More Efficient (Reusable) Private Set Union

Dov Gordon[1], Carmit Hazay[2], Phi Hung Le[1], and Mingyu Liang[1]

[1] George Mason University
{gordon,ple13,mliang5}@gmu.edu
[2] Bar-Ilan University
carmit.hazay@biu.ac.il

Abstract. We study the problem of private set union in the two-party setting, providing several new constructions. We consider the case where one party is designated to receive output. In the semi-honest setting, we provide a four-round protocol and two-round protocol, each with two variants. Our four-round protocol focusing on runtime out-performs the state-of-the-art in runtime for the majority of the medium bandwidth settings (100Mbps) and the large set size ($\geq 2^{20}$) settings, with a runtime that is 1.04X-1.25X faster than existing protocols. Our other four-round variant focusing on communication outperforms the state-of-the-art in communication by up to a factor of 1.43 when the set size $\geq 2^{18}$. On the other hand, our two-round protocol is only slightly more expensive but has the property that the receiver message can be re-used across multiple executions. All our semi-honest protocols are post-quantum secure if post-quantum primitives (OKVS and OT) are used.

In the setting where the sender is malicious, we provide the first protocols that avoid using expensive zero-knowledge proofs. Our four-round protocol costs less than double in both communication and computation compared to all other *semi-honest constructions*, showcasing great efficiency. As in the semi-honest setting, we also give a more expensive protocol that allows the receiver message to be re-used.

Our work draws on several techniques from the literature on private set intersection and helps clarify how these techniques generalize (and don't generalize) to the problem of private set union.

Keywords: Private Set Union, Secure Two-Party Computation, Reusability

1 Introduction

As the field of secure computation transitions to practice, much research has focused on building custom protocols for particular computations of interest. These custom protocols improve the cost of communication, computation, and round complexity of the generic solutions. One particular computation that has received more attention than any other is private set intersection (PSI) [33,12,26,24,27,17,25,31,44,11,10,40,23,14,45,43,34,49,42,37]. This is partly because the computation has such broad application, and partly because it is so amenable to custom techniques.

$$\mathcal{F}_{\mathbf{PSU}}^{\hat{n}_1, n_2}$$

Parameters: $\mathcal{S}$'s set size upper bound $\hat{n}_1$ and $\mathcal{R}$'s set size n_2.

Functionality:

1. Wait for $X = \{x_1, \cdots, x_{n_1}\}$ from $\mathcal{S}$, abort if $n_1 > \hat{n}_1$.
2. Wait for $Y = \{y_1, \cdots, y_{n_2}\}$ from $\mathcal{R}$.
3. Send $X \cup Y$ to $\mathcal{R}$.
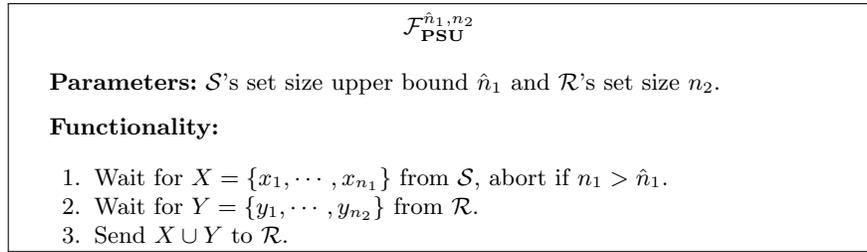
Fig. 1. Ideal Functionality for Two-party PSU. (Upper bound for sender.)

Much more recently, a focus has developed on the neighboring problem of private set union (PSU). In the PSU problem, two parties compute the union of their two sets without revealing the intersecting elements. As a natural next step from PSI, PSU and its variants already have various applications, such as cyber risk assessment and management [38,28], joint graph computation [7], distributed network monitoring [33], private DB full join [36], private ID [20], SQL queries over multiple sources [4,50] and others.

1.1   Defining PSU

In our work, we assume that only one party, the *receiver* $\mathcal{R}$, receives output. While we consider a malicious sender, we assume that the receiver is semi-honest, for reasons discussed below. Properly defining the functionality for PSU in the malicious setting is somewhat subtle, and deserves a short discussion. The functionality that we realize is described in Figure 1, and there are several points of note.

Sender set size. We have parameterized the functionality with an upper bound of $\hat{n}_1$ on the sender set size. If the sender were to use a smaller input set, the functionality would allow the computation to proceed. In the extreme case, if the sender were to submit the empty set, note that the output of the receiver would be its own input, Y. This is the same outcome that would result if the sender were somehow able to fully correlate its own input X with Y (without learning Y). A stronger functionality, which rejects the sender's input if it is not exactly of size $\hat{n}_1$, might be more desirable in many applications. As we will explain in more detail shortly, we allow this relaxation because it admits a much more efficient protocol, through the generalization of a technique used by Freedman et al. [18]. Finding an efficient realization of the functionality that enforces a fixed input size is an interesting direction for future work. We note, however, that there may be applications where we prefer not to leak the exact input set size, or the exact intersection size, and our relaxed ideal functionality could be an asset in such scenarios.

Receiver set size and malicious receivers. In this work we only consider a semi-honest receiver. Reflecting this, our functionality assumes that the size of the receiver's input set is exactly n_2. It is worth noting that when the input domain

is large (i.e. exponential in the security parameter), malicious security for the receiver is in fact *easier* to achieve than semi-honest security. While this is counter-intuitive, the reason is because the malicious receiver, who is allowed to change its input, can select a random input set, and with very high probability, it will recover the full input of the sender, as the intersection will be empty. It is therefore "secure" to simply send the sender's input in the clear. In the semi-honest setting, this would not be secure, as the corrupted receiver must use its given input, which may intersect with the sender input; a simulator cannot recover the intersection, and thus cannot correctly simulate the sender's message. Following similar reasoning, with large input domains, we *could* prove that our protocols are secure, even against a malicious receiver; we do not bother to do so, because the claim is uninteresting. With small input domains, malicious security is more meaningful. To achieve it, we would need the receiver to prove that it has not included too small of an input set in its input encoding. Doing this without the use of expensive zero knowledge proofs is left as an open problem.

1.2   Our Contribution

We study the problem of two-party private set union in both the semi-honest and malicious settings. In the semi-honest setting, we realize the stronger functionality, which fixes the input sizes of both parties. In the latter case, we provide output to a semi-honest receiver and prove security when the sender is an active adversary that can deviate arbitrarily from the prescribed protocol. In each setting, we present two protocols: a four-round and two-round protocols. The four-round protocols have faster running times due to lower computational costs, while the two-round protocols provide reusability, allowing the first receiver message to be used by multiple senders without repeating the computation. Our protocols leverage some of the most successful techniques of the PSI literature, generalizing as needed in order to provide highly efficient results. In particular, our four-round protocol focusing on runtime out-performs the state-of-the-art in runtime for the majority of the medium bandwidth settings (100Mbps) and the large set size ($\geq 2^{20}$) settings, with a runtime that is 1.04X-1.25X faster than existing protocols. The other variant focusing on communication outperforms the state-of-the-art in communication by up to a factor of 1.43 when the set size $\geq 2^{18}$. Finally, as we use ring learning with errors (RLWE) based encryption scheme, all our semi-honest protocols are post-quantum secure if the underlying oblivious key-value store (OKVS) and oblivious transfer (OT) schemes are also post-quantum secure.

In the malicious setting, our four-round protocol achieves security against malicious senders at almost no additional cost. We are not aware of any implementations of protocols that are secure against malicious senders, but since our protocols are the first to achieve this notion of malicious security without relying on expensive zero-knowledge proofs, we can confidently assert that they are state-of-the-art. Specifically, our four-round protocol costs less than twice as much in both communication and computation compared to all other *semi-*

honest constructions, showing great efficiency. As in the semi-honest setting, our two-round, malicious secure protocol also has a reusable receiver message.

1.3   Technical Overview

Background. A classic approach to computing PSI, first introduced by Freedman et al. [18], and afterward appearing in many follow-up results, proceeds as follows. The receiver encodes its input set, Y, using a random polynomial P, subject to the constraint that $P(y) = 0$ for every $y \in Y$. The receiver then encrypts the coefficients of this polynomial using a homomorphic encryption scheme and sends the ciphertexts to the sender. The sender can homomorphically evaluate the same polynomial on each of its inputs: letting $d_x = P(x)$ denote the resulting ciphertext, the sender computes $r \cdot d_x + x$, for random group element r, and sends the result back to the receiver. If $P(x) = 0$, the receiver recovers the correct element x from the intersection, whereas if $P(x) \neq 0$, the receiver recovers a random group element that it can safely discard.

Polynomial encodings have been used in many recent PSI protocols, mainly in the construction of *programmable PRFs* [35,44,9,8]. Other recent constructions have moved away from using polynomials to encode the receiver input, using garbled Bloom filters, [15,45,29], or cuckoo hashing [42,46,44]. More recently, Garimella et al. [21] generalized the basic approach described above by defining the notion of an *oblivious key-value store* (OKVS). Rather than using a polynomial to encode the input, the receiver uses a data encoding containing key/value pairs, where the keys are the items in the input set, and the values are uniformly picked from the value space. After receiving the encoded key/value pairs, the sender runs a decoding algorithm for each item x in its input set. This algorithm returns the matching value used in the key/value pairs if x is in the receiver's set. Finally, the parties perform secure comparisons of the values to determine whether it is an intersected input.

Very recently, Zhang et al. [53] have extended the use of OKVS to the PSU setting. To do this, the values that they use in the OKVS are ciphertexts: the receiver selects a single random value w from the plaintext space, and for each input y in its set, it encrypts the value w, and inserts the key/value pair $(y, \mathsf{Enc}(w))$ into the OKVS. The sender decodes using $x \in X$, re-randomizes the ciphertext, and returns the resulting ciphertexts to the receiver. If $x \in Y$, this ciphertext will decrypt to w, revealing nothing beyond that $x \in X \cap Y$. On the other hand, when $x \notin Y$, their instantiation of OKVS ensures that the returned ciphertext is random, and therefore is unlikely to decrypt to w. The semi-honest parties then engage in an oblivious transfer: the sender uses inputs $(x, \perp)$, and the receiver obliviously requests x if and only if the plaintext recovered was not w.

Reusability. In the past, several works [30,1] have focused on building efficient two-party protocols in the non-interactive secure computation (NISC) model, where only a single round of interaction between the two parties is allowed. Such a model captures the scenarios where each round of interaction is costly and/or slow. In particular, the latency of delivering a message is high, or the

synchronization between two parties is extremely hard. In addition, an achievable property in this model is the reusability of the receiver's message. This is particularly useful in settings with multiple senders running the same protocol with a single receiver that computes and broadcasts its message only once.

Semi-honest protocols. While the OKVS abstraction has helped discover more efficient protocols (in both PSI and PSU), we observe that it has also removed one well-known technique from consideration. A simple idea that appears frequently in both the PSI literature and first proposed in [18], and, more recently, in the PSU literature [36], is the use of *bucketing*: rather than executing the protocol on the full inputs sets, the two parties first agree on a hash function, and use it to partition their input sets into buckets. They then compute PSU on each bucket independently, and the receiver takes the union of the results to recover the final output. When using polynomials, the small bucket size of $O(\log n)$ greatly reduces the degree of each polynomial (from n to $\log n$), which has a big impact on the computational cost. However, when using other more successful instantiations of the OKVS, such as 3H-GCT proposed by Garimella et al. [21], and then employed in PSU by Zhang et al. [53], the value of bucketing is erased. We defer a description of the 3H-GCT instantiation to Section 2.2, and simply mention here that with this data structure, there is no advantage for improving concrete computation cost while the need to pad the bucket to its maximum size only hurts performance.

Zhang et al. [53] currently hold the state-of-the-art for PSU in the semi-honest setting, and their improvement is over the work of Kolesnikov et al. [36], which relies on polynomial encodings and bucketing. However, we find that Kolesnikov et al. missed an important improvement that bucketing offers. As described previously, we assign a different polynomial to each bucket, benefiting from the reduced degree. However, when sending the encrypted coefficients to the sender, we use Ring-LWE, which allows for plaintext packing [52]. For simplicity, let m denote both the number of buckets and the packing parameter, and let $a_{i,0}, \ldots, a_{i,k+1}$ denote the coefficients of the polynomial for the ith bucket. We pack the plaintexts vertically across the buckets, placing the jth coefficient from the polynomial of each bucket into the same plaintext: $\mathsf{Pack}(a_{1,j}, \ldots, a_{m,j})$. To perform polynomial evaluation over its inputs, the sender chooses a different value from each of its buckets, $x_1, \ldots, x_m$, and constructs the vector $(\mathsf{Pack}(x_1^0, \ldots, x_m^0), \ldots, \mathsf{Pack}(x_1^{k+1}, \ldots, x_m^{k+1}))$ (with increasing powers of the input elements in each slot). It then computes the inner product of this vector with the vector of encrypted coefficients, yielding an evaluation of the ith polynomial in the ith plaintext slot. The effect is that we save a factor of the packing parameter in both communication and in the computation times of both parties.

In our four-round protocol, the receiver proceeds as described above for the protocol of Zhang et al. That is, after decrypting the ciphertexts, it learns for each index j whether the jth input of the sender is in the intersection. The two parties then perform an oblivious transfer for each index, where the receiver requests the jth input value only if it is not in the intersection. We evaluate

the performance of this protocol in Section 5. We show that it outperforms the state-of-the-art by at least $1.04 \times -1.25\times$ depending on the network bandwidth and input set size.

We made a small modification to reduce this to two rounds and provide re-usability. When encoding its input, the receiver ensures that the roots of the polynomials lay at each of its input values. The sender evaluates the polynomials as before to arrive at packed ciphertexts of the form of $(Q_1(x_1), \ldots, Q_m(x_m))$. It additionally computes the ciphertexts of $(x_1 \cdot Q_1(x_1), \ldots, x_m \cdot Q_m(x_m))$. For $x_j \in X \cap Y$, the jth value in both plaintexts is 0, and nothing is learned about that input by the receiver. When $x_j \notin X \cap Y$, the ratio of the jth values in each plaintext reveals x_j. A similar approach for PSU, without packing or bucketing, was first described by Frikken [19]. In terms of performance, the time required to perform additional plaintext/ciphertext multiplications slightly outweighs the cost of the oblivious transfer, with the total runtime of two-round protocols being 1.22 times higher than that of four-round protocols. As for communication cost, additionally sending $O(n)$ ciphertexts in the two-round protocol incurs a higher cost compared to conducting $O(n)$ oblivious transfers (given a pre-processed silent random OT), which results in a 1.5x blowup of total communication cost.

Security against Malicious Senders. In order to realize an efficient protocol that is secure against a malicious sender, $\mathcal{S}$, we wish to avoid the usage of expensive tools such as zero-knowledge proofs. Therefore, we generalize and apply a simple and efficient technique introduced by Freedman, Nissim, and Pinkas [18] for constructing PSI protocols against malicious senders, and we refer to the generalized technique as the FNP paradigm. The main idea is to de-randomize the computation of the sender, allowing the receiver to verify the correctness of the received messages. For reasons we describe later, we do not know how to leverage this technique on packed values, so we return instead to using the OKVS abstraction. (The original FNP construction used polynomial encodings.)

Roughly speaking, using a homomorphic encryption scheme, for each input $y \in Y$, the receiver, $\mathcal{R}$, generates an encoding using key/value pairs $(y, \mathsf{Enc}(0))$. For each $x \in X$, the sender decodes to recover d_x, samples a random value r, and homomorphically computes $(e, h) \leftarrow (\mathsf{Enc}(r_1 \cdot d_x + r), H_2(r_2, x))$, where $r_1 \| r_2 = H_1(r)$, and H_1, H_2 are hash functions modelled as random oracles. Notice that if $x \in Y$, e decrypts to r. This allows $\mathcal{R}$ to re-evaluate $\mathcal{S}$'s computation, verifying that e, h are correctly computed with r and some $y \in Y$ (the intersected input).

To summarize the high-level intuition of their protocol, we first note that their protocol satisfies a "locality" property: the responses of $\mathcal{S}$ can be decomposed into multiple messages, where each message corresponds solely to one of $\mathcal{S}$'s inputs. Then, to compute each message corresponding to an input x, $\mathcal{S}$ de-randomizes the computation with some "local" randomness r. To enable $\mathcal{R}$ to verify the message is computed correctly, $\mathcal{S}$ allows $\mathcal{R}$ to recover r if x is in the intersection. This can be done in one go as in [18], or, looking ahead to our second protocol, through some membership test, followed by oblivious transfer, performed in separate rounds. Finally, $\mathcal{R}$ can reevaluate the sender's step in the

protocol using x and r. Note that this implicitly requires that it is "secure" for $\mathcal{R}$ to know x, which is the case for intersection elements in PSI.

Given the intuition above, the FNP paradigm can be extended to functions where the output of the receiver is the partial input of the sender, including PSU. Additionally, however, to leverage the FNP paradigm, the protocol must satisfy the "locality" property; for example, it does not seem to fit well with techniques like packing, as we used in our semi-honest protocols. For each of its input values x, $\mathcal{S}$ will decode to recover a ciphertext d_x, such that d_x encrypts 0 if $x \in Y$, and d_x is a random ciphertext otherwise. (This property is realized through the use of the key-value store.) $\mathcal{S}$ computes and sends: $d_x, (d_x \cdot x), (d_x \cdot r)$, where, as above, $H(x||r)$ determines the randomness used in the construction of the three ciphertexts (i.e. re-randomization). When $\mathsf{Dec}(d_x) = 0$, nothing is revealed to $\mathcal{R}$, and in all other cases, $\mathcal{R}$ can extract both x and r to verify the correctness of the computation. This forms the foundation of our 2-round protocol. Additionally, we present a four-round protocol that relies on oblivious transfer and assess the performance of both protocols in Section 5. Notably, our evaluation reveals that the communication cost of our four-round malicious protocol is merely 1.25 times that of the state-of-the-art semi-honest protocol. Furthermore, our computation cost remains under twice that of the state-of-the-art semi-honest protocol.

Here, we want to point out a limitation of the FNP paradigm, which led us to accept the relaxation described previously. Notice that $\mathcal{R}$ can only verify the messages corresponding to its output set, as it is insecure to disclose the randomness used for inputs that are not part of $\mathcal{R}$'s output. For example, in PSI, $\mathcal{R}$ can verify messages corresponding to the intersected inputs. In PSU, $\mathcal{R}$ can verify messages corresponding to elements in the union that are outside the intersection. Thus, our PSU protocols allow a malicious $\mathcal{S}$ to "inflate" the intersection size by sending encryptions of 0; this is captured in the functionality by allowing the input set of the sender to be smaller than intended. In PSI, the equivalent attack is also admissible: $\mathcal{S}$ can send encodings of random values in order to reduce the intersection size. However, when dealing with large input domains, this attack on PSI is equivalent to using random inputs (and thus isn't an attack at all). For PSU, the impact is the one described previously.

On the positive side, the FNP paradigm is extremely simple to implement and does not require any additional heavy machinery such as zero-knowledge proofs. Specifically, it enables the enhancement of semi-honest security of the sender "for free". Note that the security of the receiver can be handled separately. Specifically, in some settings (for instance, when the receiver's input set is much smaller than the sender's input or in a reusable setting where the receiver's work is captured by a one-time effort), the overhead of attaching a zero-knowledge proof to the receiver's message will be amortized away by the overall amount of work in the protocol.

It is worth comparing this to a relaxation that frequently appears in the PSI literature [49,44]: many efficient PSI functionalities allow the receiver to *increase* their input set size up to some bound, usually referred to as the "slackness" bound, possibly inflating the intersection size beyond what the honest behavior

allows.[3] In PSU, this relaxation allows the sender to correlate its input with the receiver's, but without any direct leakage. In PSI it allows the receiver to learn more about the sender's input than was intended, but without this correlation attack.

1.4 Open Questions

There are several directions that warrant further study, and their discussion helps further explain our results.

FNP with packing. As described above, we do not know how to leverage ciphertext packing, which provides considerable performance improvement, together with the FNP paradigm. The problem stems from the fact that the same encryption randomness is used for the entire plaintext: if x_i and x_j are packed together into the plaintext, we cannot reveal the randomness used for $x_i \notin Y$ while hiding the randomness used for $x_j \in Y$. Exploring modifications to the construction, and possibly to the encryption scheme, are interesting directions.

Batched OKVS. Zhang et al. were the first to achieve linear complexity, and they did this through the use of the OKVS abstraction. Although we found that abandoning the abstraction in favor of bucketing provided better performance, ideally, we would still have preferred to unify our presentation by continuing to use the OKVS abstraction. Additionally, such an approach might have helped us improve our construction in the future if new realizations of the OKVS primitive are discovered. Because our semi-honest protocols still use polynomial encodings of the input, it is tempting to view the process as batch encoding and decoding of the OKVS. Unfortunately, formalizing this becomes a bit messy. When batch decoding a set of keys, S_k, you would still recover only a single value in V (in our case, a ciphertext). Since the encoder likely did not batch the same set of inputs during encoding – in fact, very likely, only a subset of S_k was even inserted into the structure – we cannot describe the returned value as either a match or a non-match. The encrypted plaintext contains membership information for each of the values in S_k, as you would expect, but this ties the OKVS primitive to using ciphertext values with packed plaintext. The abstraction becomes sufficiently different, and, perhaps, too constrained, so we decided to abandon it. That said, the value of batch encoding and decoding is real, and some reasonable abstraction might provide new insights, just as the OKVS abstraction has done.

Extending FNP. We hinted above that the FNP paradigm can be extended to a broader class of computations. Intuitively, we can generalize this as follows. Let F_Y denote some arbitrary predicate that depends on $\mathcal{R}$'s input set Y, and let X denote $\mathcal{S}$'s input set. If the output of computation is $\{x \mid x \in X \land F_Y(x) = 1\}$, then we can construct a reusable protocol while leveraging the FNP paradigm to get sender security at almost no cost. The receiver encrypts the

---

[3] This does not stem from the FNP paradigm, but rather the use of OKVS: depending on the instantiation, a malicious server can sometimes encode more input elements without detection.

Table 1. Asymptotic costs of existing semi-honest secure PSU protocols. n is the size of the input sets, κ and λ are the computational and statistical security parameters, and ℓ is a ciphertext packing parameter. "pub" denotes the public-key operation and "sym" denotes the symmetric-key operation.

	Communication	Computation
Frikken [19]	$O(\kappa n)$	$O(n^2)$ pub
Davidson and Cid [13]	$O(\kappa \lambda n)$	$O(\lambda n)$ pub
Kolesnikov et al. [36]	$O(\kappa n \log n)$	$O(n \log n \log \log n)$ sym
Garimella et al. [20]	$O(\kappa n \log n)$	$O(n \log n)$ sym
Jia et al. [32]	$O(\kappa n \log n)$	$O(n \log n)$ sym
Zhang et al. [53]	$\boldsymbol{O(\kappa n)}$	$\boldsymbol{O(n)}$ **pub**
Ours	$\boldsymbol{O(\kappa n)}$	$O(n \log n \log \ell)$ pub

predicate $F_Y(\cdot)$ using an FHE scheme, and sends the ciphertext to the sender. The sender homomorphically evaluates the predicate on each of its inputs to recover ciphertext d_x, which is an encryption of 0 if the output does not satisfy the predicate, and an encryption of 1 if it does. It then proceeds as in our own construction, re-randomizing using $H(x||r)$, and sends $d_x, (d_x \cdot x), (d_x \cdot r)$. While the fully generic protocol is not likely to be efficient, we believe that there are more interesting applications for this paradigm, with more efficient predicate encodings, waiting to be discovered.

Laconic PSU. Recently, there have been several results on *laconic PSI* [2,3]. This is a two-round, reusable PSI protocol in which the receiver message is sublinear in its set size. Such protocols are especially appealing in the setting where a receiver has a large input and must repeatedly compute on that set with multiple senders. We fall short of achieving this, providing reusability but only with an $O(n)$ size receiver message. It is worth noting that if we sacrifice the two-round property, we could achieve succinct communication by asking the receiver to hold onto its encoding, and having the sender query the OKVS structure (or polynomial encoding) obliviously. However, achieving sublinear receiver communication in two rounds remains a very interesting question.

1.5   Related Work

We compare the asymptotic communication and computation cost of our semi-honest protocols with the previous solution in Table 1 and review previous solutions.

In the semi-honest model, Frikken [19] presents a PSU protocol that relies on polynomial representation and additively homomorphic encryption (AHE). In particular, the receiver computes a polynomial $Q(\cdot)$ such that for every y in its input set, $Q(y) = 0$. Then, it sends the encrypted coefficients of $Q(\cdot)$ to the sender. For each input x, the sender homomorphically computes the ciphertext of $Q(x)$ and $x \cdot Q(x)$ and returns them to the receiver. Thus, the receiver can recover all x that are not in the intersection, as $Q(x) \neq 0$. Their protocol requires

communicating $O(n)$ ciphertexts and $O(n^2)$ computation.[4] We describe their solution a bit more formally, and refer to it as the "naive solution," in Section 3.2.

Davidson and Cid [13] propose a PSU protocol that is based on Bloom filters and AHE. Roughly speaking, the receiver encodes its set Y using k hash functions into a bloom filter of λn bits. It inverts the filter by flipping each bit, encrypts the inverted filter using the AHE scheme, and sends it to the sender. For each input x, the sender retrieves the k ciphertexts corresponding to x from the encrypted, inverted filter. Then, it homomorphically computes the sum; let c denote the encrypted sum. The sender homomorphically computes and sends back the ciphertexts c and $x \cdot c$. Note that if $x \in Y$, $c = 0$, therefore, the receiver can only recover $x \notin Y$ from these responses. The protocol requires sending $O(\lambda n)$ ciphertexts and computing $O(\lambda n)$ public key operations, in which λ is the security parameter.

Kolesnikov et al. [36] propose a PSU protocol that relies on polynomials, OPRFs (oblivious pseudorandom functions), and OT (oblivious transfer). The parties first invoke OPRF on their inputs, and use the OPRF's outputs for polynomial interpolation/evaluation. Through comparing its own OPRF outputs with the evaluation result sent by the sender, the receiver learns a bit that indicates whether an input corresponding to its OPRF output lies in the intersection. It then invokes OT to receive the non-intersected input. Overall, it avoids the expensive public key operations required in the previous works. Also, to reduce the cost of interpolation and evaluation, they use bucketing techniques to partition the input set into multiple subsets, and run the PSU protocol on each pair of subsets. This reduces the communication complexity to $O(n \log n)$ and the computational cost to $O(n \log n \log \log n)$.

Garimella et al. [20] propose a PSU protocol that relies on an oblivious switching network to implement a permuted characteristic functionality. In this functionality, the sender receives the permutation used to permute its set and the receiver gets a vector of bits indicating which permuted input is in the intersection. Similar to the constructions above, they then invoke OT, allowing the receiver to recover all non-intersected inputs from the sender. The reliance on the oblivious switching network incurs an $O(n \log n)$ cost for both communication and computation.

Jia et al. [32] proposed a protocol based on secure shuffling, multi-point OPRF, and OT. At a high level, the two parties invoke a secure shuffling protocol to permute and secret share the receiver's input set, e.g., an input y is shared to $s \oplus y$ and s. Next, they call a multi-point OPRF to evaluate over receiver's shares while returning the PRF key to the sender. The sender then computes the xor between every pair of its own inputs and its shares of the receiver's inputs. Note that if a sender's input x equals a receiver's input y, evaluating the xor on x and y's share $s \oplus y$ gives s, which is exactly the share held by the receiver. Then, it

---

[4] The author claims that bucketing technique can reduce the computation cost to $O(n \log \log n)$, but does not give any details on how to modify the protocol, and does not discuss the subtle issue that in PSU, as opposed to PSI, one has to avoid revealing which buckets contain intersecting items.

evaluates the PRF over the xor results and sends the PRF outputs back to the receiver. Similarly to [36], the receiver compares the PRF output with its own OPRF results to learn whether the corresponding input is in the intersection. In the end, the two parties invoke OT to allow the receiver to learn all non-intersected inputs. To improve efficiency, they leverage the cuckoo hash table, which brings the communication and computation costs down to $O(n \log n)$.

Zhang et al. [53] give the most concretely efficient protocols. Their protocols are based on the OKVS abstraction, and the one with better computation efficiency relies on re-randomizable PKE. The receivers encode the pairs of keys and values into an OKVS, where the keys are its inputs and the values are fresh ciphertexts of an arbitrary value w picked by the receiver. Then, the sender decodes each of its OKVS inputs and sends back the (re-randomized) value, which decrypts to w if it is in the intersection. The receiver relies on OT to retrieve all non-intersected inputs from the sender. Their approach gives the best concrete communication and computation costs. Asymptotically, they only require communicating $O(n)$ ciphertexts and computing $O(n)$ public key operations.

We also briefly review previous protocols realizing (two-party) PSU in the malicious model. Frikken [19] also modifies its semi-honest PSU protocol to achieve malicious security using zero-knowledge proofs, which incurs $O(n^2)$ communication cost to prove the correct polynomial interpolation and evaluation. Hazay and Nissim [26] construct a PSU protocol using zero-knowledge proofs to ensure $\mathcal{R}$ correctly constructs the polynomial. They rely on a perfectly hiding commitment scheme and an OPRF evaluation to ensure that $\mathcal{S}$ correctly evaluates the protocol. Their protocol requires communicating $O(n \log n)$ group elements and performs $O(n \log n)$ modular exponentiation. Blanton and Aguiar [5] use generic MPC and incur $O(n \log n)$ communication and computation costs. Seo et al. [51] focus on multiparty set union and require $O(s^3 \cdot n^2)$ communication cost and $O(s^4 \cdot n^2)$ computation cost, where s is the number of parties. They rely on verifiable secret sharing and zero-knowledge proofs to obtain malicious security.

2 Preliminaries

2.1 Notations and Security Definition

Let κ denote the computational security parameter. Let $\mathcal{S}$ (resp. $\mathcal{R}$) denote the sender (resp. receiver) and let $X = \{x_1, \cdots, x_{n_1}\}$ (resp. $Y = \{y_1, \cdots, y_{n_1}\}$) denote the sender's (resp. receiver's) input set. Also, let $[m]$ denote the set $\{1, 2, \cdots, m\}$. Finally, we use the abbreviation PPT to denote probabilistic polynomial-time and refer to a function μ as being negligible in κ if its inverse grows faster than any polynomial in κ. Given two distribution $\mathcal{D}_1$ and $\mathcal{D}_2$, we denote $\mathcal{D}_1 \stackrel{c}{\equiv} \mathcal{D}_2$ to indicate that they are computationally indistinguishable.

We follow [22,39] for the security definitions for secure two-party computations. We further employ (re-randomizable) additively homomorphic encryption schemes. As these are quite standard, we defer them to appendix A.

2.2   Oblivious Key-Value Stores

We use the notion of oblivious key-value stores (OKVS) [42,21]. The definition goes as follows:

Definition 1. *A Key-value store is parameterized by a set $\mathcal{K}$ of keys, a set $\mathcal{V}$ of values, and a set of functions H as well as two algorithms:*

1. $\mathsf{Encode}_H(\{(k_1, v_1), \ldots, (k_n, v_n)\})$ *takes as input key-value pairs $\{(k_1, v_1), \ldots, (k_n, v_n)\} \subseteq \mathcal{K} \times \mathcal{V}$, and outputs an object D or an error indicator $\perp$ with statistically small probability.*
2. $\mathsf{Decode}_H(D, k)$ *takes an object D and a key k as inputs, and outputs a value $v \in \mathcal{V}$.*

In the rest of this paper, we omit the underlying parameter H for simplicity. An OKVS needs to satisfy the following two properties:

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

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

Obliviousness. For all distinct $(k_1^0, \ldots, k_n^0)$ and all distinct $(k_1^1, \ldots, k_n^1)$, if Encode does not output $\perp$ for $(k_1^0, \ldots, k_n^0)$ or $(k_1^1, \ldots, k_n^1)$, then

$$\mathsf{Encode}((k_1^0, v_1^0), \ldots, (k_n^0, v_n^0)) \stackrel{c}{\cong} \mathsf{Encode}((k_1^1, v_1^1), \ldots, (k_n^1, v_n^1))$$

where v_i^b is uniformly sampled from $\mathcal{V}$ for all $i \in [n]$ and $b = \{0, 1\}$.

Furthermore, the following randomness property, which is formally captured in [53], is handy when arguing the correctness of our PSU protocols.[5]

(Optional) Randomness. For any $A \subseteq \mathcal{K} \times \mathcal{V}$ with distinct keys and any k^* that is not among the keys appearing in A, the output of $\mathsf{Decode}(\mathsf{Encode}(A), k^*)$ is statistically close to uniform over $\mathcal{V}$.

Finally, by $(n, m, 2^{-\lambda})$-OKVS scheme, we mean the scheme encodes n elements into m slots, and the encode algorithm fails with no more than $2^{-\lambda}$ probability. In our work, we use the following two constructions of OKVS.

Polynomial. A simple example of OKVS is a polynomial P satisfying $P(k_i) = v_i$ for all $i \in [n]$. The (encrypted) coefficients of the polynomial are the OKVS data structure where $m = n$. It is straightforward to see the correctness property holds while the obliviousness property follows from the CPA security of the encryption scheme. The encode (resp. decode) algorithm is simply polynomial interpolation (resp. evaluation).

3-Hash garbled cuckoo table. 3-Hash Garbled Cuckoo Table (3H-GCT), first introduced in [21], is a more sophisticated scheme to instantiate OKVS. We give a high-level description of its data structure and encode/decode algorithms below, but refer to the original paper for full details.

---

[5] In fact, this property is a bit stronger than what we need, thus some instantiation of OKVS that do not has this property still suffice for correctness. We elaborate on this point after presenting our protocols, when the OKVS we instantiate is clear.

Recall that $\mathcal{K}$ (resp. $\mathcal{V}$) corresponds to the key space (resp. value space) of the OKVS. Specifically, let $\mathcal{V}$ be a finite group or a finite field. Let $m = m' + d + \lambda$ where $d \in O(\log n)$ and λ is the statistical security parameter. Let the data structure $D = L || R \in \mathcal{V}^m$, where $L \in \mathcal{V}^{m'}$ and $R \in \mathcal{V}^{d+\lambda}$.

The encode algorithm is parameterized with the functions $h_1, h_2, h_3 : \mathcal{K} \to [m']$ and a function $r : \mathcal{K} \to \{0,1\}^{d+\lambda}$. Given h_1, h_2, h_3 and a $k \in \mathcal{K}$, let $l(k)$ be the bit vector of length m' where all bits are zero except at positions $h_1(k), h_2(k), h_3(k)$. Then, the data structure D encoding $\{(k_i, v_i)\}_{i \in [n]}$ should satisfy $\langle l(k_i) || r(k_i), L || R \rangle = v_i$ for $i \in [n]$, where $\langle \cdot, \cdot \rangle$ denotes the inner product between two vectors. Note that finding a satisfying D is equivalent to solving the system of n linear equations. In particular, the encode algorithm outputs a random solution for this system of linear equations. On the other hand, decoding a key k can be done by simply retrieving the slots in D that correspond to the non-zero bits in $l(k_i) || r(k_i)$ and summing them up.

The OKVS is correct whenever the encode algorithm succeeds. It may fail when there is no solution, i.e., there exist "too many" linearly dependent rows in the linear equations. To empirically bound this failure probability, [21] explores several architectures to amplify the probabilistic guarantee of encoding success. We again refer to their paper for full details. Here we simply state their parameters. In Section 5.4 in [21], they construct a $(n, m, 2^{-\lambda})$-OKVS scheme with $m \approx 1.3n$ and $\lambda > 40$.

Roughly speaking, obliviousness follows from the fact that the encode algorithm returns a random solution to the linear equations, and values $\{v_i\}_{i \in [n]}$ are uniformly distributed. In Theorem 7 in [53], they formally prove that 3H-GCT satisfies both the obliviousness and the randomness property.

A more recent implementation of OKVS is proposed in [48]. Compared to 3H-GCT, it slightly reduces the communication cost, and thus would lead to slightly improved performance. In this paper, we use the OKVS abstraction to describe our protocol and, for simplicity, we use 3H-GCT to evaluate our performance.

3   Semi-Honest PSU Protocols

In this section, we introduce two semi-honest protocols that realize the PSU functionality (Figure 2). In Section 3.1, we provide a two-round reusable protocol relying on RLWE PKE, whereas in Section 3.2 we provide a variant protocol in an OT-hybrid model that is more efficient, in terms of both communication and computation, at the cost of adding two rounds of interaction. Finally, we implement both protocols and defer the experimental results to Section 5.

3.1   Semi-Honest Reusable PSU

We now present our two-round, reusable PSU protocol, secure against a semi-honest adversary. Informally, we refer to a two-round PSU protocol as reusable if the receiver's message can be reused across multiple executions of the protocol, potentially with different senders. This property is especially appealing when

$$\mathcal{F}_{\mathbf{PSU}}^{n_1,n_2}$$

Parameters: $\mathcal{S}$'s set size n_1 and $\mathcal{R}$'s set size n_2. The universe $\mathcal{U}$ of all elements.

Input: $\mathcal{S}$'s input set $X = \{x_1, \cdots, x_{n_1}\} \subseteq \mathcal{U}$ and $\mathcal{R}$'s input set $Y = \{y_1, \cdots, y_{n_2}\} \subseteq \mathcal{U}$.

PSU:

  − Send $X \cup Y$ to $\mathcal{R}$.
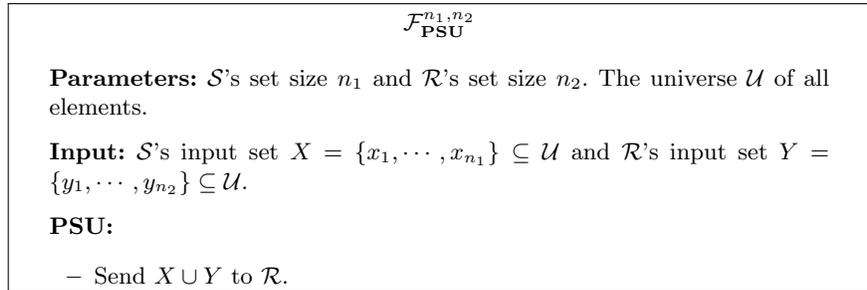
Fig. 2. Ideal Functionality of PSU.

the receiver holds a very large input set, and interacts with many senders, each possibly holding smaller inputs.

A naive solution. We describe again a simpler solution, introduced by Friekken [19]. Recall that $\mathcal{R}$'s input set is $Y = \{y_1, \cdots, y_{n_2}\}$. $\mathcal{R}$ first computes the polynomial $Q(x) = \prod_{i=1}^{n_2}(x - y_i) = \sum_{i=0}^{n_2} a_i \cdot x^i$. Next, it encrypts the polynomial coefficients $a_0, \cdots, a_{n_2}$, resulting in $c_0, \cdots, c_{n_2}$ and sends the ciphertexts to $\mathcal{S}$. For each x_j in $\mathcal{S}$'s set X, $\mathcal{S}$ homomorphically evaluates the polynomial, computing $d = \sum_{i=0}^{n_2} c_i \cdot x_j^i$, and sets $d' = d \cdot x_j$. Then, it re-randomizes d and d' to e and e' and sends the re-randomized ciphertexts back to $\mathcal{R}$. Note that if $x_j \in Y$, e and e' are re-randomized ciphertexts encrypting 0s, which are statistically indistinguishable from fresh ciphertexts encrypting 0s. Therefore, $\mathcal{R}$ could not learn anything for x_j from e and e'. To recover the extra inputs that $\mathcal{S}$ brings to the union, $\mathcal{R}$ decrypts e and e' to obtain the plaintexts, $v = Q(x_j)$ and $v' = x_j \cdot Q(x_j)$. Note that if $x_j \in Y$, both of them are equal to zero as $Q(x_j) = 0$. Again, $\mathcal{R}$ learns nothing from these intersection points from the decryption. On the other hand, if $x_j \notin Y$, $\mathcal{R}$ recovers $x_j = v'/v$ and adds it to the output set. It is trivial to see that this solution gives the correct result.

For simplicity, throughout our paper, we assume $n_1, n_2 \in O(n)$ when analyzing the complexity of our protocols. The naive solution requires $\mathcal{S}$ to evaluate the polynomial n_1 times with an $O(n)$ field operations for each evaluation[6]. Therefore, to achieve a much more efficient construction (both asymptotically and concretely), we use the bucketing technique with packing (both introduced below) on top of the naive construction.

Bucketing. We borrow the bucketing technique first used by Freedman et al. [18] used for the PSI problem. Roughly speaking, using a simple hash table, both parties partition their input sets into $m \in O(n/\log n)$ buckets, where each bucket contains no more than some $k \in O(\log n)$ inputs, except with negligible probability. To prevent additional leakage from the number of inputs in each bucket, both parties should pad it to maximum size k. However, a subtlety is observed by Kolesnikov et al. [36]. To ensure that we do not indicate to the

---

[6] In Kolesnikov et al. [36], the more efficient claim is made as FFT is applicable there. In our case, the coefficients are encrypted so we need to use the straight-forward algorithm with cost linear to the degree of the polynomial.

receiver whether an intersection occurs in any particular bucket, the sender's dummy items must look like intersection items. This is handled by ensuring the receiver includes at least 1 dummy item in each interpolation, so that it will intersect with all dummy items of the sender in the corresponding bucket. As a result, the maximum number of inputs in a receiver's bucket is actually $k + 1$. Then, the naive solution above is run m times, one for each bucket. Kolesnikov et al. also observe that the receiver can do this by using 0 coefficients, rather than interpolating over dummy inputs. In more detail, $\mathcal{R}$ interpolates a total of m polynomials of degree (at most) $k + 1$, one for each bucket, and all using $k + 2$ coefficients. Then $\mathcal{S}$ evaluates a degree $k + 1$ polynomial for each of its inputs. To hide its own bucket sizes, the sender evaluates each polynomial k times, regardless of the bucket load. Thus bucketing reduces the total computational costs of both $\mathcal{S}$ and $\mathcal{R}$ to $O(n \log n)$ field operations, while the total communication cost remains $O(n)$.

Further optimization via packing. To further improve concrete efficiency, we can use the SIMD (Single-Instruction Multiple-Data), first proposed in [52], to compute multiple values in parallel. In particular, we can encode and pack a batch of inputs or polynomial coefficients into a single ring element and use an RLWE encryption scheme to encrypt it into a single ciphertext. Specifically, for $i = [k] \cup \{0\}$, $\mathcal{R}$ packs the ith coefficient of all m buckets into a ring element, and encrypts it to a single ciphertext C_j. Also, for $j = 1, \cdots, k$, $\mathcal{S}$ packs its jth inputs from all m buckets into a ring element. Using coordinate-wise multiplication between plaintext and ciphertext (denoted as $\odot$), $\mathcal{S}$ can compute the multiplication between each pair of polynomial coefficients and $\mathcal{S}$'s input in a batch. This greatly improves concrete computational efficiency. Our semi-honest protocol incorporates the two optimizations mentioned above, and is shown in Figure 3.

Theorem 1. *The protocol Π_{PSU} (Figure 3) securely realizes the ideal functionality $\mathcal{F}_{PSU}^{n_1, n_2}$ (Figure 2), under the presence of a semi-honest adversary corrupting either $\mathcal{S}$ or $\mathcal{R}$.*

Proof. As $\mathcal{F}_{\mathrm{PSU}}$ is deterministic and we are in the semi-honest setting, it suffices to separately show (a) correctness: the protocol output is correct, and (b) privacy: each party's view in the protocol can be simulated using its own input and output.

It is straightforward to see that the correctness holds when the buckets do not overflow during hashing. In particular, in Step 4 of the protocol, $\mathcal{R}$ recovers $(0, 0)$ for each input x of $\mathcal{S}$ within the intersection, i.e., $(x \in Y)$, as a result of the polynomial evaluation yielding 0. On the other hand, every element $x_{i,j} \notin Y$ can be recovered by $\mathcal{R}$ and added to the output. Finally, by following the choices of parameters m and k according to [46,36], the probability of overflow is bounded by 2^{-40}.

To see that the privacy holds, we first show how $\mathsf{Sim}_{\mathcal{S}}(X)$ simulates the view for the corrupt $\mathcal{S}$, which consists of pk, h, and $\mathcal{R}$'s encrypted coefficients $C_0, ..., C_{k+1}$ sent in Step 2:

<div style="text-align:center">

Π_{PSU} **Semi-honest PSU**

</div>

Parameters.

- $\mathcal{S}$'s set size n_1, $\mathcal{R}$'s set size n_2.
- Number of buckets $m \in O(n/\log(n))$, maximum bucket size $k \in O(\log n)$.
- A prime p. A special dummy item $\perp \in \mathbb{Z}_p$. A finite ring $\mathbb{R}$.
- A RLWE scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReRand})$ with plaintext space $\mathbb{R}$, a reversible function $\mathsf{Pack} : \mathbb{Z}_p^m \to \mathbb{R}$.

Input. $\mathcal{S}$'s input set $X = \{x_1, \cdots, x_{n_1}\} \subseteq \mathbb{Z}_p \setminus \{\perp\}$ and $\mathcal{R}$'s input set $Y = \{y_1, \cdots, y_{n_2}\} \subseteq \mathbb{Z}_p \setminus \{\perp\}$.

Protocol.

1. **Setup.** $\mathcal{R}$ computes $pp \leftarrow \mathsf{Setup}(1^\kappa)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(pp)$, picks a random hash function $h : \mathbb{Z}_p \to [m]$.
2. **Encryption.** $\mathcal{R}$ uses h to hash inputs in Y into m buckets. Additionally, it adds a $\perp$ to each bucket. Let $B_i^{\mathcal{R}} = \{y_{i,1}, \cdots, y_{i,|B_i^{\mathcal{R}}|}\}$ denote the set of inputs in $\mathcal{R}$'s i^{th} bucket. Next, for $i \in [m]$, $\mathcal{R}$ samples $a_i \leftarrow \mathbb{Z}_p$ and computes

$$Q_i(x) = a_{i,0} + \cdots + a_{i,k+1}x^{k+1} = a_i(x - y_{i,1})\cdots(x - y_{i,|B_i^{\mathcal{R}}|})$$

   Note that $a_{i,j} = 0$ if $j > |B_i^{\mathcal{R}}|$. For $j = [k+1] \cup \{0\}$, let $A_j = \mathsf{Pack}(a_{1,j}, a_{2,j}, \cdots, a_{m,j})$, $\mathcal{R}$ computes $C_j \leftarrow \mathsf{Enc}_{\mathsf{pk}}^{\mathsf{RLWE}}(A_j)$. Finally, it sends pk, h and $C_0, \ldots, C_{k+1}$ to $\mathcal{S}$.
3. **Evaluate the encrypted polynomial.** $\mathcal{S}$ uses h to hash inputs in X into m buckets. It pads each bucket to size k using $\perp$ and randomly permutes all inputs within each bucket. Let $B_i^{\mathcal{S}} = \{x_{i,1}, \cdots, x_{i,k}\}$ denote the set of inputs in $\mathcal{S}$'s i^{th} bucket. For $j \in [k]$, $i \in [k+1] \cup \{0\}$, let $X_j^i = \mathsf{Pack}(x_{1,j}^i, \ldots, x_{m,j}^i)$. $\mathcal{S}$ computes: $D_j = (\sum_{i=0}^{k+1} C_i \bigodot X_j^i)$ and $D_j' = X_j \bigodot D_j$. Note that

$$D_j \in \mathsf{Enc}_{\mathsf{pk}}^{\mathsf{RLWE}}\left(\mathsf{Pack}(Q_1(x_{1,j}), \cdots, Q_m(x_{m,j}))\right)$$
$$D_j' \in \mathsf{Enc}_{\mathsf{pk}}^{\mathsf{RLWE}}\left(\mathsf{Pack}(Q_1(x_{1,j}) \cdot x_{1,j}, \cdots, Q_m(x_{m,j}) \cdot x_{m,j})\right)$$

   Finally, $\mathcal{S}$ re-randomizes the ciphertexts D_j, D_j' to E_j, E_j' and sends back all (E_j, E_j').
4. **Obtaining output.** For each pair of E_j, E_j', $\mathcal{R}$ decrypts and unpacks to recover $\{(Q_i(x_{i,j}), Q_i(x_{i,j}) \cdot x_{i,j})\}_{i \in [m]}$. Note that if $x_{i,j}$ is in the intersection or $x_{i,j} = \perp$, $\mathcal{R}$ recovers the tuple $(0, 0)$ as $Q_i(x_{i,j}) = 0$. Otherwise, $\mathcal{R}$ can compute $x_{i,j}$ and put it in $\hat{X}$.

Output. $\mathcal{R}$ outputs $\hat{X} \cup Y$.

<div style="text-align:center">

Fig. 3. Semi-honest PSU Protocol

</div>

1. Run $pp \leftarrow \mathsf{Setup}(1^\kappa)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(pp)$ and randomly pick a hash function h.
2. Compute $\widetilde{C}_0 \leftarrow \mathsf{Enc}_{\mathsf{pk}}^{\mathsf{RLWE}}(0), \cdots, \widetilde{C}_{k+1} \leftarrow \mathsf{Enc}_{\mathsf{pk}}^{\mathsf{RLWE}}(0)$.

Clearly, pk, h follows the same distribution, and any PPT adversary that can distinguish $C_0, \cdots, C_{k+1}$ and $\widetilde{C}_0, \cdots, \widetilde{C}_{k+1}$ given pk can break the IND-CPA security of the underlying encryption scheme.

Next, we show how $\mathsf{Sim}_{\mathcal{R}}(Y, X \cup Y)$ simulates the view for the corrupt $\mathcal{R}$, which consists of $\mathcal{S}$'s replies $\{E_j, E_j'\}_{j \in [k]}$ in Step 3:

1. Simulates $\mathcal{R}$'s random tape to run $pp \leftarrow \mathsf{Setup}(1^\kappa)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(pp)$, and randomly pick a hash function h.
2. Compute $\widetilde{X} = X \cup Y \setminus Y$. Hash $\widetilde{X}$ into the m buckets and pad each bucket to size k with $\perp$ and randomly permute all items within each bucket. Also, hash inputs in Y into the m buckets and add a $\perp$ to each bucket. Let $\widetilde{B}_i^{\mathcal{S}} = \{x_{i,1}, \cdots, x_{i,k}\}$ (resp. $B_i^{\mathcal{R}} = \{y_{i,1}, \cdots, y_{i,|B_i^{\mathcal{R}}|}\}$) denote the set of items in $\mathcal{S}$'s (resp. $\mathcal{R}$'s) i^{th} bucket.
3. For $i \in [m]$, sample $a_i \leftarrow \mathbb{Z}_p$ and compute

$$Q_i(x) = a_{i,0} + \cdots + a_{i,k+1} x^{k+1} = a_i(x - y_{i,1}) \cdots (x - y_{i,|B_i^{\mathcal{R}}|})$$

For $j = [k+1] \cup \{0\}$, let $A_j = \mathsf{Pack}(a_{1,j}, a_{2,j}, \cdots, a_{m,j})$ and compute $C_j \leftarrow \mathsf{Enc}_{\mathsf{pk}}^{\mathsf{RLWE}}(A_j)$.
4. For $j \in [k]$, and $i \in [k+1] \cup \{0\}$, let $\widetilde{X}_j^i = \mathsf{Pack}(x_{1,j}^i, \ldots, x_{m,j}^i)$, compute $\widetilde{D}_j = \left(\sum_{i=0}^{k+1} C_i \odot \widetilde{X}_j^i\right)$ and $\widetilde{D}_j' = \widetilde{X}_j \odot \widetilde{D}_j$. Re-randomize the ciphertexts $\widetilde{D}_j, \widetilde{D}_j'$ to $\widetilde{E}_j, \widetilde{E}_j'$.

First note that the key pair $(\mathsf{pk}, \mathsf{sk})$ follows the same distribution in both views. Next, due to the statistically re-randomization indistinguishable property and CPA security, E_j, E_j' in the real view and $\widetilde{E}_j, \widetilde{E}_j'$ in the simulated view are computationally indistinguishable. Moreover, in the real view, decrypting and unpacking all E_j, E_j' for $j \in [k]$ results in tuples corresponding to inputs in $\hat{X} = X \setminus Y$, as well as $n_1 - |\hat{X}|$ tuples of $(0,0)$ corresponding to intersected inputs or padding inputs $\perp$. On the other hand, in the simulated view, decrypting and unpacking all $\widetilde{E}_j, \widetilde{E}_j'$ for $j \in [k]$ results in tuples corresponding to inputs in $\widetilde{X}$, which equals to $\hat{X}$, and the same number of $(0,0)$ tuples as in the real view. Therefore, the decryption of ciphertexts gives identical outputs. This concludes our proof.

Reusable Security. We do not formally model re-usable security, but we make the following observations. When the receiver is corrupt, $\mathsf{Sim}_{\mathcal{R}}$ does not use the private randomness of the receiver when constructing $\widetilde{E}_j, \widetilde{E}_j'$ in Step 3. Therefore, when reusing $\mathcal{R}$'s message across executions, if there exists a distinguisher $\mathcal{D}_{\mathsf{many}}$ that can distinguish the real-world executions from the simulated executions, then there exists a distinguisher $\mathcal{D}$ that can do so in a single execution, in

contradiction of Theorem 1. This follows because $\mathcal{D}$ can simply run $\mathsf{Sim}_{\mathcal{R}}$ itself to generate the missing portion of $\mathcal{D}_{\mathsf{many}}$'s view.

When the sender is corrupt, reusability holds trivially for semi-honest protocols that have full correctness. Intuitively, this is because there is nothing learned by the sender after each additional execution, if the protocol always completes correctly.[7] Therefore, it suffices for $\mathsf{Sim}_{\mathcal{S}}$ to simulate the message of the receiver just one time. In subsequent protocol executions, $\mathsf{Sim}_{\mathcal{S}}$ simply extracts the sender input and submits it to the trusted party.

3.2   Semi-Honest Efficient PSU with OT

In this subsection, we show a simple trick to trade the reusability and round complexity for better communication and computational cost when we are in the OT-hybrid model. Recall that in the previous protocol, $\mathcal{R}$ recovers the set $\{(Q_i(x_{i,j}), Q_i(x_{i,j}) \cdot x_{i,j})\}_{i \in [m]}$ from each pair of E_j, E'_j sent by $\mathcal{S}$. Instead, we first let $\mathcal{R}$ recover only $\{Q_i(x_{i,j})\}_{i \in [m]}$ from E_j, and then have $\mathcal{R}$ to recover $x_{i,j}$ through OT, if and only if $Q_i(x_{i,j}) \neq 0$. This reduces the number of ciphertexts needed to be computed, thus reducing the communication and computational costs. Formally, our protocol is shown in Figure 4, with the differences highlighted.

Theorem 2. *In the $\mathcal{F}_{OT}$-hybrid model, the protocol $\Pi_{\mathrm{PSU}}^{\mathcal{F}_{OT}}$ (Figure 4) securely realizes the ideal functionality $\mathcal{F}_{PSU}^{n_1, n_2}$ (Figure 2), under the presence of a semi-honest adversary corrupting either $\mathcal{S}$ or $\mathcal{R}$.*

The proof is quite similar to the security proof for $\Pi_{\mathrm{PSU}}^{\mathsf{reuse}}$. Therefore, we only give a sketch here and defer the proof to the full version. To start with, it is easy to verify that the correctness holds. Next, to claim privacy against $\mathcal{S}$, note that the view of $\mathcal{S}$ consists of pk and encrypted coefficients $C_0, \ldots, C_{k+1}$ sent by $\mathcal{R}$, which is exactly the same as in $\Pi_{\mathrm{PSU}}^{\mathsf{reuse}}$. Finally, compared to $\Pi_{\mathrm{PSU}}^{\mathsf{reuse}}$, the view for $\mathcal{R}$ now contains $\{E_j\}_{j \in [m]}$ instead of $\{(E_j, E'_j)\}_{j \in [m]}$, so we can modify the simulator to only generate $\{\widetilde{E}_j\}_{j \in [m]}$, and argue computational indistinguishability as before. Additionally, the simulator simulates all OT executions to return every $x \in \widetilde{X}$ to $\mathcal{R}$, which is identical to the real view.

4   PSU with Malicious Senders

In this section, we present two PSU protocols that securely realize our PSU functionality (Figure 1) against a malicious sender in the random oracle model. In Section 1.1, we define and discuss the proper PSU functionality in this setting. In Section 4.1, we provide a two-round reusable protocol relying on an additive

---

[7] In contrast, if the protocol occasionally fails, one would have to argue that the failure event does not reveal something about the randomness used in the receiver's message. It is possible to construct a contrived protocol that is secure after a small number of failures, but eventually leaks all of the randomness of the receiver's message.

$\Pi_{\mathrm{PSU}}^{\mathcal{F}_{\mathbf{OT}}}$ **Semi-honest PSU with OT**

Parameters.

- $\mathcal{S}$'s set size n_1, $\mathcal{R}$'s set size n_2.
- Number of buckets $m \in O(n/\log(n))$, maximum bucket size $k \in O(\log n)$.
- A prime p. A special dummy item $\perp \in \mathbb{Z}_p$. A finite ring $\mathbb{R}$.
- A RLWE scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReRand})$ with plaintext space $\mathbb{R}$, a reversible function $\mathsf{Pack} : \mathbb{Z}_p^m \to \mathbb{R}$.
- Ideal functionality $\mathcal{F}_{\mathrm{OT}}$ (Figure 7).

Input. $\mathcal{S}$'s input set $X = \{x_1, \cdots, x_{n_1}\} \subseteq \mathbb{Z}_p \setminus \{\perp\}$ and $\mathcal{R}$'s input set $Y = \{y_1, \cdots, y_{n_2}\} \subseteq \mathbb{Z}_p \setminus \{\perp\}$.

Protocol.

1. **Setup.** $\mathcal{R}$ computes $pp \leftarrow \mathsf{Setup}(1^\kappa), (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(pp)$, picks a random hash function $h : \mathbb{Z}_p \to [m]$.
2. **Encryption.** $\mathcal{R}$ uses h to hash inputs in Y into m buckets. Additionally, it adds a $\perp$ to each bucket. Let $B_i^{\mathcal{R}} = \{y_{i,1}, \cdots, y_{i,|B_i^{\mathcal{R}}|}\}$ denote the set of inputs in $\mathcal{R}$'s i^{th} bucket. Next, for $i \in [m]$, $\mathcal{R}$ samples $a_i \leftarrow \mathbb{Z}_p$ and computes

$$Q_i(x) = a_{i,0} + \cdots + a_{i,k+1}x^{k+1} = a_i(x - y_{i,1}) \cdots (x - y_{i,|B_i^{\mathcal{R}}|})$$

    Note that $a_{i,j} = 0$ if $j > |B_i^{\mathcal{R}}|$. For $j = [k+1] \cup \{0\}$, let $A_j = \mathsf{Pack}(a_{1,j}, a_{2,j}, \cdots, a_{m,j})$, $\mathcal{R}$ computes $C_j \leftarrow \mathsf{Enc}_{\mathsf{pk}}^{\mathsf{RLWE}}(A_j)$. Finally, it sends pk, h and $C_0, \ldots, C_{k+1}$ to $\mathcal{S}$.
3. **Evaluate the encrypted polynomial.** $\mathcal{S}$ uses h to hash inputs in X into m buckets. It pads each bucket to size k using $\perp$ and randomly permutes all inputs within each bucket. Let $B_i^{\mathcal{S}} = \{x_{i,1}, \cdots, x_{i,k}\}$ denote the set of inputs in $\mathcal{S}$'s i^{th} bucket. For $j \in [k], i \in [k+1] \cup \{0\}$, let $X_j^i = \mathsf{Pack}(x_{1,j}^i, \ldots, x_{m,j}^i)$.

    $\mathcal{S}$ computes: $D_j = (\sum\limits_{i=0}^{k+1} C_i \bigodot X_j^i)$. Note that

$$D_j \in \mathsf{Enc}_{\mathsf{pk}}^{\mathsf{RLWE}}\left(\mathsf{Pack}(Q_1(x_{1,j}), \cdots, Q_m(x_{m,j}))\right)$$

    Finally, $\mathcal{S}$ re-randomizes the ciphertexts D_j to E_j and sends back all E_j.
4. **Obtaining output.** For each E_j, $\mathcal{R}$ decrypts and unpacks to recover $\{Q_i(x_{i,j})\}_{i \in [m]}$. Note that if $x_{i,j}$ is in the intersection or $x_{i,j} = \perp$, $Q_i(x_{i,j}) = 0$. Let $b_{i,j} = 0$ if $Q_i(x_{i,j}) = 0$ and $b_{i,j} = 1$ otherwise. Let $\hat{X} = \emptyset$. For each pair of $i \in [m], j \in [k]$, $\mathcal{S}$ and $\mathcal{R}$ call the OT functionality $\mathcal{F}_{\mathrm{OT}}$. Specifically, $\mathcal{S}$ serves as the sender with inputs $(\perp, x_{i,j})$ and $\mathcal{R}$ serves as the receiver with input $b_{i,j}$. Therefore, for all i, j such that $b_{i,j} = 1$, $\mathcal{R}$ receives $x_{i,j}$ and puts it in $\hat{X}$.

Output. $\mathcal{R}$ outputs $\hat{X} \cup Y$.

Fig. 4. Semi-honest PSU with OT Protocol

homomorphic encryption scheme, and in Section 4.2 we provide a more efficient protocol, in terms of both communication and computation, at the cost of adding two rounds of interaction. This protocol is in an OT-hybrid model, using a re-randomizable public key encryption scheme. We note that the two protocols are incomparable and achieve different properties and efficiency measures.

4.1   Reusable PSU

In this subsection, we show how to realize a two-round reusable PSU protocol using the FNP paradigm. Compared to our semi-honest protocol, in which we use a polynomial and simple hash table to encode $\mathcal{R}$'s input set, here we generalize our protocol by using the abstraction of OKVS to encode $\mathcal{R}$'s input set. Also, we rely on an Additive-homomorphic encryption scheme, but due to the requirement of the locality property to use the FNP paradigm, we cannot use packing as we did in the semi-honest setting.

   In more detail, $\mathcal{R}$ first encodes its input set Y to an OKVS data structure D, such that for every $y \in Y$, $\mathsf{Decode}(D, y)$ returns a fresh encryption of 0. After receiving D, for each input x, $\mathcal{S}$ homomorphically computes a tuple of (rerandomized) ciphertexts: $(d, d \cdot x, d \cdot r)$ where $d = \mathsf{Decode}(D, x)$. Recall that $h(x\|r)$ determines the randomness used by $\mathcal{S}$ to derandomize the computation of this tuple/message. When $\mathsf{Decode}(D, x) \neq 0$, i.e., $x \notin Y$, $\mathcal{R}$ can recover x and r from the decryption of the tuple, and reevaluate $\mathcal{S}$'s derivation of it. Eventually, $\mathcal{R}$ verifies the correctness of all tuples/messages corresponding to the elements in $X \setminus Y$. Our protocol is given in Figure 5.

Theorem 3. *The protocol $\Pi_{\mathrm{PSU}}^{*\mathsf{reuse}}$ (Figure 5) securely realizes the ideal functionality $\mathcal{F}_{PSU}^{\hat{n}_1, n_2}$ (Figure 1) with abort, against any malicious sender and any semi-honest receiver with abort in the random oracle model.*

   We give a high-level intuition of the proof below and defer the full proof to Appendix B.

Malicious sender. To simulate the OKVS for the malicious sender, we argue that the simulator can just encode an arbitrary set of inputs (as keys of the OKVS) with values chosen uniformly from the ciphertext space. By the obliviousness property of the OKVS, this is computationally indistinguishable from an OKVS encoding real inputs of $\mathcal{R}$ with uniformly chosen ciphertexts. Next, we can further replace the uniformly chosen ciphertexts with ciphertexts encrypting 0s, and argue that the change is computationally indistinguishable, due to single message multiple ciphertexts indistinguishability of the underlying PKE scheme.

   Finally, to show that the output distribution in the above hybrid world is statistically indistinguishable from that of the real world, we rely on the following arguments: Due to the randomness property of the OKVS, any of $\mathcal{S}$'s inputs that are not encoded by $\mathcal{R}$ decode to a random value/ciphertext, and the probability that a random ciphertext decrypts to 0 is negligible. Therefore, $\mathcal{R}$ can retrieve all $\mathcal{S}$'s non-intersected inputs except for negligible probability, assuming that $\mathcal{S}$ does not cheat. In the case where $\mathcal{S}$ cheats by sending incorrectly formed

---

$$\Pi_{\mathsf{PSU}}^{*\mathsf{reuse}}$$

Parameters.

- $\mathcal{S}$'s set size upper bound $\hat{n}_1$, $\mathcal{R}$'s set size n_2.
- Statistical security parameter λ.
- Number of slots $m \in O(n)$.
- A finite group $\mathbb{G}$ and finite field $\mathbb{F}$.
- A AH-PKE scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReRand})$ with plaintext space $\mathbb{F}$, ciphertext space $\mathbb{G}$, and randomness space ∇.
- An $(n, m, 2^{-\lambda})$-OKVS scheme $(\mathsf{Encode}, \mathsf{Decode})$ with key space $\mathbb{F}$ and value space $\mathbb{G}$.
- A hash function $h : \mathbb{F}^2 \to \nabla^3$, modeled as a random oracle.

Input. $\mathcal{S}$'s input set $X = \{x_1, \cdots, x_{n_1}\}$ and $\mathcal{R}$'s input set $Y = \{y_1, \cdots, y_{n_2}\}$ where each entry in X, Y is a field element of $\mathbb{F}$.

Protocol.

1. **KeyGen.** $\mathcal{R}$ computes $pp \leftarrow \mathsf{Setup}(1^\kappa)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(pp)$.
2. **Encrypt and encode.** $\mathcal{R}$ generates n_2 fresh ciphertexts $(c_1, \ldots, c_{n_2})$ of 0, that is, for $i \in [n_2]$, $c_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$. It then generates $D \leftarrow \mathsf{Encode}(\{(y_i, c_i)\}_{i \in [n_2]})$ and sends pk, D to $\mathcal{S}$.
3. **Decode and re-randomize.** For each element $x_i \in X$, $\mathcal{S}$ computes $d_i = \mathsf{Decode}(D, x_i)$. Next, it samples $r_i \leftarrow \mathbb{F}$ and computes $v_i \| v_i' \| v_i'' = h(x_i \| r_i)$. Then, for each $i \in [n_1]$, it computes $e_i = \mathsf{ReRand}_{\mathsf{pk}}(d_i, v_i)$, $e_i' = \mathsf{ReRand}_{\mathsf{pk}}(d_i \cdot x_i, v_i')$, and $e_i'' = \mathsf{ReRand}_{\mathsf{pk}}(d_i \cdot r_i, v_i'')$, with $\cdot$ denoting the multiplication between a ciphertext and a plaintext. Finally, it sends $\{(e_i, e_i', e_i'')\}_{i \in [n_1]}$ to $\mathcal{R}$.
4. **Decrypt and obtain output.** $\mathcal{R}$ receives $\{(e_i, e_i', e_i'')\}_{i \in [n_1]}$ and aborts if $n_1 > \hat{n}_1$. Otherwise, for each (e_i, e_i', e_i'') received, $\mathcal{R}$ computes the decryption: $u_i = \mathsf{Dec}_{\mathsf{sk}}(e_i)$, $u_i' = \mathsf{Dec}_{\mathsf{sk}}(e_i')$, $u_i'' = \mathsf{Dec}_{\mathsf{sk}}(e_i'')$. Let $\hat{X} = \emptyset$. For each tuple of u_i, u_i', u_i'' such that $u_i \neq 0$, it calculates $x = u_i'/u_i$ and $r = u_i''/u_i$. It then reevaluates $\mathcal{S}$'s previous step to verify that $\{(e_i, e_i', e_i'')\}$ was correctly computed. If so, it sets $\hat{X} \cup \{x\}$.

**Output.** $\mathcal{R}$ outputs $\hat{X} \cup Y$.

Fig. 5. Malicious sender reusable PSU

ciphertext tuples, $\mathcal{R}$ in the real world discards those recovered (non-intersected) inputs that correspond to incorrectly formed ciphertext tuples. In the (simulated) hybrid world, the simulator can extract each of $\mathcal{S}$'s inputs along with the local randomness used to re-randomize ciphertexts through h, which is modeled as a random oracle. It can then use these inputs and the corresponding randomness to verify that each tuple of ciphertexts is computed correctly, discarding those inputs that are not. Although the discarded inputs may include those intersected inputs, this will not affect the output, as it will not be returned to $\mathcal{R}$ by the ideal PSU functionality. The formal proof is in Appendix B.

Semi-honest receiver. The simulator for the semi-honest receiver is similar to the simulator we give in the semi-honest setting. Specifically, the simulator, given $\mathcal{R}$'s input set Y and the output set $X \cup Y$, computes $\widetilde{X} = X \cup Y \setminus Y$. Then, it follows $\mathcal{R}$'s steps to generate the OKVS using $\mathcal{R}$'s input set Y. Next, it follows $\mathcal{S}$'s steps to decode and generate the re-randomized ciphertext tuples using inputs in $\widetilde{X}$, i.e., the set of $\mathcal{S}$'s inputs that are not in the intersection. Also, the simulator adds "dummy" ciphertext tuples, generated by encrypting 0s, and sends all ciphertext tuples to $\mathcal{R}$. In the end, $\mathcal{R}$ recovers all inputs in $\widetilde{X}$, except with negligible probability, due to the randomness property of OKVS. Moreover, the remaining "dummy" ciphertext tuples are (statistically) indistinguishable from those ciphertext tuples generated through proper decoding and re-randomizing in the real world, due to the (statistically) re-randomization indistinguishable property of the underlying PKE.

Reusable security. For a corrupt semi-honest receiver, the argument is identical to the one made in Section 3.1: because $\mathsf{Sim}_{\mathcal{R}}$ did not use the private randomness of $\mathcal{R}$ when constructing the sender messages in Step 3, any distinguisher that can break security after multiple executions could do so after a single execution by continuing the simulation itself.

When the sender is malicious, we claim reusability by first noting that the FNP paradigm enforces honest behavior. In particular, suppose there exists some $\mathcal{D}$ that can distinguish the real world from the ideal world after ℓ executions by some adversary $\mathcal{A}$. We can show that there exists a sequence of inputs, $X_1, \ldots, X_\ell$, such that $\mathcal{D}$ can distinguish the real world from the ideal world when a semi-honest adversary corrupts $\mathcal{S}$. Concretely, these inputs are the subsets of $\mathcal{A}$'s inputs that are verified under the FNP validation. As we argued in Section 3.1, since the protocol is correct (with all but negligible probability), it follows that nothing is learned about the randomness used by the receiver in generating its single message, and a one-time simulation of that message suffices.

4.2   Efficient PSU with OT

In this subsection, we show how to construct a PSU protocol in the OT-hybrid model using the FNP paradigm. Our protocol sacrifices the reusability and round complexity, but generally achieves better concrete communication and computational costs compared to our previous reusable PSU protocol. We quickly go

over our new protocol by highlighting the differences with the previous one. Our complete protocol is given in Figure 6.

1. To start with, $\mathcal{R}$ encodes its input and sends OKVS D to $\mathcal{S}$, except that it uses fresh encryptions of a single, randomly selected plaintext w, instead of fresh encryptions of 0.
2. For each of the inputs of $\mathcal{S}$'s, instead of computing the tuple of ciphertexts (e, e', e''), $\mathcal{S}$ only computes and returns the first ciphertext e.
3. $\mathcal{R}$ decrypt e, and knows that it corresponds to a non-intersected input if $\mathsf{Dec}(e) \neq w$.
4. $\mathcal{S}$ and $\mathcal{R}$ call $\mathcal{F}_{\mathrm{OT}}$ for each input of $\mathcal{S}$. $\mathcal{R}$ to retrieves all x, r corresponds to e such that $\mathsf{Dec}(e) \neq w$ (and recovers $\bot$ elsewhere). If $\mathcal{R}$ verifies that e is correctly computed by reevaluating $\mathcal{S}$'s previous step, $\mathcal{R}$ puts x into the output set.

It is straightforward to see that this protocol still utilizes the FNP paradigm. In particular, $\mathcal{R}$ now relies on OT to recover the randomness used to derandomize $\mathcal{S}$'s computation of e. Moreover, (semi-honest) $\mathcal{R}$ only recovers the randomness corresponding to those of $\mathcal{S}$'s inputs that are not in the intersection.

In the new protocol, notice that we do not require any additive homomorphic operation on the ciphertexts, since $\mathcal{S}$ no longer needs to compute $x \cdot e$ and $r \cdot e$. Additionally, the plaintext space of the encryption scheme no longer needs to include 0. Together, this allows us to replace the stronger AH-PKE schemes in the previous protocol with weaker ReRand-PKE schemes, which allows for more concretely efficient communication. For the computational cost, the cost for additive homomorphic operation is the dominant cost in our previous protocol. By avoiding it here, the concrete computational cost is also greatly improved.

Theorem 4. *In the $\mathcal{F}_{OT}$-hybrid model, the protocol $\Pi_{\mathrm{PSU}}^{*\mathcal{F}_{OT}}$ (Figure 6) securely realizes the ideal functionality $\mathcal{F}_{PSU}^{\hat{n}_1, n_2}$ (Figure 1) with abort, against any malicious sender and any semi-honest receiver with abort in the random oracle model.*

The majority of the proof is the same as the proof of Theorem 3. In particular, as the protocol now relies on OT to transmit the input x and its corresponding randomness r, the simulator needs to adjust accordingly, but the general argument remains the same. We defer the full proof to Appendix B.

5 Implementations and Experiments

5.1 Implementation Details

We implemented all our protocols using 64-bit input items. Our implementations can be found at: https://github.com/merpsu1/merpsu_semihonest[8] and https://github.com/merpsu1/merpsu_malicious.

---

[8] We are currently awaiting the approval process to release them. Once granted, we will promptly publish them here.

$$\Pi_{\text{PSU}}^{*\mathcal{F}_{\textbf{OT}}}$$

Parameters.

- $\mathcal{S}$'s set size upper bound $\hat{n}_1$, $\mathcal{R}$'s set size n_2.
- Statistical security parameter λ.
- Number of slots $m \in O(n)$.
- Ideal functionality $\mathcal{F}_{\text{OT}}$ (Figure 7).
- Two finite groups $\mathbb{G}_1$ and $\mathbb{G}_2$.
- A ReRand-PKE scheme (Setup, KeyGen, Enc, Dec, ReRand) with plaintext space $\mathbb{G}_1$, ciphertext space $\mathbb{G}_2$, and randomness space ∇.
- An $(n, m, 2^{-\lambda})$-OKVS scheme (Encode, Decode) with key space $\mathbb{G}_1$ and value space $\mathbb{G}_2$.
- A hash function $h : \mathbb{G}_1^2 \to \nabla$, modeled as a random oracle.

Input. $\mathcal{S}$'s input set $X = \{x_1, \cdots, x_{n_1}\}$ and $\mathcal{R}$'s input set $Y = \{y_1, \cdots, y_{n_2}\}$ where all entries in X (resp., Y) are distinct group elements in $\mathbb{G}_1$.

Protocol.

1. **KeyGen.** $\mathcal{R}$ computes $pp \leftarrow \text{Setup}(1^\kappa)$, $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(pp)$.

2. **Encrypt and encode.** $\mathcal{R}$ uniformly samples $w \leftarrow \mathbb{G}_1$ and generates n_2 fresh ciphertexts $(c_1, \ldots, c_{n_2})$ of w, i.e., for $i \in [n_2]$, $c_i \leftarrow \text{Enc}_{\text{pk}}(w)$. It then generates $D \leftarrow \text{Encode}(\{(y_i, c_i)\}_{i \in [n_2]})$ and sends pk, D to $\mathcal{S}$.

3. **Decode and re-randomize.** For each element $x_i \in X$, $\mathcal{S}$ computes $d_i = \text{Decode}(D, x_i)$. Next, it samples $r_i \leftarrow \mathbb{G}_1$ and computes $v_i = h(x_i \| r_i)$. Finally, it computes $e_i = \text{ReRand}_{\text{pk}}(d_i, v_i)$ for each $i \in [n_1]$ and sends $\{e_i\}_{i \in [n_1]}$ to $\mathcal{R}$.

4. **Test equality.** $\mathcal{R}$ receives $\{e_i\}_{i \in [n_1]}$ and aborts if $n_1 > \hat{n}_1$. Otherwise, for each e_i received, $\mathcal{R}$ computes the decryption: $u_i = \text{Dec}_{\text{sk}}(e_i)$. Let $b_i = 0$ if $u_i = w$ and $b_i = 1$ otherwise.

5. **Obtain output.** Let $\hat{X} = \emptyset$. For each $i \in [n_1]$, $\mathcal{S}$ and $\mathcal{R}$ call the OT functionality $\mathcal{F}_{\text{OT}}$. Specifically, $\mathcal{S}$ serves as the sender with inputs $(\bot, x_i \| r_i)$ and $\mathcal{R}$ serves as the receiver with input b_i. Therefore, for all i such that $b_i = 1$, $\mathcal{R}$ receives $x_i \| r_i$ and reevaluates $\mathcal{S}$'s previous step 3 to verify that e_i was correctly decoded and re-randomized by $\mathcal{S}$. If so, it puts x_i in $\hat{X}$.

Output. $\mathcal{R}$ outputs $\hat{X} \cup Y$.

Fig. 6. Malicious sender PSU in the OT-hybrid model

Table 2. Bucketing parameters, those within parentheses specifically chosen to optimize under a network speed of 10Mbps.

	$n = 2^{14}$		$n = 2^{16}$		$n = 2^{18}$		$n = 2^{20}$	
Protocols	m	k	m	k	m	k	m	k
Our Protocol SA	4096	29	4096	58	4096	139	4096	396
Our Protocol SB	4096	29	4096	58	24576 (4096)	47 (139)	73728 (16384)	55 (141)

Semi-honest protocols We employ a 192-bit ciphertext modulus to accommodate noise blowup due to homomorphic operations. This encompasses a (64+12) bits expansion from multiplication of plaintext ring element (12 bits expansion from the polynomial ring degree), and an upper bound of 8 bits expansion from summing terms within buckets. To conduct noise flooding, we sample noise from a uniform distribution with $148 + 40$ bits support. In total, this puts the noise level to $188 < 192$ bits.

Unlike [20,32,53], our semi-honest protocols rely on bucketing techniques, allowing us to adjust the number of buckets m and the bucket size k to optimize either communication cost or runtime. Therefore, we provide two variants: *Protocol SA* prioritizes the communication cost, while *Protocol SB* focuses on reducing the runtime. Specifically, the communication cost of these protocols is dominated by $m \cdot k$ while the computational cost is dominated by $m \cdot k^2$. And m and k are chosen such that there is a negligible probability that overflow occurs in any of the m buckets. Consequently, *Protocol SA* opts for a smaller m, while *Protocol SB* chooses m in a way to balance the computational cost and communication cost. The specific values are detailed in Table 2.

Malicious Protocols Our protocols are implemented in JAVA on top of the open-source project *mpc4j* by Zhang et al. [53]. Both of our two-round and four-round protocols use 3H-GCT[21] to instantiate the OKVS scheme. As for the encryption scheme, our two-round protocol uses Paillier encryption while our four-round protocols use ECC ElGamal encryption. Their communication cost and runtime are presented in Table 3 and 4 under tabs *Protocol M2, Protocol M4* respectively.

5.2 Experimental Setup

We ran all our experiments in a single AWS c5.24xlarge instance. The machine has 48 cores with GPU clock speed of 3.6 GHz and 192 GB of RAM. We use single-threading for all our experiments. To simulate different network settings (LAN, 1Gbps, 100Mbps, and 10Mbps), we used the Linux *tc* command. Additionally, we add 80ms latency to all settings except for the LAN setting.

5.3 Experimental Results

We compare our results against the existing PSU protocols including Kolesnikov et al.[36], Garimella et al. [20], Jia et al. [32], and Zhang et al. [53]. For Jia

Table 3. The communication cost in MB.

Protocols	$n = 2^{14}$			$n = 2^{16}$			$n = 2^{18}$			$n = 2^{20}$		
	$R \to S$	$S \to R$	Total	$R \to S$	$S \to R$	Total	$R \to S$	$S \to R$	Total	$R \to S$	$S \to R$	Total
KRTW19[36]	4.19	29.64	33.8	17.7	122.1	139.8	69.3	562.8	632	300	2306	2606
GMR+21[20]	5.91	7.98	13.9	26.0	34.1	60.1	114	145	259	493	616	1109
JSZ+22[32]-R	4.66	5.64	10.28	20.76	24.75	45.49	92.68	107.9	200.56	405.54	467.27	872.79
ZCL+21[53]-SKE	3.17	3.36	6.52	12.6	13.4	26.0	50.3	53.5	103.8	201	214	415
ZCL+21[53]-PKE	1.17	1.59	2.75	4.63	6.37	11.0	18.5	25.5	44.0	74.0	102	176
ZCL+21[53]-PKE*	2.17	2.90	5.05	8.64	11.6	20.2	34.5	46.3	80.8	138	185	323
Our Protocol SA2	2.91	10.88	13.78	5.63	21.75	27.38	13.22	52.13	65.35	37	149	186
Our Protocol SB2	2.91	10.88	13.78	5.63	21.75	27.38	26.63	105.75	132.38	93 (53)	371 (212)	464 (265)
Our Protocol SA4	2.91	6.34	9.25	5.63	12.69	18.31	13.22	30.41	43.63	37	87	124
Our Protocol SB4	2.91	6.34	9.25	5.63	12.69	18.31	26.63	61.69	88.32	93 (53)	217 (123)	310 (176)
Our Protocol M2	36.15	15.7	51.85	NA	NA	NA	NA	NA	NA	NA	NA	NA
Our Protocol M4	1.77	1.68	3.45	7.07	6.69	13.76	28.29	26.75	55.04	113.16	107	220.16

et al. [32], we used the protocol that shuffles the receiver's set, which performs better in the balanced setting. For Zhang et al. [53], we test all three variants reported in their paper: SKE, PKE, and PKE*, where PKE performs point compression to reduce communication costs while PKE* does not. Also, their SKE implementation additionally requires a preprocessing phase to generate Boolean multiplication triples which we do not test and report the online time only. As per the authors, this phase results in approximately a 50-fold surge in the overall end-to-end runtime. Consequently, we opt to exclude their SKE version when assessing the best-performing protocol.

Using the implementations in open-source project *mpc4j* by Zhang et al. [53], we ran all experiments on the same AWS c5.24xlarge instance and recorded the run-time and communication cost. We ran all experiments with input sizes of $2^{14}, 2^{16}, 2^{18}, 2^{20}$ except for our 2-round malicious protocol, which is quite expensive due to the usage of Paillier encryption. See Table 3 for communication cost and Table 4 for runtime, with the optimal cost (except Zhang et al.'s SKE version [53]) for each setting marked in blue.

Our $SA4$ Protocol demonstrates the most efficient communication cost for input sizes of 2^{18} and 2^{20}. Particularly, for the latter, we've achieved a $1.43\times$ improvement compared to the previous state-of-the-art solution. Regarding runtime, our $SB4$ protocol outperforms other protocols for an input size of $n = 2^{20}$ except in cases with the slowest bandwidth. Specifically, our protocol is $1.04 - 1.16$ times as fast as the best previous approaches. Moreover, when the network bandwidth is 100Mbps, our $SB4$ protocol surpasses all existing protocols in terms of runtime, except for the smallest input size ($n = 2^{14}$). In particular, our protocol is $1.15 - 1.25$ times as fast as the best previous approaches. In summary, our protocols provide the flexibility to tune the balance between communication and computation costs (excluding network delays). We excel in larger set sizes scenarios and moderate bandwidth cases. In contrast, the work of Garimella et al. achieves better computational efficiency for smaller set sizes but imposes significant communication costs, making it preferable for high throughput scenarios. On the other hand, Zhang et al.'s PKE variant is communication efficient but less computationally efficient than Garimella et al.'s work and our

best variant, and therefore performs better in the lowest throughput setting (10 Mbps).

Finally, our malicious four-round protocol (M4) is less than 2x as expensive in both communication and computation compared to all other *semi-honest constructions*.

References

1. Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 387–404. Springer, Heidelberg, May 2014.
2. Navid Alamati, Pedro Branco, Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Sihang Pu. Laconic private set intersection and applications. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography*, pages 94–125, Cham, 2021. Springer International Publishing.
3. Diego Aranha, Chuanwei Lin, Claudio Orlandi, and Mark Simkin. Laconic private set-intersection from pairings. Cryptology ePrint Archive, Paper 2022/529, 2022. https://eprint.iacr.org/2022/529.
4. Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel Kho, and Jennie Rogers. Smcql: Secure querying for federated databases. *Proc. VLDB Endow.*, 10(6):673–684, feb 2017.
5. Marina Blanton and Everaldo Aguiar. Private and oblivious set and multiset operations. In Heung Youl Youm and Yoojae Won, editors, *ASIACCS 12*, pages 40–41. ACM Press, May 2012.
6. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 309–325, New York, NY, USA, 2012. Association for Computing Machinery.
7. 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, Heidelberg, December 2005.
8. Nishanth Chandran, Nishka Dasgupta, Divya Gupta, Sai Lakshmi Bhavana Obbattu, Sruthi Sekar, and Akash Shah. Efficient linear multiparty PSI and extensions to circuit/quorum PSI. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1182–1204. ACM Press, November 2021.
9. Nishanth Chandran, Divya Gupta, and Akash Shah. Circuit-psi with linear complexity via relaxed batch OPPRF. *Proc. Priv. Enhancing Technol.*, 2022(1):353–372, 2022.
10. Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1243–1255. ACM Press, October / November 2017.
11. Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In Radu Sion, editor, *Financial Cryptography and Data Security, 14th International Conference, FC 2010, Tenerife, Canary Islands, Spain, January 25-28, 2010, Revised Selected Papers*, volume 6052 of *Lecture Notes in Computer Science*, pages 143–159. Springer, 2010.

Table 4. The runtime in *seconds*.

Network	Protocols	Input Size			
		$n = 2^{14}$	$n = 2^{16}$	$n = 2^{18}$	$n = 2^{20}$
LAN (≥ 10gbps)	KRTW19[36]	4.98	18.93	92.94	363.20
	GMR+21[20]	1.31	5.75	27.95	183.62
	JSZ+22[32]-R	2.74	11.98	57.93	371.17
	ZCL+22[53]-SKE	0.98	4.37	17.07	75.84
	ZCL+22[53]-PKE	3.04	12.66	53.81	223.82
	ZCL+22[53]-PKE*	2.48	10.36	45.10	195.50
	Our Protocol SA2	3.20	11.22	60.66	486.58
	Our Protocol SB2	3.20	11.22	47.37	213.31
	Our Protocol SA4	2.59	9.39	53.17	423.70
	Our Protocol SB4	2.59	9.39	38.82	176.24
	Our Protocol M2	2471.41	NA	NA	NA
	Our Protocol M4	4.62	17.70	77.52	323.09
1Gbps	KRTW19[36]	25.19	38.87	133.17	428.39
	GMR+21[20]	3.03	9.95	41.14	214.04
	JSZ+22[32]-R	4.29	15.44	68.57	407.17
	ZCL+22[53]-SKE	12.12	15.68	33.02	101.01
	ZCL+22[53]-PKE	3.55	13.75	57.21	242.86
	ZCL+22[53]-PKE*	3.18	11.96	49.19	203.08
	Our Protocol SA2	4.41	11.95	61.25	487.40
	Our Protocol SB2	4.41	11.95	49.63	219.51
	Our Protocol SA4	3.79	10.31	54.01	426.05
	Our Protocol SB4	3.79	10.31	41.27	183.75
	Our Protocol M2	2466.03	NA	NA	NA
	Our Protocol M4	5.12	19.58	79.85	332.85
100Mbps	KRTW19[36]	26.23	41.71	138.6	456.46
	GMR+21[20]	3.64	13.13	55.3	282.17
	JSZ+22[32]-R	4.78	17.82	81.74	423.92
	ZCL+22[53]-SKE	12.45	16.86	35.81	114.97
	ZCL+22[53]-PKE	3.65	14.47	59.24	242.84
	ZCL+22[53]-PKE*	3.27	12.64	53.87	218.15
	Our Protocol SA2	4.56	12.34	61.54	487.66
	Our Protocol SB2	4.56	12.34	51.06	224.54
	Our Protocol SA4	4.04	10.69	54.86	429
	Our Protocol SB4	4.04	10.69	43.15	190.16
	Our Protocol M2	2469.66	NA	NA	NA
	Our Protocol M4	5.59	19.87	82.56	340.9
10Mbps	KRTW19[36]	55.8	161.98	660.88	2643.94
	GMR+21[20]	14.44	60.45	257.5	1165.06
	JSZ+22[32]-R	12.76	54.18	237.65	1106.37
	ZCL+22[53]-SKE	15.4	32.53	103.57	405.14
	ZCL+22[53]-PKE	5.22	19.46	80.4	329.07
	ZCL+22[53]-PKE*	5.91	22.45	90.01	368.5
	Our Protocol SA2	13.18	30.65	98.82	522.03
	Our Protocol SB2	13.18	30.65	98.82	405.77
	Our Protocol SA4	8.97	22.19	80.74	469.52
	Our Protocol SB4	8.97	22.19	80.74	333.74
	Our Protocol M2	2484.63	NA	NA	NA
	Our Protocol M4	6.81	26.8	110.08	452.93

12. Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In *ACNS*, pages 125–142, 2009.
13. Alex Davidson and Carlos Cid. An efficient toolkit for computing private set operations. In Josef Pieprzyk and Suriadi Suriadi, editors, *ACISP 17, Part II*, volume 10343 of *LNCS*, pages 261–278. Springer, Heidelberg, July 2017.
14. Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *CCS*, pages 789–800, 2013.
15. Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 789–800. ACM Press, November 2013.
16. T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.
17. Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 303–324. Springer, Heidelberg, February 2005.
18. Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan Camenisch, editors, *EURO-CRYPT 2004*, volume 3027 of *LNCS*, pages 1–19. Springer, Heidelberg, May 2004.
19. Keith B. Frikken. Privacy-preserving set union. In Jonathan Katz and Moti Yung, editors, *ACNS 07*, volume 4521 of *LNCS*, pages 237–252. Springer, Heidelberg, June 2007.
20. 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, Heidelberg, May 2021.
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, Heidelberg.
22. Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*, volume 2. Cambridge University Press, 2009.
23. S. Dov Gordon, Carmit Hazay, and Phi Hung Le. Fully secure psi via mpc-in-the-head. Cryptology ePrint Archive, Paper 2022/379, 2022. https://eprint.iacr.org/2022/379.
24. Carmit Hazay. Oblivious polynomial evaluation and secure set-intersection from algebraic PRFs. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 90–120. Springer, Heidelberg, March 2015.
25. Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 155–175. Springer, Heidelberg, March 2008.
26. Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 312–331. Springer, Heidelberg, May 2010.
27. Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. Scalable multiparty private set-intersection. In Serge Fehr, editor, *PKC 2017, Part I*, volume 10174 of *LNCS*, pages 175–203. Springer, Heidelberg, March 2017.
28. 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.

29. Roi Inbar, Eran Omri, and Benny Pinkas. Efficient scalable multiparty private set-intersection via garbled bloom filters. In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 235–252. Springer, Heidelberg, September 2018.

30. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 406–425. Springer, Heidelberg, May 2011.

31. 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, Heidelberg, March 2009.

32. Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, Jiajun Du, and Dawu Gu. Shuffle-based private set union: Faster and more secure. In *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, August 2022. USENIX Association.

33. Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 241–257. Springer, Heidelberg, August 2005.

34. Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 818–829. ACM Press, October 2016.

35. Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1257–1272. ACM Press, October / November 2017.

36. 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, Heidelberg, December 2019.

37. Phi Hung Le, Samuel Ranellucci, and S. Dov Gordon. Two-party private set intersection with an untrusted third party. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2403–2420. ACM Press, November 2019.

38. Arjen K. Lenstra and Tim Voss. Information security risk assessment, aggregation, and mitigation. In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *ACISP 04*, volume 3108 of *LNCS*, pages 391–401. Springer, Heidelberg, July 2004.

39. Yehuda Lindell. How to simulate it - a tutorial on the simulation proof technique. Cryptology ePrint Archive, Report 2016/046, 2016. https://ia.cr/2016/046.

40. 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, Heidelberg, August 2020.

41. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.

42. Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from PaXoS: Fast, malicious private set intersection. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 739–767. Springer, Heidelberg, May 2020.

43. Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In Jaeyeon Jung and Thorsten Holz, editors, *USENIX Security 2015*, pages 515–530. USENIX Association, August 2015.

44. Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 122–153. Springer, Heidelberg, May 2019.

45. Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 797–812. USENIX Association, August 2014.

46. Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on ot extension. *ACM Trans. Priv. Secur.*, 21(2), jan 2018.

47. Michael O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 2005. https://ia.cr/2005/187.

48. Srinivasan Raghuraman and Peter Rindal. Blazing fast psi from improved okvs and subfield vole. Cryptology ePrint Archive, Paper 2022/320, 2022. https://eprint.iacr.org/2022/320.

49. Peter Rindal and Mike Rosulek. Malicious-secure private set intersection via dual execution. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1229–1242. ACM Press, October / November 2017.

50. Jennie Rogers, Elizabeth Adetoro, Johes Bater, Talia Canter, Dong Fu, Andrew Hamilton, Amro Hassan, Ashley Martinez, Erick Michalski, Vesna Mitrovic, Fred Rachman, Raj Shah, Matt Sterling, Kyra VanDoren, Theresa L. Walunas, Xiao Wang, and Abel Kho. VaultDB: A Real-World Pilot of Secure Multi-Party Computation within a Clinical Research Network. *arXiv e-prints*, page arXiv:2203.00146, February 2022.

51. Jae Hong Seo, Jung Hee Cheon, and Jonathan Katz. Constant-round multi-party private set union using reversed laurent series. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 398–412. Springer, Heidelberg, May 2012.

52. Nigel Smart and Frederik Vercauteren. Fully homomorphic simd operations. *IACR Cryptology ePrint Archive*, 2011:133, 01 2011.

53. Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin. Optimal private set union from multi-query reverse private membership test. Cryptology ePrint Archive, Report 2022/358, 2022. https://ia.cr/2022/358.

$\mathcal{F}_{\mathbf{OT}}$

Parameters: Sender $\mathcal{S}$, receiver $\mathcal{R}$.

Input: $\mathcal{S}$ inputs two messages m_0, m_1, $\mathcal{R}$ inputs $b \in \{0, 1\}$.

Output: $\mathcal{R}$ receives m_b, $\mathcal{S}$ receives $\perp$.
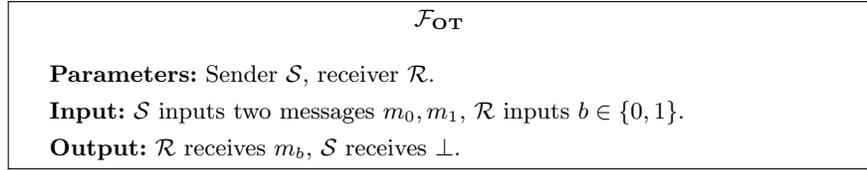
Fig. 7. Ideal Functionality of 1-out-of-2 OT.

Supplementary Materials:

A   Additional Preliminaries

A.1   Building Blocks

Permutation Hashing. We adopt the same approach described in [43] to reduce the effective length of the input items when hashing them into buckets. Let σ be the original length of the items, H a hash function, $m = 2^t$ the number of buckets. For an item x, we represent x as $x = x_L \| x_R$ where $|x_L| = t$ bits. Then the value x_R will be pushed into the bin $x_L \oplus H(x_R)$. The input length of the items in the buckets now becomes $\sigma - \log(m)$.

Oblivious Transfer. Oblivious transfer (OT) [47] is an important cryptographic primitive that allows a sender to transfer one of the two or more messages to the receiver, while remains oblivious as to which message is transferred. More in details, we give the ideal functionality for 1-out-of-2 OT in Figure 7.

A.2   Security Definition

Let $f = (f_1, f_2)$ be a probabilistic polynomial-time functionality. Let x denote the input of the first party, and y denote the input of the second party. After executing the functionality, the first party receives $f_1(x, y)$ and the second party receives $f_2(x, y)$. Let $\mathsf{view}_i^\pi(x, y, \kappa)$ be the view of the ith party ($i = \{1, 2\}$) during the execution of π on the inputs (x, y) and the security parameter κ. In particular, let $\mathsf{output}_i^\pi(x, y, \kappa)$ be the output of the ith party during the execution of π on inputs (x, y) and security parameter κ. Let $\mathsf{output}^\pi(x, y, \kappa)$ denote the output of the two parties.

Definition 2. *(Semi-honest security) Let $f = (f_1, f_2)$ be a functionality. We say π securely computes f in the presence of static semi-honest adversaries if there exist probabilistic polynomial-time simulators Sim_1 and Sim_2 such that*

$$\{(\mathsf{Sim}_1(1^\kappa, x, f_1(x, y)), f(x, y))\}_{x,y,\kappa} \overset{c}{\equiv} \{(\mathsf{view}_1^\pi(x, y, \kappa), \mathsf{output}^\pi(x, y, \kappa))\}_{x,y,\kappa}$$

$$\{(\mathsf{Sim}_2(1^\kappa, y, f_2(x, y)), f(x, y))\}_{x,y,\kappa} \overset{c}{\equiv} \{(\mathsf{view}_2^\pi(x, y, \kappa), \mathsf{output}^\pi(x, y, \kappa))\}_{x,y,\kappa}$$

where $x, y \in \{0, 1\}^*$ and $\kappa \in \mathbb{N}$.

Informally, the above guarantee that each party's view can be simulated using only its own input and output. Hence, an adversary corrupting either party cannot learn more than what it can learn from the party's input and output already.

Now we consider the malicious setting. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine and let $i \in \{1, 2\}$ be the corrupted party. The ideal execution of f on inputs (x, y), auxiliary input z to $\mathcal{A}$ and the security parameter κ, denoted by $\text{IDEAL}_{f,\mathcal{A}(z),i}(x, y, \kappa)$, is defined as the output pair of the honest party and the adversary $\mathcal{A}$ from the ideal execution, where the latter includes arbitrary PPT function of the prescribed input of the corrupted party, the auxiliary input z and $f_i(x', y')$, where x' (resp. y') may not equal to x if $i = 1$ (resp. y if $i = 2$). On the other hand, the real execution of π on inputs (x, y), auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted by $\text{REAL}_{\pi,\mathcal{A}(z),i}(x, y, \kappa)$ is defined as the output pair of the honest party and the adversary $\mathcal{A}$ from the real execution of π.

Definition 3. *(Malicious security) Let $f = (f_1, f_2)$ be a functionality. We say that π securely computes f with abort in the presence of static malicious adversaries if for every non-uniform PPT adversary $\mathcal{A}$ in the real world, there exists non-uniform PPT adversary $\mathcal{S}$ in the ideal world, such that for every $i \in \{1, 2\}$,*

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z),i}(x, y, \kappa)\right\}_{x,y,z,\kappa} \overset{c}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z),\mathcal{S}}(x, y, \kappa)\right\}_{x,y,z,\kappa}$$

where $x, y \in \{0,1\}^*$, $z \in \{0,1\}^*$ and $\kappa \in \mathbb{N}$.

Finally we also consider a hybrid world that allows π to call some trusted party to compute some ideal functionalities $g_1, \cdots, g_{p(n)}$. We denote the hybrid execution as $\text{HYBRID}_{\pi,\mathcal{A}(z),a}^{g_1,\cdots,g_{p(n)}}(x, y, \kappa)$.

Definition 4. *(Malicious security in the hybrid model) Let $f = (f_1, f_2)$ be a functionality. We say π securely computes f with abort in the $g_1, \cdots, g_{p(n)}$-hybrid model, in the presence of static malicious adversaries if for every non-uniform PPT adversary $\mathcal{A}$ in the real world, there exists a non-uniform PPT adversary $\mathcal{S}$ in the ideal world, such that for every $i \in \{1, 2\}$,*

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z),i}(x, y, \kappa)\right\}_{x,y,z,\kappa} \overset{c}{\equiv} \left\{\text{HYBRID}_{\pi,\mathcal{A}(z),\mathcal{S}}^{g_1,\cdots,g_{p(n)}}(x, y, \kappa)\right\}_{x,y,z,\kappa}$$

where $x, y \in \{0,1\}^*$, $z \in \{0,1\}^*$ and $\kappa \in \mathbb{N}$.

A.3   Re-Randomizable Public Key Encryption Schemes

A Re-randomizable public key encryption scheme is a tuple of five PPT algorithms:

- Setup(1^κ): The setup algorithm takes the security parameter 1^κ and outputs the public parameters pp, which includes the description of the message and ciphertext spaces $\mathcal{M}, \mathcal{C}$.

- KeyGen(pp): The key generation algorithm takes pp and output a pair of public and private key (pk, sk).
- $\mathsf{Enc_{pk}}(m)$: The encryption algorithm encrypts a message $m \in \mathcal{M}$ with pk to a ciphertext $c \in \mathcal{C}$.
- $\mathsf{Dec_{sk}}(c)$: The decryption algorithm decrypts a ciphertext $c \in \mathcal{C}$ with sk to a message $m \in \mathcal{M}$ or a failure symbol $\perp$.
- $\mathsf{ReRand_{pk}}(c)$: The re-randomization algorithm re-randomize a ciphertext $c \in \mathcal{C}$ with pk to another ciphertext $c' \in \mathcal{C}$. We also use the notation $\mathsf{ReRand_{pk}}(c, r)$ to de-randomize this alogrithm, in which r is the randomness used to re-randomize.

Correctness. A PKE scheme is re-randomization correct if for any $pp \leftarrow \mathsf{Setup}(1^\kappa)$, any $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(pp)$, and any $m \in M$, any $c \leftarrow \mathsf{Enc_{pk}}(m)$ and $c' \leftarrow \mathsf{ReRand_{pk}}(c)$, we have $\mathsf{Dec_{sk}}(c) = \mathsf{Dec_{sk}}(c') = m$.

Re-randomization Indistinguishability. A PKE scheme is re-randomization indistinguishable if for any $pp \leftarrow \mathsf{Setup}(1^\kappa)$, any $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(pp)$, and any $m \in M$, the distribution of $c \leftarrow \mathsf{Enc_{pk}}(m)$ and $c' \leftarrow \mathsf{ReRand_{pk}}(c)$ are identical.

Statistically re-randomization Indistinguishability. A PKE scheme is statistically re-randomization indistinguishable if for any $pp \leftarrow \mathsf{Setup}(1^\kappa)$, any $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(pp)$, and any $m \in M$, the distribution of c'_1 and c'_2, where $c_1 \leftarrow \mathsf{Enc_{pk}}(m), c'_1 \leftarrow \mathsf{ReRand_{pk}}(c)$ and $c_2 \leftarrow \mathsf{Enc_{pk}}(m), c'_2 \leftarrow \mathsf{ReRand_{pk}}(c)$, are statistically indistinguishable.

Similar to [53], we define an extra property for some re-randomizable PKE schemes, which comes in handy when we prove the security for some of our protocols:

(Optional) Single-message multi-ciphertext pseudorandomness. Roughly speaking, we require that a vector of n ciphertexts encrypting the same plaintext be indistinguishable from n random values in the ciphertext space. More formally, a PKE scheme is single-message multi-ciphertext pseudorandom if for any PPT $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:

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

is negligible in κ.

In particular, it suffices to show that a PKE satisfies the above definition for $n = 1$, i.e., single-ciphertext pseudorandomness, and the case with generic n can be shown via a standard hybrid argument.

An instantiation that we used in our paper is the ElGamal [16] encryption scheme. It is straightforward to show that it satisfies both correctness and re-randomization indistinguishability. Additionally, it also satisfies single-message

multi-ciphertext pseudorandomness, which can be proved by reducing to the hardness of DDH problem.

A.4   Additively Homomorphic Public Key Encryption Schemes

On top of re-randomizable PKE schemes, we define additively homomorphic PKE schemes by requiring them to satisfy the following additively homomorphic properties:

- There is a homomorphic addition operation $\boxplus$, such that for any m, m', any $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m)$, $c' \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m')$, we have $\mathsf{Dec}(c \boxplus c') = m + m'$.

In this paper, we usually just use $+$ instead of $\boxplus$ when the context is clear. This additively homomorphic property also allows us to compute $k \cdot c$, where k is a scalar and c is a ciphertext. In particular, we have $\mathsf{Dec}_{\mathsf{sk}}(k \cdot c) = k \cdot \mathsf{Dec}_{\mathsf{sk}}(c)$.

An instantiation that we use is the Paillier [41] encryption scheme. It satisfies correctness and re-randomization indistinguishability, and the additive homomorphic property. Additionally, it also satisfies single-message multi-ciphertext pseudorandomness, which can be proved by a reduction to the hardness of decisional composite residuosity.

The other instantiation we use is BGV Encryption Scheme [6]. It satisfies correctness and statistical re-randomization indistinguishability (using noise flooding), and the additive homomorphic property. Plus, it also satisfies single-message multi-ciphertext pseudorandomness, due to the hardness of RLWE.

B   Proofs of security in the malicious setting

B.1   Proof of Theorem 3

Claim. Our protocol is secure against any malicious sender.

Proof. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine. The execution of $\Pi_{\mathrm{PSU}}^{*\mathsf{reuse}}$ on inputs (X, Y), auxiliary input z to $\mathcal{A}$ and computational security parameter κ, denoted by $\mathrm{HYBRID}_{\Pi, \mathcal{A}(z)}(X, Y, \kappa)$, is defined as the view of the adversary $\mathcal{A}$ and the output of honest $\mathcal{R}$ from the hybrid execution. On the other hand, the ideal execution of $\mathcal{F}_{\mathrm{PSU}}^{\hat{n}_1, n_2}$ on inputs (X, Y), auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted by $\mathrm{IDEAL}_{\mathcal{F}, \mathsf{Sim}_{\mathcal{S}}(z)}(X, Y, \kappa)$, is defined as the view of the adversary $\mathcal{A}$ and the output of honest $\mathcal{R}$ from the ideal execution.

Note that OKVS successfully encodes the input set except for $2^{-\lambda}$ probability. Therefore in our analysis, we simply neglect the failure event. To show that our protocol is secure against malicious $\mathcal{S}$, we show that for any adversary $\mathcal{A}$ corrupting $\mathcal{S}$, there exists a simulator $\mathsf{Sim}_{\mathcal{S}}$ (Figure 8) such that
$$\left\{ \mathrm{HYBRID}_{\Pi, \mathcal{A}(z)}(X, Y, \kappa) \right\}_{X, Y, z, \kappa} \overset{c}{\equiv} \left\{ \mathrm{IDEAL}_{\mathcal{F}, \mathsf{Sim}_{\mathcal{S}}(z)}(X, Y, \kappa) \right\}_{X, Y, z, \kappa}$$
where each entry in X (resp., Y) is a distinct field element in $\mathbb{F}$, $z \in \{0, 1\}^*$ and $\kappa \in \mathbb{N}$. Let rt and V (resp., $\widetilde{rt}$ and $\widetilde{V}$) denote the random tape and the random oracle's response in the hybrid world (resp., ideal world). We have

$$\left\{ \text{HYBRID}_{\Pi,\mathcal{A}(z)}\left(X,Y,\kappa\right)\right\}_{X,Y,z,\kappa} = \left\{X, rt, \mathsf{pk}, D, V, \hat{X} \cup Y\right\}_{X,Y,z,\kappa}$$

$$\left\{ \text{IDEAL}_{\mathcal{F},\mathsf{Sim}_{\mathcal{S}}(z)}(X,Y,\kappa)\right\}_{X,Y,z,\kappa} = \left\{X, \widetilde{rt}, \widetilde{\mathsf{pk}}, \widetilde{D}, \widetilde{V}, \widetilde{X} \cup Y\right\}_{X,Y,z,\kappa}$$

To argue the above probability ensembles are computationally indistinguishable, we fix any X, Y, z, κ and consider the following sequence of hybrid joint distributions:

Hyb_0: Same as $\text{IDEAL}_{\mathcal{F},\mathsf{Sim}_{\mathcal{S}}(z)}(X,Y,\kappa)$.

Hyb_1: Same as Hyb_0, except that the simulator is given the honest $\mathcal{R}$'s input $Y = \{y_1, \ldots, y_{n_2}\}$. In Step 3 of $\mathsf{Sim}_{\mathcal{S}}$, it computes $\widetilde{D}' \leftarrow \mathsf{Encode}(\{(y_i, \widetilde{c}_i)\}_{i \in [n_2]})$ instead and sends $\widetilde{D}'$ to $\mathcal{A}$, where $\widetilde{c}_i$ is uniformly sampled from the ciphertext space, just as in Hyb_0.

Hyb_2: Same as Hyb_1, except that the simulator follows $\mathcal{R}$'s step to generate n_2 fresh ciphertexts $c_1, \ldots, c_{n_2}$ of 0. Then, it computes $D \leftarrow \mathsf{Encode}(\{(y_i, c_i)\}_{i \in [n_2]})$ and sends D to $\mathcal{A}$.

Hyb_3: Same as Hyb_2, except that the simulator uses the secret key $\widetilde{\mathsf{sk}}$ to decrypt the ciphertext tuples and extract the inputs (rather than through the random oracle), which is the same to how $\mathcal{R}$ recovers the inputs in the hybrid world. Note that Hyb_3 is identical to $\text{HYBRID}_{\Pi,\mathcal{A}(z)}(X,Y,\kappa)$ (the simulator essentially acts according to honest $\mathcal{R}$'s protocol), except that rt, pk, V are replaced by $\widetilde{rt}, \widetilde{\mathsf{pk}}, \widetilde{V}$. Nevertheless, they are only semantically different, and that their distributions are exactly the same. Also, in both hybrids, abort is triggered when more than $\hat{n}_1$ ciphertexts are returned by $\mathcal{A}$.

We start by arguing that Hyb_0 and Hyb_1 are computationally indistinguishable. In particular, we show that if there exists some PPT distinguisher $\mathcal{D}$ that distinguishes Hyb_0 and Hyb_1 with non-negligible probability, there exists an adversary $\mathcal{A}_{\mathsf{obl}}$ that breaks the obliviousness property of OKVS:

1. The challenger $\mathcal{C}$ computes $pp \leftarrow \mathsf{Setup}(1^\kappa)$, $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{KeyGen}(pp)$ and sends $\widetilde{\mathsf{pk}}$ to $\mathcal{A}_{\mathsf{obl}}$.[9] $\mathcal{A}_{\mathsf{obl}}$ sends $Y_0 = \{\widetilde{y}_i\}_{i \in n_2}$ and $Y_1 = Y$ to the challenger $\mathcal{C}$, where $\widetilde{y}_i$ are random input values, as generated by the simulator in Hyb_0. $\mathcal{C}$ uniformly picks $b \leftarrow \{0,1\}$, parses Y_b as $\{y_i^b\}_{i \in n_2}$ and uniformly samples $\{c_i^b\}_{i \in n_2}$ from the ciphertext space. Finally, $\mathcal{C}$ computes and returns $D_b \leftarrow \mathsf{Encode}(\{(y_i^b, c_i^b)\}_{i \in n_2})$.

2. $\mathcal{A}_{\mathsf{obl}}$ employs $\mathcal{A}$ on X and a uniformly picked $\widetilde{rt}$. It then sends $\widetilde{\mathsf{pk}}$ and D_b to $\mathcal{A}$. Similarly to Step 4 of $\mathsf{Sim}_{\mathcal{S}}$, it emulates the random oracle, receiving ciphertext tuples and extracting $\mathcal{A}$'s "effective" input set $\widetilde{X}$. Finally, it computes honest $\mathcal{R}$'s output $\widetilde{X} \cup Y$ and sends the joint view $(X, \widetilde{rt}, \widetilde{\mathsf{pk}}, D_b, \widetilde{V}, \widetilde{X} \cup Y)$ to $\mathcal{D}$. Note that when $b = 0$, the joint view follows the joint distribution in Hyb_0. Otherwise, it follows the joint distribution in Hyb_1.

3. $\mathcal{A}_{\mathsf{obl}}$ uses $\mathcal{D}$'s outcome to decide whether D_b is built from Y_0 or Y_1. It responds to $\mathcal{C}$ accordingly and wins with non-negligible probability.

---

[9] For this reduction, we do not distinguish whether $\mathcal{C}$ or $\mathcal{A}_{\mathsf{obl}}$ choose the key pairs as $\widetilde{\mathsf{sk}}$ is not used in either Hyb_0 or Hyb_1.

Next, we show that hybrids Hyb_1 and Hyb_2 are computationally indistinguishable. Similar to the reduction above, we show that if there exists some PPT distinguisher $\mathcal{D}$ that distinguishes Hyb_1 and Hyb_2 with non-negligible probability, there exists a PPT adversary $\mathcal{A}_{\mathsf{pseu}}$ that breaks the single-message multi-ciphertexts pseudorandomness property of the underlying PKE scheme:

1. The challenger $\mathcal{C}$ computes $pp \leftarrow \mathsf{Setup}(1^\kappa)$, $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{KeyGen}(pp)$ and sends $\widetilde{\mathsf{pk}}$ to $\mathcal{A}_{\mathsf{pseu}}$. $\mathcal{A}_{\mathsf{pseu}}$ sends 0 to $\mathcal{C}$. $\mathcal{C}$ uniformly picks $b \leftarrow \{0,1\}$. If $b=0$, $\mathcal{C}$ uniformly samples $\{c_i^0\}_{i \in [n_2]}$ from the ciphertext space. Otherwise, $\mathcal{C}$ generates n_2 fresh ciphertexts of 0, i.e., $\{c_i^1\}_{i \in [n_2]}$ where $c_i^1 \leftarrow \mathsf{Enc}_{\widetilde{\mathsf{pk}}}(0)$. Then, $\mathcal{C}$ sends $\{c_i^b\}_{i \in [n_2]}$ to $\mathcal{A}_{\mathsf{pseu}}$.

2. $\mathcal{A}_{\mathsf{pseu}}$ employs $\mathcal{A}$ on input X and a uniformly picked $\widetilde{rt}$. Next, it computes $D_b \leftarrow \mathsf{Encode}(\{(y_i, c_i^b)\}_{i \in n_2})$ and sends $\widetilde{\mathsf{pk}}$ and D_b to $\mathcal{A}$. Similarly to Step 4 of $\mathsf{Sim}_\mathcal{S}$, it emulates the random oracle, receiving ciphertext tuples and extracting $\mathcal{A}$'s "effective" input set $\widetilde{X}$. Finally, it computes the honest $\mathcal{R}$'s output $\widetilde{X} \cup Y$ and sends the joint view $(X, \widetilde{rt}, \widetilde{\mathsf{pk}}, D_b, \widetilde{V}, \widetilde{X} \cup Y)$ to $\mathcal{D}$. Note that when $b=0$, the joint view follows the joint distribution as in Hyb_1. Otherwise, it follows the joint distribution as in Hyb_2.

3. $\mathcal{A}_{\mathsf{pseu}}$ uses $\mathcal{D}$'s outcome to decide whether the view is according to Hyb_1 or Hyb_2, and uses the outcome to answer $\mathcal{C}$ and wins the above game with non-negligible probability.

Moreover, we show that Hyb_2 and Hyb_3 are statistically indistinguishable. Notice that X, $\widetilde{rt}$, $\widetilde{\mathsf{pk}}$, and D are identically distributed in the two hybrids. Thus, $\mathcal{A}$'s response (random oracle queries and ciphertext tuples) follows the same distribution in both hybrids. As a result, it suffices to analyze a fixed $\mathcal{A}$'s response. First notice that the simulator's responses to the random oracle queries are identically distributed in the two hybrids. It remains to argue that the honest $\mathcal{R}$'s output $\widetilde{X} \cup Y$ in Hyb_2 is equal to $\hat{X} \cup Y$ in Hyb_3, except for negligible probability. Recall that in Hyb_3, the simulator follows $\mathcal{R}$'s step to extract the sender's inputs. Specifically, for each tuple (e, e', e'') sent by $\mathcal{A}$, $\mathcal{R}$ decrypts them to (u_i, u_i', u_i''). If $u_i \neq 0$, it recovers $x = u_i'/u_i$ and $r = u_i''/u_i$. At the end, it verifies that (e, e', e'') are correctly computed with x and r, i.e., it emulates $\mathcal{S}$'s step to see whether it generated (e, e', e''). If so, x is included in $\hat{X}$. On the other hand, in Hyb_2, the simulator recovers $\widetilde{X}$ by parsing each query made to the random oracle into $\widetilde{x}$ and $\widetilde{r}$, and emulating $\mathcal{S}$'s steps to generate the ciphertext tuple. Similarly, it only includes $\widetilde{x}$ in $\widetilde{X}$ if the generated tuple matches one of the tuples in $\mathcal{A}$'s response. For analysis purpose, we further partition $\widetilde{X}$ in Hyb_2 to $\widetilde{X}_1$ and $\widetilde{X}_2$ where the former contains all values x such that the first entry of its corresponding tuple decrypts to a non-zero value. Notice that as long as we show that both $\widetilde{X}_1 = \hat{X}$ and $\widetilde{X}_2 \subseteq Y$ except with negligible probability, we have $\widetilde{X} \cup Y = (\widetilde{X}_1 + \widetilde{X}_2) \cup Y = \widetilde{X}_1 \cup Y = \hat{X} \cup Y$ except with negligible probability and our proof for the indistinguishability between Hyb_2 and Hyb_3 is done.

First, we argue that $\widetilde{X}_1 = \hat{X}$ except with negligible probability. For simplicity, we assume that the PKE scheme has perfect decryption correctness[10]. We start by showing that $\widetilde{X}_1 \subseteq \hat{X}$ except with negligible probability. For each $x \in \widetilde{X}_1$, there is a string r (queried together with x to the random oracle), and a ciphertext tuple in $\mathcal{A}$'s response that can be generated with x and r following $\mathcal{S}$'s step. In Hyb_3, this tuple correctly decrypts to (u_i, u'_i, u''_i) where the same x and r can be recovered from u'_i/u_i and u''_i/u_i, (recall that $u_i \neq 0$ by the definition of $\widetilde{X}_1$). Furthermore, as guaranteed by the definition of $\widetilde{X}$, the ciphertext tuple can be correctly generated from x and r following $\mathcal{S}$'s step. Therefore, x is also included in $\hat{X}$.

Next, we show that $\hat{X} \subseteq \widetilde{X}_1$ except with negligible probability. For each $x \in \hat{X}$, there is a string r and a ciphertext tuple in $\mathcal{A}$'s response such that the ciphertext tuple can be generated with x and r following $\mathcal{S}$'s step. As $\mathcal{A}$ has a negligible probability of guessing the outcome of the random oracle, there must be a query of $x\|r$ to the random oracle with overwhelming probability. In Hyb_2, the simulator parses this query into $x\|r$ and generates the matching ciphertext tuple following $\mathcal{S}$'s step. Therefore, x is also included in $\widetilde{X}_1$.

Second, we argue $\widetilde{X}_2 \subseteq Y$, i.e. all inputs in $\widetilde{X}_2$ are in the intersection, except with negligible probability. For each $x \in \widetilde{X}_2$, the first entry of its corresponding (correctly formed) tuple is a ciphertext re-randomized from $\mathsf{Decode}(D, x)$, which is an encryption of 0 by how we defined $\widetilde{X}_2$. Due to the randomness property of OKVS, for any input $x \notin Y$, $\mathsf{Decode}(D, x)$ is an encryption of 0 only with negligible probability. Therefore, $\widetilde{X}_2 \subseteq Y$ except with negligible probability. This concludes the proof of indistinguishability between Hyb_2 and Hyb_3.

Claim. Our protocol is secure against any semi-honest receiver.

Proof. As $\mathcal{F}_{\mathrm{PSU}}^{\hat{n}_1, n_2}$ is deterministic and we want to show security against a semi-honest $\mathcal{R}$, it suffices to separately show (a) correctness: the protocol's output is correct except for negligible probability, and (b) privacy: $\mathcal{R}$'s view in the protocol can be simulated using its own input and output.

To see our protocol is correct, first notice that Encode only fails with $2^{-\lambda}$ probability and we simply neglect the failure event. Then, in Step 4 of the protocol, for each (e, e', e'') corresponding to a $x \in X$, if $x \notin Y$, there is only a negligible probability that e decrypts to 0. This is due to the randomness property of the OKVS, which requires e to be statistically indistinguishable from a random element in the ciphertext space. As a result, $\mathcal{R}$ can recover x and r from the tuple except for negligible probability. Therefore, $\hat{X} = X \setminus Y$ except for negligible probability so $\hat{X} \cup Y = X \cup Y$.

To see our protocol is private, we define the following simulator $\mathsf{Sim}_{\mathcal{R}}(Y, X \cup Y)$ to generate the view for the semi-honest $\mathcal{R}$:

1. The simulator internally runs the adversary $\mathcal{A}$ on input Y and a uniformly picked rt.

---

[10] In fact, it suffices that the PKE scheme decrypts correctly except for negligible probability. The adversary cannot take advantage of this as the randomness used in a correctly re-randomized ciphertext tuple comes from the random oracle.

<div align="center">

Simulator $\mathsf{Sim}_{\mathcal{S}}$

</div>

1. **Initialize.** The simulator internally runs the adversary $\mathcal{A}$. It starts with writing X on $\mathcal{A}$'s input tape and a uniformly picked rt on $\mathcal{A}$'s random tape.
2. **KeyGen.** The simulator computes $pp \leftarrow \mathsf{Setup}(1^{\kappa}), (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{KeyGen}(pp)$.
3. **Encode.** The simulator randomly generates key-value pairs $\{(\widetilde{y}_i, \widetilde{c}_i)\}_{i \in [n_2]}$. Specifically, $\{\widetilde{y}_1, \ldots, \widetilde{y}_{n_2}\}$ are uniformly picked distinct field elements from $\mathbb{F}$ and each $\widetilde{c}_i$ is uniformly picked from the ciphertext space. It then computes $\widetilde{D} \leftarrow \mathsf{Encode}(\{(\widetilde{y}_i, \widetilde{c}_i)\}_{i \in [n_2]})$ and sends $\widetilde{\mathsf{pk}}, \widetilde{D}$ to $\mathcal{A}$.
4. **Extract input.** The simulator emulates the random oracle to receive and reply the queries from $\mathcal{A}$. Specifically, the simulator maintains a dictionary of all past queries and the values it returns. For a new query $\widetilde{x}||\widetilde{r}$, the simulator uniformly picks a value from the randomness space ∇^3 to reply and writes down the query value pair in the dictionary. For a repeated query, the simulator looks it up in the dictionary and replies with the same value. Next, the simulator receives $\{(\widetilde{e}_i, \widetilde{e'}_i, \widetilde{e''}_i)\}_{i \in [n_1]}$ from $\mathcal{A}$. If $n_1 > \hat{n}_1$, it signals the ideal functionality to abort (can be triggered by sending an arbitrary input set with size n_1) and terminates the simulation. For each $\widetilde{x}||\widetilde{r}$ stored in the dictionary, it emulates $\mathcal{A}$'s steps to decode and build the ciphertext tuple, followed by re-randomizing the ciphertext tuple using the value corresponding to $\widetilde{x}||\widetilde{r}$ in the dictionary. If the resulting tuple is included in $\{(\widetilde{e}_i, \widetilde{e'}_i, \widetilde{e''}_i)\}_{i \in [n_1]}$, the simulator puts $\widetilde{x}$ in $\widetilde{X}$.
5. **Generate output.** The simulator submits $\widetilde{X}$ to the ideal functionality.

Fig. 8. Simulator for corrupted $\mathcal{S}$.

2. The simulator receives pk and an OKVS D from $\mathcal{A}$. Next, it computes $\bar{X} = X \cup Y \setminus Y$, i.e., the set of extra elements that $\mathcal{S}$ adds to the union. For all $x \in \bar{X}$, the simulator follows Step 3 of the protocol to decode and re-randomize them to $\{(e_i, e_i', e_i'')\}_{i \in [|\bar{X}|]}$. In particular, to re-randomize the ciphertext tuples, the simulator emulates the random oracle by maintaining a dictionary of all $\widetilde{x}||\widetilde{r}$ and their corresponding random values it picks to re-randomize. Also, it generates additional tuples $\{(e_i, e_i', e_i'')\}_{|\bar{X}|+1 \leq i \leq n_1}$, where each tuple is generated from $e_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$, $e_i' \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$, $e_i'' \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$. Finally, it sends $\{e_i, e_i', e_i''\}_{i \in [|\bar{X}|]} \cup \{e_i, e_i', e_i''\}_{|\bar{X}|+1 \leq i \leq n_1}$ in a random order to $\mathcal{A}$.

3. The simulator replies $\mathcal{A}$'s queries to the random oracle according to the dictionary.

Clearly, Y and rt are identical to those in a real execution. We start by arguing that the ciphertext tuples $\{(e_i, e_i', e_i'')\}_{i \in [|\bar{X}|]}$ are indistinguishable from those in a real execution. Specifically, for each ciphertext tuple corresponds to a $x \in \bar{X}$, the simulator generates it by following the exact protocol of $\mathcal{S}$ in the real execution. Next, we argue that the remaining ciphertext tuples (corresponding to the intersected inputs) are also identically distributed to those in a real execution. In both hybrids, these ciphertext tuples are encryptions of 0, and since the randomness used to re-randomized the ciphertexts is hidden from $\mathcal{A}$, each ciphertext is (statistically) indistinguishable from another re-randomized cihpertext of 0 due to the (statistical) re-randomization indistinguishability property. Finally, it is straightforward to see that the simulator perfectly emulates the random oracle so the responses to the random oracle queries are perfectly distinguishable from those in the real execution. This concludes our proof.

B.2   Proof of Theorem 4

Since the proof is largely similar to the proof of Theorem 3 above, we mainly highlight the differences here.

Claim. Our protocol is secure against any malicious sender.

Proof. To show that our protocol is secure against malicious $\mathcal{S}$, we show that for any adversary $\mathcal{A}$ corrupting $\mathcal{S}$ and interacting with the functionality $\mathcal{F}_{\mathrm{OT}}$, there exists a simulator $\mathsf{Sim}_\mathcal{S}$ such that

$$\left\{\mathrm{HYBRID}_{\Pi, \mathcal{A}(z)}^{\mathcal{F}_{\mathrm{OT}}}(X, Y, \kappa)\right\}_{X,Y,z,\kappa} \stackrel{c}{\equiv} \left\{\mathrm{IDEAL}_{\mathcal{F}, \mathsf{Sim}_\mathcal{S}(z)}(X, Y, \kappa)\right\}_{X,Y,z,\kappa}$$

First, notice that the adversary's view contains exactly the same entries as in the previous proof. Therefore, we basically use the same hybrids as before to argue the indistinguishability among them. The only difference occurs when we argue Hyb_2 and Hyb_3 are indistinguishable, as both the simulator and the honest $\mathcal{R}$ recover input in slightly different ways as before. Specifically, in Hyb_2, the simulator still extracts potential inputs and their corresponding randomness in the form of $\widetilde{x}||\widetilde{r}$ from the random oracle queries. Next, it verifies that decoding the OKVS D with $\widetilde{x}$ and re-randomizing the resulting ciphertext with $h(\widetilde{x}||\widetilde{r})$ yields a ciphertext in the response from $\mathcal{A}$. Additionally, the simulator, emulated

as $\mathcal{F}_{\mathrm{OT}}$, receives a pair of OT messages corresponding to $\widetilde{x}$. It verifies that the second message is $\widetilde{x}||\widetilde{r}$. If both verifications pass, the simulator includes $\widetilde{x}$ in $\widetilde{X}$. In Hyb_3, $\mathcal{R}$ decrypts each e_i sent from $\mathcal{S}$ to recover a value u. If $u \neq w$, i.e., it indicates the underlying input is not in the intersection, $\mathcal{R}$ will try to retrieve the corresponding $x||r$ through $\mathcal{F}_{\mathrm{OT}}$. To verify, it use x and r to verify that e is correctly generated from decoding D at x and re-randomizing the ciphertext with $h(x||r)$. If the verification passes, it places x in $\hat{X}$.

Similar to the previous proof, we partition $\widetilde{X}$ in Hyb_2 to $\widetilde{X}_1$ and $\widetilde{X}_2$ where the former contains all values x such that the corresponding ciphertext e decrypts to a non-w value. As before, it suffices to show that both $\widetilde{X}_1 = \hat{X}$ and $\widetilde{X}_2 \subseteq Y$ except with negligible probability. First, we argue that $\widetilde{X}_1 = \hat{X}$ except with negligible probability. For simplicity, we assume that the PKE scheme has perfect decryption correctness as before. We start by showing that $\widetilde{X}_1 \subseteq \hat{X}$ except with negligible probability. For each $x \in \widetilde{X}_1$, there is a string r (queried together with x to the random oracle), a corresponding ciphertext e in $\mathcal{A}$'s response that can be correctly generated with $\widetilde{x}$ and $\widetilde{r}$ following $\mathcal{S}$'s step, and a corresponding OT message $x||r$. In Hyb_3, e correctly decrypts to some non-w value and the corresponding x and r can be recovered from the corresponding OT message. Furthermore, as guaranteed by the definition of $\widetilde{X}$, the ciphertext e can be correctly generated from x and r following $\mathcal{S}$'s step. Therefore, x is also included in $\hat{X}$. Similar to before, $\hat{X} \subseteq \widetilde{X}_1$ except with negligible probability as $\mathcal{A}$ has a negligible probability of guessing the outcome of the random oracle. Second, to argue $\widetilde{X}_2 \subseteq Y$, i.e. all inputs in $\widetilde{X}_2$ are in the intersection, except with negligible probability, we similarly rely on the randomness property of OKVS. In particular, each correctly formed ciphertext is a ciphertext re-randomized from $\mathsf{Decode}(D, x)$, which is an encryption of w by the definition of $\widetilde{X}_2$. Due to the randomness property of OKVS, for any input $x \notin Y$, $\mathsf{Decode}(D, x)$ is an encryption of w only with negligible probability. This concludes the proof of indistinguishability between Hyb_2 and Hyb_3.

Claim. Our protocol is secure against any semi-honest receiver.

Proof. Similar to the previous proof, it suffices to separately show (a) correctness: the protocol's output is correct except for negligible probability, and (b) privacy: $\mathcal{R}$'s view in the protocol can be simulated using its own input and output.

We start by showing correctness. In Step 4 of the protocol, for each e_i corresponding to x_i, if $x_i \notin Y$, there is only a negligible probability that e_i decrypts to w. This is due to the randomness property of the OKVS, which requires e to be statistically indistinguishable from a random element in the ciphertext space. As a result, $\mathcal{R}$ requests $x_i||r_i$ through OT functionality except for negligible probability. Therefore, $\hat{X} = X \setminus Y$ except for negligible probability and $\hat{X} \cup Y = X \cup Y$.

To see that the privacy holds, we define the simulator $\mathsf{Sim}_{\mathcal{R}}(Y, X \cup Y)$ similar to the one we used in the previous proof. One difference is that for each input, the simulator now sends a single ciphertext, as opposed to a ciphertext tuple. The other difference is that the simulator emulates as $\mathcal{F}_{\mathrm{OT}}$ and thus needs to send one of the OT messages to the semi-honest $\mathcal{R}$.

We similarly argue that these single ciphertexts are indistinguishable between the ideal and hybrid worlds. First, $\{(e_i)\}_{i \in [|\bar{X}|]}$ are indistinguishable from those in the hybrid world. Specifically, for each ciphertext that corresponds to a $x \in \bar{X}$, the simulator generates it by following the exact protocol of $\mathcal{S}$ in the hybrid world. Next, we argue that the remaining ciphertexts (corresponding to the intersected inputs) are also identically distributed to those in the hybrid world. In both hybrids, these ciphertext tuples are encryptions of w. Since the randomness used to re-randomize the ciphertexts are hidden from $\mathcal{A}$ in hybrid worlds (they are not recovered from the OT), each ciphertext is (statistically) indistinguishable from a fresh re-randomized ciphertext of w in the ideal world due to the (statistically) re-randomization indistinguishability property. Finally, it is straightforward to see that the simulator perfectly emulates the OT by simply sending the correct message. This concludes our proof.