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.com
[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 two protocols. Our four-round protocol out-performs the state-of-the-art in both communication and computation, and has a runtime that is 1.3X-2X faster than existing protocols. Our two-round protocol is only slightly more expensive, but it is still faster than existing protocols and has the property that the receiver message can be re-used across multiple executions. All our semi-honest protocols are post-quantum secure.

In the setting where the sender is malicious, we provide the first protocols that avoid the use of expensive zero-knowledge proofs. We estimate (conservatively) that our constructions are less than 2X more expensive than the best known *semi-honest constructions.* As in the semi-honest setting, we describe two protocols: a faster one that requires four rounds of communication, and a slightly 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 PSU.

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

1 Introduction

As the field of secure computation transitions to practice, a lot of 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) [28,9,23,21,24,14,22,26,38,8,7,34,20,11,39,37,29,42,36,32]. This is partly because the computation has such broad application, and partly because it is so amenable to custom techniques.

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, but without revealing the intersecting elements. This is a natural

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

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 of Two-party PSU. (Upper bound for sender.)

next step from PSI: some very interesting techniques have been developed in the study of PSI, and while not all of them will extend to find application in broad classes of computation, many of them will likely extend to other private computations on sets.

1.1 Defining PSU

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 things to 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 here. 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. [15]. We note that there may be applications where we prefer not to leak the exact input set size, or the exact intersection size. Our relaxed ideal functionality could be an asset in such scenarios. Finding an efficient realization of the functionality that enforces a fixed input size is an interesting direction for future work.
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, since the corrupted receiver is assumed

to use its true input, which may be correlated with the sender input. 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.

In lieu of this, it is tempting to consider relaxing the functionality as we did with sender security, parameterizing with only an upper bound on the receiver set size, with the aim of claiming security against a malicious receiver. However, this functionality admits trivial solutions, just as the current functionality does in the large domain setting; if the receiver (or simulator) uses the empty-set, they recover the full input of the sender. Finally, we note that the current discussion assumes that only the receiver has output. If both parties receive output, then this discussion relating to a malicious receiver applies to both parties. In that case, there are only two security models that make sense: semi-honest security for both parties, or restricting to small domain, and requiring both parties to prove that their input sets are sufficiently large.

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 inputs 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 protocol, and a two-round protocol. The four-round protocols have the 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. Our four-round semi-honest construction out-performs the state-of-the-art by $1.3\times-2\times$ in LAN network with the bandwidth of 10Gbps, and by $1.5\times-1.8\times$ in a 10 Mbps network. We achieve 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, as 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 constitute the state-of-the-art. 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. [15], 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* [30,38,6,5]. Other recent constructions have moved away from using polynomials to encode the receiver input, using garbled Bloom filters, [12,39,25], or cuckoo hashing [36,40,38]. More recently, Garimella et al. [18] generalized the basic approach described above by defining the notion of an *oblivious key value store*. 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 decode 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 sender and receiver perform secure comparisons of the values to determine whether it is an intersected input.

Very recently, Zhang et al. [47] 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 sends the resulting ciphertexts back to the receiver. If $x \in Y$, this ciphertext will decrypt to w, revealing nothing more than the fact that $x \in X \cap Y$. On the other hand, when $x \notin Y$, their instantiation of OKVS ensures that the returned ciphertext is a random one, 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.

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 [15], and, more recently, in the PSU literature [31], 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. [18], and then employed in PSU by Zhang et al. [47], 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. [47] 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. [31], 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 [44]. 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 out-performs the state-of-the-art by at least $1.3\times - 1.5\times$ depending on the network bandwidth.

To reduce this to two rounds and provide re-usability, we make a small modification. 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 [16]. Although it is only moderately more expensive, the runtime of performing the additional plaintext/ciphertext multiplications is greater than the cost of the oblivious transfer. In terms of communication cost, sending extra $O(n)$ ciphertexts is also more expensive than executing $O(n)$ OTs. If the RLWE ciphertext modulus is 128-bit long, and the plaintext mod-

ulus is 44-bit long, then using OT will be about 6× less expensive (assuming a pre-processed silent random OT).

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 [15] 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 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 [15], 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 3 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.

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 allows to enhance the 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 [42,38]: 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

[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.

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 $\{x \mid x \in X \wedge F_Y(x) = 1\}$, then we can construct a reusable protocol while leveraging the FNP paradigm to get sender security at almost no cost. The receiver encrypts the 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 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* [1,2]. This is a two-round, reusable PSI protocol in which the receiver message is sublinear in their set size. Such protocols are especially appealing in the setting where a receiver has 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.

Table 1. Asymptotic costs of existing semi-honest secure PSU protocols. n is the size of the input sets, λ denotes a security parameter, and ℓ is a ciphertext packing parameter. Communication is measured in field elements, and computation is measured in field multiplications.

	Communication	Computation
Frikken [16]	$O(n)$	$O(n^2)$
Davidson and Cid [10]	$O(\lambda n)$	$O(\lambda n \log \lvert F \rvert)$
Kolesnikov et al. [31]	$O(n \log n)$	$O(n \log n \log \log n)$
Garimella et al. [17]	$O(n \log n)$	$\boldsymbol{O(n \log n)}$
Jia et al. [27]	$O(n \log n)$	$\boldsymbol{O(n \log n)}$
Zhang et al. [47]	$\boldsymbol{O(n)}$	$O(n \log \lvert F \rvert)$
Ours	$\boldsymbol{O(n)}$	$O(n \log n \log \ell)$

1.5 Related Work

In the semi-honest model, Frikken [16] 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 [10] 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 homomoprhically 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. [31] 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 in-

[4] The author claims that bucketing technique can reduce the computation cost to $O(\log \log n)$, 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.

tersection. 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 runs 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. [17] 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 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. [27] 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 evaluates the PRF over the xor results and sends the PRF outputs back to the receiver. Similarly to [31], 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. [47] 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 [16] 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 [23] 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 [3] use generic MPC and incur $O(n \log n)$ communication and computation cost. Seo et al. [43] 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.

We follow [19,33] 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) [36,18]. 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 key-value pairs* $\{(k_1, v_1), \ldots, (k_n, v_n)\} \subseteq \mathcal{K} \times \mathcal{V}$ *as input, 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 need 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 any tuple of distinct keys $(k_1^0, \ldots, k_n^0)$ and $(k_1^1, \ldots, k_n^1)$, if Encode does not output $\perp$ for $(k_1^0, \ldots, k_n^0)$ or $(k_1^1, \ldots, k_n^1)$, then

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

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

Furthermore, the following randomness property, which is formally captured in [47], 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 is 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 [18], 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.

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 $r : \mathcal{V} \to \{0,1\}^{d+\lambda}$. 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 corresponds 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, [18] 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 [18], 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

[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.

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

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.

uniformly distributed. In Theorem 7 in [47], they formally prove that 3H-GCT satisfies both the obliviousness and the randomness property.

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

A naive solution. We describe again a simpler solution, introduced by Friekken [16]. 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 $\mathcal{R}$. 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 equal to zero as $Q(x_j) = 0$. Therefore, $\mathcal{R}$ learn nothing from these intersection points. 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 $\mathcal{R}$ to interpolate a polynomial of degree n_2 and $\mathcal{S}$ to evaluate the polynomial n_1 times, which accounts for $O(n^2)$ field operations for interpolation and amortized $O(n)$ field operations for each evaluation. Therefore, to achieve a much more efficient construction (both asymptotically and conretely), 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. [15] 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. [31]. To ensure that we do not indicate to the receiver whether an intersections 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 observes 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[6], 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 [44], 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 i^{th} 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 j^{th} 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 coefficient and $\mathcal{S}$'s input in a batch. This greatly improves the concrete computational efficiency. Our semi-honest protocol incorporates the two optimizations mentioned above, and is shown in Figure 3.

[6] In Kolesnikov et al. [31], 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 amortized cost linear to the degree of the polynomial.

Π_{PSU} **Semi-honest PSU**

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 $a_{i,0} \leftarrow \mathbb{Z}_p$ 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

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

   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 $\hat{X} \cup Y$.

Fig. 3. Semi-honest PSU Protocol

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 $\mathcal{F}_{\text{PSU}}$ 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 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, $\mathcal{R}$ recover $(0, 0)$ for every $\mathcal{S}$'s input x is in the intersection, i.e. $(x \in Y)$, due to the polynomial evaluated to 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 [40,31], 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:

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 $\widetilde{D}_j = (\sum_{i=0}^{k+1} C_i \bigodot \widetilde{X}_j^i)$ and $\widetilde{D}'_j = \widetilde{X}_j \bigodot \widetilde{D}_j$. 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 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 $\hat{X} = X \cup Y \setminus Y$, 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 to the trusted party.

3.2 Efficient PSU with OT in the Semi-honest Setting

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 and the overall communicated by a half, 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

[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.

$\Pi_{\mathrm{PSU}}^{\mathcal{F}_{\mathrm{OT}}}$ 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 $a_{i,0} \leftarrow \mathbb{Z}_p$ 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: $D_j = (\sum_{i=0}^{k+1} C_i \bigodot X_j^i)$. Note that

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

   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

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 $\Pi_{\text{PSU}}^{\mathsf{reuse}}$. Finally, compared to $\Pi_{\text{PSU}}^{\mathsf{reuse}}$, 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 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 $\Pi_{\text{PSU}}^{*\mathsf{reuse}}$ (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.

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

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

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 $\mathcal{R}$.
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. Reusable PSU

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 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-randomized 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 indistinguishable from those ciphertext tuples generated through proper decoding and re-randomizing in the real world, due to the 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 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.

Instantiation and cost. Using 3H-GCT for the OKVS scheme and Paillier encryption for the AH-PKE scheme, let σ_{Pai} denote the size of a Paillier ciphertext and let σ denote the size of an input. To achieve 2^{-40} failure probability for 3H-GCT, we need $m \approx 1.3n$ using the star architecture technique introduced in [18], and the communication cost can be divided into the following:

- $\mathcal{R}$ sends the 3H-GCT to $\mathcal{S}$: $\approx 1.3 \cdot n_2 \cdot \sigma_{\mathsf{Pai}}$.
- $\mathcal{S}$ sends back the tuples of re-randomized ciphertexts: $3 \cdot n_1 \cdot \sigma_{\mathsf{Pai}}$.

The total computational cost is $O(n \log n)$ due to the $3 + O(\log n) + \lambda$ group operations[8] required to secret share (resp. reconstruct) a ciphertext/value when we encode (resp. decode) the 3H-GCT. However, the concrete cost is dominated by the cost of modular exponentiation. And we requires O(n) modular exponentiation for computing encryption, decryption, re-randomization and plaintext/ciphertext multiplication.

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 $\mathsf{Dec}(e) \neq w$.

[8] While the ciphertext space $\mathbb{Z}_{N^2}^*$ of Paillier encryption is a multiplicative group, instead, we can use $\mathbb{Z}_{N^2}$ as the value space, which allows for sharing and reconstructing secret with more efficient additive group operation. This is based on the observation that a random group element in $\mathbb{Z}_{N^2}$ is a group element in $\mathbb{Z}_{N^2}^*$ except for negligible probability.

4. $\mathcal{S}$ and $\mathcal{R}$ call $\mathcal{F}_{\text{OT}}$ 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 $\perp$ 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 straight forward 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 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 $\Pi_{\text{PSU}}^{*\mathcal{F}_{OT}}$ (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.

Instantiation and cost. We analyze the communication cost and roughly compare our computation cost with the previous state-of-the-art semi-honest protocol by Zhang et al. [47]. Similarly to their protocol, we use 3H-GCT for the OKVS scheme and ECC ElGamal encryption for the re-randomizable PKE scheme. Furthermore, our OT functionality can be implemented efficiently using Ferret OT [45] in the malicious setting. Let σ_{EG} denote the size of an ECC ElGamal ciphertext, and σ denote the size of an input. To achieve 2^{-40} failure probability for 3H-GCT, we need $m \approx 1.3n$ using the star architecture technique introduced in [18], and the communication cost can be divided into to the following:

- $\mathcal{R}$ sends the 3H-GCT to $\mathcal{S}$: $\approx 1.3 \cdot n_2 \cdot \sigma_{\mathsf{EG}}$.
- $\mathcal{S}$ sends back the re-randomized ciphertexts: $n_1 \cdot \sigma_{\mathsf{EG}}$.
- Oblivious transfer: $2 \cdot n_1 \cdot (\sigma + \sigma_{\mathsf{EG}})$.

For computational cost, when comparing to the protocol based on re-randomizable PKE by Zhang et al. [47], our protocol requires the following tweaks: (1) The OT protocol needs to be maliciously secure, (2) The OT protocol needs to transfer a message twice the size as theirs, as our message include both the input and its local randomness used to de-randomize the computation of the corresponding message. (3) The receiver needs to reevaluate $\mathcal{S}$'s step to decode and re-randomize, in order to verify the ciphertexts received are correctly computed.

$\Pi^{*\mathcal{F}_{\mathrm{OT}}}_{\mathrm{PSU}}$

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 $\mathcal{F}_{\mathrm{OT}}$ (Figure 7).
- Two finite groups $\mathbb{G}_1$ and $\mathbb{G}_2$.
- A ReRand-PKE scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReRand})$ with plaintext space $\mathbb{G}_1$, ciphertext space $\mathbb{G}_2$, and randomness space ∇.
- An $(n, m, 2^{-\lambda})$-OKVS scheme $(\mathsf{Encode}, \mathsf{Decode})$ 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 $X = \{x_1, \cdots, x_{n_1}\} \subseteq \mathbb{G}_1$ and $\mathcal{R}$'s input set $Y = \{y_1, \cdots, y_{n_2}\} \subseteq \mathbb{G}_1$.

Protocol.

1. **KeyGen.** $\mathcal{R}$ computes $pp \leftarrow \mathsf{Setup}(1^\kappa)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(pp)$.
2. **Encrypt and encode.** $\mathcal{R}$ uniformly samples $w \leftarrow \mathbb{G}_1$ and generates n_2 fresh ciphertexts $(c_1, \ldots, c_{n_2})$ of w, i.e., for $i \in [n_2]$, $c_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(w)$. It then generates $D \leftarrow \mathsf{Encode}(\{(y_i, c_i)\}_{i\in[n_2]})$ and sends pk, D to $\mathcal{R}$.
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 $r_i \leftarrow \mathbb{G}_1$ and computes $v_i = h(x_i||r_i)$. Finally, it computes $e_i = \mathsf{ReRand}_{\mathsf{pk}}(d_i, v_i)$ 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: $u_i = \mathsf{Dec}_{\mathsf{sk}}(e_i)$. 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 $\mathcal{F}_{\mathrm{OT}}$. Specifically, $\mathcal{S}$ serves as the sender with inputs $(\perp, x_i||r_i)$ 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. PSU in the OT-hybrid model

Item (1) can be achieved almost for free. In [45], they show that their maliciously secure protocol only adds about 5 to 14 percent of runtime compared to their semi-honest protocol, while OT itself is relatively cheap compared to the rest of the computations. Item (2) incurs no more than twice the run time. In the worst case, item (3) doubles the total cost of decoding and re-randomizing the ciphertext. In conclusion, we estimate our protocol incurs no more than 2x computation cost compared to [47], which is the previous state of the art for the semi-honest protocol.

5 Semi-Honest PSU Experimental Results

5.1 Experimental Setup

We ran 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 do not use multi-threading in our implementation where each party runs with a single thread. We used the linux *tc* command to simulate the network bandwidth. In all our experiments, the input items are represented as a 64-bit value. Using permutation-based hashing, the effective length of each item becomes $64-\log(m)$ where m is the number of buckets. Upon hashing the elements, all buckets will be padded to k items such that $m \cdot \sum_{i=k+1}^{n} \binom{n}{i} \frac{1}{m}^{i} (1-\frac{1}{m})^{n-i} \leq 2^{-\lambda}$, where λ is a statistical security parameter. In our experiments, we choose $\lambda = 40$.

Table 2. Parameters for our hashing scheme

	$n = 2^{14}$		$n = 2^{16}$		$n = 2^{18}$		$n = 2^{20}$	
Protocols	m	k	m	k	m	k	m	k
Our Protocol A	4096	29	4096	58	4096	139	32768	90
Our Protocol B	4096	29	4096	58	16384	59	65536	60

We implemented two variations of our two-round and four-round semi-honest protocols. *Protocol A* optimizes on the communication cost while *Protocol B* aims at reducing the computation cost. We denote the two-round protocols by *Protocol A2, Protocol B2* and the four-round ones by *Protocol A4, Protocol B4*. The difference between the two variations is the number of buckets m used in the hashing scheme. For each m, we denote by k the maximum bin size after padding. See Table 2 for the concrete values of $m = O(n/\log n)$ and $k = O(n/m) = O(\log n)$ for each protocol. The communication cost of the protocols is dominated by $m \cdot k$ while the computation is dominated by $m \cdot k^2$. To minimize the computation cost, *Protocol B* chooses m such that $m \cdot k^2$ is small. On the other hand, *Protocol A* aims at minimizing $O(m \cdot k)$.

5.2 Experimental Results

We compare our results against the existing PSU protocols such as Kolesnikov et al.[31], Garimella et al. [17], and Zhang et al. [47]. We took the reported

run-time and communication cost of the existing protocol from Zhang et al. [47] directly. The authors ran all the protocols on a single Intel Core i9-9900K with CPU clock speed of 3.3GHz and 128GB RAM and also used linux *tc* command to simulate the network.

Protocol $A4$ has the lowest communication cost for input of size $n \geq 2^{18}$. Compared with [46], ours uses 15% less bandwidth, but runs 2.35× faster in LAN and 1.5× faster in a slower network with the bandwidth of $10Mbps$. Our protocols have higher communication cost for small input size. As the number of buckets needs to be greater than or equal to the RLWE packing parameter (which is 4096 in our implementation), when the input size is small, there is not much room for optimization. Both our protocols $A4$ and $B4$ out-perform all existing protocols in terms of computation cost. Protocol $B4$ is $1.3\times - 2\times$ faster than existing protocol in LAN, and the protocol $A4$ is $1.5\times - 1.8\times$ faster when the network bandwidth is $10Mbps$. See Table 3 for a comparison of the communication overhead and Table 4 for the computation cost.

Table 3. The communication cost in MB.

	$n = 2^{14}$			$n = 2^{16}$			$n = 2^{18}$			$n = 2^{20}$		
Protocols	$R \to S$	$S \to R$	Total	$R \to S$	$S \to R$	Total	$R \to S$	$S \to R$	Total	$R \to S$	$S \to R$	Total
KRTW19[31]	4.19	29.64	33.8	17.7	122.1	139.8	69.3	562.8	632	300	2306	2606
GMR+21[17]-IKNP	5.91	7.98	13.9	26.0	34.1	60.1	114	145	259	493	616	1109
GMR+21[17]-Silent	2.03	14.93	17.0	6.9	44.6	51.5	26.4	157.1	183.5	104	619	723
ZCL+21[47]-SKE	3.17	3.36	6.52	12.6	13.4	26.0	50.3	53.5	103.8	201	214	415
ZCL+21[47]-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[47]-PKE*	2.17	2.90	5.05	8.64	11.6	20.2	34.5	46.3	80.8	138	185	323
Our Protocol A2	1.84	7.25	9.09	**3.66**	14.50	18.16	**8.72**	34.75	43.5	**45.1**	180	225
Our Protocol B2	1.84	7.25	9.09	**3.66**	14.50	18.16	14.8	59.0	73.8	60.0	240	300
Our Protocol A4	1.84	4.30	6.14	**3.66**	8.61	12.27	**8.72**	**20.6**	**29.4**	**45.1**	**107**	**152**
Our Protocol B4	1.84	4.30	6.14	**3.66**	8.61	12.27	14.8	35.0	49.8	60.0	143	203

5.3 Parallelizing Our Protocols

The main building blocks of our protocols are hashing, polynomial interpolation, packed encode, packed decode, and RLWE operations. All these operations are fully parallelizable. After hashing the input items into buckets, the polynomial interpolation in each bucket can be executed in parallel. RLWE addition or subtraction is clearly parallelizable. RLWE plaintext-ciphertext multiplication, packed encode, and packed decode all use a number theoretic transform as a sub-routine which is fully parallelizable with the Butterfly algorithm. We expect around t times gain in the running time in a LAN setting with t threads.

References

1. Navid Alamati, Pedro Branco, Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Sihang Pu. Laconic private set intersection and applications. In Kobbi Nissim

Table 4. The computation cost in *seconds*.

Network	Protocols	Input Size			
		$n = 2^{14}$	$n = 2^{16}$	$n = 2^{18}$	$n = 2^{20}$
LAN ($\geq$ 10gbps)	KRTW19[31]	4.41	14.0	65.1	247
	GMR+21[17]-IKNP	1.82	5.64	22.4	101
	GMR+21[17]-Silent	4.70	11.8	40.3	177
	ZCL+21[47]-SKE	2.39	6.28	21.7	85.5
	ZCL+21[47]-PKE	7.90	15.7	48.1	174
	ZCL+21[47]-PKE*	7.43	14.3	42.2	149
	Our Protocol A2	1.20	4.77	27.6	98.2
	Our Protocol B2	1.20	4.77	20.8	84.5
	Our Protocol A4	**0.92**	**3.65**	21.1	73.8
	Our Protocol B4	**0.92**	**3.65**	**15.7**	**64.2**
1Gbps	KRTW19[31]	18.8	8.33	91.2	284
	GMR+21[17]-IKNP	4.34	9.37	30.3	1239
	GMR+21[17]-Silent	10.7	18.1	51.3	198
	ZCL+21[47]-SKE	18.8	24.6	41.7	108
	ZCL+21[47]-PKE	8.73	17.0	49.8	177
	ZCL+21[47]-PKE*	8.33	15.8	44.6	153
	Our Protocol A2	1.92	5.32	28.4	98.9
	Our Protocol B2	1.92	5.32	21.5	86.9
	Our Protocol A4	**1.48**	**4.19**	21.9	75.4
	Our Protocol B4	**1.48**	**4.19**	**16.6**	**66.9**
100Mbps	KRTW19[31]	31.4	44.3	53.2	369
	GMR+21[17]-IKNP	7.75	16.7	53.2	212
	GMR+21[17]-Silent	17.3	28.5	71.5	257
	ZCL+21[47]-SKE	33.4	40.8	62.1	145
	ZCL+21[47]-PKE	10.1	18.5	53.1	188
	ZCL+21[47]-PKE*	9.86	17.6	49.9	173
	Our Protocol A2	2.31	5.64	29.3	99.5
	Our Protocol B2	2.31	5.64	22.6	94.4
	Our Protocol A4	**1.82**	**4.50**	22.8	78.0
	Our Protocol B4	**1.82**	**4.50**	**17.9**	**73.8**
10Mbps	KRTW19[31]	60.3	168	448	2649
	GMR+21[17]-IKNP	17.9	63.6	255	1077
	GMR+21[17]-Silent	29.7	68.3	223	837
	ZCL+21[47]-SKE	33.8	53.1	112	391
	ZCL+21[47]-PKE	11.6	23.9	74.5	274
	ZCL+21[47]-PKE*	13.2	29.4	93.3	350
	Our Protocol A2	8.9	17.3	53.3	256
	Our Protocol B2	8.9	17.3	78.0	318
	Our Protocol A4	**6.44**	**13.7**	**40.6**	**182**
	Our Protocol B4	**6.44**	**13.7**	54.6	221

and Brent Waters, editors, *Theory of Cryptography*, pages 94–125, Cham, 2021. Springer International Publishing.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In *ACNS*, pages 125–142, 2009.
10. 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.
11. 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.
12. 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.
13. T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.
14. 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.
15. Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 1–19. Springer, Heidelberg, May 2004.
16. 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.

17. 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.
18. 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.
19. Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*, volume 2. Cambridge University Press, 2009.
20. 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.
21. 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.
22. 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.
23. 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.
24. Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. Scalable multi-party private set-intersection. In Serge Fehr, editor, *PKC 2017, Part I*, volume 10174 of *LNCS*, pages 175–203. Springer, Heidelberg, March 2017.
25. 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.
26. 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.
27. 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.
28. 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.
29. 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.
30. 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.
31. Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. Scalable private set union from symmetric-key techniques. In Steven D. Galbraith and Shiho Mo-

riai, editors, *ASIACRYPT 2019, Part II*, volume 11922 of *LNCS*, pages 636–666. Springer, Heidelberg, December 2019.
32. 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.
33. 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.
34. 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.
35. 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.
36. 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.
37. 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.
38. 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.
39. 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.
40. Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on ot extension. *ACM Trans. Priv. Secur.*, 21(2), jan 2018.
41. Michael O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 2005. https://ia.cr/2005/187.
42. 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.
43. 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.
44. Nigel Smart and Frederik Vercauteren. Fully homomorphic simd operations. *IACR Cryptology ePrint Archive*, 2011:133, 01 2011.
45. Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1607–1626. ACM Press, November 2020.
46. Bin Zhang. Cryptanalysis of GSM encryption in 2G/3G networks without Rainbow tables. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019,*

Part III, volume 11923 of *LNCS*, pages 428–456. Springer, Heidelberg, December 2019.

47. 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 [37] 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) [41] 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 i^{th} 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 i^{th} 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*

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

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

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 $i \in \{1,2\}$,*

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

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 $\text{HYBRID}^{g_1,\cdots,g_{p(n)}}_{\pi,\mathcal{A}(z),a}(x,y,\kappa)$.

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 $i \in \{1,2\}$,*

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

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

A.3 Re-Randomizable Public Key Encryption Schemes

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

- $\mathsf{Setup}(1^\kappa)$: 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 $\mathcal{M},\mathcal{C}$.

- $\mathsf{KeyGen}(pp)$: The key generation algorithm takes pp and output a pair of public and private key $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}_{\mathsf{pk}}(m)$: The encryption algorithm encrypts a message $m \in \mathcal{M}$ with pk to a ciphertext $c \in \mathcal{C}$.
- $\mathsf{Dec}_{\mathsf{sk}}(c)$: The decryption algorithm decrypts a ciphertext $c \in \mathcal{C}$ with sk to a message $m \in \mathcal{M}$ or a failure symbol $\perp$.
- $\mathsf{ReRand}_{\mathsf{pk}}(c)$: 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 $\mathsf{ReRand}_{\mathsf{pk}}(c, r)$ 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 $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m)$ and $c' \leftarrow \mathsf{ReRand}_{\mathsf{pk}}(c)$, we have $\mathsf{Dec}_{\mathsf{sk}}(c) = \mathsf{Dec}_{\mathsf{sk}}(c') = m$.

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 $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m)$ and $c' \leftarrow \mathsf{ReRand}_{\mathsf{pk}}(c)$ are identical.

Similar to [47], 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 $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:

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

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 [13] 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 [35] 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 reduction to the hardness of decisional composite residuosity.

The other instantiation we use is BGV Encryption Scheme [4]. It satisfies correctness and re-randomization indistinguishability, and the additive homomorphic property.

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 $\Pi^{*\mathsf{reuse}}_{\mathrm{PSU}}$ on inputs (X, Y), auxiliary input z to $\mathcal{A}$ and computational security parameter κ, denoted by $\textsc{hybrid}_{\Pi,\mathcal{A}(z)}(X, Y, \kappa)$, is defined as the output pair of the honest party and the adversary $\mathcal{A}$ of the hybrid execution. On the other hand, the ideal execution of $\mathcal{F}^{\hat{n}_1,n_2}_{\mathrm{PSU}}$ on inputs (X, Y), auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted by $\textsc{ideal}_{\mathcal{F},\mathcal{A}(z)}(X, Y, \kappa)$, is defined as the output pair of the honest party and the adversary $\mathcal{A}$ from the ideal execution.

Note that OKVS successfully encodes the input set except for $2^{-\lambda}$ probability. In our following analysis, we simply treat Encode as perfect. 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

$$\left\{\textsc{hybrid}_{\Pi,\mathcal{A}(z)}(X, Y, \kappa)\right\}_{X,Y,z,\kappa} \overset{c}{\equiv} \left\{\textsc{ideal}_{\mathcal{F},\mathsf{Sim}_{\mathcal{S}}(z)}(X, Y, \kappa)\right\}_{X,Y,z,\kappa}$$

where $X, Y \subseteq \mathbb{F}$, $z \in \{0,1\}^*$ and $\kappa \in \mathbb{N}$. And

$$\left\{\textsc{hybrid}_{\Pi,\mathcal{A}(z)}(X, Y, \kappa)\right\}_{X,Y,z,\kappa} = \left\{\mathsf{pk}, D, \hat{X} \cup Y\right\}_{X,Y,z,\kappa}$$

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

To argue the above probability ensembles are computationally indistinguishable, we use the following hybrid joint distributions for any X, Y, z, κ:

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

Hyb_1: Same as Hyb_0, except that the simulator is given the honest $\mathcal{R}$'s input Y. It computes $\widetilde{D}' \leftarrow \mathsf{Encode}(\{(y_i, \widetilde{c}_i)\}_{i\in[n_2]})$ 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 $\text{HYBRID}_{\Pi,\mathcal{A}(z)}(X, Y, \kappa)$, except pk is replaced by $\widetilde{\mathsf{pk}}$, note that pk and $\widetilde{\mathsf{pk}}$ are only semantically different, and the distributions are exactly the same.

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

1. $\mathcal{A}_{\mathsf{obl}}$ sends $Y_0 = \{\widetilde{y}_i\}_{i\in n_2}$ and $Y_1 = Y$ to the challenger $\mathcal{C}$. $\mathcal{C}$ 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. $\mathcal{A}_{\mathsf{obl}}$ uniformly samples a public key $\widetilde{\mathsf{pk}}$, using $\mathcal{A}$ as a black box, sends $\widetilde{\mathsf{pk}}$ and D_b to $\mathcal{A}$. Upon response from $\mathcal{A}$, it extracts $\mathcal{S}$'s input set $\widetilde{X}$ (same as how it is done through verification in $\mathsf{Sim}_{\mathcal{S}}$, step 3 and 4), abort if necessary. Otherwise, it computes $\widetilde{X} \cup Y$ and sends $(\widetilde{\mathsf{pk}}, D_b, \widetilde{X} \cup Y)$ to $\mathcal{D}$.
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.

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

1. $\mathcal{A}_{\mathsf{pseu}}$ receives $\widetilde{\mathsf{pk}}$ from the challenger. $\mathcal{A}_{\mathsf{pseu}}$ 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, $\mathcal{C}$ and 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. $\mathcal{A}_{\mathsf{pseu}}$, computes $D_b \leftarrow \mathsf{Encode}(\{(y_i, c_i^b)\}_{i\in n_2})$ and sends $\widetilde{\mathsf{pk}}$ and D_b to $\mathcal{A}$. Upon response from $\mathcal{A}$, it extracts $\mathcal{S}$'s input set $\widetilde{X}$ (same as how it is done through verification in $\mathsf{Sim}_{\mathcal{S}}$, step 3 and 4), abort if necessary. Otherwise, it computes $\widetilde{X} \cup Y$ and sends $(\widetilde{\mathsf{pk}}, D_b, \widetilde{X} \cup Y)$ to $\mathcal{D}$.
3. $\mathcal{A}_{\mathsf{pseu}}$ uses $\mathcal{D}$'s outcome to decide whether the view is in Hyb_1 or Hyb_2, and use the outcome to answer $\mathcal{C}$ and wins with non-negligible probability.

Finally, we show that Hyb_2 and Hyb_3 are statistically indistinguishable. Notice that both $\widetilde{\mathsf{pk}}$ and D follow the same distributions in the two hybrids. Thus, $\mathcal{A}$'s response follows the same distribution in both hybrids. Now, let X^* denote $\mathcal{A}$'s set of "effective" inputs in both worlds, where each input $x \in X^*$ has a corresponding tuple (e, e', e'') that can be formed by correctly decoded x with

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

1. **KeyGen.** The simulator computes $pp \leftarrow \mathsf{Setup}(1^\kappa), (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{KeyGen}(pp)$.
2. **Encrypt and encode.** The simulator randomly generates key-value pairs $\{(\widetilde{y}_i, \widetilde{c}_i)\}_{i\in[n_2]}$, in which $\{\widetilde{y}_1, \ldots, \widetilde{y}_{n_2}\} \subseteq \mathbb{F}$ and $\widetilde{c}_i$ 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 $\mathcal{S}$.
3. **Decode and re-randomize.** The simulator keeps a list L for all queries to the random oracle, i.e., all $\widetilde{x}||\widetilde{r}$. Then, it receives $\{(\widetilde{e}_i, \widetilde{e'}_i, \widetilde{e''}_i)\}_{i\in[n_1]}$ from $\mathcal{S}$. 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.
4. **Decrypt and obtain output.** Let $\widetilde{X} = \emptyset$. For each tuple $(\widetilde{e}_i, \widetilde{e'}_i, \widetilde{e''}_i)$, it verifies that it is computed from some $\widetilde{x}||\widetilde{r} \in L$ i.e., it matches with an earlier query to the random oracle. If the verification passes, the simulator sets $\widetilde{X} = \widetilde{X} \cup \{\widetilde{x}\}$. Finally, the simulator submits $\widetilde{X}$ to the ideal functionality.

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

D and re-randomized using $h(x||r)$. In Hyb_2, by definition, $\widetilde{X}$ is a subset X^* as it additionally requires that each $x \in \widetilde{X}$, along with their corresponding r, has been queried by $\mathcal{A}$ to the random oracle. As $\mathcal{A}$ has negligible probability to guess the outcome of a random oracle, thus a negligible probability to correctly form a tuple (e, e', e'') without querying a random oracle on $x||r$, $\widetilde{X} = X^*$ except with negligible probability. In Hyb_3, with overwhelming probability[9], $\hat{X} = X^* \setminus Y$, i.e., $\hat{X}$ is $\mathcal{S}$'s set of "effective" inputs minus those inputs in $\mathcal{R}$'s set. Combining the observations above, we have $\widetilde{X} \cup Y = X^* \cup Y = (X^* \setminus Y) \cup Y = \hat{X} \cup Y$ except with negligible probability. Finally, Hyb_3 is perfectly indistinguishable from $\text{HYBRID}_{\Pi,\mathcal{A}(z)}(X, Y, \kappa)$. Firstly, pk and $\widetilde{\mathsf{pk}}$ are only semantically different and follow the same distribution. Secondly, in both worlds, abort is triggered when more than $\hat{n}_1$ ciphertexts are returned by $\mathcal{A}$.

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

Proof. As $\mathcal{F}_{\text{PSU}}^{\hat{n}_1,n_2}$ is deterministic and we want to show security against a semi-honest $\mathcal{R}$, it suffices to separately show (a) correctness: the protocol 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 scheme is correct, first notice that Encode only fails with $2^{-\lambda}$ probability and we treat it as perfect in our analysis. 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

[9] There is a negligible probability that there exists some e_i decrypted to 0, even the corresponding $x \notin Y$.

with 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 and $\hat{X} \cup Y = X \cup Y$.

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

1. Run $pp \leftarrow \mathsf{Setup}(1^\kappa)$, and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(pp)$.
2. Compute $\bar{X} = X \cup Y \setminus Y$, i.e., the set of extra elements that $\mathcal{S}$ brings to to the union.
3. Follow step 2 of the protocol to generate D.
4. For all $x \in \bar{X}$, it follows step 3 of the protocol to decode and re-randomize them to $\{(e_i, e'_i, e''_i)\}_{i \in [|\bar{X}|]}$. Additionally, it generates $\{(e_i, e'_i, e''_i)\}_{|\bar{X}|+1 \le i \le n_1}$, where each $e_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$, $e'_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$, $e''_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$.
5. Output $\{e_i\}_{i \in [|\bar{X}|]} \cup \{e'_i\}_{|\bar{X}|+1 \le i \le n_1}$ in a random order.

First notice that all $\{(e_i, e'_i, e''_i)\}_{i \in [|\bar{X}|]}$ are perfectly indistinguishable with those non-intersected inputs in the real view. Then, both $\{e'_i\}_{|\bar{X}|+1 \le i \le n_1}$ and the remaining ciphertexts in the real view decrypt to 0s. And due to the re-randomization indistinguishability property of the ciphertexts, the ciphertexts themselves follows the identical distribution. This concludes our proof.

B.2 Proof of Theorem 4

Claim. Our protocol is secure against any malicious sender.

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

Note that OKVS successfully encodes the input set except for $2^{-\lambda}$ probability. In our following analysis, we simply treat Encode as perfect. 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 $\mathcal{F}_{\text{OT}}$, there exists a simulator $\mathsf{Sim}_{\mathcal{S}}$ (Figure 9) such that

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

where $X, Y \subseteq \mathbb{G}_1$, $z \in \{0,1\}^*$ and $\kappa \in \mathbb{N}$. And

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

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

To argue the above probability ensembles are computationally indistinguishable, we use the following hybrid joint distributions for any X, Y, z, κ:

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. It computes $\widetilde{D}' \leftarrow \mathsf{Encode}(\{(y_i, \widetilde{c}_i)\}_{i\in[n_2]})$ and sends $\widetilde{D}'$ to $\mathcal{A}$, where $\widetilde{c}_i$ is uniformly sampled from the ciphertext space, just like in Hyb_0.

Hyb_2: Same as Hyb_1, except that the simulator follows $\mathcal{R}$'s instructions to sample w and generate n_2 ciphertexts $c_1, \ldots, c_{n_2}$ of w. 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 $\text{HYBRID}^{\mathcal{F}_{\text{OT}}}_{\Pi,\mathcal{A}(z)}(X, Y, \kappa)$, except pk is replaced by $\widetilde{\mathsf{pk}}$, note that pk and $\widetilde{\mathsf{pk}}$ are only semantically different, and the distributions are exactly the same.

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

1. $\mathcal{A}_{\mathsf{obl}}$ receives $\widetilde{\mathsf{pk}}$ from the challenger. $\mathcal{A}_{\mathsf{obl}}$ sends $Y_0 = \{\widetilde{y}_i\}_{i\in n_2}$ and $Y_1 = Y$ to the challenger $\mathcal{C}$. $\mathcal{C}$ 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. $\mathcal{A}_{\mathsf{obl}}$, using $\mathcal{A}$ as a black box, sends $\widetilde{\mathsf{pk}}$ and D_b to $\mathcal{A}$. Upon response from $\mathcal{A}$, it extracts $\mathcal{S}$'s input set $\widetilde{X}$ (same as how it is done through verifications in $\mathsf{Sim}_{\mathcal{S}}$, step 3 and 4), abort if necessary. Otherwise, it computes $\widetilde{X} \cup Y$ and sends $(\widetilde{\mathsf{pk}}, D_b, \widetilde{X} \cup Y)$ to $\mathcal{D}$.
3. $\mathcal{A}_{\mathsf{obl}}$ uses $\mathcal{D}$'s outcome to decide whether D_b is built from Y_0 or Y_1. It answers $\mathcal{C}$ accordingly and wins with non-negligible probability.

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

1. $\mathcal{A}_{\mathsf{pseu}}$ receives $\widetilde{\mathsf{pk}}$ from the challenger. $\mathcal{C}$ 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, $\mathcal{C}$ samples $w \leftarrow \mathbb{G}_1$ and generates n_2 ciphertexts of w, i.e., $\{c_i^1\}_{i\in[n_2]}$ where $c_i^1 \leftarrow \mathsf{Enc}_{\widetilde{\mathsf{pk}}}(w)$. Then, $\mathcal{C}$ sends $\{c_i^b\}_{i\in[n_2]}$ to $\mathcal{A}_{\mathsf{pseu}}$.
2. $\mathcal{A}_{\mathsf{pseu}}$, computes $D_b \leftarrow \mathsf{Encode}(\{(y_i, c_i^b)\}_{i\in n_2})$ and sends $\widetilde{\mathsf{pk}}$ and D_b to $\mathcal{A}$. Upon response from $\mathcal{A}$, it extracts $\mathcal{S}$'s input set $\widetilde{X}$ (same as how it is done through verifications in $\mathsf{Sim}_{\mathcal{S}}$, step 3 and 4), abort if necessary. Otherwise, it computes $\widetilde{X} \cup Y$ and sends $(\widetilde{\mathsf{pk}}, D_b, \widetilde{X} \cup Y)$ to $\mathcal{D}$.
3. $\mathcal{A}_{\mathsf{pseu}}$ uses $\mathcal{D}$'s outcome to decide whether the view is in Hyb_1 or Hyb_2, and use the outcome to answer $\mathcal{C}$ and wins with non-negligible probability.

Finally, we show that Hyb_2 and Hyb_3 are statistically indistinguishable. Notice that both $\widetilde{\mathsf{pk}}$ and D follow the same distributions in the two hybrids. Thus, $\mathcal{A}$'s response follows the same distribution in both hybrids. Now, let X^* denote $\mathcal{A}$'s set of "effective" inputs in both worlds, where each input $x \in X^*$

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

1. **KeyGen.** The simulator computes $pp \leftarrow \mathsf{Setup}(1^\kappa), (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{KeyGen}(pp)$.
2. **Encrypt and encode.** The simulator randomly generates key-value pairs $\{(\widetilde{y}_i, \widetilde{c}_i)\}_{i\in[n_2]}$, in which $\{\widetilde{y}_1, \ldots, \widetilde{y}_{n_2}\} \subseteq \mathbb{G}_1$ and $\widetilde{c}_i$ 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 $\mathcal{S}$.
3. **Decode and re-randomize.** The simulator keeps a list L for all queries to the random oracle, i.e., all $\widetilde{x}||\widetilde{r}$. Then, it receives $\{\widetilde{e}_i\}_{i\in[n_1]}$ from $\mathcal{S}$. 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.
4. **Obtain output.** Let $\widetilde{X} = \emptyset$. For i^{th} call of $\mathcal{F}_{\mathrm{OT}}$, the simulator receives $\perp, \widetilde{x}'_i||\widetilde{r}'_i$ and verifies that $\widetilde{x}'_i||\widetilde{r}'_i \in L$, i.e., it matches with an earlier query to the random oracle. Additionally, it follows $\mathcal{S}$'s instruction in step 3 to compute $\widetilde{e}'_i$ using $\widetilde{D}, \widetilde{x}'_i$, and $\widetilde{r}'_i$ and verifies that $\widetilde{e}'_i$ equals to $\widetilde{e}_i$ received in the previous step. If both verifications succeed, the simulator sets $\widetilde{X} = \widetilde{X} \cup \{\widetilde{x}'_i\}$. Finally, the simulator submits $\widetilde{X}$ to the ideal functionality.

Fig. 9. Simulator for corrupted $\mathcal{S}$.

has a corresponding OT with message $x||r$, and its corresponding e is correctly decoded and re-randomized using D and $h(x||r)$, respectively. In Hyb_2, by definition, $\widetilde{X}$ is a subset X^* as it additionally requires that each $x \in \widetilde{X}$, along with their corresponding r, has been queried by $\mathcal{A}$ to the random oracle. As $\mathcal{A}$ has negligible probability to guess the outcome of a random oracle, thus a negligible probability to correctly form a e without querying a random oracle on $x||r$, $\widetilde{X} = X^*$ except with negligible probability. In Hyb_3, with overwhelming probability[10], $\hat{X} = X^* \setminus Y$, i.e., $\hat{X}$ is the set of "effective" inputs minus those inputs in $\mathcal{R}$'s set. Combining the observations above, we have $\widetilde{X} \cup Y = X^* \cup Y = (X^* \setminus Y) \cup Y = \hat{X} \cup Y$ except with negligible probability.

Finally, Hyb_3 is perfectly indistinguishable from $\textsc{hybrid}^{\mathcal{F}_{\mathrm{OT}}}_{\Pi,\mathcal{A}(z)}(X, Y, \kappa)$. Firstly, pk and $\widetilde{\mathsf{pk}}$ are only semantically different and follow the same distribution. Secondly, in both worlds, abort is triggered when more than $\hat{n}_1$ ciphertexts are returned by $\mathcal{S}$.

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

Proof. As $\mathcal{F}^{\hat{n}_1,n_2}_{\mathrm{PSU}}$ is deterministic and we want to show security against a semi-honest $\mathcal{R}$, it suffices to separately show (a) correctness: the protocol 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.

[10] There is a negligible probability that there exists some e_i decrypted to w, even the corresponding $x \notin Y$.

To see our scheme is correct, first notice that Encode only fails with $2^{-\lambda}$ probability. Then, 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 with a random element in the ciphertext space. As a result, $\mathcal{R}$ requests $x_i||r_i$ from 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 privacy holds, we define the following simulator $\mathsf{Sim}_{\mathcal{R}}(Y, X \cup Y)$ to generate the view for the semi-honest $\mathcal{R}$:

1. Run $pp \leftarrow \mathsf{Setup}(1^\kappa)$, and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(pp)$.
2. Compute $\bar{X} = X \cup Y \setminus Y$, i.e., the set of extra elements that $\mathcal{S}$ brings to to the union.
3. Follow step 2 of the protocol to sample w and generate D.
4. For all $x \in \bar{X}$, it follows step 3 of the protocol to decode and re-randomize them to $\{e_i\}_{i\in[|\bar{X}|]}$. Additionally, it generates $\{e'_i\}_{i\in[n_1-|\bar{X}|]}$, where each $e'_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(w)$.
5. The simulated view includes $\{e_i\}_{i\in[|\bar{X}|]} \cup \{e'_i\}_{i\in[n_1-|\bar{X}|]}$ in a random order and the corresponding OT outputs $\{(x_i||r_i)_{i\in[|\bar{X}|]}\} \cup \{\perp, ..., \perp\}$ in the same random order.

First notice that all $\{e_i\}_{i\in[|\bar{X}|]}$ and $\{(x_i||r_i)_{i\in[|\bar{X}|]}\}$ are perfectly indistinguishable with the real view. Also, the simulator gives OT outputs $\{\perp, ..., \perp\}$ to $\mathcal{R}$ for each $x \notin Y$, and this is exactly the same as the real world. Finally, both $\{e'_i\}_{i\in[n_1-|\bar{X}|]}$ in the simulated view and $\{e_i : e_i \leftarrow \mathsf{ReRand}(d_i), d_i = \mathsf{Decode}(D, x_i)\}_{x_i\in X\setminus\bar{X}}$ in the real view are ciphertext of w. And due to the re-randomization indistinguishability property of the ciphertexts, the ciphertexts themselves follows the identical distribution. This concludes our proof.