Scalable Private Set Union from Symmetric-Key Techniques

Vladimir Kolesnikov[*] Mike Rosulek[†] Ni Trieu[†] Xiao Wang[‡]

September 4, 2019

Abstract

We present a new efficient protocol for computing private set union (PSU). Here two semi-honest parties, each holding a dataset of known size (or of a known upper bound), wish to compute the union of their sets without revealing anything else to either party. Our protocol is in the OT hybrid model. Beyond OT extension, it is fully based on symmetric-key primitives. We motivate the PSU primitive by its direct application to network security and other areas.

At the technical core of our PSU construction is the reverse private membership test (RPMT) protocol. In RPMT, the sender with input x^* interacts with a receiver holding a set X. As a result, the receiver learns (only) the bit indicating whether $x^* \in X$, while the sender learns nothing about the set X. (Previous similar protocols provide output to the opposite party, hence the term "reverse" private membership.) We believe our RPMT abstraction and constructions may be a building block in other applications as well.

We demonstrate the practicality of our proposed protocol with an implementation. For input sets of size 2^{20} and using a single thread, our protocol requires 238 seconds to securely compute the set union, regardless of the bit length of the items. Our protocol is amenable to parallelization. Increasing the number of threads from 1 to 32, our protocol requires only 13.1 seconds, a factor of 18.25× improvement.

To the best of our knowledge, ours is the first protocol that reports on large-size experiments, makes code available, and avoids extensive use of computationally expensive public-key operations. (No PSU code is publicly available for prior work, and the only prior symmetric-key-based work reports on small experiments and focuses on the simpler 3-party, 1-corruption setting.) Our work improves reported PSU state of the art by factor up to $7,600\times$ for large instances.

1 Introduction

Private set union (PSU) is a special case of secure two-party computation. PSU allows two parties holding sets X and Y respectively, to compute the union $X \cup Y$, without revealing anything else, namely what are the items in the intersection of X and Y.

[*] Georgia Institute of Technology, kolesnikov@gatech.edu

[†] Oregon State University, {rosulekm,trieun}@eecs.oregonstate.edu

[‡] MIT and Boston University, wangxiao@cs.northwestern.edu

1.1 Motivation

PSU (like the well-researched private set intersection, PSI) has numerous applications in practice, and tailored efficient solutions are highly desirable. Consider the following use cases. (We note that these use cases cover a wide range of PSU settings, such as multi-party or shared-output PSU. Our work does not address all of the settings, of course, but provides a building block and a baseline for the entire research direction.)

Cyber risk assessment and management via joint IP blacklists and joint vulnerability data. As noted in [LV04, HLS$^+$16], organizations aim to optimize their security updates to minimize vulnerabilities in their infrastructure. Crucial role in the above is played by *joint* lists of blacklisted IP addresses, characteristic network traces and other associated data, as well as *joint* lists of data points reported by vulnerability scanners. At the same time, organizations are understandably reluctant to reveal details pertaining to their current or past attacks or sensitive network data. As convincingly argued in [HLS$^+$16], the use of MPC in computing set unions of the above data sets will mitigate the organizations' concerns. [HLS$^+$16] implements the computation of such set union and related data aggregation as generic MPC in the VIFF framework. As noted by the authors, the major performance bottleneck in their work is private computation of set union. Our tailored PSU algorithms will be applicable to this computation as the main building block.

More generally, privacy-preserving data aggregation is a well-appreciated goal in the network security and other communities. For example, SEPIA [BSMD10] is a library aimed to optimize generic MPC to securely and in real-time compute event correlation and aggregation of network traffic statistics. Our PSU protocol can potentially be helpful in that setting too.

Other applications and use cases. Imagine two Internet providers considering a merger, and they would like to calculate how efficient the resulting joint network would be without revealing the information of their existing networks [BS05]. Another application of combining set-intersection and set-union is the following scenario discussed in [KS05]. A social services organization wants to determine the list of cancer patients who are on welfare. Some patients may have cancer treatment at multiple hospitals. By using a private set union protocol, the union of each hospital's lists of cancer patients can be computed (while removing duplicate patients without leaking the details of the patients), then a secure set intersection operation between the resulting union and the welfare rolls can be performed.

More generally, PSU is an essential building block for private DB supporting full join. Suppose there are two tables owned by two principals, say DMV (Department of Motor Vehicles) and SSA (Social Security Administration). With a PSU-based implementation, a query such as

$$\begin{array}{l}\texttt{SELECT } ssn, dob\\ \texttt{FROM } dmv\_db \texttt{ FULL JOIN } ssa\_db\\ \texttt{ON } dmv\_db.ssn = ssa\_db.ssn \texttt{ WHERE } dob \geq Jan\ 1, 1980\end{array}$$

will allow the players to learn the two columns of the union, but not learn whether the other player has the matching record.

Malicious model is of course the ultimate goal in this line of research. At the same time, we believe semi-honest guarantee is sufficient in many scenarios. Further, our work may serve as a stepping stone to the malicious-secure solution where it is required. We believe that our performance improvement of *four* orders of magnitude is surprising for a reasonably researched problem, and sets the baseline for the PSU performance.

1.2 Contribution

Over the last decade, there has been a significant amount of work on private set intersection [DKT10, DT10, ADT11, HEK12, DCW13, PSZ14, PSZ18, KKRT16, CDJ16, KMP$^+$17, RR17a, CLR17, HV17, RA18, CCS18, FNO18, CO18, DRRT18, PSTY, PRTY19, IKN$^+$19]. However, there has been little work on PSU, with current PSU state-of-the-art not scalable for big data. Despite similarities between the two functionalities, many effective PSI techniques do not directly apply to PSU. We give a brief discussion about the unsuitability for PSU of several popular PSI techniques in Section 5.4 as well as throughout the paper.

We design a truly scalable PSU protocol, building on newly developed building blocks. In detail, our contribution can be summarized as follows:

1. We identify that existing fast private membership tests, used in leading PSI protocols are not immediately applicable for computing PSU (cf. Section 2.1), and a richer PMT of [CO18] carries 125× performance penalty (cf. Section 1.3). We propose a new building block *reverse private membership test* (PMT) in Section 4. We present an efficient instantiation of this building block, which serves as the basis of our symmetric-key based PSU protocol.

2. We apply the bucketing technique to further reduce the computation and communication overhead. We identify and overcome several new challenges unique to bucketing in the context of PSU (but not PSI). Details can be found in Section 5.

3. Integrating the above two components, we build a truly scalable system for PSU computation that is *three orders of magnitude* faster than the current reported performance for large two-party PSU instances. Specifically, we are $\approx$ 7,600× faster than [DC17], which is the current best reported numbers for larger sets of 1 million elements. [BA12] consider an easier setting with three parties and one corruption. Although our protocol works in a stronger model than [BA12], we are still 30× faster in terms of running time on sets of 2^{12} elements and have $100 - 125\times$ smaller communication (cf. Table 3). Our protocol evaluates PSU of two million-element datasets in about a minute on WAN and 13 seconds on a LAN.

4. Our implementation is released on Github: `https://github.com/osu-crypto/PSU`. To our knowledge, this is the first publicly available PSU implementation.

1.3 Related Work

We start by reviewing previous PSU protocols, with particular emphasis on the semi-honest model.

Kissner and Song [KS05]. To our knowledge, the first PSU protocol was proposed by Kissner and Song [KS05]. The PSU of [KS05] is based on polynomial representations and additively homomorphic encryption (AHE). The core idea of their protocol is that if the sets X (respectively, Y) is represented as a polynomial f (respectively, g) whose roots are the set's elements, then the polynomial representation of the union $X \cup Y$ is $f \times g$. An important property is that an item x is in the set X if and only if $f(x) = 0$. Consequently, for each item e that appears in either set X or Y, it holds that $(f \times g)(e) = f(e) \times g(e) = 0$. The players compute the polynomial $f \times g$ under AHE, and figure out the set of elements based on a procedure called "element reduction", which can reduce the degree of the roots.

Frikken [Fri07]. Relying on the polynomial representation, Frikken [Fri07] proposed a faster PSU protocol with linear communication complexity in the size of the dataset. At the high level,

Protocol	Comm. (bits)	Comp. [#Ops symm/pub-key]
[KS05]	$O(\kappa^3 n^2)$	$O(n^2)$ pub-key
[Fri07]	$O(\kappa n)$	$O(n \log\log(n))$ pub-key
[BA12]	$O(\kappa \ell n \log(n))$	$O(n\ell \log(n))$ symm
[DC17]	$O(\kappa \lambda n)$	$O(\lambda n)$ pub-key
Ours	$O(\kappa n \log(n))$	$O(n \log(n))$ symm

Table 1: Asymptotic communication (bits) and computation costs of two-party PSU protocols in the semi-honest setting. Pub-key: public-key operations; symm: symmetric cryptographic operations. n is the size of the parties' input sets. ℓ is the bit-length item. λ is statistical security parameters. In [BA12] and our protocol, $\kappa = 128$ is computational security parameter, while $\kappa = 2048$ is the public key length in other protocols. We ignore the pub-key cost of κ base OTs.

it proceeds as follows. Suppose that $E(f)$ is an encrypted polynomial representation for the set X, a tuple of the form $(xE(f(x)), E(f(x)))$ achieves the specific property that this tuple will be $(0; 0)$ if $x \in X$. In other words, $x \notin X$ can be recovered from the decrypted tuple values. Therefore, instead of computing the encrypted $f \times g$ in [KS05], Bob just computes the above tuples after receiving the encrypted polynomial representation $E(f)$ from Alice, and sends them back to Alice in random order. Alice now decrypts the tuples and learns the value that is not in the intersection. The work of Frikken [Fri07] requires $O(n\kappa)$ communication, where n is the size of the parties' input sets and κ is the length of public-key/ciphertext. Computational cost of generating each tuple is $O(n)$, thus this protocol requires $O(n^2)$ computation. Moreover, their protocol [Fri07] is expensive due to the multi-point evaluation on the encrypted polynomial, which requires the depth of the arithmetic circuit (leveled fully homomorphic encryption) to be logarithmic of the input set size. The authors claimed that the computation of their protocol can be reduced to $O(n \log\log(n))$ by using the bucketing technique with minor modifications to their protocol, but it is not clear how to modify it. Indeed, using bucketing is quite tricky for PSU until our work. Based on the polynomial representation, Hazay and Nissim [HN12] extended the Frikken's protocol in the presence of malicious adversaries.

Blanton and Aguiar [BA12]. In 2012, Blanton and Aguiar [BA12] proposed a faster PSU protocol based on oblivious sorting and generic MPC protocols. The core idea of their protocol consists of combining the input sets into a new set, then sorting the resulting set, and comparing adjacent items of the sorted set in order to eliminate duplicates. They focus on constructing the circuit for PSU (and several other set operations) and relegate its evaluation to generic protocols. Their paper provides experimental results on small input set in a three-party and honest majority setting for $32-$bit sized elements. Their largest experiment, $n \leq 2^{11}$, runs in 25 seconds; our $n \leq 2^{12}$ experiment on larger element sizes runs in 1.42 seconds. Importantly, they run the experiments in the three-party setting, where evaluation is much faster as wire secrets can be 1-bit long.

We sketch approximate communication cost of their two-party garbled-circuit-based protocol based on state-of-the-art OT extension and half gates [ZRE15]. Oblivious sorting of 2^{22} elements per player involves sorting a 2^{23} array. Considering 32-bit elements, such 2PC will require approximately $23 \cdot (2^{23}) \cdot (32 + 32) \cdot 256 = 3,161,095,929,856$ bits $= 395$ GB. Here 256 is the half-gates garbled table size and 32 is the element size. Subsequent duplicate elimination will cost approximately the same as oblivious sort, so total communication cost is $\approx$ 790GB. Considering larger element size, say, 128 bits, results in the corresponding 4× cost increase, bringing total to $\approx 3.1TB$. Transferring

$3TB$ over a 400Mbps WAN will take $\frac{3 \cdot 8 \cdot 10^6}{400} = 60000$ seconds = 16.67 hours. For comparison, our protocol for this size runs in 250 seconds, a 240× improvement.

[BA12] should perhaps be seen as an improvement over current public key-based protocols. As discussed above, our tailored solution outperforms [BA12] by a large factor even in the setting that is the most unfavorable for us. Because there is no reported data on the performance of [BA12] on larger set sizes and no existing generic MPC/2PC system supports large circuits generated by [BA12], we use calculated numbers in our comparison to [BA12] in Table 3.

Davidson and Cid [DC17]. Recently, Davidson and Cid [DC17] proposed an efficient protocol based on an encrypted Bloom filter and additively homomorphic encryption (AHE). In the [DC17] protocol, the receiver represents its input set Y using Bloom Filter (BF) with k hash functions, and inverts this filter by flipping the bit value of each entry. It then encrypts the inverted Bloom filter by using an IND-CPA secure AHE scheme, and sends it to the sender. For each item x of its input set X, the sender uses the k hash functions to retrieve k encrypted BF entries corresponding to x. He then uses AHE homomorphism to sum up under encryption the k retrieved ciphertexts. Let c be the obtained (AHE-encrypted) sum. The sender sends (AHE-encrypted) pairs $\{cx, c\}$ to receiver. Receiver decrypts them and is able to obtain x iff $c \neq 0$. Indeed, if $x \in Y$, all k entries of x are not set in the inverted BF, resulting in $c = 0$. Therefore, the receiver only obtains $X \setminus Y$, from which it computes $X \cup Y$. [DC17] requires $O(\kappa\lambda n)$ communication and $O(\lambda n)$ modular exponentiations, where λ is the statistical security parameters, and κ is the length of public-key/ciphertext, which is in the range 1024-2048 due to their use of public-key primitives. In concrete terms, encrypted BF for the set size $n = 2^{20}$ requires 8.05 GB and 16.1 GB when using a $\kappa = 1024$ bit and $\kappa = 2048$ bit key length, respectively.

Other related work. We note that recent work [CO18] proposed private membership test with shared output, which can be used to instantiate our reverse private membership test. Our RPMT is much faster. For specific parameters used in our work (bucket size 61, bit length 128), [CO18] requires 80KB communication per test while our RPMT construction only needs 0.64KB, a 125× improvement in terms of communication. In addition, our construction requires 140× fewer symmetric-key operations than [CO18]. Because we work with small bucket sizes, our polynomial-based RPMT is fast computationally as well.

Outsourcing PSU was considered in the work of Canetti et al. [CPPT14]. In this problem, users outsource their encrypted data and computation to an untrusted cloud server, while keeping their data private. The main purpose is to minimize the computational overhead of the users by utilizing the powerful resources of the cloud server.

Table 1 provides a brief comparison to the prior highest-performing PSU protocols in the semi-honest setting. We emphasize that public-key operations are the workhorse of all prior work, while we do only $\kappa = 128$ such operations to initiate OT extension. This is the main reason for 7,600× performance improvement over prior work we observe. We report in detail the performance results and comparisons in Section 6.

2 Overview of Our Results & Techniques

We start with a special case. Suppose that the sender has only one item y in its set Y and the receiver holding the set X will receive the resulting union $\{y\} \cup X$. The protocol must satisfy the following:

Parameters: Set sizes m and n; two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$

Functionality:

- Wait for an input $X = \{x_1, x_2, \ldots, x_n\} \subseteq \{0,1\}^*$ from sender $\mathcal{S}$, and an input $Y = \{y_1, y_2, \ldots, y_m\} \subseteq \{0,1\}^*$ from receiver $\mathcal{R}$
- Give output $X \cup Y$ to the receiver $\mathcal{R}$.

Figure 1: Private Set Union Functionality $\mathcal{F}_{\mathsf{psu}}^{m,n}$.

Parameters: A set size n; two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$

Functionality:

- Wait for an input $x^* \in \{0,1\}^*$ from sender $\mathcal{S}$, and an input $X = \{x_1, x_2, \ldots, x_n\} \subseteq \{0,1\}^*$ from receiver $\mathcal{R}$
- Give the receiver $\mathcal{R}$ output 1 if $x^* \in X$ and 0 otherwise.

Figure 2: Reverse Private Membership Test Functionality $\mathcal{F}_{\mathsf{rpmt}}^{n}$.

(1) if $y \notin X$, the receiver is allowed to learn y as it is implied by the output. The sender learns nothing.

(2) if $y \in X$, the receiver knows that $y \in X$ (implied by the output), but not allowed to learn which is the sender's item y. Sender learns nothing.

Receiver learns which of the cases (1) or (2) occurs. Based on the case, the sender's item y can be conditionally sent to the receiver using a "one-sided" OT, a version of OT that requires transfer of a single encrypted secret, rather than the usual transfer of two encrypted secrets, exactly one of which the receiver can decrypt.

2.1 Reverse Private Membership Test (RPMT)

We formalize the above basic functionality as the RPMT functionality (cf. Figure 2) and design a corresponding tailored efficient protocol, which we believe to be of independent interest. RPMT is related to the traditional Private Membership Test (PMT) [PSZ14], which is a two-party protocol in which the party with input y learns whether or not its item is in the input set X of other party (who learns nothing). In a RPMT, the output is given to the opposite party, i.e. the party holding the set X will learn whether $y \in X$ (and nothing else). We formally describe the ideal RPMT functionality in Figure 2.

We emphasize that, unlike PSI, use of PMT is not very natural for PSU. This is because the PMT output receiver holds an element, and gets the answer in plaintext whether the element belongs to a set held by the sender. This is implied by the PSI output, and hence can be used there. However, this is extra information in the PSU functionality. We don't know of a natural way to efficiently use PMT with PSU.

This seemingly simple functionality adjustment (PMT→RPMT) doesn't seem to be fixable by a small tweak of PMT. This is because the underlying primitive used to implement fast

PMT [KKRT16] is a variant of OT extension, and the role of OT receiver naturally belongs to the player with a single-element input y; it is not clear how to amend the protocol to allow (only) the other player to receive the output.

The basic idea for our RPMT is to have the receiver represent a dataset X as a polynomial $\widetilde{P}(x)$ whose roots are its elements, and send the (plaintext) coefficients of the polynomial $P(x) = \widetilde{P}(x) + s$ to the other party, where s is a secret value chosen at random by the receiver. The sender evaluates the received polynomial on y and obtains $P(y) = s'$. It is easy to see that $s' = s$ if $y \in X$, i.e. y is a root of $\widetilde{P}(x)$. At this point, the receiver could compare s' and s in the clear and learn the output of RPMT. However, if $y \notin X$, the value $P(y)$ may leak partial information about y. To prevent this, instead of the receiver sending s to the sender, the parties perform a *private equality test* (PEQT) to determine whether two strings s and s' are equal. The PEQT guarantees that the sender learns nothing about whether $y \in X$ while the polynomial presentation allows receiver to determine whether $y \in X$ but not the value of y (beyond what is implied by $y \in^? X$).

We note that full PEQT is actually not required, and a weaker and slightly efficient subprotocol is sufficient. For uninterrupted flow, we return to this observation in Sect. 2.2.

This brief overview of RPMT ignores an important security issue. In particular, suppose $y \in X$, so the sender can evaluate $P(y) = s$. Then he/she can compute $P(\cdot) - s$: a polynomial whose roots are all of the elements of X! To address this issue, the parties invoke oblivious PRF (OPRF) on their inputs, and use the OPRF's outputs for the polynomial interpolation/evaluation. Recall that OPRF is a 2-party protocol in which the OPRF sender learns a PRF key k and the OPRF receiver learns $F_k(z)$, where F is a pseudorandom function (PRF) and z is the receiver's input. In RPMT, the RPMT sender acts as the OPRF receiver to receive $F_k(y)$ and the RPMT receiver acts as the OPRF sender to obtain the PRF key k. Now, the receiver interpolates a polynomial P over points[1] $\{(x, s \oplus F_k(x))\}$ $\forall x \in X$, and sends the coefficients of this polynomial to the other party, who evaluates it on y, and outputs $P(y) \oplus F_k(y)$. Thanks to OPRF, the important properties needed for RPMT still hold: (i) $F_k(y) = F_k(x)$ if $x = y$. Therefore, the sender obtains the secret value s chosen by the receiver; (ii) even if $y \in X$, other elements of X can no longer be inferred from $P(\cdot)$ and $P(y)$. This is intended to make finding roots of $P(\cdot) - P(y)$ useless to the sender. Moreover, to learn X, the sender has to know its OPRF value $F_k(x)$, which is not possible because of the OPRF guarantees. A detailed overview of the RPMT protocol is presented in Section 4.

We note that RPMT and OPRF are fast cryptographic tools. Recently, Kolesnikov et al. [KKRT16] proposed an efficient protocol which performs many OPRF or PEQT with amortized cost of 5 μs. Therefore, the main computation cost of our RPMT is the multiplication/evaluation of the polynomial, which requires time $O(n \log^2(n))$ using FFT or $O(n^2)$ using a more straight-forward algorithm. This is expensive for large set size $n = |X|$. We avoid the need to work with high-degree polynomials by hashing/bucketing (see below). The communication overhead is small and is equal to $O(n)$.

We can summarize the above gadget for the simple case of PSU (union of a set X and a single element y) as follows: using RPMT on X and y, the receiver learns a bit $b \in \{0, 1\}$ indicating whether $y \in X$. Next, the parties perform one-sided OT to allow receiver obliviously obtain y if $b = 0$ (i.e. $y \notin X$), nothing otherwise.

[1] Of course, $x \in \{0, 1\}^*$ needs to be "hashed down" to an element of the field we are working with. This can be done,e.g., by applying a collision resistant hash function. For simplicity, here we mention, but don't formalize this step.

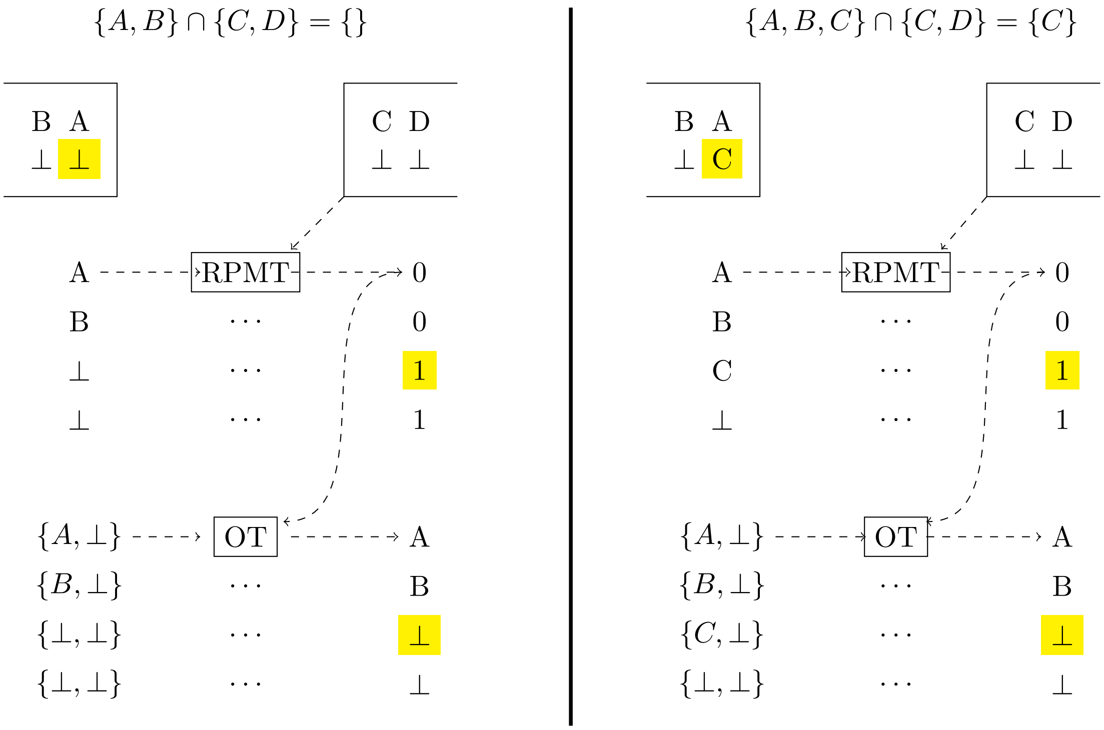

Figure 3: Illustration of the main idea behind our protocol: using RPMT and oblivious transfer to perform PSU on a sample bin. The left-hand side illustrates that the sender's bin contains 2 real items $\{A, B\}$ and the receiver's bin contains 2 real items $\{C, D\}$, these sets are disjointed. The right-hand side shows that the sender's bin contains 3 real items $\{A, B, C\}$ and the receiver's bin contains 2 real items $\{C, D\}$, these sets have a common item C. An item $\perp$ denotes the global item known by both parties.

2.2 An Efficiency Optimization

Going back to the discussion of our RPMT protocol in the previous section, while it uses a PEQT protocol to compare the output of the polynomial, this is in fact overkill for our application to PSU.

Indeed, suppose the sender instead just sends the output of the polynomial s' in the clear to the receiver. Consider the two cases. First, if $y \in X$, we have $s' = s$, so no information about y would be leaked, as desired. In the other case that $y \notin X$, we want (in the overall PSU protocol) the receiver to learn y anyway! So even if s' leaks information about y, this is fine. Hence, for the purpose of PSU, our protocol can conclude by a plaintext comparison, where the sender sends s' to the receiver.

As it turns out, this optimization, while elegant, is not substantial in terms of overall performance, providing $3-5\%$ improvement in running time and $\sim$10% improvement in communication. This can be seen by sketching relative costs of our subprotocols, and is also supported by our experiments (See Table 5 in the Appendix for more details). Because of this, we chose to present the paper in terms of the more general and conceptually simpler RPMT primitive.

However, we did formalize and prove secure the improved protocol. It is presented, together with a proof of security and experimental results in Appendix A. We feel that this presentation structure allows to focus our main presentation on the simpler primitives, while at the same time devote sufficient attention to an interesting optimization.

2.3 General Case from RPMT

We now discuss how to extend the above approach to the general case of PSU with $|Y| > 1$. The idea is natural: for each item $y \in Y$ simply execute the above gadget on y and X. As a result, the receiver obliviously obtains all items in $Z \leftarrow Y \setminus X$ which directly allows him to learn the union $X \cup Y = X \cup Z$. However, this approach requires n instances of RPMT and n instances of OT (here, we assume that $|X| = |Y| = n$). This results in communication and computation complexity of $O(n^2)$ and $O(n^2 \log(n))$, respectively. Therefore, this PSU construction is only efficient when n is small. Our next trick is to use a hashing technique to overcome this limitation.

At the high level, the idea is that the parties use a hashing scheme to assign their items into bins, and then perform the quadratic-cost PSU on each bin efficiently. By applying a balls-into-bins analysis and minimizing the overall cost, our hashing scheme has $O(n/\log n)$ bins, where each bin contains $O(\log n)$ items. We review the hashing scheme in detail in Section 5.2. This optimization reduces costs to $O(n \log n)$ in communication and $O(n \log n \log\log n)$ in computation. However, bucketing introduces a challenge specific to the PSU – the receiver learns additional information on the intersection items, namely, the bucket where the match occurred/did not occur. Consider an example where the receiver's first bin X_1 contains three items and the sender's first bin contains y_1. In our protocol, parties perform RPMT on X_1 and y_1. Suppose $y_1 \in X$, which means, because of bucketing, that $y_1 \in X_1$. From RPMT output, the receiver learns that $y_1 \in X_1$, which cannot be inferred from just the PSU output.

To address this issue, both parties add dummy items $\perp$ into each of their bins to fill them to their maximal size prior to executing RPMT on the bins. Then even if the output of RPMT on $(X_1 \cup \perp)$ and y_1 gives the receiver a bit $b = 1$ (i.e. indicating that $y_1 \in X_1$), the receiver will not learn any information on y_1 since y_1 may be the dummy item $\perp$. We note that this high-level description of the use of dummy items hides some technical nuance, which is explained in detail in Section 5.

Figure 3 illustrates the main idea behind our protocol. It is easy to see from the Figure 3 that the receiver's view in both important cases (two bins are disjoint or two bins have a common item) are exactly same. As noted above, each bin must be padded with $\perp$ to the maximum number of items expected in a bin. In Figure 3, the maximum bin size is 4. Section 5 formally describes the full construction of our PSU.

2.4 Efficiency

Our PSU protocol requires only $O(\kappa)$ public-key operations to perform base OT (which can run in the offline phase). In the online phase, our protocol consists of $O(n)$ OPRF instances, $O(n)$ PEQT instances, and $O(n)$ OT instances. These building blocks are based on symmetric-key operations, and can use the same base OTs. In terms of communication, our protocol requires $O(\kappa n \log(n))$, where κ is the computational security parameter.

Our protocol is 3-4 orders of magnitude faster than previous state-of-the art. We present detailed performance analysis and comparisons in Section 6.

PARAMETERS: A PRF F, and two parties: sender and receiver

FUNCTIONALITY:

- Wait for input x from the receiver.
- Sample a random PRF seed k and give it to the sender. Give $F_k(x)$ to the receiver.

Figure 4: OPRF Ideal Functionality.

PARAMETERS: Two parties: sender and receiver

FUNCTIONALITY:

- Wait for input $\boldsymbol{x}_0 \in \{0,1\}^*$ from the sender, and input $\boldsymbol{x}_1 \in \{0,1\}^*$ from the receiver.
- Give the receiver $\mathcal{R}$ output 1 if $\boldsymbol{x}_0 = \boldsymbol{x}_1$ and 0 otherwise.

Figure 5: The Private Equality Test Ideal Functionality $\mathcal{F}_{\mathsf{peqt}}$

2.5 Using Padding to Hide Input Set Sizes

If desired, it is easy to add padding to our protocol so as to hide the actual sizes of players input sets. This is done simply by setting the protocol parameters (number of bins, maximal bin size) based on the known upper bound of set size. It is easy to verify that this (higher parameter values) do not cause correctness or security violations. Intuitively, players will process more bins with higher maximal bin sizes, but fewer actual items. However, the number of actual items per bin is hidden by our protocol.

3 Preliminaries

3.1 Oblivious Transfer

Oblivious Transfer (OT) is a ubiquitous cryptographic primitive and is a foundation for almost all efficient secure computation protocols. In OT, a sender with two input strings (x_0, x_1) interacts with a receiver who has an input choice bit b. The result is that the receiver learns x_b without learning anything about x_{1-b}, while the sender learns nothing about b.

The very first OT protocol was proposed by Rabin [Rab81]. While several OT protocols were proposed, they all essentially relied on public key operations (necessarily so, due to lower bound [IR89]). The OT extension protocols [Bea96, IKNP03] went around the lower bound by considering a *batched* OT evaluation. OT extension protocol works by evaluating a small number (namely, computational security parameter κ) of expensive OTs that are used as a base for performing many OTs using only cheap symmetric-key operations. In 2013, Kolesnikov and Kumaresan [KK13] proposed an optimization and generalization of the IKNP OT extension, which achieved $O(\log \kappa)$ factor performance improvement in communication and computation. In the same year, Asharov et al. [ALSZ13] proposed several IKNP optimizations (some overlapping with [KK13]) and provided optimized implementation of OT extension. In this work, we are interested in an specific variant of OT (one-sided OT), in which the sender has only one message to send, and which is received by the receiver based on its choice bit. The sender remains oblivious as to whether or not the receiver received the message. With this OT variant, one can reduce the bandwidth requirement by sending only one secret instead of two. As a result, we can perform many OT instances with amortized cost of 50 nanoseconds and 129 bits transmitted.

3.2 Oblivious PRF & Private Equality

Oblivious PRF: An oblivious PRF (OPRF) [FIPR05] is a 2-party protocol in which the sender learns (or chooses) a random PRF key k and the receiver learns $F_k(x_1), \ldots, F_k(x_t)$, where F is a PRF and $(x_1, \ldots, x_t) \subseteq \{0,1\}^*$ are inputs chosen by the receiver. Here, we consider a slightly weaker variant of OPRF due to [KKRT16] where the PRF keys are related. We describe the ideal functionality for an OPRF in Figure 4.

OPRF and BaRK-OPRF instantiation. While many OPRF protocols exist, we focus on the protocol (BaRK-OPRF) of Kolesnikov et al. [KKRT16]. This protocol has the advantage of being based on oblivious-transfer (OT) extension. As a result, it uses only inexpensive symmetric-key cryptographic operations (apart from a constant number of initial public-key operations for base OTs). The protocol efficiently generates a large number of OPRF instances, which makes it a particularly good fit for our eventual PSI application that uses many OPRF instances. Concretely, the amortized cost of each OPRF instance costs roughly 500 bits in communication and a few symmetric-key operations.

Technically speaking, the protocol of [KKRT16] achieves a slightly weaker variant of OPRF than what we have defined in Figure 4. In particular, (1) PRF instances are generated with *related keys*, and (2) the protocol reveals slightly more than just the PRF output $F_k(q)$. We stress that the resulting PRF of [KKRT16] remains a secure PRF even under these restrictions. More formally, let $leak(k, q)$ denote the extra information that the protocol leaks to the receiver. [KKRT16] gives a security definition for PRF that captures the fact that outputs of F, under related keys $k_1, \ldots, k_n$, are pseudorandom even given $leak(k_i, q_i)$. This guarantee is sufficient for our purpose.

For the ease of presentation and reasoning, we work with the cleaner security definitions that capture the main spirit of BaRK-OPRF. We emphasize that, although cumbersome, it is possible to incorporate all of the [KKRT16] relaxations into the definitions. We stress that our eventual application of PSU is secure *in the standard sense* when built from BaRK-OPRF, and we make corresponding remarks in the proof of security outlining how security holds for BaRK-OPRF.

Private Equality Test (PEQT). Fagin, Naor, and Winkler [FNW] introduced one of the first PEQT protocols. PEQT is a 2-party protocol in which a receiver who has an input string x_0 interacts with a sender holding an input string x_1. The result is that the receiver learns a bit indicating whether $x_0 = x_1$ and nothing else, whereas the sender learns nothing. We formally define the PEQT functionality in Figure 5. [FNW] protocol was based on public-key cryptography. A long list of follow-up works [NP, BST01, Lip03, PSZ14, PSSZ15, KKRT16, PSZ18] improved the efficiency of PEQT. Some of them were introduced in the context of PSI. PEQT can be immediately obtained from BaRK-OPRF by computing and comparing BaRK-OPRF output in the clear, cf. [KKRT16] (i.e., one party learns $F_k(x_1)$; the other party learns k and sends $F_k(x_0)$). We will use the latter most efficient instantiation.

4 Reverse Private Membership Test (RPMT)

PARAMETERS:

- Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$
- Set X is of size n of elements.
- The bit-length of field elements $\sigma = \lambda + \log(n)$.
- Ideal OPRF, $\mathcal{F}_{\mathsf{peqt}}$ primitives specified in Figure 4, and Figure 5, respectively. Let $F_k(x) : \{0,1\}^* \mapsto \{0,1\}^\sigma$ be the underlying OPRF function.
- A collision-resistant hash function $h(x) : \{0,1\}^* \mapsto \{0,1\}^\sigma$.

INPUT OF $\mathcal{S}$: $x^* \in \{0,1\}^*$

INPUT OF $\mathcal{R}$: $X = \{x_1, x_2, \dots, x_n\} \subseteq \{0,1\}^*$

PROTOCOL:

1. $\mathcal{S}$ acts as OPRF *receiver*, sends x^* to OPRF. $\mathcal{S}$ receives $q^* = F_k(x^*)$ and receiver $\mathcal{R}$ receives k.
2. $\mathcal{R}$ chooses $s \leftarrow \mathbb{F}(2^\sigma)$ at random. $\mathcal{R}$ interpolates a $\mathbb{F}(2^\sigma)$-polynomial $P(x)$ over points $\{(h(x_i), s \oplus q_i)\}$, where $q_i = F_k(x_i), \forall i \in [n]$. Here $s \oplus q_i$ is computed as operation on σ-bit strings.
3. $\mathcal{R}$ sends the coefficients of P to $\mathcal{S}$.
4. $\mathcal{S}$ computes $s^* = P(h(x^*)) \oplus q^*$.
5. $\mathcal{S}$ and $\mathcal{R}$ invoke the $\mathcal{F}_{\mathsf{peqt}}$-functionality:
   - $\mathcal{R}$ acts as *receiver* with input s.
   - $\mathcal{S}$ acts as *sender* with input s^*.
   - $\mathcal{R}$ receives output from $\mathcal{F}_{\mathsf{peqt}}$.

Figure 6: Reverse Private Membership Test Protocol $\mathcal{F}^n_{\mathsf{rpmt}}$.

We describe our efficient construction of Reverse Private Membership Test (RPMT), which is a semi-honest secure protocol for the functionality specified in Figure 2. Throughout the paper we use the notations κ, λ for the computational and statistical security parameters, respectively. Our RPMT protocol is described in Figure 6. The formal protocol follows the intuition presented in the first part of Section 2. Polynomial arithmetic is done in field $\mathbb{F}(2^\sigma)$ for some appropriate σ. We discuss using smaller field size in Section 5.3.

RPMT protocol is presented in Figure 2. We next argue it computes $\mathcal{F}^n_{\mathsf{rpmt}}$ correctly. Afterwards, we state and prove the security properties of the protocol.

Correctness. The main observation of OPRF is that the RPMT sender (acting as OPRF's receiver) obtains the output q^* which is equal to q_i, if $x^* = x_i$. In this case, it is not hard to see that $s^* = P(h(x^*)) \oplus q^* = P(h(x_i)) \oplus q^* = s$. From the $\mathcal{F}_{\mathsf{peqt}}$-functionality, the receiver outputs 1. In case $x^* \notin X$, the OPRF functionality gives the sender q^* which is not in $\{q_i \mid i \in [n]\}$, thus $s^* \neq s$ and the receiver gets 0 from the $\mathcal{F}_{\mathsf{peqt}}$-functionality.

We remark that our RPMT protocol is correct except in case of a collision $P(h(x^*)) = P(h(x_i))$ for $x^* \neq x_i$, which occurs with probability is $2^{-\sigma}$. By setting $\sigma = \lambda + \log(n)$, a union bound shows probability of collision is negligible $2^{-\lambda}$.

Security. We now state and prove security properties of RPMT.

Theorem 1. *The construction of Figure 6 securely implements functionality $\mathcal{F}^n_{\mathsf{rpmt}}$ in the semi-honest model, given the OPRF and Private Equality Test primitives defined in Figure 4, and Figure 5, respectively.*

Proof. We exhibit simulators $\mathsf{Sim}_{\mathcal{R}}$ and $\mathsf{Sim}_{\mathcal{S}}$ for simulating corrupt $\mathcal{R}$ and $\mathcal{S}$ respectively, and argue the indistinguishability of the produced transcript from the real execution.

Corrupt Sender. $\mathsf{Sim}_{\mathcal{S}}(x^*)$ simulates the view of corrupt $\mathcal{S}$, which consists of $\mathcal{S}$'s randomness, input, output and received messages. $\mathsf{Sim}_{\mathcal{S}}$ proceeds as follows. It first chooses $q' \in_R \{0,1\}^\sigma$, calls OPRF simulator $\mathsf{Sim}_{\mathcal{S}_{\mathsf{OPRF}}}(x^*, q')$, and appends its output to the view. We note that BaRK-OPRF is behaving the same as OPRF with respect to the security guarantee needed for simulating this step, namely that q^* obtained in Step 1 is pseudorandom. This is the only direct use of BaRK-OPRF in this protocol, and hence the rest of the argument made w.r.t. OPRF applies to our instantiation as well.

$\mathsf{Sim}_{\mathcal{S}}$ simulates Step 3 as follows. It generates random $s' \in \{0,1\}^\sigma$, and n random points $(x'_i, q'_i) \in_R (\{0,1\}^\star, \{0,1\}^\sigma)$. $\mathsf{Sim}_{\mathcal{S}}$ then interpolates polynomial P over points $\{h(x'_i), s' \oplus q'_i\}$ and appends its coefficients to the generated view.

Finally, to simulate Step 5, $\mathsf{Sim}_{\mathcal{S}}$ runs simulator $\mathsf{Sim}_{\mathsf{PEQT}}$ on input $(s' = P(h(x^*)) \oplus q')$ and appends the output of $\mathsf{Sim}_{\mathsf{PEQT}}$ to its output of the view.

We now argue that the output of $\mathsf{Sim}_{\mathcal{S}}$ is indistinguishable from the real execution. For this, we formally show the simulation by proceeding the sequence of hybrid transcripts T_0, T_1, T_2, T_3, where T_0 is real view of $\mathcal{S}$, and T_3 is the output of $\mathsf{Sim}_{\mathcal{S}}$.

Hybrid 1. Let T_1 be the same as T_0, except that the OPRF execution is replaced as follows. By the OPRF/BaRK-OPRF pseudorandomness guarantee and the indistinguishability of the output of $\mathsf{Sim}_{\mathcal{S}_{\mathsf{OPRF}}}$, we replace $F(k, x^*)$ and $F(k, x_i), \forall i \in [n]$, with q' and $q'_i, \forall i \in [n]$, respectively. We note that if $x^* = x_i$, then $q' = q'_i$. It is easy to see that T_0 and T_1 are indistinguishable.

Hybrid 2. Let T_2 be the same as T_1, except that the polynomial is a uniform polynomial of degree $n-1$ (sampled by interpolating over random points). Consider two following cases:

- $x^* \notin X$: Since all values q'_i are uniformly random from the $\mathcal{S}$'s point of view, so are the $s \oplus q'_i$.
- $x^* = x_i$ (consequently, $q' = q'_i$): Since other values $q'_{j\in[n]}, \forall j \neq i$, are uniformly random from $\mathcal{S}$'s point of view, we replace these $s \oplus q'_j$ with random. Then s is used only in the expression $s \oplus q'_i$. Since s is uniform, $s \oplus q'_i$ is also uniformly random from the $\mathcal{S}$'s view even though the adversary knows $q' = q'_i$.

In summary, the polynomial from the real execution can be replaced with polynomial P sampled over random points. T_1 and T_2 are indistinguishable.

Hybrid 3. Let T_3 be the same as T_2, except the PEQT execution is replaced with running the simulator $\mathsf{Sim}_{R_{\mathsf{PEQT}}}(s')$. Because $\mathsf{Sim}_{R_{\mathsf{PEQT}}}$ is guaranteed to produce output indistinguishable from real, T_3 and T_2 are indistinguishable.

Corrupt Receiver. $\mathsf{Sim}_{\mathcal{R}}(x_1, ..., x_n, out)$ simulates the view of corrupt $\mathcal{R}$, which consists of $\mathcal{R}$'s randomness, input, output and received messages. $\mathsf{Sim}_{\mathcal{R}}$ proceeds as follows. It chooses a random $k' \in_r \{0,1\}^\kappa$, calls OPRF simulator $\mathsf{Sim}_{S_{\mathsf{OPRF}}}(\perp, k')$, and appends its output to the view. Finally, to simulate Step 5, $\mathsf{Sim}_{\mathcal{S}}$ runs simulator $\mathsf{Sim}_{\mathsf{PEQT}}$ on input (k', out) and appends the output of $\mathsf{Sim}_{\mathsf{PEQT}}$ to its output of the view.

The view generated by $\mathsf{Sim}_{\mathcal{R}}$ in indistinguishable from a real view because of the indistinguishability of the transcripts of the underlying simulators.

□

Communication Cost. Ignoring the fixed cost of base OTs for OT extension, the PMT communication cost (prior to further optimizations discussed in Section 5.3) includes:

- OPRF in Step 1: ρ bits, where ρ is the width of the pseudorandom code defined in Table 2 by referencing parameters from [KKRT16].
- Sending the coefficients of P in Step 3: $(n+1)\sigma$ bits
- $\mathcal{F}_{\mathsf{peqt}}$ in Step 5: $\rho + \lambda$ bits

Therefore, the overall communication cost of our PMT protocol is

$$\Phi(n) = 2\rho + \lambda + (n+1)\sigma \tag{1}$$

5 Private Set Union

We now present our main result, an application of our RPMT to PSU. The construction closely follows the high-level overview presented in the second part of Section 2. Recall, the RPMT functionality allows the receiver to learn one-bit output indicating whether the sender's item is in its (receiver's) set, while keeping this item secret (i.e. the receiver will not know which sender's item is among its set). The performance of our RPMT protocol is linear in the size of the receiver's set, resulting in a quadratic costs for PSU.

Next, in Section 5.1, we show how to use a hashing/bucketing technique to overcome this limitation. At the high level, the idea is that each party maps their items into bins using a public hash function. Each bin contains a small number of items which allows the two parties to evaluate RPMT on the elements of each bin separately.

Let m denote the maximum sender's bin size when mapping n items to β bins with no (expected) overflow. Within each bin, the protocol requires $(m+1)$ invocations of RPMT. Section 5.2 analyses hashing parameters to minimize the overall cost of our PSU.

PARAMETERS:

- Set sizes n_1 and n_2, and two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$
- A bit-length ℓ. Let $n = \max(n_1, n_2)$.
- Number of bins $\beta = \beta(n)$, and max bin size m, suitable for our hashing scheme (Table 2)
- Ideal $\mathcal{F}_{\mathsf{rpmt}}$ primitive defined in Figure 2, and ideal OT primitive.
- A special dummy item $\perp \in \{0,1\}^*$

INPUT OF $\mathcal{S}$: $X = \{x_1, x_2, \ldots, x_{n_1}\} \subseteq \{0,1\}^\ell$

INPUT OF $\mathcal{R}$: $Y = \{y_1, y_2, \ldots, y_{n_2}\} \subseteq \{0,1\}^\ell$

PROTOCOL:

1. Randomly pick a hash function H from all hash functions with domain $\{0,1\}^\ell$ and range $[\beta]$.
2. $\mathcal{S}$ and $\mathcal{R}$ hash elements of their sets X and Y into β bins under hash function H. Let $B_{\mathcal{S}}[i]$ and $B_{\mathcal{R}}[i]$ denote the set of items in the sender's and receiver's i-th bin, respectively.
3. $\mathcal{S}$ pads each bin $B_{\mathcal{S}}[i]$ with (several copies, as needed) the special item $\perp$ up to the maximum bin size $m+1$, and randomly permutes all items in this bin.
4. $\mathcal{R}$ pads each bin $B_{\mathcal{R}}[i]$ with one special item $\perp$ and (several, as needed) different dummy items to the maximum bin size $m+1$.
5. $\mathcal{R}$ initializes set $Z = \{\}$.
6. For each bin $i \in [\beta]$, for each item $x_j \in B_{\mathcal{S}}[i]$:
   (a) $\mathcal{S}$ and $\mathcal{R}$ invoke the $\mathcal{F}_{\mathsf{rpmt}}$-functionality:
      - $\mathcal{S}$ acts as *sender* with input x_j
      - $\mathcal{R}$ acts as *receiver* with input set $B_{\mathcal{R}}[i]$
      - $\mathcal{R}$ obtains bit b_j.

   (b) $\mathcal{S}$ and $\mathcal{R}$ invoke the OT-functionality:
      - $\mathcal{S}$ acts as *sender* with pair-input $\{x_j, \perp\}$
      - $\mathcal{R}$ acts as *receiver* with bit input b_j
      - $\mathcal{R}$ obtains the OT output z_j and sets $Z = Z \cup z_i$.
7. $\mathcal{R}$ outputs $Y \cup Z$.

Figure 7: Private Set Union Protocol $\mathcal{F}_{\mathsf{psu}}^{n_1,n_2}$.

5.1 PSU Construction

As described above, in our PSU protocol we place players' elements into β buckets of maximum size $m+1$ each.

We describe the main construction of PSU in Figure 7. Correctness of our PSU protocol follows from the fact that the RPMT functionality gives the receiver the zero-bit output if its set does not contain the sender's item. In Step 6b, the receiver obliviously receives that item from OT functionality.

We now state and prove security of our PSU construction.

Theorem 2. *The construction of Figure 7 securely implements the Private Set Union functionality $\mathcal{F}_{\mathsf{psu}}^{n_1,n_2}$ of Figure 1 in the semi-honest model, given the OT and Reverse Private Equality Test primitives defined in Figure 2.*

Proof. We exhibit simulators $\mathsf{Sim}_{\mathcal{S}}$ and $\mathsf{Sim}_{\mathcal{R}}$ for simulating corrupt $\mathcal{S}$ and $\mathcal{R}$ respectively, and argue the indistinguishability of the produced transcript from the real execution.

Corrupt Sender. When employing the abstraction of the RPMT and OT functionalities, simulating corrupt $\mathcal{S}$ is elementary. $\mathsf{Sim}_{\mathcal{S}}(X)$ simulates the view of corrupt $\mathcal{S}$, which consists of $\mathcal{S}$'s randomness, input, output and received messages. The simulator simulates an execution of the protocol in which $\mathcal{S}$ receives nothing from the PTM and OT ideal functionality in Step 4. Thus, it is straightforward to see that the simulation is perfect.

Corrupt Receiver. $\mathsf{Sim}_{\mathcal{R}}(Y,Z)$ simulates the view of corrupt $\mathcal{R}$, which consists of $\mathcal{R}$'s randomness, input, output and received messages. We will view $\mathsf{Sim}_{\mathcal{R}}$'s input Z as the set $Z = Y \setminus X$, i.e. the set of elements that X "brings to the union." $\mathsf{Sim}_{\mathcal{R}}$ proceeds as follows.

$\mathsf{Sim}_{\mathcal{R}}$ simulates protocol of Figure 7 bucket-by-bucket. Consider the i-th bucket. Let X_i (respectively Y_i, Z_i) be the set of elements of X (respectively, Y, Z) that are mapped to the i-th bucket. $\mathsf{Sim}_{\mathcal{R}}$ pads Y_i to $m+1$ elements as is done in Step 4. Now, $\mathsf{Sim}_{\mathcal{R}}$ has all the information to simulate Step 6. $\mathsf{Sim}_{\mathcal{R}}$ constructs the sequence simulating when $\mathcal{R}$ discovers new elements in the union. This is an m-element sequence S, where $\mathsf{Sim}_{\mathcal{R}}$ puts $|Z_i|$ elements z_i at randomly chosen slots, and fills the remaining $m - |Z_i|$ elements of the sequence with $\perp$.

$\mathsf{Sim}_{\mathcal{R}}$ then goes through the elements of S. Consider the j-the such element S_j. $\mathsf{Sim}_{\mathcal{R}}$ sets $out_j = 0$ if $S_j = \perp$, and otherwise sets $out_j = 1$. $\mathsf{Sim}_{\mathcal{R}}$ invokes the simulator of $\mathcal{F}_{\mathsf{rpmt}}$ with input (Y_i, out_j), and appends the output of the simulator to its own output. This simulates Step 6a.

$\mathsf{Sim}_{\mathcal{R}}$ proceeds by simulating Step 6b, as follows. $\mathsf{Sim}_{\mathcal{R}}$ invokes the simulator of OT with input (out_j, S_j). This corresponds to $\mathcal{R}$ providing input out_j and receiving output S_j from OT. $\mathsf{Sim}_{\mathcal{R}}$ appends the output of the simulator to its own output.

$\mathsf{Sim}_{\mathcal{R}}$ proceeds simulating each of β bins and terminates. This completes the description of the simulator.

We now argue that the output of $\mathsf{Sim}_{\mathcal{R}}$ is indistinguishable from the real execution. This is easy to see. $\mathsf{Sim}_{\mathcal{R}}$'s reconstruction of how/when the elements of $Z = Y \setminus X$ are discovered by $\mathcal{R}$ is distributed identically to the real execution. The remainder of the simulation refers to simulators of implementations of ideal functionalities.

□

5.2 Hashing Parameters

A natural first attempt is to hash n items into n bins, where each bin will contain $O(1)$ items on average. If we could have $O(1)$ items per bin in PSU, this would result in $O(n)$ total RPMT

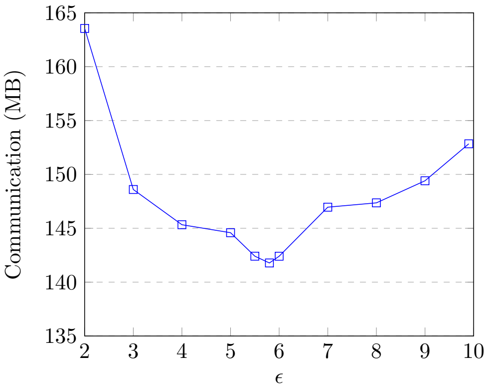

Figure 8: Communication cost (MB) of our PSU protocol for $n = 2^{16}$ given the number of bins $\beta = 10^{-2}\epsilon n$

instances, a low cost. However, we must hide the actual number of items in each bin, and hence all bins must be padded to an upper bound m. Gonnet [Gon81] showed $m = \frac{\ln(n)}{\ln\ln(n)}(1 + o(1))$. The coefficient of little-o is not specified in [Gon81]; Pinkas *et al.* [PSZ18] empirically determined the concrete m given the number of bins β. In our case, n bins is not an optimal strategy. For example, hashing $n = 2^{20}$ elements into n bins, bin size $m = 20$ is required to ensure that overflow occurs with probability $\leq 2^{-40}$. As a result, for $n = 2^{20}$ our PSU protocol performs $21n$ RPMT instances in total, which requires 2^{28} OPRF ciphertexts sent and received. We can do better.

In the following, we analyze the effect of the number of bins β and maximum bin size $m+1$ on the communication overhead of our protocol, and choose the best parameters to minimize our cost. We recall that the overall communication cost of our PSU protocol is equal to $\beta m\Phi(m+1) + \beta m(\kappa+\sigma)$, where $\Phi(m+1)$ is the RPMT communication cost specified in Equation (1). To guarantee that mapping n items to β bins with no overflow, we compute the probability that there exists a bin with more than m items:

$$\Pr(\exists \text{bin with} \geq m \text{ items}) \leq \beta \sum_{m+1}^{n} \binom{n}{i} \big(\frac{1}{\beta}\big)^{i} \big(1 - \frac{1}{\beta}\big)^{n-i} \tag{2}$$

Bounding (2) to be negligible in the statistical security parameter $\lambda = 2^{40}$, we obtain the required bin size m without overflow for a given n and β. To minimize the overall communication cost, we choose $\beta = O(n/\log n)$. According to standard balls-and-bins argument, the maximum bin size is $O(\log(n))$. To determine the coefficients in the big "O", we first fix the number of bins with an initialization value $\beta = \epsilon n = 0.01n$, evaluate Equation (2) to obtain the necessary m, and calculate the required communication cost given β and m. In order to find "sweet spot" for our communication cost, we increase the scale ϵ by 0.001 after each time. We observe that our protocol yields the lowest communication when ϵ is in a range $[0.4, 0.6]$. Figure 8 shows the result for $n = 2^{16}$: we choose $\beta = \epsilon n = 0.058n$ and require $m = 60$ to achieve 2^{-40} hashing failure probability. We also report the set of our hashing parameters in Table 2.

parameters & comm.	set size n 2^8	 2^{10}	 2^{12}	 2^{14}	 2^{16}	 2^{18}	 2^{20}	 2^{22}
ρ	424	432	432	440	440	448	448	448
β/n	0.043	0.055	0.05	0.053	0.058	0.052	0.06	0.051
m	63	58	63	62	60	65	61	68
Comm. cost (MB)	0.39	1.81	7.84	33.43	141.78	602.20	2544.7	10748

Table 2: Hashing parameters for different set sizes n, and our PSU's communication cost (MB). ρ is OT extension matrix width in OPRF ($\approx$ number of bits required per OPRF call) as reported in Table 1 [KKRT16], β is the number of bins, $m+1$ is max bin size PSU with n elements per party. Total PSU communication reported in MB and excludes the fixed cost of base OTs for OT extension.

5.3 Discussion and Optimization

In our RPMT protocol described in Figure 6, the receiver computes a polynomial of degree $(n-1)$ with the field of $\mathbb{F}(2^\sigma)$, where $\sigma = \lambda + \log(n)$. With hash-to-bin technique used in PSU, we are able to reduce the degree from $(n-1)$ to $O(\log(n))$, which avoids an expensive computation at the cost of manipulating polynomials with high degree. However, we increase the field size by $10\% - 12\%$.

Recall that our PSU protocol requires $\beta(m+1)$ RPMT instances in total. For each RPMT protocol, its correctness is violated when a collision event occurs: $P(h(x_i)) = P(h(y_j))$ for $x_i \neq y_j$. To yield collision probability 2^λ over all bins, which is suited for most applications, the size of q_i values is $\sigma = \lambda + \log(\beta(m+1)^2)$. For example, for $n = 2^{20}$, we use the polynomial field size $\mathbb{F}(2^{68})$.

Polynomials with Dummy Points In Step 4, Figure 7, receiver pads each bin with one special item $\perp$ and additional different dummies to the maximum bin size $m+1$. This padding serves the purpose of hiding the number of items that were mapped to a specific bin, which would leak some information about the input set. In RPMT protocol (Step 2, Figure 6), the receiver generates the polynomial over points $\{h(y_i), s \oplus q_i\}$ where some of q_i are the OPRF of the dummy items d_i. Therefore, we simply replace these $q_i = F_k(d_i)$ by random values.

Another optimization, inspired by [KMP+17], is that the receiver computes $P(x)$ by first interpolating the polynomial over the non-dummy items only. That is, receiver interpolates P_0 over $m' \leq (m+1)$ points $\{h(y_i), s \oplus F_k(y_i)\}$, and also computes $P_1(x) = \prod_{i=1}^{m'}(x - h(y_i))$ over m' roots $h(y_i)$, where y_i are real items. Then receiver chooses a random polynomial $P_r(x)$ of degree $(m - m' + 1)$; and computes $P(x) = P_0(x) + P_1(x)P_r(x)$. It is easy to see that $P(h(x_i)) = s \oplus F_k(x_i)$, $\forall x_i \in X$. Using hashing parameters from Table 2, the expected value of m' is only 18 for $n = 2^{18}$, while the worst-case $m = 65$. This optimization reduces the cost of expensive polynomial generation (by approximately 200% in our implementation).

Relaxing RPMT Finally, as discussed in Section 2, the use of full-fledged RPMT for PSU is slightly overkill. It would suffice to use an RPMT protocol which leaked some information about the sender's item (in the case that $x^* \notin X$), since the PSU protocol will release that value anyway. In Appendix A we describe a simple change to the RPMT protocol that remains secure in the context of our PSU protocol. Basically, instead of using PEQT to compare polynomial outputs, the sender just sends it polynomial output in the clear. This is safe in the context of PSU since the

PSU simulator will have access to the sender's RPMT input x^* whenever the polynomial output leaks information about x^*.

5.4 Discussion: Difficulties in Applying Other PSI Techniques

In addition to the optimizations mentioned above, we also explored other commonly used techniques developed in the context of PSI [FNP04, PSSZ15, PSZ18, KKRT16, KMP+17, CLR17, HV17]. Interestingly, we found that many standard techniques for PSI do not directly work for our PSU paradigm, despite the apparent similarity of the two problems. In the following, we will discuss PSU-specific obstacles in applying these techniques. The reader may safely skip this section on the first reading as we discuss here only techniques that we *did not use* in our protocol.

Cuckoo hashing This hashing scheme was introduced by Pagh and Rodler [PR04]. It is the standard hashing scheme in current PSI protocols. At the high level, the receiver uses two (optionally, more) public hash functions h_1, h_2 to store its item in one of the bins $\{h_1(x), h_2(x)\}$. The hashing process uses eviction and the choice of which of the bins is used depends on the entire set. Using the same hash functions and simple hashing, the sender maps its item y into *both* bins $\{h_1(y), h_2(y)\}$ (i.e., item y appears twice in the hash table). Then the parties evaluate PSI bin-by-bin. This is efficient since the receiver has only one item per bin. This hashing scheme avoids a quadratic-cost PSI within a bin.

Unfortunately, this hashing scheme (and the corresponding performance improvement) does not immediately fit in the PSU case. The reason is that the receiver may learn the Cuckoo hash positions of the sender's items, which may reveal information about sender's entire input. Concretely, suppose that in our protocol the sender uses Cuckoo hashing to map its item x into bin $h_1(x)$. If $x \notin Y$, the receiver will learn which bin x is mapped to. As noted above, the bin storing x depends on the whole input set of the sender and this leaks some information about the party input set that cannot be simulated.

Phasing Permutation-based hashing (phasing) was introduced by Arbitman *et al.* [ANS10] to reduce the bit length of the items that are mapped to bins (in our PSU, this would help reduce the polynomial field size). Phasing was used in [PSSZ15, HV17, RR17b, CLR17] to improve PSI performance when input items has short bit length. The idea is to view each item x as two parts: first $\log(\beta)$ bits used to define the bin to which the item is mapped, and the last bits used as a representation to store the item in the bin.

Concretely, the item x can be presented as $x = x_L | x_R$, where x_L has $\log(\beta)$ bit-length. The item x is mapped into bin $x_L \oplus f(x_R)$, where f is a random function that maps arbitrary strings to a range of $[0, \beta]$. That bin will store x_R as a representative of x. Clearly, x_R has $\log(\beta)$ bits shorter than the original item x. This permutation-based hashing technique achieves significant savings, especially when the original item x has small length (e.g. 32 bits or 64 bits). For instance, assume that the item x has 32-bit length, the set size is $n = 2^{20}$. Then bin elements are only 17 bits long, instead of 32 bits. As a result, we might hope to use the polynomial field size of only $\mathbb{F}(2^{17})$ in RPMT, yielding a significant improvement.

Unfortunately, this general phasing technique does not yield any performance benefit in our PSU paradigm. The underlying reason is that the items in each bin are first given as input to an OPRF for that bin, however the state-of-the-art OPRF protocol that we use ([KKRT16]) is

	Protocol	Bit key length	Cryptographic strength	Set size n 2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}
Time	[DC17]	1024	Legacy	11.78	44.73	175.7	702.4	2836.5	11341.2
		2048	112	78.02	312.44	1233.59	4952.94	19881.51	79272.48
	[BA12]	128	128	2.41	11.88	24.88	–	–	–
	Ours	128	128	**0.57**	**0.66**	**0.83**	**1.15**	**2.65**	**10.42**
		Speedup		4×	18×	30×	4306×	7502×	7607×
Comm.	[DC17]	1024	Legacy	2.83	11.32	45.28	181.12	724.49	2897.97
		2048	112	4.06	16.25	65.01	260.04	1040.18	4160.74
	[BA12]	128	128	75.5	369.1	1744.83	8053.06	36507.22	163208.76
	Ours	128	128	**0.45**	**2.05**	**8.48**	**34.98**	**144.65**	**652.09**
		Speedup		9.02×	7.92×	7.66×	7.43×	7.19×	6.38×

Table 3: Comparison of total runtime (in seconds) and communication (in MB) between our protocol, [DC17] and [BA12]. Both parties have n 128-bit elements as input, except [BA12] running time is based on 32-bit elements. [DC17] implementation is in Go, using 8 threads. Our implementation is in C++, 8 threads. [DC17] and us use fast emulated LAN (10Gbps, 0.02ms RTT). Cryptographic strength refers to the computational security of the protocol, according to NIST recommendations. [BA12] runtime is taken from their 3-party experiments, and [BA12] communication is calculated by us for 2PC and 128-bit elements. Best results are marked in bold.

Setting	T	Set size n 2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}
LAN	1	0.66	0.86	1.42	3.54	12.41	61.34	238.88	1039.64
	4	0.59	0.69	0.98	1.46	4.03	17.94	69.07	301.76
	16	0.55	0.66	0.78	0.97	1.82	6.29	21.9	90.99
	32	0.53	0.63	0.69	0.84	1.56	4.1	13.09	54.63
WAN	1	1.38	1.73	2.61	6.96	23.29	102.5	406.15	1679.85
	4	1.33	1.56	1.99	3.29	8.58	31.05	118.79	463.51
	16	1.25	1.39	1.76	2.55	5.61	18.67	70.55	280.15
	32	1.22	1.33	1.57	2.4	5.02	17.08	62.96	250.97
Speedup		1.13-1.24 ×	1.3-1.36×	1.66-2.06×	2.9-4.22×	4.64-7.98×	6-14.9×	6.5-18.2×	6.7-19.1×

Table 4: Scaling of our protocol with set size and number of threads. Total running time is in seconds. n elements per party, 128-bit length element, and threads $T \in \{1, 4, 16, 32\}$ threads. LAN setting with 10Gbps network bandwidth, 0.02ms RTT. WAN setting with 400Mbps network bandwidth, 40ms RTT.

insensitive to the item length. It is only the OPRF output length that determines the field size for polynomial interpolation. Since the OPRF outputs are random, their length must be chosen to avoid collisions with probability $1 - 2^{-\lambda}$.

6 Implementation

Our protocol requires the receiver to generate a polynomial of degree m, and the sender to evaluate it on one point, where $m+1$ is the maximum bin size. Since the degree $m = O(\log(n))$ of the polynomial is relatively small, we use the straightforward Lagrange interpolation and evaluation algorithm which requires $O(m^2)$ field operations. As parties use the bit-string output of the OPRF as input to the polynomial operations, it is natural to interpolate and evaluate the polynomial over $GF(2^\sigma)$. Our polynomial implementation uses the NTL library [Sho] with GMP library and GF2X [GBZT] library installed for speeding up the running time. Inspired by Huang *et al.* [HEKM11], we applied pipelining optimization when the receiver sending all polynomials to the sender. In more detail, we find that by sending polynomial coefficients for 2^8 bins in a batch to the sender, we can minimize the overall wall-clock time of the execution.

As detailed in Section 2, our PSU protocol builds on a specific OPRF variant [KKRT16] and OT extension. We do $\kappa = 128$ Naor-Pinkas OTs [NP01]. We use the source code (OPRF and OT) from [KKRT16, Rin]. Our complete implementation will freely available on GitHub.

We implement our protocol in C++, and run our protocol on a single Intel Xeon with 2.30GHz and 256GB RAM. We emulate the network by using Linux tc command. In the following, we compare our protocol to the state-of-the-art PSU protocol [DC17] which provides empirical experiments for a larger set, and the work of [BA12] which reports experimental numbers for PSU of small sets $n \leq 2^{12}$). Additionally, we demonstrate the scalability and parallelizability of our protocol by evaluating it on sets of up to 2^{22} 128-bit items each.

All comparisons are total running time. We note that our protocols are very amenable to pre-computation (by precomputing and pre-sending OT extension and OPRF matrices).

6.1 Comparison with Prior Work

Since implementation of [DC17] and [BA12] are not publicly available, we use their reported experimental numbers. We perform a comparison on the range of set sizes $n = \{2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}\}$ to match the parameters used in [DC17, Table 3&4] and [BA12, Table 3]. [DC17] ran experiments on Intel Xeon 3.30GHz 256GB RAM and 10Gbps LAN; we use a similar (1.32× slower) machine as reported above and same LAN. [BA12] reports running on 2.4GHz AMD Opteron.

Runtime Comparison In the [DC17] protocol, a Bloom filter (BF) of $44n$ elements is used to yield the false-positive probability 2^{-30}. Each element requires expensive encryption, decryption and further manipulation under an additively-homomorphic encryption (AHE).

We report detailed comparisons in Table 3, and here we highlight some numbers. Our protocol runs in 0.94 seconds for $n = 2^{10}$, while [BA12] requires 11.88 seconds, a factor of 18× improvement; and [DC17] requires 312.44 seconds with 2048-bit key length (which corresponds to the security level considered in our protocol), a factor of 332× improvement. As the set size n increases, [DC17] runs correspondingly slower. When increasing the set size to $n = 2^{18}$, [DC17]'s overall running time is $79,272.48$ seconds while ours is only 10.42 seconds.

This is a 7607× improvement in running time compared to [DC17] (2048-bit key length). A higher improvement factor as we move to higher set size likely indicates that non-protocol-essential system overheads take a higher fraction of resources in smaller set size executions in our protocol. In Section 6.2, we demonstrate the scalability and parallelizability of our protocol.

Bandwidth Comparison The receiver in [DC17] sends a large encrypted BF. For $n = 2^{20}$, BF size is 8.05 GB and 16.1 GB when encrypted with 1024-bit and 2048-bit key, respectively. [BA12] relies on generic 2PC/MPC to run their protocol. We sketch approximate communication cost of their protocol in the two-party setting based on state-of-the-art OT extension and half gates (cf. discussion in Section 1.3). Oblivious sorting of n elements per party involves sorting an array of size $2n$. Considering ℓ-bit elements, this will require approximately $2n \cdot \log(2n) \cdot 2\ell \cdot 256$ bit. Here 256 is the half-gates garbled table size. The communication complexity of the duplicate elimination [BA12] costs approximately the same as oblivious sort. For the bandwidth comparison, we only report the [BA12]'s communication cost of oblivious sorting and duplicate elimination, which is in favor of their protocol.

We compare bandwidth for the set sizes explored in [DC17], and summarize their and our results in Table 3. The communication cost of our protocol is significantly less than that of the prior work. Concretely, for $n = 2^{18}$, our protocol requires 652.09 MB of communication, a 6.38 × improvement. For very small set size $n = 2^8$, our protocol requires only 0.45 MB while [DC17] needs 4.06 MB and [BA12] requires at least 75.5 MB.

Correctness error probability In [DC17] protocol, Bloom filter introduces a false positive error in the output. Recall, the false positive rate (FPR) is the probability that a *single* element is mistakenly marked as being in the set. The [DC17]'s implementation chooses FPR of 2^{-30}. Thus, computing the set union of 2^{-18} items each, the probability that the entire output includes a false positive is 2^{-12}. We use simple hashing with probability of existence of an overflowed bin of 2^{-40}. Thus, in our protocol, the correctness error probability 2^{-40} is per *whole* set, not per single item.

6.2 Scalability and Parallelizability

We demonstrate the scalability and parallelizability of our protocol by evaluating it on set sizes $n = \{2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, 2^{20}, 2^{22}\}$. We run each party in parallel with $T \in \{1, 4, 16, 32\}$ threads. We report the performance of our protocol in Table 4, showing running time in both LAN/WAN settings: a LAN setting with 10 Gbps network bandwidth and 0.02 ms round-trip latency; a WAN setting with 400 Mbps network bandwidth and a simulated 40 ms round-trip latency.

Our protocol indeed scales well. Small-size problems are sub-second; medium-size problems ($n = 2^{14}$) are 3.54 seconds and larger sizes ($n = 2^{20}$) is under 4 minutes, all single-threaded. Increasing the number of threads runs the $n = 2^{20}$ instance in 13.09 seconds, a four orders of magnitude improvement over prior work. Benchmarking our implementation in the WAN setting, our protocol also scales well due to the fact that the communication cost is reasonable (for $n = 2^{18}$, our protocol needs 652.09 MB of communication).

Our protocol is very amenable to parallelization. Specifically, our algorithm can be parallelized at the level of bins. For example, when increasing T from 1 to 32, our protocol shows a factor of 19× improvement as the running time reduces from 1039.64 seconds to 54.63 seconds for an input of $n = 2^{22}$ elements.

Of particular interest is the last row, which presents the ratio between the runtime of the single thread and 32 threads. Our protocol yields a better speedup when the set size is larger. For smallest set size of $n = 2^8$, the protocol achieves a moderate speed up of about 1.13. When considering the larger database size $n = 2^{22}$, the speed up of $3.4 - 3.6$ is obtained at 4 threads and $6.7 - 19.1$ at 32 threads.

References

[ADT11] Giuseppe Ateniese, Emiliano De Cristofaro, and Gene Tsudik. (If) size matters: Size-hiding private set intersection. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and

Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 156–173. Springer, Heidelberg, March 2011.

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 535–548. ACM Press, November 2013.

[ANS10] Yuriy Arbitman, Moni Naor, and Gil Segev. Backyard Cuckoo hashing: Constant worst-case operations with a succinct representation. In *51st FOCS*, pages 787–796. IEEE Computer Society Press, October 2010.

[BA12] 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.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *28th ACM STOC*, pages 479–488. ACM Press, May 1996.

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

[BSMD10] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. Sepia: Privacy-preserving aggregation of multi-domain network events and statistics. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, pages 15–15, Berkeley, CA, USA, 2010. USENIX Association.

[BST01] Fabrice Boudot, Berry Schoenmakers, and Jacques Traoré. A fair and efficient solution to the socialist millionaires' problem. *Discrete Applied Mathematics*, 111:2001, 2001.

[CCS18] Andrea Cerulli, Emiliano De Cristofaro, and Claudio Soriente. Nothing refreshes like a RePSI: Reactive private set intersection. In *ACNS 18*, LNCS, pages 280–300. Springer, Heidelberg, 2018.

[CDJ16] Chongwon Cho, Dana Dachman-Soled, and Stanislaw Jarecki. Efficient concurrent covert computation of string equality and set intersection. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 164–179. Springer, Heidelberg, February / March 2016.

[CLR17] 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 17*, pages 1243–1255. ACM Press, October / November 2017.

[CO18] Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. In *SCN 18*, LNCS, pages 464–482. Springer, Heidelberg, 2018.

[CPPT14] Ran Canetti, Omer Paneth, Dimitrios Papadopoulos, and Nikos Triandopoulos. Verifiable set operations over outsourced databases. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 113–130. Springer, Heidelberg, March 2014.

[DC17] 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.

[DCW13] 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 13*, pages 789–800. ACM Press, November 2013.

[DKT10] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 213–231. Springer, Heidelberg, December 2010.

[DRRT18] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. Pir-psi: Scaling private contact discovery. *Proceedings on Privacy Enhancing Technologies*, 2018.

[DT10] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In Radu Sion, editor, *FC 2010*, volume 6052 of *LNCS*, pages 143–159. Springer, Heidelberg, January 2010.

[FIPR05] 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.

[FNO18] Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. Private set intersection with linear communication from general assumptions. Cryptology ePrint Archive, Report 2018/238, 2018. https://eprint.iacr.org/2018/238.

[FNP04] 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.

[FNW] Ronald Fagin, Moni Naor, and Peter Winkler. Comparing information without leaking it. *Commun. ACM*.

[Fri07] 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.

[GBZT] Pierrick Gaudry, Richard Brent, Paul Zimmermann, and Emmanuel Thomé. https://gforge.inria.fr/projects/gf2x/.

[Gon81] Gaston H. Gonnet. Expected length of the longest probe sequence in hash code searching. *J. ACM*, 28(2):289–304, April 1981.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS 2012*. The Internet Society, February 2012.

[HEKM11] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security 2011*, 2011.

[HLS$^+$16] K. Hogan, N. Luther, N. Schear, E. Shen, D. Stott, S. Yakoubov, and A. Yerukhimovich. Secure multiparty computation for cooperative cyber risk assessment. In *2016 IEEE Cybersecurity Development (SecDev)*, pages 75–76, Nov 2016.

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

[HV17] 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.

[IKN$^+$19] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Mariana Raykova, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. On deploying secure computing commercially: Private intersection-sum protocols and their business applications. Cryptology ePrint Archive, Report 2019/723, 2019.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003.

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *21st ACM STOC*, pages 44–61. ACM Press, May 1989.

[KK13] Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 54–70. Springer, Heidelberg, August 2013.

[KKRT16] 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 16*, pages 818–829. ACM Press, October 2016.

[KMP$^+$17] 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 17*, pages 1257–1272. ACM Press, October / November 2017.

[KS05] 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.

[Lip03] Helger Lipmaa. Verifiable homomorphic oblivious transfer and private equality test. In Chi-Sung Laih, editor, *ASIACRYPT 2003*, volume 2894 of *LNCS*, pages 416–433. Springer, Heidelberg, November / December 2003.

[LV04] Arjen K. Lenstra and Tim Voss. Information security risk assessment, aggregation, and mitigation. In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *ACISP 04*, volume 3108 of *LNCS*, pages 391–401. Springer, Heidelberg, July 2004.

[NP] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, STOC '99.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In S. Rao Kosaraju, editor, *12th SODA*, pages 448–457. ACM-SIAM, January 2001.

[PR04] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

[PRTY19] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: Lightweight private set intersection from sparse ot extension. In *CRYPTO 2019*, 2019.

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: private set intersection using permutation-based hashing. In *Proceedings of the 24th USENIX Conference on Security Symposium*, pages 515–530. USENIX Association, 2015.

[PSTY] 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*.

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on ot extension. In *Proceedings of the 23rd USENIX conference on Security Symposium*, pages 797–812. USENIX Association, 2014.

[PSZ18] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on ot extension. *ACM Trans. Priv. Secur.*, 21(2), January 2018.

[RA18] Amanda C. D. Resende and Diego F. Aranha. Faster unbalanced private set intersection. In *FC 2018*, LNCS. Springer, Heidelberg, 2018.

[Rab81] Michael O. Rabin. How to exchange secrets by oblivious transfer. Aiken Computation Laboratory, Harvard U., 1981.

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

[RR17a] Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 235–259. Springer, Heidelberg, April / May 2017.

[RR17b] 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 17*, pages 1229–1242. ACM Press, October / November 2017.

[Sho] Victor Shoup. http://www.shoup.net/ntl/.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015.

	Our	Set size n							
	Protocol	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}
Time	with PEQT	0.66	0.86	1.42	3.54	12.41	61.34	238.88	1039.64
	without PEQT	0.65	0.86	1.41	3.51	11.02	49.12	229.22	1015.23
Comm.	with PEQT	0.45	2.05	8.48	34.98	144.65	652.09	2693.30	11,077.83
	without PEQT	0.41	1.86	7.72	31.80	131.16	600.62	2470.10	10,233.27

Table 5: Comparison of total runtime (in seconds) and communication (in MB) between the RPMT version and the optimized version of our protocol. n elements per party, 128-bit length element, and single thread in LAN setting with 10Gbps network bandwidth, 0.02ms RTT.

A RPMT Optimization

In the RPMT protocol, the receiver computes a polynomial P with special output s. The sender computes $s^* = P(h(x^*)) \oplus q^*$, where q^* is its OPRF output. Then the parties use PEQT to securely compare s to s^*.

In the context of PSU, it is not necessary to use PEQT for this step. Instead, the sender can simply send s^* to the receiver. The logic is as follows: If $x^* \in X$, the sender should learn only this fact (and nothing about x^*). This is still the case after the optimization because the sender will compute the same polynomial output s^* for any such $x^* \in X$. If $x^* \in X$, it means that the receiver will eventually learn x^* as part of the PSU output (and the sender can infer that x^* was contributed by the receiver). The PSU simulator will therefore have the value x^*, and it can perfectly simulate the polynomial output $s^* = P(h(x^*)) \oplus q^*$.

We now formalize the details of this modification. Rather than define a weaker/leaky version of RPMT, we instead introduce a protocol for 1-vs-n PSU. Such a functionality is quite similar to RPMT, which can be thought of as revealing only the cardinality of $|\{x^*\} \cup X|$, which is equivalent to revealing the cardinality of $|\{x^*\} \setminus X|$ (either 0 or 1).

The details of the 1-vs-n PSU protocol are given in Figure 9. Now, using 1-vs-n PSU as a building block instead of RPMT, our full-fledged PSU protocol can be written as in Figure 10.

The security proof of the full-fledged PSU protocol is essentially the same as in the pre-optimization protocol. The security of the 1-vs-n protocol is given below:

Theorem 3. *The construction of Figure 9 securely implements functionality $\mathcal{F}^{1,n}_{\mathsf{psu}}$ in the semi-honest model, given the OPRF primitive defined in Figure 4.*

Proof. We exhibit simulators $\mathsf{Sim}_{\mathcal{R}}$ and $\mathsf{Sim}_{\mathcal{S}}$ for simulating corrupt $\mathcal{R}$ and $\mathcal{S}$ respectively, and argue the indistinguishability of the produced transcript from the real execution.

Corrupt Sender. $\mathsf{Sim}_{\mathcal{S}}(x^*)$ simulates the view of corrupt $\mathcal{S}$, which consists of $\mathcal{S}$'s randomness, input, output and received messages. $\mathsf{Sim}_{\mathcal{S}}$ proceeds as follows. It first chooses $q' \in_R \{0,1\}^\sigma$, calls OPRF simulator $\mathsf{Sim}_{\mathcal{S}_{\mathsf{OPRF}}}(x^*, q')$, and appends its output to the view.

$\mathsf{Sim}_{\mathcal{S}}$ simulates Step 3 as follows. It generates random $s' \in \{0,1\}^\sigma$, and n random points $(x'_i, q'_i) \in_R (\{0,1\}^\star, \{0,1\}^\sigma)$. $\mathsf{Sim}_{\mathcal{S}}$ then interpolates the polynomial P over these points $\{h(x'_i), s' \oplus q'_i\}$ and appends its coefficients to the generated view.

We argue that the output of $\mathsf{Sim}_{\mathcal{S}}$ is indistinguishable from the real execution. For this, we formally show the simulation by proceeding the sequence of hybrid transcripts T_0, T_1, T_2, where T_0 is real view of $\mathcal{S}$, and T_2 is the output of $\mathsf{Sim}_{\mathcal{S}}$.

Hybrid 1. Let T_1 be the same as T_0, except that the OPRF execution is replaced as follows. By the OPRF/BaRK-OPRF pseudorandomness guarantee and the indistinguishability of the output of $\mathsf{Sim}_{\mathcal{S}_{\mathsf{OPRF}}}$, we replace $F(k, x^*)$ and $F(k, x_i), \forall i \in [n]$, with q' and $q'_i, \forall i \in [n]$, respectively. We note that if $x^* = x_i$, then $q' = q'_i$. It is easy to see that T_0 and T_1 are indistinguishable.

Hybrid 2. Let T_2 be the same as T_1, except that the polynomial is an uniform polynomial of degree $n - 1$. Consider two following cases:

- $x^* \notin X$: Since all values q'_i are uniformly random from the $\mathcal{S}$'s point of view, so are the $s \oplus q'_i$.
- $x^* = x_i$ (consequently, $q' = q'_i$): Since other values $q'_{j \in [n]}, \forall j \neq i$, are uniformly random from $\mathcal{S}$'s point of view, we replace these $s \oplus q'_j$ with random. Then s is used only in the expression $s \oplus q'_i$. Since s is uniform, $s \oplus q'_i$ is also uniformly random from the $\mathcal{S}$'s view even though the adversary knows $q' = q'_i$.

In summary, the polynomial from the real execution can be replaced with a polynomial P over random points. T_1 and T_2 are indistinguishable.

Corrupt Receiver. $\mathsf{Sim}_{\mathcal{R}}(x_1, ..., x_n, out)$ simulates $\mathcal{R}$'s view, which includes $\mathcal{R}$'s randomness, input, output and received messages. $\mathsf{Sim}_{\mathcal{R}}$ proceeds as follows.

First, if $out = \{x_1, \dots, x_n, x^*\}$ for some x^*, then the simulator knows $\mathcal{S}$'s input x^* and can trivially simulate all of $\mathcal{S}$'s actions honestly. This case of simulation is clearly perfect.

Otherwise, $\mathsf{Sim}_{\mathcal{R}}$ chooses a random $k' \in_r \{0, 1\}^\kappa$, calls OPRF simulator $\mathsf{Sim}_{\mathcal{S}_{\mathsf{OPRF}}}(\perp, k')$, and appends its output to the view. It simulates a message $s^* = s$ from $\mathcal{S}$ in Step 4. Finally, to simulate Step 5, $\mathsf{Sim}_{\mathcal{S}}$ runs simulator $\mathsf{Sim}_{\mathsf{OT}}$ on input $(1, \perp)$ and appends the output of $\mathsf{Sim}_{\mathsf{OT}}$ to its output of the view.

The view generated by $\mathsf{Sim}_{\mathcal{R}}$ in indistinguishable from a real view because of the indistinguishability of the transcripts of the underlying simulators.

□

PARAMETERS:

- Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$
- Set X is of size n of elements.
- The field size $\sigma = \lambda + \log(n)$.
- Ideal OPRF primitive specified in Figure 4. Let $F_k(x) : \{0,1\}^* \mapsto \{0,1\}^\sigma$ be the underlying OPRF function.
- A collision-resistant hash function $h(x) : \{0,1\}^* \mapsto \{0,1\}^\sigma$

INPUT OF $\mathcal{S}$: $x^* \in \{0,1\}^*$
INPUT OF $\mathcal{R}$: $X = \{x_1, x_2, \ldots, x_n\} \subseteq \{0,1\}^*$
PROTOCOL:

1. $\mathcal{S}$ acts as OPRF *receiver*, sends x^* to OPRF. $\mathcal{S}$ receives $q^* = F_k(x^*)$ and receiver $\mathcal{R}$ receives k.
2. $\mathcal{R}$ randomly picks $s \leftarrow \mathbb{F}(2^\sigma)$. $\mathcal{R}$ interpolates a $\mathbb{F}(2^\sigma)$-polynomial $P(x)$ over points $\{(h(x_i), s \oplus q_i)\}$, where $q_i = F_k(x_i), \forall i \in [n]$. Here $s \oplus q_i$ is computed as operation on σ-bit strings.
3. $\mathcal{R}$ sends the coefficients of P to $\mathcal{S}$.
4. $\mathcal{S}$ computes $s^* = P(h(x^*)) \oplus q^*$ and sends it to $\mathcal{R}$.
5. $\mathcal{S}$ and $\mathcal{R}$ invoke the oblivious transfer functionality:
   - $\mathcal{R}$ acts as *receiver* with input 1 if $s^* = s$ and input 0 otherwise.
   - $\mathcal{S}$ acts as *sender* with input $(x^*, \perp)$.
6. If $s^* = s$, then $\mathcal{R}$ gives output X. Otherwise, it learned x^* as output from the oblivious transfer, and outputs $X \cup \{x^*\}$.

Figure 9: 1-vs-n PSU Protocol.

PARAMETERS:

- Set sizes n_1 and n_2, and two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$
- A bit-length ℓ. Let $n = \max(n_1, n_2)$.
- Number of bins $\beta = \beta(n)$, hash function $H : \{0,1\}^\ell \to [\beta]$, and max bin size m, suitable for our hashing scheme (Table 2)
- A special dummy item $\perp \in \{0,1\}^*$

INPUT OF $\mathcal{S}$: $X = \{x_1, x_2, \ldots, x_{n_1}\} \subseteq \{0,1\}^\ell$
INPUT OF $\mathcal{R}$: $Y = \{y_1, y_2, \ldots, y_{n_2}\} \subseteq \{0,1\}^\ell$
PROTOCOL:

1. $\mathcal{S}$ and $\mathcal{R}$ hash elements of their sets X and Y into β bins under hash function H. Let $B_\mathcal{S}[i]$ and $B_\mathcal{R}[i]$ denote the set of items in the sender's and receiver's i-th bin, respectively.
2. $\mathcal{S}$ pads each bin $B_\mathcal{S}[i]$ with (several copies, as needed) the special item $\perp$ up to the maximum bin size $m+1$, and randomly permutes all items in this bin.
3. $\mathcal{R}$ pads each bin $B_\mathcal{R}[i]$ with one special item $\perp$ and (several, as needed) different dummy items to the maximum bin size $m+1$.
4. $\mathcal{R}$ initializes set $Z = \{\}$.
5. For each bin $i \in [\beta]$, for each item $x_j \in B_\mathcal{S}[i]$:
   (a) $\mathcal{S}$ and $\mathcal{R}$ invoke the $\mathcal{F}_{\mathsf{psu}}^{1,m+1}$-functionality:
      - $\mathcal{S}$ acts as *sender* with input x_j
      - $\mathcal{R}$ acts as *receiver* with input set $B_\mathcal{R}[i]$
      - $\mathcal{R}$ obtains output $Z_{i,j}$ and sets $Z = Z \cup Z_{i,j}$.
6. $\mathcal{R}$ outputs Z.

Figure 10: Private Set Union Protocol $\mathcal{F}_{\mathsf{psu}}^{n_1,n_2}$.