Concretely Efficient Private Set Union via Circuit-based PSI

Gowri R Chandran, Thomas Schneider, Maximilian Stillger
Technical University of Darmstadt, Germany

Christian Weinert
Royal Holloway, University of London

Abstract

Private set intersection (PSI) is a type of private set operation (PSO) for which concretely efficient linear-complexity protocols do exist. However, the situation is currently less satisfactory for other relevant PSO problems such as private set union (PSU): For PSU, the most promising protocols either rely entirely on computationally expensive public-key operations or suffer from substantial communication overhead.

In this work, we present the first PSU protocol that is mainly based on efficient symmetric-key primitives yet enjoys comparable communication as public-key-based alternatives. Our core idea is to re-purpose state-of-the-art circuit-based PSI to realize a multi-query reverse private membership test (mq-RPMT), which is instrumental for building PSU. We carefully analyze a privacy leakage issue resulting from the hashing paradigm commonly utilized in circuit-based PSI and show how to mitigate this via oblivious pseudorandom function (OPRF) and new shuffle sub-protocols. Our protocol is modularly designed as a sequential execution of different building blocks that can be easily replaced by more efficient variants in the future, which will directly benefit the overall performance.

We implement our resulting PSU protocol, showing a run-time improvement of 10% over the state-of-the-art public-key-based protocol of Chen et al. (PKC'24) for input sets of size 2^{20}. Furthermore, we improve communication by 1.6× over the state-of-the-art symmetric-key-based protocol of Zhang et al. (USENIX Sec'23).

Keywords

private set union; private set intersection; circuit-based PSI

1 Introduction

In general, the problem of private set operations (PSO) is to compute some function over private input sets X and Y held by parties A and B, respectively. An extensively studied PSO problem is that of private set intersection (PSI), where the goal is to compute $X \cap Y$ without revealing any set elements that are not in the intersection. After more than 40 years of research on PSI, there exist very efficient protocols. For example, the PSI protocol of [41] requires less than 1 s and only around 30 MB of communication to compute the intersection between two sets with one million elements each. There exist several variants of PSI, including circuit-based PSI protocols that allow to compute arbitrary functions of the intersection of two sets [36, 37, 41].

Recently, the study of other PSO problems such as computing the private set union (PSU) has received more attention. In PSU, the goal is to compute the union $X \cup Y$ without revealing which elements are in the intersection, i.e., are held by both parties. This functionality has various applications, e.g., implementing cross-organizational access control lists or generating combined intrusion detection reports in a privacy-preserving way. See, e.g., [25] for a discussion of several real-world use cases for PSU.

In PSU, the receiving party learns only additional elements from the sending party that it does not already have as input. Intuitively, this is much harder to achieve than PSI, where the parties learn a subset of elements they already know, i.e., the intersection.

The first PSU protocol was proposed in [27] and is based on polynomial representations and additively homomorphic encryption. Further PSU protocols include, e.g., [6, 15, 18, 22], most of which are again based on computationally inefficient homomorphic encryption or have unfavorable computation and/or communication complexities.

Recent works such as [25, 28] were game-changers in this respect as they proposed the first protocols that utilize mainly efficient symmetric-key primitives. However, their communication complexity of $O(n \log n)$ and the concrete performance was still sub-optimal.

This was changed very recently by works such as [13] and [47] that achieve $O(n)$ communication complexity. The fundamental idea of these works is to use a functionality called multi-query reverse private membership test (mq-RPMT) that in combination with oblivious transfer (OT) can be turned into a PSU protocol. However, this break-through was achieved by relying mostly on public-key operations for the instantiation of the mq-RPMT functionality, thereby requiring to perform $O(n)$ exponentiations.

1.1 Our contributions

We observe that the mq-RPMT functionality introduced in [13, 47] can be instantiated very efficiently using state-of-the-art circuit-based PSI [41]. However, as circuit-based PSI protocols commonly hash inputs to bins to improve complexity, this turns out to allow the PSU receiver to infer some information about the sender's set. A concurrent and independent work by Liu et al. [29] already showed an attack for a similar leakage issue in the unbalanced PSU protocol of [44]. However, their solution to mitigate leakage, which is to increase hashing-related parameters, results in a significant performance penalty, specifically with respect to computation costs.

We, too, carefully analyze potential privacy leakage and propose suitable countermeasures. More precisely, we propose to randomize the mapping of inputs by first running a lightweight multi-point oblivious pseudo-random function (OPRF) protocol [12]. Additionally, we propose to obliviously shuffle the output of the circuit-based PSI step to hide the mapping of new items that the PSU receiver learns.

For the shuffle step, we discuss multiple options, ranging from oblivious evaluation of traditional Benes [4] or Waksman [45] permutation networks to different efficient instantiations of a "Combine-and-Permute" (CnP) functionality. This allows the reconstruction of a vector of shares towards the receiver with a permutation known only to the sender. One possible instantiation of CnP is a new modified version of the oblivious punctured vector (OPV)-based shuffle protocol of [11]. Alternatively, we show how to utilize a very recent work on permutation correlations [33] for CnP. For the latter option, our PSU protocol achieves linear computation as well as communication while being mostly based on efficient symmetric-key primitives.

Importantly, the modularity of our approach allows to upgrade the building blocks easily. Thereby, we can immediately benefit from future improvements in terms of OT, OPRF, circuit-based PSI, and shuffling protocols. Moreover, by building on circuit-based PSI, we can easily extend our PSU protocol to accommodate more functionalities such as PSU with associated values (aka labeled PSU) and PSU cardinality (PSU-CA). We briefly describe how to design and integrate such extensions in Appendix B.

Finally, we implement our protocoland benchmark the individual components as well as overall performance in comparison to related work. Thereby, we show that we concretely improve in run-time over the state-of-the-art public-key-based PSU protocol of [13]. More precisely, for sets of size 2^{20}, we improve by 10 % over [13] in a LAN network configuration. Furthermore, we show communication improvements of 1.6× compared to the state-of-the-art symmetric-key-based PSU protocol of [47], thereby presenting an interesting new trade-off between the two protocol classes.

In short, we summarize our contributions as follows: We

- show how to construct PSU by instantiating the mq-RPMT functionality in the framework of [13] with state-of-the-art circuit-based PSI [41];
- analyze the privacy leakage resulting from input hashing and mitigate this leakage by using combinations with OPRF and shuffle protocols;
- propose new, efficient instantiations for the "Combine and Permute" (CnP) functionality, resulting in linear-complexity PSU, relying mostly on symmetric-key primitives;
- implement and evaluate our PSU protocol, showing run-time improvements over state-of-the-art public-key-based PSU [13] and communication improvements over symmetric-key-based PSU [47].

1.2 Outline

In § 2, we introduce necessary preliminaries. Related works, including further PSU variants, are discussed in § 3. In § 4, we give a high-level overview of our PSU protocol based on circuit-based PSI, explain why additional mitigations are necessary to prevent privacy leakage, and rigorously prove security as well as correctness. One of our mitigations relies on a shuffling functionality called "combine-and-permute" (CnP), for which we we present a new efficient protocol in § 5. In § 6, we discuss possible instantiations for all other required building blocks as well as their complexity. Finally, we present our implementation and evaluation results in § 7, before concluding in § 8.

2 Preliminaries

In this section, we introduce some of the primitives that are used in our work.

2.1 Secret Sharing

We will use a simple form of Boolean 2-out-of-2 secret sharing. For this, a secret value $s \in \{0,1\}$ is split into two random shares $s_0, s_1 \in \{0,1\}$ such that $s = s_0 \oplus s_1$. Note that a party possessing only one of the shares cannot learn any information about the value of s.

2.2 Oblivious Transfer

The concept of oblivious transfer (OT) [40], in its simplest form, allows a sender S and a receiver R to obliviously transfer one bit x_c from S to R such that S does not learn the choice bit c and R does not learn the other bit x_{1-c} (cf. Fig. 1). A large number of OTs can be efficiently generated via OT extension [24]. Notably, "silent" OT extension protocols [7, 8] manage to do this with sub-linear communication overhead, leading to amortized costs as low as 0.1 bit communication per (random) OT. We utilize OT (extension) primarily on its own for oblivious data transfers. However, OT is also a fundamental building block for secure two-party protocols such as Yao's garbled circuits [46] or the secret-sharing-based GMW protocol [21], which we use for obliviously evaluating circuits.

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

Parameters: Sender S has input $\{x_0, x_1\}$ and receiver R has choice bit $c \in \{0,1\}$.

Functionality: Upon receiving $\{x_0, x_1\}$ from S and c from R, output x_c to R.

Figure 1: Ideal functionality for oblivious transfer (OT).

2.3 Oblivious Pseudo-Random Functions

An oblivious pseudo-random function (OPRF) allows a sender S to receive a PRF key k and the receiver R to receive the PRF evaluations $F_k(y_1), \ldots, F_k(y_n)$ on its input set $Y = \{y_1, \ldots, y_n\}$. Here, the sender does not learn anything about the receiver's input, and the receiver does not learn anything about the key k. We provide the formal ideal functionality in Fig. 2.

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

Parameters: Pseudorandom function $F : \{0,1\}^l \times \{0,1\}^\lambda \to \{0,1\}^\lambda$, where l is the length of the input and λ is the computational security parameter. Receiver R inputs set $Y = \{y_1, \ldots, y_n\}$.

Functionality: Upon receiving Y from R, sample $k \leftarrow \{0,1\}^\lambda$, and return $F_k(Y) = \{F_k(y_1), \ldots, F_k(y_n)\}$ to R and k to S.

Figure 2: Ideal functionality for OPRF.

2.4 (Circuit-based) Private Set Intersection and Cuckoo Hashing

PSI protocols enable two parties to compute the intersection $X \cap Y$ for private input sets X and Y such that the PSI receiver learns no information about elements of the PSI sender that are not in the intersection.

A variant of PSI, called circuit-based PSI, additionally allows to securely compute any (symmetric) function f over the intersection such that only the output $f(X \cap Y)$ is revealed, but no information about the intermediate intersection result (cf. Fig. 3). There exist many works that construct increasingly efficient circuit-based PSI protocols [10, 35–38, 41, 43].

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

Parameters: Sender S has input set X and receiver R has input set Y, where $|X| = |Y| = n$. f is a symmetric function known to S and R.

Functionality: Upon receiving X from S and Y from R, compute $f(X \cap Y)$ and return it to R.

Figure 3: Ideal functionality for circuit-based PSI.

Starting with [38], many of these protocols first place the items of the input sets in hash tables to reduce the number of required oblivious equality checks. Specifically, one party uses Cuckoo hashing [32], which is a hashing scheme with two (or more) hash functions h_1, h_2 to map items to bins such that it is guaranteed that in each bin of the hash table T_Y there is at most one item stored. The other party would follow a "simple hashing" scheme where each item x is stored in all bins indicated by the hash functions, i.e., in locations $h_1(x)$ as well as $h_2(x)$ of table T_X. Then, obliviously checking item $\mathsf{T}_Y[i]$ (for all i bins) against all items in the corresponding simple hashing bin $\mathsf{T}_X[i]$ for equality allows to produce a bit vector $\vec{Z}$. This vector indicates for each item in T_Y whether it is in the intersection or not. In the context of circuit-based PSI, the resulting bit vector $\vec{Z}$ would usually be in a secret-shared form such that with generic secure computation protocols, it is possible to obliviously evaluate any (symmetric) function f on it.

2.5 (Reverse) Private Membership Testing

A functionality related to PSI is private membership testing (PMT). Here, the membership of a single item x (instead of a set X) is obliviously checked against a set Y, resulting in a single-bit output indicating if a match was found or not, i.e., $x \in Y$ or not. Obviously, PMT can be utilized to build PSI by running multiple queries for all $x \in X$ in parallel. If multiple x are tested in one batch, this is referred to as multi-query PMT (mq-PMT).

A variant of PMT is reverse PMT (RPMT), where the party holding set Y receives the output. This becomes interesting in the corresponding multi-query variant called mq-RPMT. Here, the sender provides its set $X = \{x_1, \ldots, x_n\}$ in random order and the receiver obtains a bit vector $\vec{Z}$ indicating the positions in X that resulted in a match with Y, i.e., $\vec{Z}[i] = 1$ iff $x_i \in Y$ (cf. Fig. 4 for the ideal functionality). This can be used to build various private set operations, including PSU, as we will see later in § 4. Note that since the receiver is not aware of the ordering of X, this functionality does not reveal any information about the set intersection.

3 Related Works

In the following, we provide more details on existing PSU protocols. Note that we do not explain "historic" protocols such as [6, 15, 18,

Functionality $\mathcal{F}_{\mathsf{mq-RPMT}}$

Parameters: Sender S has input set X and receiver R has input set Y, where $|X| = |Y| = n$.

Functionality: Upon receiving X from S and Y from R, return a bit vector $\vec{Z}$ to R such that $\vec{Z}[i] = 1$ if $x_i \in Y$, and $\vec{Z}[i] = 0$ otherwise.

Figure 4: Ideal functionality for mq-RPMT.

22, 27] in detail but restrict the discussion to recent and state-of-the-art protocols. Furthermore, we discuss existing works on leakage analysis of PSU protocols.

Symmetric-key Protocols. Kolesnikov et al. [28] were the first to build PSU primarily from symmetric-key techniques. In their protocol, both parties hash their input sets to regular hash tables; for all bins, they execute RPMT instances; the PSU receiver then uses the RPMT output as OT input to obtain all *non*-matching elements. Notably, the RPMT protocol used in [28] relies on efficient symmetric-key OPRF- and equality test executions. Nevertheless, their concrete performance is not compelling with single-threaded run-times of more than 200 seconds for sets with 2^{20} elements.

The more recent work of [25] addresses an inherent information leakage in the protocol of [28], which occurs due to the utilized hashing technique: as PSU is computed on a bucket level, the receiver can locate and therefore infer information about $X \cap Y$ – which should not be possible according to the definition of PSU. To avoid this leakage, the authors of [25] first hash the inputs sets following the Cuckoo/simple hashing paradigm and then run a "Permute and Share" protocol on the Cuckoo table; the following (multi-query) OPRF operations are then carried out over the permuted and secret-shared table. The run-times of this protocol are competitive with around one minute for sets with 2^{20} elements; however, the communication overhead is very high with more than one gigabyte. Note that we also utilize permutations to prevent information leakage from hashing. However, we apply this step at a later protocol stage where we have to permute only a bit vector instead of full-length inputs.

Conceptually, the closest to our work is the work of [20]. They implement several PSO functionalities, including PSU, by combining the OPPRF pre-processing step of the circuit-based PSI protocol of [36] with oblivious shuffle and private equality testing protocols. In contrast, our approach is more abstract and works directly with any circuit-based PSI protocol following the Cuckoo/simple hashing paradigm. Furthermore, we have significantly better performance than [20], e.g., for set of size 2^{20}, we require 4× less communication.

Public-key Protocols. The work of [47] achieved a major breakthrough by proposing the first linear complexity PSU protocols. The core of their protocols is the multi-query RPMT (mq-RPMT) functionality (cf. § 2.5), which they introduce in their work. After executing mq-RPMT, the PSU receiver can then learn the missing elements in the union by running OTs using the mq-RPMT output vector as choice vector.

For the instantiation of the mq-RPMT functionality, [47] proposes a public-key and a symmetric-key variant, both of them utilizing an oblivious key-value store (OKVS) data structure [34] for the encoding of the receiver's input set. The *public-key* variant has run-times in the order of 2.5 minutes for sets of size 2^{20}. The *symmetric-key* variant has more competitive run-times of under one minute, but the communication overhead of over 400MB for sets of size 2^{20} is substantial. This is due to the fact that $O(n)$ decryption circuits of the MPC-friendly PRF "LowMC" [1] must be evaluated obliviously. Our protocol results in similar run-times, but almost halves the communication overhead.

The work of [5] proposes a computation/communication trade-off for the OKVS data structure used in the mq-RPMT implementation of [47] to achieve a better rate when representing input sets, and therefore slightly lower communication.

Very recently, the work of [13] has proposed multiple new, purely public-key-based instantiations for the mq-RPMT framework of [47] and demonstrated applications to various PSOs. Their most efficient PSU variant, including new implementation tricks, achieves a single-threaded run-time of roughly one minute with a communication overhead of roughly 100MB for sets of size 2^{20}. In this work, we present a new trade-off: with mainly symmetric-key techniques, we get 10 % better run-times than [13] at the cost of about twice the communication. Note that we expect the run-time benefit of our modular protocol to increase over time with improved circuit-based PSI building blocks.

Unbalanced and Multi-Party PSU. Similar to PSI, there exist further variants of PSU. For example, works such as [17, 44] study unbalanced PSU protocols that are optimized for use cases where the input set of one party is much smaller than for the other party. Furthermore, multi-party PSU is a natural generalization of two-party PSU [16, 19, 30]. In this work, we focus on (balanced) use cases with two sets of roughly similar size.

Leakage Analysis. Recently, Jia et al. [26] revisited information leakage in PSU protocols in more detail, which was already discussed in works such as [25]. In particular, they highlight "during-execution" leakage issues that are inherent in most state-of-the-art PSU protocols (including [5, 13, 20, 25, 44, 47]) where the elements in $X \setminus Y$ are obtained by the PSU receiver via OT in the final protocol step. This type of leakage occurs as the PSU receiver learns the intersection size in the form of (randomly permuted) OT choice bits before the protocol concludes. In practice, this might result in unintended leakage of elements in the intersection $X \cap Y$. For example, a maliciously acting receiver can abort the execution after learning the intersection size (blaming an unreliable network connection) and request repeating the protocol run; in such repeated runs, the malicious receiver can modify its input set, observe changes in the intersection size, and thus arrive at conclusions about the intersection content. Note that this type of attack has the potential to bypass conventional rate-limiting approaches that typically limit the number of successful protocol executions but might allow unsuccessful attempts to be repeated.

To mitigate such leakage issues, the authors of [26] propose extensions to the standard PSU ideal functionality (cf. Fig. 5) and a suitable protocol for implementing this extended version based mainly on symmetric-key operations. In more details, their protocol produces encrypted outputs such that the receiver only learns the intersection size in the very last decryption step. However, the resulting communication performance is not concretely efficient with more than 2GB of communication for sets of size 2^{20}.

From a high-level perspective, we operate in the same OT-based PSU framework as [13, 47], and our protocol is therefore subject to the "during-execution" leakage discussed by [26]. However, given that our protocol construction utilizes circuit-based PSI as the core building block (and therefore can be easily reprogrammed to implement slightly different functionalities), we are optimistic that a similar methodology as in the protocol of [26] (i.e., providing encrypted intermediate outputs) can be applied. Furthermore, as a practical fix, deployments of our protocol can extend rate limiting to consider partially completed protocol executions.

In a concurrent and independent work, Liu et al. [29] propose an attack on the Cuckoo hashing-based unbalanced PSU protocol of [44]. In particular, they show how the hashing-based RPMT protocol might leak information about the intersection even after using permutation to hide the location of the matches. Furthermore, they show how the computational security of the protocol depends on the size of the hash table and the number of hash functions. More precisely, increasing these parameters will provide stronger security guarantees. However, this negatively impacts the PSU performance. For instance, using the parameter recommendations from [39], the computation cost of [44] increases by 70%. Another solution briefly mentioned in [29] is to apply an OPRF before the RPMT protocol. This is precisely what we propose and evaluate, to mitigate one of the two occurring leakage issues. We give a detailed analysis of how using an OPRF in our protocol mitigates the leakage and show experimentally that the performance of our leakage-resilient protocol is very close to that of the leaky version of [44].

		Runtime in s		Communication
	Protocol	LAN	WAN	in MB
Public-key	[47]	173.44	173.96	176
	[5]	-	345.59	160
	[13]	**61.66**	**73.78**	**103**
Symm.-key	[28]	238.88	406.15	2470
	[20]	114.42	319.87	1155
	[25]	48.70	67.76	1339
	[47]	**44.78**	**59.78**	414
	[26]	49.38	225.32	2430
	Our work	53.19	77.10	**252**

Table 1: Comparison of PSU protocols. All results are taken from the respective papers for single-threaded execution and two sets with 2^{20} elements each. Best in class results are marked in bold. Note that [5] uses fairly incomparable network settings in their evaluation.

4 Our Private Set Union (PSU) Protocol

A two-party private set union (PSU) protocol comprises of a sender S and a receiver R having private input sets X and Y, respectively. At the end of the protocol execution, the receiver obtains $X \cup Y$, without learning anything else about set X. We formally define the ideal PSU functionality in Fig. 5.

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

Parameters: The sender S has as input set X and the receiver R has input set Y, where $|X| = |Y| = n$.

Functionality: Upon receiving X from S and Y from R, compute $X \cup Y$ and send it to R.

Figure 5: Ideal functionality for private set union (PSU).

In this work, we show how to construct PSU from any circuit-based PSI protocol that follows the Cuckoo/simple hashing paradigm (cf. § 2.4) and has the secret-shared intersection result as an intermediate output [41, 43]. For this, in § 4.1, we first intuitively illustrate how mq-RPMT can be instantiated from such protocols. Additionally, we point out arising information leakage issues, and how to address them. Then, in § 4.3, we formally define our PSU protocol and prove correctness as well as security.

4.1 Intuition

In Fig. 7, we recall how many circuit-based PSI protocols (including state-of-the art protocols such as [41]) work: The input sets X and Y are mapped to hash tables using Cuckoo and simple hashing, respectively. Then, some form of PMT is carried out between corresponding bins to obliviously check if the item in the Cuckoo table bin is contained in the corresponding simple hashing bin. The results of these PMT executions form a vector $\vec{Z}$, which is then usually made available in secret-shared form for further circuit computations over the intersection, e.g., computing the cardinality by counting the 1s in the vector. We call this functionality, where the parties receive the secret shares of $\vec{Z}$, "intermediate" circuit-based PSI (cf. Fig. 6).

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

Parameters: Sender S has input set X and receiver R has input set Y, where $|X| = |Y| = n$.

Functionality: Upon receiving X from S and Y from R, compute $X \cap Y$ and return secret shares z_0 and z_1 of the bit-vector $\vec{Z}$, where $Z[i] = 1$ if $x_i \in Y$, and 0 otherwise.

Figure 6: Ideal functionality for "intermediate" circuit-based PSI.

Here, we make an important observation: if the vector $\vec{Z}$ is reconstructed towards the party using simple hashing (cf. Fig. 7), this effectively is a mq-RPMT output and could be immediately used for computing PSU by requesting the elements from the Cuckoo table for which $\vec{Z}$ is 0 via OT.

However, somewhat similar to [28], this PSU construction can leak information about $X \cap Y$. Intuitively, this is because the hashing schemes place restrictions on the order of the PSU sender's vector (in mq-RPMT terminology). This can be exploited by the PSU receiver, for example, if the vector $\vec{Z}$ has a 1 in a position where there is only one element y mapped to the corresponding bin, then y must be in the intersection.

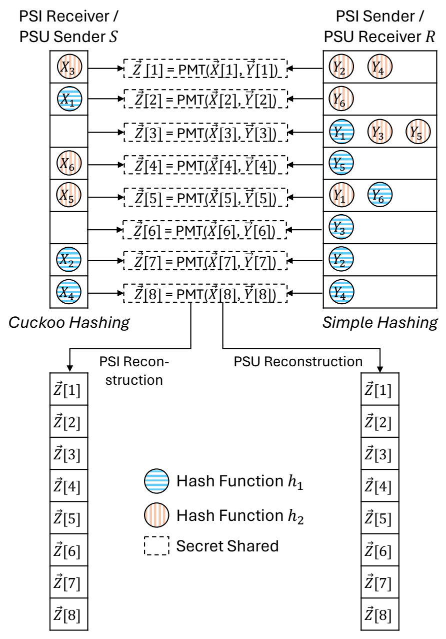


Figure 7: Conceptual visualization of "intermediate" circuit-based PSI constructions, and PSU based on circuit-based PSI, simplified to use two hash functions. Note that $\vec{Z}$ for both PSI [23, 38] and PSU cannot be directly reconstructed towards the respective receiving party (the party using Cuckoo hashing for PSI and the party using simple hashing for PSU) as this would leak additional information about private inputs. Instead, in regular circuit-based PSI (cf. Fig. 3) a (symmetric) function f would be obliviously computed over $\vec{Z}$, whereas in PSU, a permuted version of $\vec{Z}$ is reconstructed towards the PSU receiver.

To circumvent this obvious leakage, we must shuffle the intermediate circuit-based PSI result, i.e., the bit vector $\vec{Z}$, in a way such that the PSU receiver learns the shuffled bit vector in the clear and the PSU sender knows the permutation. This way, the PSU sender can re-arrange its OT inputs according to the permutation.

However, even with shuffling, there still exists a more subtle leakage issue: Once the PSU receiver learns all the new elements, the receiver can use knowledge about the hash functions to identify all of the new elements' possible locations in the Cuckoo table, thereby effectively reversing the mitigating effect of the shuffle step. To prevent this, we can apply a (multi-point) OPRF step before mapping the elements to the tables. In this step, the PSU sender learns a single key k and the PSU receiver learns $F_k(y)$ for each $y \in Y$. The inputs for the circuit-based PSI protocol are then the sets $F_k(X)$ and $F_k(Y)$. Due to not knowing k, the PSU receiver cannot speculate about the mapping of the newly learned elements.

Note that none of the two additional steps alone (OPRF and shuffling) is sufficient: if we would only use the OPRF step but not shuffling, then the PSU receiver can still infer information about the intersection from the positions of 1s in $\vec{Z}$.

4.2 Combine and Permute (CnP) Functionality

The shuffling that mitigates leakage occurring due to hashing in circuit-based PSI can be realized using a so-called "Combine and Permute" (CnP) functionality $\mathcal{F}_{\mathsf{CnP}}$ (cf. Fig. 8). This functionality can be seen as a new variant of the "Permute and Share" functionality introduced in [11]. It takes the shares of the bit vector $\vec{Z}$ as input, combines the shares, and permutes the combined vector using a random permutation π. We provide details of our protocol Π_{CnP} that instantiates this functionality in § 5, and discuss different instantiations in § 6.

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

Parameters: The sender S has as input a random permutation π and share z_0, and the receiver R has as input share z_1.

Functionality: Upon receiving (π, z_0) from S and z_1 from R, compute $\pi(z_0 \oplus z_1)$ and send it to R.

Figure 8: Ideal functionality for CnP.

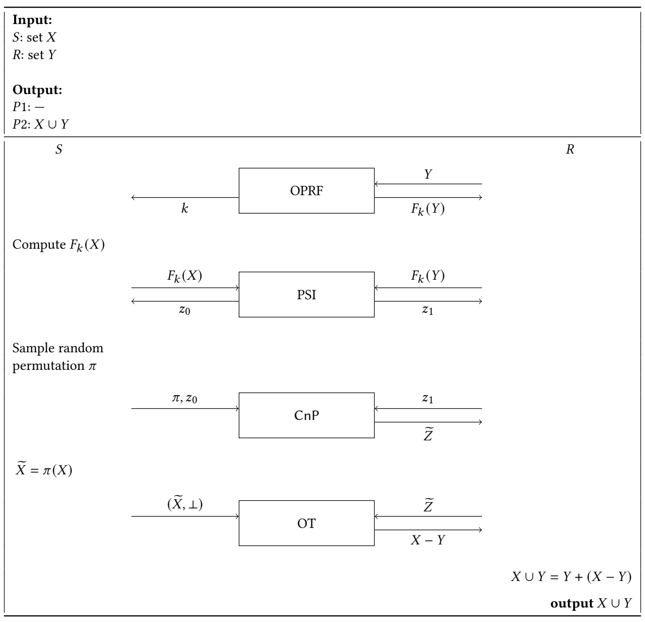


Figure 9: Overview of our PSU protocol.

4.3 Protocol Specification

Following our intuitive explanation on the ingredients we need and *why* we need them to construct PSU from circuit-based PSI, we provide an overview of the full protocol in Fig. 9. On a high level, the steps are as follows:

- The sender S and the receiver R engage in an OPRF protocol, where R inputs Y. S obtains a key k, and R obtains $F_k(Y)$.
- S and R run an "intermediate" circuit-based PSI protocol on their respective inputs $F_k(X)$ and $F_k(Y)$. The parties receive secret shares z_0, z_1 of the intermediate PSI result bit vector $\vec{Z}$.
- Then, the parties run the Combine and Permute (CnP) protocol Π_{CnP}, and R receives the combined and permuted bit vector $\widetilde{Z}$ as output.
- S and R run an OT protocol, where R inputs the permuted bit vector $\widetilde{Z}$ as the choice bit vector, and S inputs the Cuckoo hash table values in permuted order as its input.
- R combines the result of OT with its set Y, and receives $X \cup Y$.

We provide a full formal protocol definition in Fig. 10.

Protocol Π_{PSU}

Input: The sender S inputs set X and the receiver R inputs set Y. $|X| = |Y| = n$.

Protocol:

(1) The sender S and the receiver R run a multi-point OPRF protocol on their input sets X and Y, respectively. S obtains the PRF key k and R obtains $F_k(Y)$.
(2) S and R run a circuit-based PSI protocol, where S uses Cuckoo hashing and R uses simple hashing, with inputs $F_k(X)$ and $F_k(Y)$, respectively. S and R obtain the secret shares z_0 and z_1 of the intersection bit vector $\vec{Z}$ of length $(1+\epsilon)n$, respectively. Here, ϵ is a small constant Cuckoo hashing table expansion parameter.
(3) S and R run a CnP protocol, where S inputs a random permutation π and z_0, and R inputs z_1. R obtains the shuffled and combined bit vector $\widetilde{Z}$.
(4) S and R perform $(1+\epsilon)n$ OTs, where S inputs tuples $(\mathsf{T}_X[\pi(i)], \perp)$, where T_X is the Cuckoo hash table constructed by S, and R inputs choice bit $\widetilde{Z}[i]$. R obtains all $x_i \in X$ for which $x_i \notin X \bigcap Y$.
(5) R adds the received x_i to Y.

Output: The receiver R obtains $X \bigcup Y$.

Figure 10: Semi-honest PSU protocol from "intermediate" circuit-based PSI.

4.4 Leakage Analysis

Without using OPRF in the first step of our protocol, the protocol would leak some information about the sender's set. In particular, when constructing hash tables based on plaintext inputs, this leaks information to the receiver regarding which elements in the receiver's set are not in the intersection, and hence not in the sender's set.

This can be exploited by a corrupted receiver as follows: When the receiver obtains the set of elements in $X - Y$, it can hash these elements using the agreed hash functions and find the possible bin locations for each element. Corresponding to these bin locations, the receiver can infer the elements in its simple hashing table that did not match for an intersection. These elements are, therefore, not in the sender's set.

More formally, the leakage works as follows: Consider a bin i such that $x \in X$ was mapped to $\mathsf{T}_X[i]$ due to hash function h_j, i.e., $\mathsf{T}_X[i] = x$ for $h_j(x) = i$. Moreover, for some $y \in Y$ with $x = y$, $\mathsf{T}_Y[i] = y$ and $h_j(y) = i$. Now consider another element $x' \in X$ such that $x' \notin Y$. Let $h_j(x') = i'$, then for all $y = \mathsf{T}_Y[i']$, $x' \neq y \implies y \notin X$. This is because $y \notin X \cap Y$ and $y \notin X - Y$, which implies $y \notin X$.

Using an OPRF prevents the receiver from obtaining this additional knowledge about the sender's set X as follows. Let $F_k(x) = \psi$ and $F_k(y) = \phi$. Then, for $x = y$, $\psi = \phi$ whereas for $x \neq y$, $\psi \neq \phi$. Therefore, when the receiver obtains the elements in $X - Y$, it cannot compute the correct OPRF output for the elements as for each $x' \in X - Y, x' \notin Y$, and therefore $F_k(x') \neq F_k(y)$. This implies $h_j(F_k(x')) \neq h_j(F_k(y))$. Therefore, the receiver cannot compute the possible bin locations for the sender's elements.

How to quantify the leakage. The leakage described in § 4.4 is probabilistic and an adversary can only determine an element to be in the intersection or not with a certain probability.

In [29], Liu et al. show that the leakage due to the hashing-based approach depends on the number of hash functions and the size of the hash table. In this section, we analyze the leakage in detail and show that it depends not only on these parameters but also on the size of the sets and the size of the intersection. We additionally analyze the probability of this leakage and show that the probability is worse than random selection. The notations for the analysis are as follows: $s = |X \cap Y|$, $n = |X - Y|$, $m = |Y|$, and t is the size of the hash table. We consider the number of hashes to be constant, i.e. 3.

In a random selection, for any $y \in Y$,

$$P[y \in X \cap Y] = \frac{s}{m}. \tag{1}$$

Now, it remains to show that the probability of $y \in X \cap Y$ is larger than s/m in the case of the aforementioned leakage. More precisely, we have to calculate the probability of $y \in X \cap Y$ given that for some $x \in X - Y$, $h_i(x) = h_{i'}(y)$ for $i, i' \in \{1, 2, 3\}$ (considering the most common case of Cuckoo/simple hashing with three hash functions). Let event $A : y \in X \cap Y$ and event $B : h_i(x) = h_{i'}(y)$, then

$$P[B] = \frac{3}{t} \quad \text{and} \quad P[B|A] = \frac{2}{t-1}. \tag{2}$$

Using Equations (1) and (2), we get

$$\begin{aligned} P[A|B] &= \frac{P[A \cap B]}{P[B]} \\ &= \frac{P[A] \cdot P[B|A]}{P[B]} \\ &= \frac{\frac{s}{m} \cdot \frac{2}{t-1}}{\frac{3}{t}} \\ &= \frac{s}{m} \cdot \frac{2t}{3(t-1)} < \frac{s}{m}. \end{aligned} \tag{3}$$

Since in a Cuckoo hashing table, each element's location depends on other elements in X as well, it is essential to take this dependency into consideration while calculating the probability. Let us now generalize the above probability by considering n elements $x_1, \ldots, x_n \in X - Y$. The event B is described as $B : h_i(x_j) = h_{i'}(y)$ and event A is described as before. Then,

$$P[B] = \left(\frac{3}{t}\right)^n \quad \text{and} \quad P[B|A] = \left(\frac{2}{t-1}\right)^n. \tag{4}$$

Similar to Equation (3), using Equations (1) and (4), we get

$$P[A|B] = \frac{s}{m} \cdot \left(\frac{2t}{3(t-1)}\right)^n < \frac{s}{m}. \tag{5}$$

From Equation (5), we can see that as n increases, the probability of $y \in X \cap Y$ decreases. This in turn implies that the probability of $y \notin X \cap Y$ increases. In simple terms, this means that knowing the concrete values of s, m, and t, an adversary can compute the probability of some $y \notin X \cap Y$ (i.e., $1 - P[A|B]$) by finding the optimal number of x_j's that satisfy $h_i(x_j) = h_{i'}(y)$, for $i, i' \in \{1, 2, 3\}$.

In this context, note that our protocol (via the number of 1s in $\vec{Z}$) as well as state-of-the-art PSU protocols inherently leak the size of the intersection (i.e., s), and m as well as t are known protocol parameters.

4.5 Correctness and Security

We first prove the correctness of our PSU protocol Π_{PSU} (Fig. 10).

Theorem 4.1 (Correctness). *Protocol* Π_{PSU} *correctly computes the functionality* $\mathcal{F}_{\text{PSU}}$.

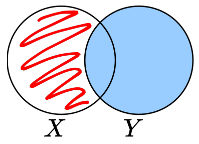


Figure 11: Venn diagram for set union.

Proof. From Fig. 11 it can be seen that the set union $X \cup Y$ can be computed by adding $X - Y$ (red area) to Y (blue area). Now we show that the protocol Π_{PSU} correctly computes and sends $X - Y$ to the receiver R, who combines it with their own set Y to get $X \cup Y$.

Let $F_k()$ be an (O)PRF. Then, by the correctness of the OPRF, we get that for any $x \in X$ and $y \in Y$, $x = y \implies F_k(x) = F_k(y)$ and $x \neq y \implies F_k(x) \neq F_k(y)$. In the next step, the OPRF outputs $F_k(X)$ and $F_k(Y)$, which are the inputs for the PSI protocol. The output of the PSI protocol is a vector $\vec{Z}$ indicating if matches were found in the bins. In our case, this would translate to: for some $x \in X$ and $y \in Y$, $F_k(x) = F_k(y)$ implies $x = y$. Therefore, the output of the PSI protocol is correct. The bit $Z[i] = 0$ if $x_i \notin Y$, and $Z[i] = 1$ if $x_i \in Y$. Permuting both the vector $\vec{Z}$ and the sender's (Cuckoo) hashing table T_X by the same permutation π would give us the same result. Now, in the last step, the parties perform OT, where the sender's input is $\{\mathsf{T}_X[\pi(i)], \perp\}$ and the receiver's input is $Z[\pi(i)]$. This ensures that when $Z[j] = 0$, the receiver obtains $x_j = \mathsf{T}_X[j]$ and when $Z[j] = 1$, the receiver obtains $\perp$, i.e., the receiver obtains all $x_i \notin X \cap Y \implies x_i \in X - Y$.

Therefore, the protocol Π_{PSU} correctly computes $X \cup Y$. □

We now prove the security of our PSU protocol in the presence of semi-honest adversaries.

Theorem 4.2 (Security). *Protocol* Π_{PSU} *securely computes the functionality* $\mathcal{F}_{\text{PSU}}$ *in a* Π_{OPRF}*, Circuit-PSI,* Π_{CnP} *and OT hybrid model.*

Proof. For proving the security of protocol Π_{PSU}, we consider an adversary $\mathcal{A}$ that corrupts either the sender S or the receiver R, and construct a simulator Sim that generates the view of the corrupted party. We also consider a trusted party TP that computes the functionalities used as subprotocols in the hybrid model, i.e., the functionalities $\mathcal{F}_{\text{OPRF}}, \mathcal{F}_{\text{CnP}}, \mathcal{F}_{\text{PSI}}$, and $\mathcal{F}_{\text{OT}}$. Then two cases arise:

Case 1: We start with the more complex case where the receiver R is corrupted. Here, the simulator Sim_R is given as input 1^n, Y and $X \cup Y$. The simulation works as follows:

(1) Sim_R sends 1^n to the TP computing the functionality $\mathcal{F}_{\mathsf{OPRF}}$, and $\mathcal{A}$ sends Y to TP. Sim_R receives key k_{Sim} from TP, and $\mathcal{A}$ receives $F_{k_{\mathsf{Sim}}}(Y)$.
(2) Sim_R randomly chooses $|X \cap Y|$ values $y'_i \in Y$ and sets $X' = (X - Y) + \{y'_i\}$.[1]
(3) Sim_R constructs a Cuckoo hashing table T'_X using $F_{k_{\mathsf{Sim}}}(X')$, sends $F_{k_{\mathsf{Sim}}}(X')$ to the TP computing the functionality $\mathcal{F}_{\mathsf{PSI}}$ and receives z'_0, and $\mathcal{A}$ receives z'_1.
(4) Sim_R selects a random permutation π', sends z'_0 and π' to the TP computing $\mathcal{F}_{\mathsf{CnP}}$, and $\mathcal{A}$ receives $\pi'(z'_0 \oplus z'_1)$.
(5) Sim_R sends $(\pi'(\mathsf{T}'_X), \perp)$ to the TP computing $\mathcal{F}_{\mathsf{OT}}$, and $\mathcal{A}$ receives $X' - Y$.

The simulated view of the receiver R is

$$\mathsf{Sim}_R(1^n,\ Y,\ X \cup Y) = (F_{k_{\mathsf{Sim}}}(Y),\ z'_1,\ \pi'(z'_0 \oplus z'_1),\ X' - Y),$$

and the view of the receiver in the real execution is

$$\mathsf{view}^{\pi}_R(Y) = (F_k(Y),\ z_1,\ \pi(z_0 \oplus z_1),\ X - Y).$$

Now, comparing the view of R in the two executions, first, we see that $X' - Y = X - Y$. Then, $F_{k_{\mathsf{Sim}}}(Y)$ and $F_k(Y)$ are indistinguishable due to the security of the OPRF protocol. z'_1 and z_1 are indistinguishable as these are secret shares, and $\pi'(z'_0 \oplus z'_1)$ and $\pi(z_0 \oplus z_1)$ are indistinguishable due to the security of the CnP protocol. Therefore, we can conclude that

$$\mathsf{Sim}_R(1^n,\ Y,\ X \cup Y) \overset{C}{\equiv} \mathsf{view}^{\pi}_R(Y).$$

Case 2: The sender S is corrupted. Here, the simulator Sim_S gets the set X as input. The simulation works as follows:

(1) Sim_S samples random elements y'_i and sends it to the TP that computes the functionality $\mathcal{F}_{\mathsf{OPRF}}$ and receives $F_{k_{\mathsf{Sim}}}(Y)$, and $\mathcal{A}$ receives k_{Sim}.
(2) Sim_S sends $F_{k_{\mathsf{Sim}}}(Y)$ to the TP computing $\mathcal{F}_{\mathsf{PSI}}$ and receives z'_1, and $\mathcal{A}$ receives z'_0.
(3) Sim_S sends z'_1 to the TP computing $\mathcal{F}_{\mathsf{CnP}}$ and receives $\pi'(z'_0 \oplus z'_1)$.
(4) Sim_S sends $\pi'(z'_0 \oplus z'_1)$ to the TP computing $\mathcal{F}_{\mathsf{OT}}$ and receives $X - Y'$.

The simulated view of the sender S is

$$\mathsf{Sim}_S(1^n,\ X) = (k_{\mathsf{Sim}},\ z'_0),$$

and the view of the receiver in the real execution is

$$\mathsf{view}^{\pi}_S(X) = (k,\ z_0).$$

From the two views, we can see that k_{Sim} and k are computationally indistinguishable due to the security of the OPRF protocol, and z'_0 and z_0 are indistinguishable as they are secret shares. Therefore,

$$\mathsf{Sim}_S(1^n,\ X) \overset{C}{\equiv} \mathsf{view}^{\pi}_S(X).$$

Thus, the protocol Π_{PSU} securely computes the functionality $\mathcal{F}_{\mathsf{PSU}}$. □

[1]$X - Y = (X \cup Y) - Y$.

5 Combine and Permute (CnP) Protocol

As output of the circuit-based PSI step, the PSU sender S and receiver R obtain shares of a bit vector that indicates the bins of the Cuckoo hashing table which were positive for intersection. In order for the receiver to obtain the elements of the union, i.e., the elements in X that are not in the intersection, R requires the combined bit vector that can later be used for the OT phase. However, naïvely sending the combined bit vector to the receiver leaks information about the bin locations that gave an intersection (cf. our leakage discussion in § 4.4).

Therefore, we propose a novel "Combine and Permute" CnP protocol to instantiate the functionality $\mathcal{F}_{\mathsf{CnP}}$ described in § 4.2. In this protocol, the combined bit vector is first permuted before sending to the receiver, thus dissociating the previous relation to the bin locations. For this protocol, the sender chooses the random permutation and permutes its own set accordingly, so that the correctness of the PSU protocol is maintained.

Our protocol is formally described in Fig. 12. It uses the "Permute and Share" protocol of [11] as inspiration, where a vector of elements is permuted and then secret shared among two parties. We modify this protocol to take two secret shares as input, combine the shares, and permute the combined vector. The resulting permuted vector is then provided to the receiver. Note that in our case, the sender knows the random permutation π that is used for the permutation.

Protocol Π_{CnP}

Input: The sender S has as input a random permutation π and share x_0, and the receiver R has as input share x_1.

Protocol:

(1) S and R run one instance of ShTr with S providing as input permutation π. R obtains a and b, and S obtains Δ.
(2) R sends $m = x_1 \oplus a$ to S.
(3) S computes $\tilde{m} = \pi(m \oplus x_0) \oplus \Delta$ and sends it to R.
(4) R computes $\tilde{x} = \tilde{m} \oplus b$.

Output: The receiver R obtains the permuted combined vector $\tilde{x}$.

Figure 12: Our semi-honest protocol for CnP.

5.1 Share Translation (ShTr)

Similar to [11], our CnP protocol has a "share translation" (ShTr) protocol as the core building block. The ideal functionality $\mathcal{F}_{\mathsf{ShTr}}$ is depicted in Fig. 13. It takes a random permutation as input from the sender, and returns some correlated random values to the sender and the receiver. These random values are then later used for the instantiation of CnP. This functionality can be implemented using different techniques as will be discussed in § 6.3.2.

5.2 Correctness and Security of CnP

In this section, we prove the correctness and security of the protocol Π_{CnP} (cf. Fig. 12) that instantiates the CnP functionality $\mathcal{F}_{\mathsf{CnP}}$.

Correctness: The correctness of the CnP protocol Π_{CnP} depends on the correctness of the ShTr protocol. The ShTr protocol takes π as input from the sender S and returns random values a, b to the

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

Parameters: The sender S has as input a random permutation π.

Functionality: Upon receiving π form the sender, the functionality returns two random vectors a and b to the receiver R, and $\Delta = \pi(a) \oplus b$ to S.

Figure 13: Ideal functionality for share translation.

receiver R, and $\Delta = \pi(a) \oplus b$ to S. R sends $m = x_1 \oplus a$ to S. Then, S computes $\tilde{m} = \pi(m \oplus x_0) \oplus \Delta$ and sends it to R, where

$$\begin{aligned}\tilde{m} &= \pi(m \oplus x_0) \oplus \Delta \\ &= \pi(x_1 \oplus a \oplus x_0) \oplus \pi(a) \oplus b \\ &= \pi(x_0 \oplus x_1) \oplus \pi(a) \oplus \pi(a) \oplus b \\ &= \pi(x) \oplus b\end{aligned}$$

R then computes $\tilde{x} = \tilde{m} \oplus b$ to get $\tilde{x} = \pi(x)$, therefore obtaining the permuted and combined bit vector.

Security: Here, we prove the security of protocol Π_{CnP}.

Theorem 5.1. *The protocol Π_{CnP} securely computes the functionality $\mathcal{F}_{\mathsf{CnP}}$ in a* ShTr-*hybrid.*

Proof. For proving the security of protocol Π_{CnP}, we first consider an adversary that corrupts either the sender S or the receiver R. Then two cases arise:

Case 1: The sender S is corrupted by $\mathcal{A}$. Consider a simulator Sim_S that has as inputs the inputs of S, i.e., the share x_0 and π. As S does not receive any output, the simulation in this case is straightforward and goes as follows:

- The sender S and the simulator Sim_S run the ShTr protocol, acting as sender and receiver, respectively, and receive the corresponding outputs.
- Sim_S samples a random vector $x'_1 \overset{\$}{\leftarrow} \{0,1\}^{|x_0|}$, and sends $x'_1 \oplus b'$ to S.
- S continues the protocol as in the real execution.

The view of the sender S in the simulation and in the real execution differ only in the message S receives. Since the share x_1 is random and x'_1 is sampled randomly from the same message space, and since b and b' are generated by the ShTr protocol and are random, the messages $x_1 \oplus b$ and $x'_1 \oplus b'$ are indistinguishable. Therefore, the view of S in the simulation is indistinguishable to its view in the real execution.

Case 2: The receiver R is corrupted by the adversary $\mathcal{A}$. Consider a simulator Sim_R for the receiver that has as inputs the input and output of R, i.e., x_1 and $\pi(x)$. The simulation works as follows:

- The simulator Sim_R samples a random bit vector x' such that it has the same Hamming weight as $\pi(x)$. Then, Sim_R computes $x'_0 = x' \oplus x_1$. Also, Sim_R selects a random permutation π'.
- The receiver R and the simulator Sim_R run the ShTr protocol as the receiver and the sender, respectively. Sim_R gives as input π' and receives Δ', and R receives a' and b'.
- Upon receiving $x_1 \oplus a'$ from R, the simulator computes $\widetilde{m}' = \pi'(x_1 \oplus a' \oplus x'_0) \oplus \Delta$ and sends it to R.
- R computes $\pi'(x') = \widetilde{m}' \oplus b'$.

In this simulation, the view of the receiver R differs in the received $\widetilde{m}'$ and the output $\pi'(x')$. However, since all involved values in $\widetilde{m}'$ are randomly selected and due to the security of the ShTr protocol, $\widetilde{m}$ and $\widetilde{m}'$ are indistinguishable. With respect to $\pi(x)$ and $\pi'(x')$, since x and x' have the same Hamming weight, and π and π' are randomly chosen permutations, they are indistinguishable as well. Thus, the view of R in the real and simulated executions are indistinguishable.

Therefore, the protocol Π_{CnP} is secure in a ShTr-hybrid. □

6 Instantiating the Building Blocks

In this section, we discuss how the building blocks of our protocol can be instantiated efficiently. Note that each of our building blocks is sequentially executed in a black-box fashion, and therefore can easily be upgraded with any future protocol that securely and efficiently implements the respective ideal functionality.

6.1 OPRF

For instantiating the ideal OPRF functionality $\mathcal{F}_{\mathsf{OPRF}}$ (cf. Fig. 2), we rely on the SoK in [9] to identify the most suitable protocol. The required properties are to be efficient on batched inputs and to be a multi-point OPRF, i.e., use the same PRF key for all inputs. As per [9, Tab. 3], the state-of-the-art work in this category is the lightweight OPRF construction of [12]. We refer to [12] for the full protocol description along with correctness and security proofs.

Complexity. The multi-point OPRF protocol of [12] has an asymptotic complexity of $O(n)$, and the concrete communication cost for 2^{20} elements of 128 bits each is 84.79 MB. From the experimental results given in Tab. 2, it can be seen that the implementation gives the expected value.

6.2 Circuit-based PSI

We suggest to utilize the state-of-the-art circuit-based PSI protocol of [41], for which the authors report unprecedented run-times and communication results. Integrating this protocol only requires minor modifications in terms of notation and implementation to suit our PSU protocol. Specifically, we interchange the functions of the sender and the receiver such that the sender constructs a Cuckoo hashing table and the receiver a simple hashing table. Moreover, the inputs of both parties are the OPRF evaluations of their respective sets. Let the sender set be X and the receiver's set be Y, then $\Psi = F_k(X)$ and $\Phi = F_k(Y)$ are the OPRF evaluations of their respective sets. The modified PSI protocol works as follows:

(1) The sender constructs a Cuckoo hashing table of their set Ψ and the receiver constructs a simple hashing table of their set Φ.

(2) For each $i \in [m]$, the receiver samples a random value $r_i \leftarrow \{0,1\}^l$, where $l = \lambda + \log_2 m$ and m is the size of the Cuckoo hashing table. For all i and $y \in \mathsf{T}_Y[i]$, the receiver constructs a list $L = \{(\phi', r_i)\}$, where $\phi' = h_j(\phi)$.
The sender constructs a set Ψ', which is defined as the collection of all $h_j(\psi)$ such that ϕ is stored at $\mathsf{T}_Y[h_j(\psi)]$.

(3) The sender and the receiver then evaluate a programmable OPRF (OPPRF) with Ψ' and L as their respective inputs,

and the sender obtains Ψ^* as output, where $\Psi^* = r'_i$ such that $r'_i = r_i$ if $\psi = \phi$, and $r'_i \neq r_i$ otherwise.

(4) The sender and the receiver use a generic MPC protocol to check equality between each r'_i and r_i pair and secret share the result between the two parties.

Complexity. One of the first circuit-based PSI protocols of [23] uses a "sort-compare-shuffle" (SCS) circuit to compute the intersection. This SCS protocol has an asymptotic complexity of $O(n \log n)$ and a very high concrete communication cost of more than 800 MB. Instead, the state-of-the-art "blazing fast" circuit-based PSI protocol of [41] achieves an asymptotic communication complexity of $O(nl)$, where l is the bit length of the elements. The concrete communication cost for 2^{20} elements of 128 bits is 120.72 MB. However, the run-time results reported in the original publication are achieved using "silver" OT [14], which was shown to be insecure [42]. For our experiments, we therefore use "silent" OT [7, 42] instead.

6.3 Shuffling

For the shuffling step, the ideal functionality for "Combine and Permute" ($\mathcal{F}_{\mathsf{CnP}}$, cf. Fig. 8) is realized. We explore two main options for this. For the first option, we acknowledge that it is possible to simply implement this as a classical permutation network in a circuit. The second option is to use our new protocol Π_{CnP} (cf. Fig. 12), for which various instantiation options do exist.

6.3.1 Permutation Networks. Implementing permutation networks in a circuit is a classical approach to shuffling, often proposed in the context of circuit-based PSI protocols when there is no symmetric function computed on top of the intersection result [23]. There are two prominent approaches in this category: Benes [4] and Waksman [2, 45] permutation networks. For an $n \times n$ permutation network, the Waksman network requires $\frac{n}{2} - 1$ fewer switches than the Benes network. Therefore, we recommend to use Waksman networks when pursuing this option.

Complexity. This approach has an asymptotic complexity of $O(n \log n)$.

6.3.2 Combine and Permute Protocols. As an alternative to classic permutation networks, there exist dedicated two-party shuffling protocols, e.g., [11, 31]. The state-of-the-art work in this category is [11], proposing the first shuffle protocol with mainly symmetric-key techniques and $O(n \log n)$ complexity. We closely analyze the protocol of [11] for applying it to our PSU protocol, and realize it can be significantly modified and optimized for our purposes, resulting in our CnP protocol shown in Fig. 12. As discussed in § 5.1, this protocol requires an underlying sub-protocol ShTr (cf. Fig. 13), which in turn can be instantiated in a number of ways:

- **Permutation networks.** Interestingly, not only the entire $\mathcal{F}_{\mathsf{CnP}}$ functionality, but also the $\mathcal{F}_{\mathsf{ShTr}}$ functionality as a sub-component of our CnP protocol Π_{CnP} can be instantiated by implementing permutation networks in a circuit. Here, the sender selects a permutation π, and the receiver selects two random strings a and b. Now we can permute a using permutation π in a GMW [21] circuit, add b to $\pi(a)$, and return the result to the sender.
Complexity. This method has $O(n \log n)$ complexity and is the one we implement (using the Waksman [2, 45] permutation network) and evaluate in § 7.
- **Oblivious Punctured Vectors (OPV).** The OPV-based approach of [11] for the ShTr protocol can be applied to our setting with some modifications. More precisely, the OPV primitive is used to construct two $n \times n$ matrices V_s and V_r for the sender and the receiver, respectively. The sender then uses its matrix V_s to construct Δ as follows: for $i \in [n]$, $\Delta[i] = \sum_{j \neq \pi(i)} V_s[i][j] - \sum_{j \neq i} V_s[j][\pi(i)]$. The receiver uses its matrix V_r to obtain a and b as follows: for $i \in [n]$, $a[i] = \sum_i V_r[i][j]$, $b[i] = \sum_j V_r[j][i]$.
Complexity. This approach has $O(n \log n)$ complexity. However, this method has a high concrete communication cost of approximately 1.8 GB. This is mainly due to the inefficiency of the protocol for shuffling 1-bit inputs.
- **Permutation Correlations.** In the recent work of [33], a permutation correlation generator based on MPC-friendly PRFs is proposed. This correlation generator can be used to instantiate the ShTr protocol using n evaluations of such PRFs. More precisely, the sender inputs $\pi(i)$, and the receiver inputs the PRF key k; then, the sender and the receiver obtain secret shares of $\pi(a)$ as outputs. I.e., the sender receives $\Delta = \pi(a) + b$, and the receiver obtains b and computes $a[i] = F_k(i)$.
Complexity. The PRF-based solution of [33] has an asymptotic complexity of $O(n)$ and a concrete communication cost of approximately 16 MB.[2]

6.4 OT

To obliviously transfer missing elements of X from S to R, we need to instantiate 1-out-of-2 OT. For this, we can rely on communication efficient silent OT [8, 42] in the setup phase to generate sufficiently many random OTs. Using Beaver's OT precomputation [3], we can derandomize these precomputed random OTs using the actual inputs in the online phase of the protocol.

The derandomization is done as follows: After running random OT, the sender gets two random values m_0 and m_1; the receiver obtains m_r, corresponding to its random choice bit r. Now, in the online phase, the receiver reports $c' = c \oplus r$ to the sender, where c is the actual choice bit. The sender gives either $(x'_0 = x_0 \oplus m_0, x'_1 = x_1 \oplus m_1)$ to the receiver if $c' = 0$, or $(x'_0 = x_0 \oplus m_1, x'_1 = x_1 \oplus m_0)$ if $c' = 1$. The receiver can then recover x_c as $x_c = x'_c \oplus m_r$.

Since the sender's second input in our OT phase is always $\perp$, we propose an optimization to significantly reduce the amount of communication. For this, we only send the first element of the correction step (i.e., x'_0) in the online phase. The receiver can then still reconstruct the output (if necessary) using the precomputed random OT messages. This optimization reduces the communication in the OT phase by half.

Complexity. For transferring the missing elements, we require $1.4n$ OTs, where $1.4n$ is the concrete size of the Cuckoo hashing

[2]Since their implementation is not yet available online, we could not verify this result experimentally.

table (setting $\epsilon = 0.4$ for "stashless" hashing when using three hash functions [41]). Using the optimization mentioned above, the sender needs to send $1.4n$ elements to the receiver. Using silent OT for precomputation, the OT step costs about 23 MB for $1.4 \cdot 2^{20}$ elements of 128 bits each.

7 Performance Evaluation

In this section, we describe the implementation of our PSU protocol, the evaluation setup, provide (micro) benchmarks, and compare our results to previous works. We provide additional benchmark results for varying set sizes and the unbalanced setting in Appendix A.

7.1 Implementation

For our implementation, we rely on multiple existing implementations of protocol building blocks. Our OPRF implementation of the protocol of [12] is based on the implementation included in [25][3]. For the circuit-based PSI protocol, we utilize the implementation of [41][4]. We replace the silver OT component with the latest version of silent OT [8], provided as part of libOTe[5]. For shuffling, we implement our CnP protocol with the share translation instantiated with a Waksman network in the BetaCircuit framework of the cryptoTools library[6].

7.2 Setup

We execute our implementation on two Linux servers with Intel(R) Core(TM) i9-7960X CPU @ 2.80GHz CPU and 128 GB RAM, connected via a 10 Gbit/s switch. Our run-times are reported in two network configurations, simulated by restricting bandwidth and increasing RTT via the Linux `tc` command: A LAN configuration with 10 Gbit/s bandwidth and 1 ms RTT, and a WAN configuration with 100Mbit/s bandwidth and 100 ms RTT. We use the same setup to evaluate the PSU protocol of [13]. We evaluate all implementations as single-threaded programs.

7.3 (Micro) Benchmarks

In Tab. 2, we report on the offline/online run-time as well as the communication overhead for each of the protocol phases described in § 6. We note that 15-20% of our protocol's run-time is in the setup phase, which is an advantage compared to purely public-key-based protocols where almost all overhead (except for the OT precomputation) occurs in the online phase. Furthermore, we note that the protocol overhead is dominated by the circuit-based PSI protocol (approx. 30 seconds). This is *unexpected*, as the paper [41] reports half the run-time (approx. 15 seconds) in a comparable hardware configuration. We speculate that this is mainly due to the replacement of silver OT in the original protocol with silent OT, which is necessary as the silver codes turned out to be insecure [42].

7.4 Comparison to Related Works

In Tab. 4, we compare the total run-time and communication overhead of our protocol with previous works. We run the implementation of the state-of-the-art protocol of [13] in our benchmark environment. For the remaining works (marked with *), we take the run-times and communication numbers from the original publications.

Protocol-Phase	Run-Time in ms		Communication
	LAN	WAN	in MB
Setup			
CPSI	0.05	0.05	0.00
CNP	10 616.43	11 622.09	0.39
OT	181.05	1 189.91	0.32
Total	10 805.50	12 812.10	0.72
Online			
MP-OPRF	11 403.41	12 717.22	84.81
CPSI	27 644.70	39 945.82	128.58
CNP	3 265.50	9 611.30	16.86
OT	66.00	2 012.84	21.23
Total	42 379.60	64 287.17	251.48
Total	53 185.10	77 099.27	252.19

Table 2: Run-time and communication overhead for all phases of our PSU protocol for $n = 2^{20}$ elements. Here, CPSI=Circuit-based PSI, CnP=Combine and Permute, MP-OPRF=Multi-point OPRF.

Protocol	Run-Time in ms		Communication
	LAN	WAN	in MB
Chen et al. [13]	61 660.98	73 782.90	103.21
Our work	53 185.10	77 099.27	252.19

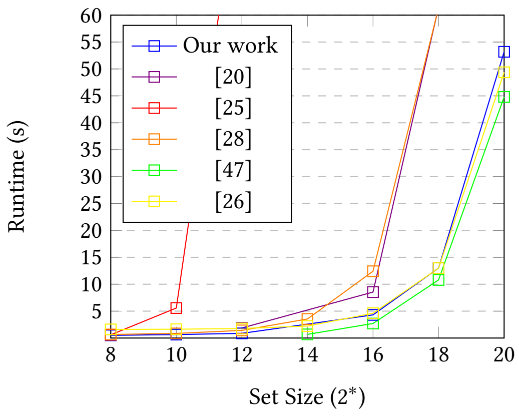
Table 3: Total run-time and communication overhead for our and previous PSU protocol of [13] for $n = 2^{20}$ elements.

Furthermore, we calculate the cost of our linear complexity protocol (marked with †) using a theoretical cost analysis, but have not implemented it because the code for the underlying novel permutation correlation generator of [33] is not yet publicly available. As shown in Tab. 4, while using the technique of [33] gives a nice asymptotic result (linear complexity), the overall concrete communication cost for relevant set sizes is virtually equivalent to the naïve approach based on permutation networks. Evaluating potential run-time differences is part of future work.

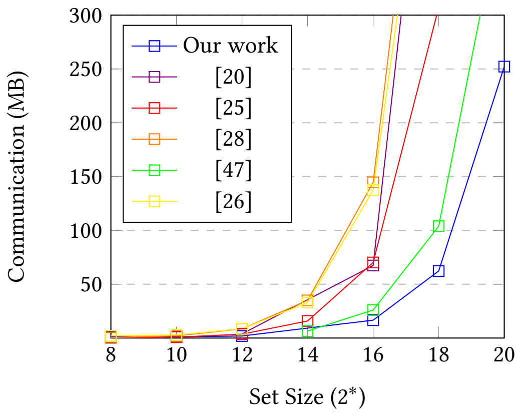
Compared to the state-of-the-art protocol of [13], we achieve better total run-time in the LAN setting, while incurring 2.5× more communication. The main benefit of our modular protocol is that it can be upgraded easily with further advances in all building blocks. With respect to the other public key-based protocol of [47], we achieve approximately 3× better run-time in LAN and 2× in WAN.

Now, comparing to the symmetric key-based protocols [25, 28, 47], our protocol has 1.6 − 9.8× less communication, and has 3.2× and 4.5× better run-times compared to [25] and [28], respectively. In the LAN setting, our protocol is 0.9× slower than [47]. However, our protocol has an online run-time of 42.38s, which is very close to the reported online run-time of [47] and, more importantly, has 1.6× lower communication. We also compare the scaling of protocol with

[3]https://github.com/dujiajun/PSU
[4]https://github.com/Visa-Research/volepsi
[5]https://github.com/osu-crypto/libOTe
[6]https://github.com/ladnir/cryptoTools



(a) Run-time (LAN) vs Size



(b) Communication vs Size

Figure 14: Scaling of run-time (a) and communication (b) with increasing set sizes $n = \{2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, 2^{20}\}$ for symmetric key-based PSU protocols.

Protocol	**Operations *-key**	**Asymptotic Comm.**	**Total Comm. in MB**	**Run-Time in s**	
				LAN	**WAN**
Kolesnikov et al., ASIACRYPT'19 [28]*	Symm	$O(n \log n)$	2 470.00	238.88	406.15
Garimella et al., PKC'21 [20]*	Symm	$O(n \log n)$	1 155.00	114.42	319.87
Jia et al., USENIX Sec'22 [25]*	Symm	$O(n \log n)$	1 339.00	173.15	266.98
Zhang et al., USENIX Sec'23 [47]*	Public	$O(n)$	176.00	173.40	184.55
Zhang et al., USENIX Sec'23 [47]*	Symm	$O(n)$	414.00	44.78	86.13
Chen et al., PKC'24 [13]	Public	$O(n)$	103.00	61.66	73.78
Jia et al., USENIX Sec'24 [26]	Symm	$O(n \log n)$	2 430.00	49.38	225.32
Our work (w/ Perm. Network)	Symm	$O(n \log n)$	252.00	53.19	77.09
Our work (w/ [33])†	Symm	$O(n)$	252.00	–	–

Table 4: Comparison of PSU protocols in "chronological" order with respect to concrete and asymptotic communication cost and run-time. All costs are for $n = 2^{20}$ elements. $*$ indicates that the results are taken from the original paper, and † indicates that the result is theoretical. The result of [5] is excluded from this table due to incomparable network settings in their benchmarks.

the other symmetric-key based protocols with respect to increasing set sizes (cf. Fig. 14). In Fig. 14a, it can be seen that our runtime scales almost similar to that of [26, 47]. Fig. 14b shows that our communication costs scale better than all other symmetric-key based protocols. The benchmark results of our protocol for different set sizes are given in Tab. 5.

We also compare our results to the leaky PSU protocol of [44]. For the balanced setting, where the parties have 2^{10} elements each, [44] presents a runtime of 0.86s in LAN and 3.73s in WAN, while we have 0.64s (1.3× faster) in LAN and 6.16s (1.6× slower) in WAN. Their protocol requires 2.42 MB communication, while ours is 1.21 MB (2× lower). This shows that, contrary to the result of [29], we mitigate the leakage caused by Cuckoo hashing without adding significant performance overhead. We also benchmark our protocol in the unbalanced setting and discuss the results in Tab. 6 in § A.2. Moreover, as mentioned earlier, our protocol will directly benefit from any future improvement in the sub-protocols.

8 Conclusion and Future Work

We propose an efficient symmetric-key-based PSU that employs circuit-based PSI as a core building block. Our protocol is built on four sequentially executed sub-components, allowing to replace each component with a suitable protocol. For now, in terms of performance, we present an appealing new trade-off between public- and symmetric-key-based PSU protocols. In the future, our modular protocol has a great potential for further improvements as each component's upgrades will directly boost the overall performance. In addition, more functions can be computed on top of the simple PSU using straightforward changes to our protocol.

Acknowledgements

This project received funding from the ERC under the EU's Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI). It was co-funded by the DFG within SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230.

References

[1] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. 2015. Ciphers for MPC and FHE. In *EUROCRYPT*.

[2] Bruno Beauquier and Éric Darrot. 2002. On Arbitrary Size Waksman Networks and Their Vulnerability. *Parallel Process. Lett.* (2002).

[3] Donald Beaver. 1995. Precomputing Oblivious Transfer. In *CRYPTO*.

[4] V. E. Beneš. 1964. Optimal rearrangeable multistage connecting networks. *The Bell System Technical Journal* (1964).

[5] Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. 2023. Near-Optimal Oblivious Key-Value Stores for Efficient PSI, PSU and Volume-Hiding Multi-Maps. In *USENIX Security Symposium*.

[6] Marina Blanton and Everaldo Aguiar. 2012. Private and oblivious set and multiset operations. In *ASIACCS*.

[7] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. 2019. Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation. In *CCS*.

[8] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. 2019. Efficient Pseudorandom Correlation Generators: Silent OT Extension and More. In *CRYPTO*.

[9] Sílvia Casacuberta, Julia Hesse, and Anja Lehmann. 2022. SoK: Oblivious Pseudorandom Functions. In *EUROS&P*.

[10] Nishanth Chandran, Divya Gupta, and Akash Shah. 2022. Circuit-PSI With Linear Complexity via Relaxed Batch OPPRF. *PoPETs*.

[11] Melissa Chase, Esha Ghosh, and Oxana Poburinnaya. 2020. Secret-Shared Shuffle. In *ASIACRYPT*.

[12] Melissa Chase and Peihan Miao. 2020. Private Set Intersection in the Internet Setting from Lightweight Oblivious PRF. In *CRYPTO*.

[13] Yu Chen, Min Zhang, Cong Zhang, and Minglang Dong. 2024. Private Set Operations from Multi-Query Reverse Private Membership Test. In *PKC*.

[14] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. 2021. Silver: Silent VOLE and Oblivious Transfer from Hardness of Decoding Structured LDPC Codes. In *CRYPTO*.

[15] Alex Davidson and Carlos Cid. 2017. An Efficient Toolkit for Computing Private Set Operations. In *ACISP*.

[16] Minglang Dong, Yu Chen, Cong Zhang, and Yujie Bai. 2024. Breaking Free: Efficient Multi-Party Private Set Union Without Non-Collusion Assumptions. *CoRR* (2024).

[17] Jean-Guillaume Dumas, Alexis Galan, Bruno Grenet, Aude Maignan, and Daniel S. Roche. 2024. Communication Optimal Unbalanced Private Set Union. *CoRR* abs/2402.16393 (2024).

[18] Keith B. Frikken. 2007. Privacy-Preserving Set Union. In *ACNS*.

[19] Jiahui Gao, Son Nguyen, and Ni Trieu. 2023. Toward A Practical Multi-party Private Set Union. *Cryptology ePrint Archive* (2023).

[20] Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh. 2021. Private Set Operations from Oblivious Switching. In *PKC*.

[21] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *STOC*.

[22] Carmit Hazay and Kobbi Nissim. 2010. Efficient Set Operations in the Presence of Malicious Adversaries. In *PKC*.

[23] Yan Huang, David Evans, and Jonathan Katz. 2012. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?. In *NDSS*.

[24] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending Oblivious Transfers Efficiently. In *CRYPTO*.

[25] Yanxue Jia, Shifeng Sun, Hong-Sheng Zhou, Jiajun Du, and Dawu Gu. 2022. Shuffle-based Private Set Union: Faster and More Secure. In *USENIX Security Symposium*.

[26] Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, and Dawu Gu. 2024. Scalable Private Set Union, with Stronger Security. In *USENIX Security Symposium*.

[27] Lea Kissner and Dawn Xiaodong Song. 2005. Privacy-Preserving Set Operations. In *CRYPTO*.

[28] Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. 2019. Scalable Private Set Union from Symmetric-Key Techniques. In *ASIACRYPT*.

[29] Keyang Liu, Xingxin Li, and Tsuyoshi Takagi. 2024. Review the Cuckoo Hash-Based Unbalanced Private Set Union: Leakage, Fix, and Optimization. In *ESORICS*.

[30] Xiang Liu and Ying Gao. 2023. Scalable Multi-party Private Set Union from Multi-query Secret-Shared Private Membership Test. In *ASIACRYPT*.

[31] Payman Mohassel and Seyed Saeed Sadeghian. 2013. How to Hide Circuits in MPC an Efficient Framework for Private Function Evaluation. In *EUROCRYPT*.

[32] Rasmus Pagh and Flemming Friche Rodler. 2001. Cuckoo Hashing. In *ESA*.

[33] Stanislav Peceny, Srinivasan Raghuraman, Peter Rindal, and Harshal Shah. 2024. Efficient Permutation Correlations and Batched Random Access for Two-Party Computation. *Cryptology ePrint Archive* (2024).

[34] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2020. PSI from PaXoS: Fast, Malicious Private Set Intersection. In *EUROCRYPT*.

[35] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. 2015. Phasing: Private Set Intersection Using Permutation-based Hashing. In *USENIX Security Symposium*.

[36] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. 2019. Efficient Circuit-Based PSI with Linear Communication. In *EUROCRYPT*.

[37] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. 2018. Efficient Circuit-Based PSI via Cuckoo Hashing. In *EUROCRYPT*.

[38] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2014. Faster Private Set Intersection Based on OT Extension. In *USENIX Security Symposium*.

[39] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2018. Scalable Private Set Intersection Based on OT Extension. *ACM Trans. Priv. Secur.* (2018).

[40] Michael O. Rabin. 1981. How To Exchange Secrets with Oblivious Transfer. *Technical Report TR-81, Aiken Computation Laboratory, Harvard University* (1981).

[41] Srinivasan Raghuraman and Peter Rindal. 2022. Blazing Fast PSI from Improved OKVS and Subfield VOLE. In *CCS*.

[42] Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. 2023. Expand-Convolute Codes for Pseudorandom Correlation Generators from LPN. In *CRYPTO*.

[43] Peter Rindal and Phillipp Schoppmann. 2021. VOLE-PSI: Fast OPRF and Circuit-PSI from Vector-OLE. In *EUROCRYPT*.

[44] Binbin Tu, Yu Chen, Qi Liu, and Cong Zhang. 2023. Fast Unbalanced Private Set Union from Fully Homomorphic Encryption. In *CCS*.

[45] Abraham Waksman. 1968. A Permutation Network. *J. ACM* (1968).

[46] Andrew Chi-Chih Yao. 1986. How to Generate and Exchange Secrets (Extended Abstract). In *FOCS*.

[47] Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin. 2023. Linear Private Set Union from Multi-Query Reverse Private Membership Test. In *USENIX Security Symposium*.

A Additional Benchmarks

A.1 Different Set Sizes

We also run our protocol for varying set sizes $n = \{2^8, 2^{10}, 2^{12}, 2^{16}, 2^{18}, 2^{20}\}$, and benchmark the run-time and communication overhead (cf. Tab. 5)[7]. We furthermore compare the scaling of our protocol with that of the other symmetric-key-based protocols, using performance results reported in the original publications. Fig. 14a shows that our protocol is slightly worse than that of [47] in terms of run-time. However, the communication cost of our protocol scales much better than that of [47], as we demonstrate in Fig. 14b.

Set Size	Run-Time in ms		Communication
	LAN	WAN	in MB
2^8	520.00	5 790.00	0.98
2^{10}	642.00	6 162.00	1.21
2^{12}	870.00	7 021.00	1.89
2^{16}	4 305.00	12 979.00	16.60
2^{18}	13 028.00	26 795.00	62.27
2^{20}	53 185.00	77 099.00	252.20

Table 5: Total run-time and communication overhead for our PSU protocol for increasing set sizes from $n = 2^8$ up to $n = 2^{20}$ elements.

A.2 Unbalanced PSU

Although our protocol is designed for the balanced case, it can still be used to compute the set union in the unbalanced setting. We benchmark the communication and computation costs for the unbalanced case and present the results in Tab. 6. We compare our results with that of [44] and observe that our runtime in the LAN setting is consistently faster, while in the WAN setting, it is slower. Our communication cost for the smaller size differences is lower than that of [44], but it increases steadily when the difference

[7]For $n = 2^{14}$, the circuit-based PSI implementation of [43] is faulty and, therefore, could not be used for our benchmark.

$\|Y\|$	Run-Time in s				Communication in MB	
	LAN		WAN			
	Tu et al. [44]	Ours	Tu et al. [44]	Ours	Tu et al. [44]	Ours
2^{10}	0.87	0.64	3.73	6.16	2.42	1.21
2^{11}	0.87	0.64	3.78	6.96	2.42	1.31
2^{12}	0.87	0.70	3.73	6.13	2.42	1.51
2^{14}	1.17	0.81	4.43	6.39	3.00	2.74
2^{15}	1.12	0.96	4.38	6.64	3.00	4.39
2^{16}	1.37	1.24	4.39	7.00	3.00	7.68

Table 6: Comparison of run-time and communication overhead of our PSU protocol with that of [44] for the unbalanced setting where the sender has a set of size $|X| = 2^{10}$, and the receiver has varying set sizes from $|Y| = 2^{10}$ to $|Y| = 2^{16}$.

between the set sizes increases. In contrast, the communication cost remains essentially constant for [44]. This can mostly be attributed to the fact that, although we support unbalanced cases, our protocol is optimized for the balanced scenario, while the protocol in [44] is optimized for the unbalanced one.

B Extensions to PSU

Due to the modular construction of our protocol, we can easily replace the intermediate protocols to accommodate more functionalities in addition to the simple set union. Here, we briefly sketch some of the extensions that are possible to our PSU protocol.

PSU with associated values. To obtain associated values along with the normal PSU, also known as *labeled PSU*, our PSU protocol can be easily extended by simply including the associated values in the final OT step.

PSU Cardinality. We can also compute the PSU cardinality by omitting the last two steps, i.e., the CnP and the OT step, and using a circuit-based PSI cardinality (PSI-CA) protocol instead of simple circuit-based PSI. The result of PSI-CA can then be used to compute the PSU cardinality. The computation of PSU-CA can be done in the circuit itself to hide the intermediate PSI-CA.

PSU Sum. The PSU Sum functionality lets the receiver learn the sum of the associated values of all elements in the union. Since the receiver knows its own associated values, all that remains is to send the sum of the associated values for elements in $X - Y$. This can be done as follows: The sender samples n random values r_i. Then, in the OT step, the sender sets its input to be $(r_i + a_i, r_i)$, where a_i is the associated value, and the receiver has the CnP step's output as its choice bit string z_i. At the end of the OT phase, the receiver can calculate $S' = \Sigma r_i + z_i \cdot a_i$. The sender can send Σr_i to the receiver, and the receiver can finally calculate the sum of $X - Y$ as $S' - \Sigma r_i$.