cao, and Yu Chen. Fast enhanced private set union in the balanced and unbalanced scenarios. *Cryptology ePrint Archive, to appear at USENIX 2025*, 2025.

Vai. Loup Vaillant. The elligator map and inverse map. https://elligator.org/map.

YLS+24. Yaxi Yang, Xiaojian Liang, Xiangfu Song, Ye Dong, Linting Huang, Hongyu Ren, Changyu Dong, and Jianying Zhou. Maliciously secure circuit private set intersection via SPDZ-compatible oblivious PRF. PoPETS 2025, to appear, 2024.

ZCL+23. Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin. Linear private set union from multi-query reverse private membership test. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 337–354. USENIX Association, August 2023.

A More Related Works

Kolesnikov, Rosulek, Trieu, and Wang [KRTW19] proposed an efficient two-party PSU protocol primarily based on symmetric-key techniques. The core building block of their protocol is the reverse private membership test (RPMT), which is a two-party protocol where a receiver holding a set V learns whether a given element w from the sender belongs to V, while the sender learns nothing. In the context of PSU, by invoking RPMT, the receiver obtains a bit $b \in \{0, 1\}$ indicating the membership of $w \in V$. Subsequently, they perform an OT, allowing the receiver to learn the value of w if $b = 1$ and nothing otherwise ($b = 0$).

To further optimize performance, [KRTW19] introduced a bucketing technique. However, Jia et al. [JSZ+22] later identified that this technique introduces information leakage. Specifically, they showed that the bucketing approach based on hashing leaks partial information about the intersection during execution. In particular, the receiver may infer whether certain bins contain elements that belong to the intersection. Furthermore, they demonstrated through experiments that the dummy element solution proposed by [KRTW19] is insufficient to completely eliminate such leakage.

Jia et al. [JSZ+22] proposed a new PSU protocol that relies on the permute-share functionality in combination with an OPRF. In their protocol, the sender inputs a permutation π and learns the receiver's input set in a permuted order, masked by random values, i.e., $\{v_{\pi(j)} \oplus r_{\pi(j)}\}$. Meanwhile, the receiver learns the corresponding mask values in the same permuted order, i.e., $\{r_{\pi(j)}\}$. The sender then sends to the receiver the set of masked values: $\{w_i \oplus v_{\pi(1)} \oplus r_{\pi(1)}, \ldots, w_i \oplus v_{\pi(n)} \oplus r_{\pi(n)}\}$. If any mask value $s_{\pi(j)}$ matches a value obtained from the sender, the receiver learns a bit indicating that the item at $\pi(j)$ is part of the intersection. Consequently, this item can be discarded in the subsequent execution of OT. To further improve efficiency, the authors use a combination of cuckoo hashing and simple hashing.

The cuckoo/simple hashing scheme plays a crucial role in designing private set operations. Garimella, Mohassel, Rosulek, Sadeghian, and Singh [GMR+21] utilized the same OT-based approach and introduced a new building block called the permuted characteristic. This is a two-party protocol where the sender learns a permutation π of their set W, while the receiver, holding a set V, obtains a vector e, where $e_i = 1$ if the sender's $\pi(i)$-th item is in Y. Their protocol is built upon commonly used techniques such as cuckoo/simple hashing, oblivious shuffling, and special-purpose equality tests.

More recently, [CSSW25] proposed an efficient PSU protocol that can be instantiated using any cuckoo/simple hashing-based circuit PSI protocol. In this approach, the circuit PSI protocol is executed over the PRF values of the original input sets, enabling the sender and receiver to obtain shares of the intersection indicator bits. They further introduced a functionality called combine and permute, which allows the receiver to learn the combined bits in a random order. This randomization enables the subsequent execution of OT to reveal the differences between the two sets. The idea of leveraging the shared output of circuit PSI has gained increasing attention and is often referred to as the secret-shared private membership test (SS-PMT) in various PSU works [LG23, GNT24, DCZB24, JSZG24].

Although efficient PSU protocols can be constructed using the cuckoo/simple hashing scheme, their complexity remains super-linear in terms of the input set size. This is due to the need for a permutation step to break the correspondence between hash bins and elements. To address this limitation, [ZCL+23] proposed the first OT-based PSU protocol with linear computation and communication complexity. Their approach leverages the oblivious key-value store (OKVS) [GPR+21] and introduces a multi-query reverse private membership test (mq-RPMT) that combines both public-key and symmetric-key techniques. Unlike hashing-based schemes, their protocol eliminates

the need for explicit hashing, as the order-hiding property of OKVS removes the necessity for computationally expensive permutations.

Building on this idea, Bienstock, Patel, Seo, and Yeo [BPSY23] further optimized PSU performance by designing a more efficient OKVS. More recently, Chen et al. [CZZ+24] developed an efficient PSU protocol based on a Diffie-Hellman (DH)-based shuffle-OPRF technique. Their approach enables permutation to be performed directly over DH-PRF values, allowing the protocol to achieve linear complexity as well.

B Additional Preliminaries

B.1 Universal Composability

The Universal Composability (UC) framework [Can01] provides a strong security guarantee by ensuring that a protocol remains secure even when composed with arbitrary other protocols. In this model, security is defined by comparing a real-world execution of the protocol with an ideal-world execution involving a trusted functionality. An adversary interacts with parties in both settings, and the goal is to construct a simulator that makes these interactions indistinguishable.

In our security proof, we focus on static corruption, where the adversary chooses which parties to corrupt before the execution begins and cannot change this choice dynamically. This simplifies the analysis while still capturing relevant real-world attack scenarios. We prove the security of our protocols within the UC framework, where an environment $\mathcal{Z}$ provides inputs to two parties, P_0 and P_1, respectively.

B.2 Idealized Models

PARAMETERS: A hash function $H : \mathcal{D} \mapsto \mathcal{M}$.

FUNCTIONALITY:

- Initialize a set $L := \varnothing$.
- Upon receiving input queries $(H, \alpha \in \mathcal{D})$:
  - If there exists some β' such that $(\alpha, \beta') \in L$, set $\beta = \beta'$; Otherwise sample $\beta \xleftarrow{\$} \mathcal{M}$ and update $L = L \cup (\alpha, \beta)$.
  - Return β.

Fig. 9: Ideal Functionality $\mathcal{F}_{\mathsf{RO}}$

Random Oracle. We recall the random oracle functionality $\mathcal{F}_{\mathsf{RO}}$ from [HM04] in Figure 9, adapt for our usage, and omit the session id for simplicity.

Ideal Permutation. We recall the ideal permutation functionality $\mathcal{F}_{\mathsf{IP}}$ in Figure 10, which is a weaker model than ideal ciphers.

C Trivial Instantiation of Standard PSU Functionality

Consider the following Trivial Protocol:

PARAMETERS: A permutation $\Pi : \mathcal{M} \mapsto \mathcal{M}$.

FUNCTIONALITY:

- Initialize a set $L := \varnothing$.
- Upon receiving input queries $(\Pi, \alpha \in \mathcal{M})$:
  - If there exists some β' such that $(\alpha, \beta') \in L$, set $\beta = \beta'$; Otherwise sample $\beta \xleftarrow{\$} \mathcal{M} \setminus \{\beta' \mid (\cdot, \beta') \in L\}$ and update $L = L \cup (\alpha, \beta)$.
  - Return β.
- Upon receiving input queries $(\Pi^{-1}, \beta \in \mathcal{M})$:
  - If there exists some α' such that $(\alpha', \beta) \in L$, set $\alpha = \alpha'$; Otherwise sample $\alpha \xleftarrow{\$} \mathcal{M} \setminus \{\alpha' \mid (\alpha', \cdot) \in L\}$ and update $L = L \cup (\alpha, \beta)$.
  - Return α.

Fig. 10: Ideal Functionality $\mathcal{F}_{\mathsf{IP}}$

- Receiver holds a set X, and Sender a set Y.
- Sender sends Y to Receiver.
- Receiver outputs $X \cup Y$.

Obviously, this protocol is insecure against a semi-honest adversary. However, it can UC-realize the PSU functionality (Figure 2 including highlighted code) against malicious adversaries.

Considering the following cases:

- Corrupted Receiver: Simulating a corrupted receiver in an ideal world is simple. Imaging the environment $\mathcal{Z}$ sets inputs Y for the honest sender, and instructs $\mathcal{A}$ to corrupt the receiver. In the ideal world, the simulator $\mathcal{S}$ samples a random set $R = \{r_1, \ldots, r_n\}$ where $r_i \leftarrow_{\$} \{0,1\}^\ell$. $\mathcal{S}$ inputs R to the functionality $\mathcal{F}_{\mathsf{PSU}}$ and receives $R \cup Y$. Given that each r_i is sampled uniformly from $\{0,1\}^\ell$, the probability that some $r_i \in Y$ exists is negligible from the union bound:

$$\Pr[\exists\, r_i \in Y] \leq n \cdot \frac{N}{2^\ell} \tag{1}$$

  when $\ell = \Omega(\log(n) + \log(N) + \lambda)$. After receiving the output from $\mathcal{F}_{\mathsf{PSU}}$, the simulator knows the honest party's input Y. Then it can perfectly simulate the transcripts using Y.
- Corrupted Sender: The simulator observes Sender's message (i.e., also its effective set Y'), and sends Y' to $\mathcal{F}_w$ hich will set the output for Receiver.

D The Enhanced PSU Functionality from JSZG24

The functionality introduced in [JSZG24, Figure 11] is recalled in Figure 3, with modifications highlighted in blue to show the differences from the standard $\mathcal{F}_{\mathsf{PSU}}$ functionality (Figure 2). For clarity, we simplify the description by omitting unnecessary notions (e.g., session id, states of each party).

This functionality can be interpreted as having a "two-sided" output, except that P_1's output is trivial—it is constant and does not depend on the other party's input. However, since the function being evaluated (the union of sets) is *deterministic*, the simulator only needs to simulate the transcripts, as the output is determined solely by both parties' inputs in the semi-honest setting. As a result, $\mathcal{F}_{e\mathsf{PSU}}$ does not offer any meaningful distinction (even for malicious adversaries) from $\mathcal{F}_{\mathsf{PSU}}$ given the following theorem.

Theorem D.1 (Recall from Theorem 1). *The standard functionality $\mathcal{F}_{\mathsf{PSU}}$ defined in Figure 2 is equivalent to the "enhanced" functionality $\mathcal{F}_{e\mathsf{PSU}}$ (recalled in Figure 3) within the UC framework, against both semi-honest and malicious adversaries. That is, any protocol realizing $\mathcal{F}_{\mathsf{PSU}}$ can also realize $\mathcal{F}_{e\mathsf{PSU}}$, and vice versa.*

Proof. We begin by showing how to realize $\mathcal{F}_{e\mathsf{PSU}}$ using $\mathcal{F}_{\mathsf{PSU}}$. In the $\mathcal{F}_{\mathsf{PSU}}$-hybrid model, this is simple by having P_1 also output $\langle\mathsf{Finished}\rangle$ after sending inputs to $\mathcal{F}_{\mathsf{PSU}}$—Remember an interactive Turing machine (ITM) can always write the `subroutine-output` tape of the environment $\mathcal{Z}$ during its execution (i.e., sending output to $\mathcal{Z}$). First, when the sender P_1 is corrupted, it is trivial since in both hybrid and ideal games there are no transcripts, and honest party P_0's output is identical in hybrid and ideal games given the same P_1's input. To see why this $\mathcal{F}_{\mathsf{PSU}}$-hybrid game perfectly emulates $\mathcal{F}_{e\mathsf{PSU}}$ when P_0 is corrupted, let us consider two cases:

- Semi-honest adversaries. In the hybrid world, P_1 always outputs $\langle\mathsf{Finished}\rangle$. The simulator $\mathcal{S}$ proceeds as follows: when $\mathcal{Z}$ gives P_0's input, $\mathcal{S}$ directly forwards it to $\mathcal{F}_{e\mathsf{PSU}}$ (since a semi-honest adversary will *never* halt or delay the input), and then immediately issues the $\langle\mathsf{Response}\rangle$ query to $\mathcal{F}_{e\mathsf{PSU}}$. Because P_1 outputs $\langle\mathsf{Finished}\rangle$ in both worlds, P_0 receives exactly the same result from $\mathcal{F}_{\mathsf{PSU}}$ and $\mathcal{F}_{e\mathsf{PSU}}$, and no transcript is exchanged between the parties. Therefore, $\mathcal{Z}$ cannot distinguish the hybrid from the ideal world.
- Malicious adversaries. This is a more interestig case: here, P_0 could halt and never send its input, while P_1 still outputs $\langle\mathsf{Finished}\rangle$ in the hybrid world. To handle this, we exploit the fact that the malicious receiver is powerful (as discussed in Section 4.2): $\mathcal{S}$ first inputs a random set into $\mathcal{F}_{e\mathsf{PSU}}$ and then immediately sends the $\langle\mathsf{Response}\rangle$ query to $\mathcal{F}_{e\mathsf{PSU}}$, without waiting for $\mathcal{Z}$'s input to P_0. In this way, P_0 actually extracts P_1's input. Since P_1 outputs $\langle\mathsf{Finished}\rangle$ in both worlds and no transcript is generated, we only need to ensure P_0's view is identical. Whenever $\mathcal{Z}$ supplies P_0's input, $\mathcal{S}$ computes the union itself using the extracted set from P_1. This matches P_0's view exactly in both worlds, so $\mathcal{Z}$ has no advantage in distinguishing.

Conversely, to realize $\mathcal{F}_{\mathsf{PSU}}$ in the $\mathcal{F}_{e\mathsf{PSU}}$-hybrid model, it suffices to have P_1 *not* output anything. In other words, P_1 never produces $\langle\mathsf{Finished}\rangle$, which matches $\mathcal{F}_{\mathsf{PSU}}$'s behavior. Again, there is no transcript, so $\mathcal{Z}$ sees the same execution in both the hybrid and ideal worlds.

Thus, we conclude that $\mathcal{F}_{\mathsf{PSU}}$ and $\mathcal{F}_{e\mathsf{PSU}}$ are equivalent. □

E Missing Proofs

E.1 Proof of Theorem 3

Theorem E.2 (Recall from Theorem 3). *Asssuming H_{com} and Π instantiate $\mathcal{F}_{\mathsf{RO}}$ (Figure 9) and $\mathcal{F}_{\mathsf{IP}}$ (Figure 10) respectively, the protocol presented in Figure 6 UC-realizes the two-sided PSU functionality $\mathcal{F}_{\mathsf{PSU}_2}$ (Figure 4) in the $\{\mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{IP}}, \mathcal{F}_{\mathsf{AoK}}\}$-hybrid model, against malicious adversaries with $q \geq 2^{\gamma} \cdot (1 - 2^{-\lambda})$, $\gamma \geq \kappa + \max(\ell, \log(nN \cdot c_{\max}))$, and $\ell \geq \lambda + \log(nN)$.*

Proof. We prove security within the UC framework, where an environment $\mathcal{Z}$ chooses input V, W for two parties P_0 and P_1, respectively. Without loss of generality, we assume P_1 is corrupted since the protocol is symmetric. Our proof follows the typical strategy: The simulator $\mathcal{S}$ first simulates the random oracle $\mathcal{F}_{\mathsf{RO}}$ and idealized permutation $\mathcal{F}_{\mathsf{IP}}$ to extract P_1's effective input; then a functionality $\mathcal{F}_{\mathsf{PSU}_2}$ is added to the layout but its output wire is not attached to the honest party P_0 and it is only used to help simulate transcripts; in the final hybrid where $\mathcal{S}$ can completely simulate the transcripts, let P_0 accept output from $\mathcal{F}_{\mathsf{PSU}_2}$ to finish the ideal world.

$\mathbf{Hyb_0}$: The Real World. This is the real world where P_0 behaves exactly the same as in Figure 6, and P_1 is corrupted by a dummy adversary $\mathcal{A}$ who is completely instructed by the environment $\mathcal{Z}$. If P_0 aborts during the execution, it outputs $\perp$. Queries of $\Pi(x)$ or $\Pi(y)^{-1}$ are made through (Π, x) or (Π^{-1}, y) to an idealized permutation $\mathcal{F}_{\mathsf{IP}}$. Queries of $H_{\mathsf{com}}(x)$ are made through (H_{com}, x) to a random oracle $\mathcal{F}_{\mathsf{RO}}$. Every AoK is computed by querying and receiving output from $\mathcal{F}_{\mathsf{AoK}}$: P_0 sends

$$(\mathcal{R}_\pi, x := (\{y_i\}, \{u_i\}), w := (\pi_0, k_0))$$

to $\mathcal{F}_{\mathsf{AoK}}$ in Step III, and sends

$$(\mathcal{R}_{\mathsf{DDH}}, x := (g, h_0, \{e_i\}, E'), w := k_0)$$

to $\mathcal{F}_{\mathsf{AoK}}$ in Step IV; If P_0 receives $(\mathcal{R}, \cdot, 0)$ from $\mathcal{F}_{\mathsf{AoK}}$, P_0 aborts.

$\mathbf{Hyb_1}$: Dummy Honest Party P_0. This hybrid is identical to the previous one, except that:

1. The honest party P_0 is a dummy party now, and it just passes input (output) to (from) the simulator $\mathcal{S}$.
2. $\mathcal{S}$ runs the entire code of P_0 inside its own machine, accepts instructions from $\mathcal{Z}$ (i.e., messages of P_1), and responds accordingly. If any abort occurs, $\mathcal{S}$ sets the output to $\perp$. In the end, $\mathcal{S}$ sends the final output to P_0.

This hybrid should be perfectly indistinguishable from $\mathbf{Hyb_0}$ to the environment.

$\mathbf{Hyb_2}$: Simulating the (Local) Random Oracles and Idealized Permutation. This hybrid is the same as the previous one, except that $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{RO}}$ and $\mathcal{F}_{\mathsf{IP}}$ itself. Particularly, to simulate $\mathcal{F}_{\mathsf{IP}}$ for Π and Π^{-1}, $\mathcal{S}$ does the following:

1. $\mathcal{S}$ initializes a set $L_\Pi := \varnothing$.
2. Upon receiving $(\Pi, \alpha \in \{0,1\}^\gamma)$:
   - If there is $(\alpha, \beta', \cdot) \in L_\Pi$, set $\beta = \beta'$;
   - Otherwise, keep sampling $r \xleftarrow{\$} \mathbb{Z}_p^*$ until $\beta = \mathsf{M2P}^{-1}(g^r) \neq \perp \wedge (\alpha, \beta, \cdot) \notin L_\Pi$, then updates $L_\Pi = L_\Pi \cup (\alpha, \beta, r)$.
   - Return β.
3. Upon receiving $(\Pi^{-1}, \beta \in \{0,1\}^\gamma)$:
   - If there is $(\alpha', \beta, \cdot) \in L_H$, then sets $\alpha = \alpha'$;
   - Otherwise samples $\alpha \xleftarrow{\$} \{0,1\}^\gamma \setminus \{\alpha' \mid (\alpha', \cdot, \cdot) \in L_\Pi\}$ and updates $L_\Pi = L_\Pi \cup (\alpha, \beta, \perp)$.
   - Return α.

Simulating $\mathcal{F}_{\mathsf{RO}}$ for H_{com} is exactly identical to Figure 9. This hybrid is statistically indistinguishable from $\mathbf{Hyb_1}$. We focus on $\mathcal{F}_{\mathsf{IP}}$ since the behavior of $\mathcal{F}_{\mathsf{RO}}$ remains unchanged. According to Lemma 3, each $\beta = \mathsf{M2P}^{-1}(g^r) \neq \perp$ is uniformly random over $\mathbb{F}_q$. Given $q \approx 2^\gamma$, precisely, $|1 - \frac{q}{2^\gamma}| \leq 2^{-\lambda}$, then the distribution of β over $\{0,1\}^\gamma$ is statistically close to uniformly random. Therefore, the behavior of $\mathcal{F}_{\mathsf{IP}}$ is statistically indistinguishable from the previous hybrid.

$\mathbf{Hyb_3}$: Extracting Witness from $\mathcal{F}_{\mathsf{AoK}}$. This hybrid is identical to the previous one, except $\mathcal{S}$ extracts witness of each AoK by observing inputs from $\mathcal{F}_{\mathsf{AoK}}$.

1. Step III: $\mathcal{S}$ extracts k_1, π_1 from χ_1; $\mathcal{S}$ aborts if $x_{\pi_1(i)}^{k_1} \neq e_i$ for any $i \in [n]$ or $g^{k_1} \neq h_1$.
2. Step IV: $\mathcal{S}$ extracts k_1 from ϕ_1; $\mathcal{S}$ aborts if $u_i' \neq u_i^{1/k_1}$ for any $(i, u_i') \in U'$, or $|U'| \neq |A|$, or $g^{k_1} \neq h_1$.

Obviously, this hybrids is perfectly indistinguishable from $\mathbf{Hyb_2}$.

$\mathbf{Hyb_4}$: Extracting Effective Input from P_1. This hybrid is identical to the previous one, except that $\mathcal{S}$ additionally tries to extract the effective input of P_1 as follows:

1. Upon observing the message σ_1 from P_1 (i.e., instructions from $\mathcal{Z}$), $\mathcal{S}$ starts looking for a record of the form $((h_1, y_1 \dots y_N), \sigma_1)$ in L_{com}.
2. If no such records are found, $\mathcal{S}$ sets $\mathsf{AbortFlag}_\sigma = 1$.
3. Otherwise, $\mathcal{S}$ sets $\mathsf{AbortFlag}_\sigma = 0$, and proceeds for each y_i with $i \in [N]$:
   - $\mathcal{S}$ starts looking for a record of the form $(w_i \,\|\, 0^{\gamma-\ell}, \cdot, r_i) \in L_{\Pi}$ such that $h_1^{r_i} = y_i$.
   - If no such records are found, $\mathcal{S}$ sets $w_i = \perp$.

This hybrid is perfectly indistinguishable from $\mathbf{Hyb_3}$ since the extraction does not affect the behavior and is unobservable by $\mathcal{Z}$.

$\mathbf{Hyb_5}$: Simulating σ_0 for P_0. This hybrid is identical to the previous one, except that $\mathcal{S}$ behaves differently while running the code of P_0 in Step II:

1. $\mathcal{S}$ samples and sends $\sigma_0 \xleftarrow{\$} \{0,1\}^\gamma$ directly.
2. After observing σ_1, if $\mathsf{AbortFlag}_\sigma = 1$, $\mathcal{S}$ sets $x_i \xleftarrow{\$} \mathbb{G}$ for every $i \in [n]$; otherwise, $\mathcal{S}$ computes $\{x_i\}_{i\in[n]}$ normally as in the previous hybrid.
3. $\mathcal{S}$ programs H_{com} such that $\sigma_0 = H_{\mathsf{com}}(g^{k_0}, x_1, \dots, x_n)$ and sends out $g^{k_0}, \{x_i\}_{i\in[n]}$.
4. If $\mathsf{AbortFlag}_\sigma = 1$, or the 2nd round message from P_1 does not equal to the record found in L_{com}, $\mathcal{S}$ aborts.

This hybrid is computationally indistinguishable from $\mathbf{Hyb_4}$ under DDH assumption (Definition 1). First, when $\mathcal{S}$ does not abort in either hybrid, two hybrids are identical and therefore perfectly indistinguishable.

Then, consider the case that $\mathcal{S}$ aborts in both $\mathbf{Hyb_4}$ and $\mathbf{Hyb_5}$. This means $\sigma_1 \neq H_{\mathsf{com}}(h_1, y_1, \dots, y_N)$ and P_1 didn't query H_{com} with such inputs. In this case, two hybrids are computationally indistinguishable by $\mathcal{Z}$ since the only transcripts are $\sigma_0, h_0, \{x_i\}_{i\in[n]}$: σ_0, h_0 are perfectly indistinguishable as they are uniformly random over $\{0,1\}^\gamma \times \mathbb{G}$ in both hybrids, whereas $\{x_i\}_{i\in[n]}$ is uniformly random in $\mathbf{Hyb_5}$ and pseudorandom in $\mathbf{Hyb_4}$ under $\mathcal{D}_{\mathbb{G}}$-DDH assumption (Definition 2). Moreover, according to Lemma 1 and Lemma 2, this assumption is equivalent to standard DDH.

Next, we consider the case that $\mathcal{S}$ aborts in only one of two hybrids. If $\mathcal{S}$ only aborts in $\mathbf{Hyb_5}$, then P_1 (and $\mathcal{A}$) produces consistent σ_1 without querying H_{com}, or there is a collision. Assuming $\mathcal{A}$ issues Q_{com} queries to H_{com} in total, finding a preimage or collision happens with probability at most $\frac{Q_{\mathsf{com}}+Q_{\mathsf{com}}^2}{2^\gamma}$. After normalization (by dividing the adversary's running time $\mathcal{T} > Q_{\mathsf{com}}$ to calculate bit-security), the advantage of $\mathcal{Z}$ to distinguish between $\mathbf{Hyb_4}$ and $\mathbf{Hyb_5}$ is $\mathsf{negl}(\kappa)$.

On the other hand, if $\mathcal{S}$ only aborts in $\mathbf{Hyb_4}$, this means $\mathsf{AbortFlag}_\sigma = 0$ and there is no collision for σ_1, but $\mathcal{S}$ still aborts in $\mathbf{Hyb_4}$. This cannot happen according to the behavior of $\mathcal{F}_{\mathsf{RO}}$.

$\mathbf{Hyb_6}$: Adding the Ideal Functionality $\mathcal{F}_{\mathsf{PSU}_2}$. This hybrid is the same as the previous one, except that:

1. An additional ideal functionality $\mathcal{F}_{\mathsf{PSU}_2}$ is added to the layout.
2. Dummy party P_0 forwards its input to *both* $\mathcal{F}_{\mathsf{PSU}_2}$ and $\mathcal{S}$.
3. $\mathcal{S}$ sets $\mathsf{AbortFlag}_w = 1$ if there is $w_i = \perp$ for any $i \in [N]$ (which are extracted since $\mathbf{Hyb_4}$), and sets $\mathsf{AbortFlag}_w = 0$ otherwise.
4. If $\mathsf{AbortFlag}_w = 0$, $\mathcal{S}$ sends $W' := W$ (as P_1's input) to $\mathcal{F}_{\mathsf{PSU}_2}$; otherwise, $\mathcal{S}$ sends $W' := \{w_i' \xleftarrow{\$} \{0,1\}^\ell\}_{i\in[N]}$ to $\mathcal{F}_{\mathsf{PSU}_2}$.

5. After receiving the union Q from $\mathcal{F}_{\mathsf{PSU}_2}$, $\mathcal{S}$ initializes a new set $V' := \varnothing$, and for each $q_i \in Q \backslash W'$, it updates $V' = V' \cup q_i$.
6. If $|V'| < n$ and $\mathsf{AbortFlag}_w = 0$, $\mathcal{S}$ pads V' to size n by selecting random items from W.

This hybrid is perfectly indistinguishable from $\mathbf{Hyb_5}$ since adding $\mathcal{F}_{\mathsf{PSU}_2}$ or recovering V' does not affect the transcripts or final output, and is unobservable by $\mathcal{Z}$.

Regarding the extracted v', we argue that if $\mathsf{AbortFlag}_w = 1$ (thus $w'_i \xleftarrow{\$} \{0,1\}^\ell$), then $w'_i \in V$ with probability $\frac{n}{2^\ell}$ which implies

$$\Pr[\exists\, w'_i \in W' \mid w'_i \in V] \leq \frac{nN}{2^\ell} \leq 2^{-\lambda}.$$

This means $\mathcal{S}$ successfully recovers the honest party P_0's input $V' = V$ with overwhelming probability when $\mathsf{AbortFlag}_w = 1$, and recovers the set difference $V \setminus W$ when $\mathsf{AbortFlag}_w = 0$. In the latter case, $\mathcal{S}$ might need to simulate the intersection by randomly choosing items from W.

$\mathbf{Hyb_7}$: Dummy Party P_0 Stops Forwarding Input to $\mathcal{S}$. This hybrid is the same as the previous one, except that:

1. P_0 does not forward its input to $\mathcal{S}$, but only to $\mathcal{F}_{\mathsf{PSU}_2}$.
2. $\mathcal{S}$ simulates transcripts by using the extracted V' as the input of P_0.

This hybrid is computationally indistinguishable from $\mathbf{Hyb_6}$ for $\mathcal{Z}$. If $\mathsf{AbortFlag}_w = 1$, then $\mathcal{S}$ uses $V' = V$ in this hybrid as P_0's input which is identical to $\mathbf{Hyb_6}$ and two hybrids are perfectly indistinguishable.

We focus on that $\mathsf{AbortFlag}_w = 0$, where each y_i binds to a valid w_i, and $\mathcal{S}$ replaces the intersection in V to be some random items in W. We enumerate the following cases:

- in Step II: Note that $\mathbf{Hyb_7}$ is identical to $\mathbf{Hyb_6}$ when $\mathsf{AbortFlag}_\sigma = 1$ since $\mathcal{S}$ will not use P_0's input (no matter it is V or V') as discussed in $\mathbf{Hyb_5}$. If $\mathsf{AbortFlag}_\sigma = 0$, $\mathcal{S}$ would abort in both $\mathbf{Hyb_7}$ and $\mathbf{Hyb_6}$, or does not abort in either hybrid, since transcripts are computationally indistinguishable: given $\mathsf{AbortFlag}_w = 0$ and V' differs from V with some items in W, $\{x_i\}_{i\in[n]}$ are pseudorandom in both hybrids according to $\mathcal{D}_{\mathbb{G}}$-DDH assumption.
- in Step III: In this case, two hybrids are identical as $\mathcal{S}$ aborts or not in Step III is independent of V', and transcripts $\{u_i\}_{i\in[N]}$ are perfectly indistinguishable.
- in Step IV.2: We argue $\mathcal{S}$ must abort in both $\mathbf{Hyb_7}$ and $\mathbf{Hyb_6}$, or does not abort in either hybrid. First, $e_i = x_{\pi_1(i)}^{k_1}$ implies $|A| = |W \setminus V|$ which is identical in both hybrids as $V \setminus W = V' \setminus W$; then, E' is identical in both hybrids as it is consisted of $\mathsf{M2P}\left(\Pi(v_i)\right)^{k_1}$ for $v_i \in V \setminus W$. Therefore, transcripts are perfectly indistinguishable for $\mathcal{Z}$ in two hybrids.
- in Step IV.3: Given $\mathsf{AbortFlag}_w = 0$, each u'_i binds to a valid w_i, therefore $\mathcal{S}$ won't abort in either hybrid, and the final output is identical in $\mathbf{Hyb_7}$ and $\mathbf{Hyb_6}$, except with probability $\frac{(c_{\max}-1)\cdot Q_{\Pi,\mathsf{M2P}}}{2^{\gamma-\ell}}$ (when more than one $r_i^{(j\in[c_{\max}])}$ maps to valid items) where $Q_{\Pi,\mathsf{M2P}}$ is the number of queries to $\mathsf{M2P}$ and Π. After normalization (with $\mathcal{T} > Q_{\Pi,\mathsf{M2P}}$), the adversary's computational advantage is $\mathsf{negl}(\kappa)$.

$\mathbf{Hyb_8}$: Dummy Party P_0 Obtains Output from $\mathcal{F}_{\mathsf{PSU}_2}$. This hybrid is the same as the previous one, except that:

1. $\mathcal{S}$ no longer forwards output to P_0 who gets output from $\mathcal{F}_{\mathsf{PSU}_2}$ directly.
2. If any abort occurs, $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathsf{PSU}_2}$.

3. in Step IV.4, $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathsf{PSU}_2}$ if $\mathsf{AbortFlag}_w = 1$, and sends $\mathsf{deliver}$ otherwise.

This hybrid is exactly the ideal world and computationally indistinguishable from $\mathbf{Hyb_7}$ for $\mathcal{Z}$. Transcripts are not affected at all, thus we only need to argue the honest party's output is identical in both hybrids. Let us focus on the case that $\mathsf{AbortFlag}_w = 1$ as this hybrid is identical to $\mathbf{Hyb_7}$ when $\mathsf{AbortFlag}_w = 0$. Moreover, given transcripts are identical to $\mathbf{Hyb_7}$, $\mathcal{S}$ would behave identically in both hybrids before Step IV.3.

We argue that $\mathcal{S}$ must abort in Step IV.3 if it does not abort earlier in $\mathbf{Hyb_7}$. $\mathsf{AbortFlag}_w = 1$ means there exists at least one y_* such that $\{\beta_1 \dots \beta_{c_{\max}} \mid \mathsf{M2P}(\beta_{i_* \in [c_{\max}]}) = y_*^{1/k_1}\}$ are not obtained by querying Π with valid format. Since $\mathcal{S}$ does not abort earlier, there is

$$e_i^{1/k_0} = \mathsf{M2P}\left(\Pi(v_j \,\|\, 0^{\gamma-\ell})\right)^{k_1}$$

for every $i \in [n], j = \pi_1(i)$. Then the probability that existing $i \in [n]$ such that $y_* = e_i^{1/k_0}$ is at most $\frac{n}{p} \leq 2^{-\lambda}$. This means $y_* \in A$ (in Step IV.1) with overwhelming probability and there's a corresponding $u'_* = u_*^{1/k_1} = y_*^{k_0/k_1}$. Considering the preimages of $(u'_*)^{1/k_0} = y_*^{1/k_1}$, that is, $\beta_1 \dots \beta_{c_{\max}}$. They mapped to valid items only with probability $\frac{c_{\max}}{2^{\gamma-\ell}} = \mathsf{negl}(\kappa)$ (after normalization by $\mathcal{T}$). Therefore, $\mathcal{S}$ would abort in $\mathbf{Hyb_7}$ with overwhelming probability.

□

F PSU with Malicious Sender Realizing the Standard One-Sided Functionality

This protocol is a simplified version of Figure 6 and it realizes the standard one-sided functionality $\mathcal{F}_{\mathsf{PSU}}$ (incorporating the highlighted code from Figure 2). Its security naturally follows from the analysis in Section 6.

PARAMETERS:

- Party P_0 holds $\{v_i\}$ and P_1 holds $\{w_i\}$ as before.
- $\mathsf{M2P} : \mathbb{F}_q \mapsto \mathbb{G}$ with $c_{\max}$ preimages, a hash function $H_{\mathsf{com}} : \{0,1\}^\star \mapsto \{0,1\}^\gamma$, and an ideal permutation $\Pi : \{0,1\}^\gamma \mapsto \{0,1\}^\gamma$.
- AoK ideal functionalities for relations $\mathcal{R}_{\mathsf{DDH}}$ (Section 6.5)

PROTOCOL:

I. **Offline Setup:**
   1. P_0 and P_1 samples random private exponents $k_0 \xleftarrow{\$} \mathbb{Z}_p^*$ and $k_1 \xleftarrow{\$} \mathbb{Z}_p^*$, respectively.

II. **Exchanging Committed Inputs (1st Round):**
   1. For each $i \in [n]$, P_0 pads v_i with $\gamma - \ell$ zeros and computes $x_i = \mathsf{M2P}\left(\Pi(v_i \,\|\, 0^{\gamma-\ell})\right)^{k_0}$. P_1 does the same to have y_j for each $w_j \in W$.
   2. P_0 sends the set $\{x_i\}_{i\in[n]}$ to P_1, while P_1 sends $h := g^{k_1}$ and $\{y_j\}_{j\in[N]}$ to P_0.

III. **Shuffling and Response (2nd Round):**
   1. P_0 computes $\{u_j = y_{\pi(j)}^{k_0}\}_{j\in[N]}$ and sends $\{u_j\}_{j\in[N]}$ to P_1, where $\pi : [N] \mapsto [N]$ is a random permutation.
   2. P_1 initializes a set $U' := \{(i, u_i^{1/k_1})\}_{j\in[N]}$. For each $j \in [N]$, if $u_j^{1/k_1} = x_i$ for some $i \in [n]$, P_0 updates $U' = U' \setminus (j, u_j^{1/k_1})$. P_1 generates an argument for $\mathcal{R}_{\mathsf{DDH}}$: $\phi = \mathsf{AoK}\{k_1 \mid \forall\, (j, u'_j) \in U',\, g^{k_1} = h \wedge u'_j = u_j^{1/k_1}\}$, and sends back U', ϕ to P_0.
   3. P_0 initializes a set $T := \varnothing$. For each $(j, u'_j) \in U'$, compute $r_j := \mathsf{M2P}^{-1}\left((u'_j)^{1/k_0}\right)$, and denote $\{r_j^{(1)}, \ldots, r_j^{(c_{\max})}\}$ the preimages mapped to the same point as r_j. Then P_0 updates $T = T \cup t_i$ if there is any $d \in [c_{\max}]$ such that $\Pi^{-1}(r_j^{(d)}) = t_j \,\|\, 0^{\gamma-\ell}$; otherwise, P_0 **aborts**.
   4. P_0 outputs $V \cup T$.

Fig. 11: Our PSU Protocol Realizing the Standard $\mathcal{F}_{\mathsf{PSU}}$