Circuit-PSI with Linear Complexity via Relaxed Batch OPPRF

Nishanth Chandran, Divya Gupta, and Akash Shah

Microsoft Research, India.
Email: {nichandr, divya.gupta, t-akshah}@microsoft.com.

Abstract. In 2-party Circuit-based Private Set Intersection (Circuit-PSI), P_0 and P_1 hold sets S_0 and S_1 respectively and wish to securely compute a function f over the set $\mathsf{S}_0 \cap \mathsf{S}_1$ (e.g., cardinality, sum over associated attributes, or threshold intersection). Following a long line of work, Pinkas *et al.* (PSTY, Eurocrypt 2019) showed how to construct a concretely efficient Circuit-PSI protocol with linear communication complexity. However, their protocol requires super-linear computation. In this work, we construct concretely efficient Circuit-PSI protocols with linear computational and communication cost. Further, our protocols are more performant than the state-of-the-art, PSTY – we are $\approx 2.3\times$ more communication efficient and are up to $2.8\times$ faster. We obtain our improvements through a new primitive called *Relaxed Batch Oblivious Programmable Pseudorandom Functions* (RB-OPPRF) that can be seen as a strict generalization of Batch OPPRFs that were used in PSTY. This primitive could be of independent interest.

1 Introduction

Private Set Intersection. Consider parties P_0 and P_1 who hold sets S_0 and S_1 respectively. Private set intersection (PSI) [50,35] allows the parties to compute the intersection of these 2 sets, $\mathsf{S}_0 \cap \mathsf{S}_1$, without revealing anything else to each other. This problem has received much attention [11,42,31,45,40,8] (also see references therein) and practical solutions to this problem are now known. However, in most applications, typically P_0 and P_1 would like to compute $f(\mathsf{S}_0 \cap \mathsf{S}_1)$, where f is a *symmetric* function. That is, f operates only on $\mathsf{S}_0 \cap \mathsf{S}_1$ and is oblivious to the order of the elements in $\mathsf{S}_0 \cap \mathsf{S}_1$. Some examples of popular and well-studied symmetric functions are set cardinality, set intersection sum [33] (where every element has an associated attribute and the output is the sum of these attributes for all the elements in the intersection), and threshold cardinality/set intersection [18,24,53,54,44,20,21,2] (which computes the intersection size/intersection respectively if the intersection size is larger than a threshold).

Circuit-PSI. To enable the computation of arbitrary symmetric functions securely over the intersection (including the aforementioned applications), Huang *et al.* [25] introduced the notion of *Circuit-PSI*. In a Circuit-PSI protocol, P_0 and P_1 receive shares of the set intersection instead of learning the intersection in the clear. These shares can be used to securely compute any symmetric function using generic 2-party secure computation protocols [23,4,52]. More specifically, for every element $z \in$ (say) S_0, P_0 and P_1 receive random bits a_0 and a_1 respectively as output where $a = a_0 \oplus a_1 = 1$ if $z \in \mathsf{S}_0 \cap \mathsf{S}_1$ (and is 0 otherwise). Following a sequence of works [42,44,9,16], the work of Pinkas *et al.* [43] somewhat surprisingly showed how to construct a Circuit-PSI protocol with *linear communication complexity* in n, the size of the input sets. Unfortunately, the computational complexity of their protocol is super-linear in n (specifically, $\mathcal{O}\left(n(\log n)^2\right)$) and is stated as one of the major bottlenecks for performance in [43].

1.1 Our Contributions

Linear Complexity Circuit-PSI. We construct Circuit-PSI protocols with communication and computational costs linear in n (this asymptotically matches recent work [27]). We demonstrate that our protocols are concretely better than the state-of-the-art [43] (which in turn outperforms [27]) – as an example, our protocol is $\approx 2.3\times$ more communication efficient and $2.3 - 2.8\times$ faster (in LAN/WAN settings) than [43], when P_0's and P_1's sets comprise of 2^{22} elements. We also extend our protocol to support computing functions on intersection of input sets with associated payloads.

Main Technical Contributions. As a core technical contribution, we introduce the notion of *Relaxed-Batch Oblivious Programmable Pseudorandom Functions* ($\mathsf{RB\text{-}OPPRF}$), which can be seen as a strict generalization of Batch Oblivious Programmable Pseudorandom Functions ($\mathsf{B\text{-}OPPRF}$), used in [43]. The linear communication construction of $\mathsf{B\text{-}OPPRF}$s in [43] required expensive polynomial interpolation of large degree polynomials, and hence was the source of the main computational (super-linear) inefficiency. In contrast, we show how to construct $\mathsf{RB\text{-}OPPRF}$ using cheap (and linear time) operations such as Cuckoo hashing. Secondly, we also construct new and more efficient protocols for the task of *Private Set Membership* (PSM) [17], and by coupling this with our $\mathsf{RB\text{-}OPPRF}$ construction, we obtain concretely efficient Circuit-PSI protocols with linear communication and computation. In PSM, one party has a list B and the other party has a single element a and they wish to test if $a \in B$.

Applications of Circuit-PSI. As discussed earlier, in many applications, P_0 and P_1 need to compute a function f over the intersection of their input sets and circuit-PSI protocols can be used to securely realize such applications. We now discuss two such applications:

- PSI-CAT/Threshold PSI. [18,24,53,54,44,43,20,21,2]. In the problem of PSI-CAT (resp. Threshold-PSI), the intersection set size (resp. intersection set) is revealed only if the size of the intersection is larger than a certain threshold. These problems have applications to privacy-preserving ridesharing [24].
- PSI-Sum. [33,44,43]. In this, P_0 has an input set with payloads and P_1 only has an input set. The parties must compute the sum of the payloads for all the elements in the intersection set. This problem has applications to revenue computation in ad conversion rates [33].

The work of [43] is the state-of-the-art protocol to realize the above applications; our new circuit-PSI protocols improve over [43] for these applications as well. As observed in [44,43], the cost to compute these functions over the output of the circuit-PSI protocol is tiny compared to computing the circuit-PSI output itself. Hence, we expect our $2.8\times$ improvement over [43] to translate to at least a $2\times$ improvement in the performance of securely realizing these functions.

Other Related Works. Recently, Karakoç and Küpçü [27], using very different techniques, also provide a Circuit-PSI protocol with linear communication and computational cost. However, their protocol has worse concrete efficiency ($\approx 4\times$ communication) than even [43] and is 5–12$\times$ slower than [43] in the LAN setting. Hence, our circuit-PSI protocols are $9\times$ more communication efficient and upto $33\times$ faster than [27].

In a concurrent and independent work, Rindal and Schoppmann [48] build a Circuit-PSI protocol using techniques from the PaXoS data structure [41]. The communication complexity of their protocol is $2.5\times$ and $1.2\times$ higher than our Circuit-PSI protocols when using IKNP-style OT Extension protocols [26,30] and Silent-OT Extension protocols [5,51] respectively.

1.2 Technical Overview

Before describing our construction, we discuss the protocol blueprint from [43].

The Protocol From [43]. First, P_0 uses Cuckoo hashing (with d hash functions, $\{h_i\}_{i=1}^d$) to hash the elements from set S_0 into a hash table HT_0 with β bins (where β is linear in n). Cuckoo hashing guarantees that every bin contains only a single element and that each element x is present in a location $h_i(x)$ for some $i \in [d]$ or in a separate set known as the stash. It will be instructive to first consider the stashless setting ([43] provide a technique to handle the stash separately). Next, P_1 employs standard hashing using all the hash functions $\{h_i\}_{i=1}^d$ to hash the set S_1 into a hash table HT_1 with the same number of bins. That is, each element $y \in \mathsf{S}_1$ will appear in d bins in HT_1, namely, $\{h_i(y)\}_{i\in[d]}$. Note that every bin can have many elements and it can be shown that for universal hash functions, each bin of HT_1 will have at most $\mathcal{O}(\log n)$ elements, with all but negligible probability. In the stashless setting, observe that if some element $z \in \mathsf{S}_0 \cap \mathsf{S}_1$ then if $\mathsf{HT}_0[j] = z$ for some j, then $z \in \mathsf{HT}_1[j]$ as well. Hence the circuit-PSI problem is reduced to β instances of the private set membership problem each with a set of size at most $\mathcal{O}(\log n)$ - i.e., for each bin, we need to compare a single element in HT_0's bin to the corresponding elements in HT_1's bin. Since comparing each of the $\mathcal{O}(\log n)$ elements in HT_1's bin with an element in HT_0's bin, for a linear number of bins would result in super-linear communication cost, [43] introduced a primitive known as Batch Oblivious Programmable Pseudorandom Functions (B-OPPRF) that reduces the number of comparisons per bin from $\mathcal{O}(\log n)$ to 1 with only linear in n communication.

B-OPPRF. Informally, an Oblivious PRF is a 2-party functionality that provides the sender with a key to a PRF, and the receiver with the outputs of the PRF on points of his choice. The Oblivious Programmable Pseudorandom Function (OPPRF) functionality additionally provides a "hint" hint to both the sender and the receiver that allows the sender to "program" the PRF to output specific values on certain (private) input points. When invoking β independent instances of OPPRF, where the number of programmed points in each instance could vary, but the total programmed points, $N = dn$, across all instances is fixed, [43], showed how to provide a single hint whose size is linear in N (and hence n) and further hides the number of programmed points in every instance. Such a primitive is known as a B-OPPRF. [43] then showed that P_1 can play the role of the sender in an instance of B-OPPRF protocol comprising of β independent instances of OPPRF with $\mathsf{HT}_1[j]$ as the programmed points and by setting *all the programmed outputs to a single random value* t_j in the j^{th} OPPRF instance. Now, P_0 will obliviously evaluate the j^{th} OPPRF with $\mathsf{HT}_0[j]$. With this, P_0 and P_1 will each hold a single value that is equal if $\mathsf{HT}_0[j] \in \mathsf{HT}_1[j]$ and unequal otherwise. They can then employ a standard equality protocol to compare these 2 elements. In summary, using B-OPPRF, they reduce the number of comparisons per bin from $\mathcal{O}(\log n)$ to 1, at the cost of linear communication; however, this unfortunately results in super-linear computation. This is because the creation of hint in the B-OPPRF construction of [43] requires polynomial interpolation that results in super-linear computational complexity.

Relaxed B-OPPRF and our Circuit-PSI Protocol. In this work, we introduce the notion of *Relaxed* B-OPPRF (or RB-OPPRF) that is a strict generalization of B-OPPRF; we then show how to efficiently construct RB-OPPRFs and then use them to realize a circuit-PSI protocol. The RB-OPPRF primitive reduces the number of comparisons per bin from $\mathcal{O}(\log n)$ to 3 while incurring only *linear computation cost*. In a d-RB-OPPRF instance, the functionality provides a set of "d" PRF outputs to the receiver on every input point. For programmed points, this set is guaranteed to include the programmed output. When the output set size is 1, this primitive is exactly the B-OPPRF primitive. Surprisingly, such a simple generalization makes constructing it much more efficient. In particular, we show how to construct an RB-OPPRF with output set size 3 that has linear computation and communication (in n) using Cuckoo hashing (with 3 hash functions), thereby avoiding the computationally expensive polynomial interpolation required in [43]. Applying a RB-OPPRF to the blueprint of [43] gives us the following guarantee for every bin: P_0 will hold a set $B = \{b_1, b_2, b_3\}$ and P_1 will hold a single value a such that $a \in B$ if $\mathsf{HT}_0[j] \in \mathsf{HT}_1[j]$ and different otherwise. Now, computing whether this is the case or not is a simple instance of a *private set membership* [17] with a set size of only 3. While many protocols for this task exist [25,12,45,9,10], we construct 2 new protocols, PSM1 and PSM2 that have 4.2× and 6.4× lower communication than prior works (see Table 1, Section 4.3). Protocol PSM1, uses techniques from tree-based comparison protocols [19,13,47] and has better computation but worse communication than PSM2 that uses the table-based OPPRF construction [32] on small sets.

Circuit-PSI With Stash. As mentioned earlier, the above discussion assumed that no stash is created during the cuckoo hashing phase. The work of [43] showed a novel dual-execution technique to compare stash elements of P_0 with elements of P_1 with linear cost. We show that the idea of dual-execution can be used with our stashless protocol as well in order to obtain an overall linear computation and communication protocol even with stash. Crucial to achieving this is the observation that when the construction in [43] is used with P_0 and P_1 having different sized-sets (say n_0 and n_1 with $n_1 < n_0$), then the computational cost of the protocol is super-linear in n_1 but linear in n_0.

1.3 Organization

We begin in Section 2 by formally defining the security model and describing the building blocks (such as cuckoo hashing, secret sharing schemes, and oblivious transfer) used by our protocol. In Section 3 we define our new primitive Relaxed Batch Programmable Pseudorandom Functions along with the corresponding oblivious 2-party functionality and efficient constructions for the same. Section 4 is devoted to our two new protocols for private set membership. We describe our circuit-PSI protocol in Section 5. In Section 6, we experimentally validate our circuit-PSI protocols and show that it outperforms prior works in both LAN and WAN settings.

2 Preliminaries

Notation. The computational security parameter and statistical correctness parameter are denoted by λ and σ respectively. The function $\mathsf{neg}(\gamma)$ denotes a negligible function in γ. For a finite set X, $x \xleftarrow{\$} X$ means that x is uniformly sampled from X, $|X|$ denotes the cardinality of set X and $X(i)$ denotes the i^{th} element of set X. We use the notion of sets and lists interchangeably in this paper. $x \leftarrow y$ denotes that variable x is assigned the value of variable y and operator $\|$ denotes concatenation. For a positive integer ℓ, $[\ell]$ denotes the set of integers $\{1, \ldots, \ell\}$. Let $\mathbf{1}\{b\}$ denote the indicator function that is 1 when b is *true* and 0 when b is *false*.

2.1 Problem Setting and Security Model

Problem Setting. Consider two parties P_0 and P_1 with private sets S_0 and S_1, respectively, each of size n and each element in the input sets are of bit-length μ. As is standard in secure multiparty computation (MPC), P_0 and P_1 agree on a function f to be computed on the intersection, i.e., the parties wish to compute $f(\mathsf{S}_0 \cap \mathsf{S}_1)$. For this, the two parties agree on a circuit C that computes f. Prior works consider two settings [44,43]. While in the first setting, f (or, C) takes only the elements as input, in the second setting, both elements and their associated payloads are considered.

Security Model. Following prior works on Circuit-PSI [25,42,44,16,43,27], we provide security against static probabilistic polynomial time (PPT) semi-honest adversaries in the real/ideal simulation paradigm [23,34]. A static semi-honest (or, honest-but-curious) adversary $\mathcal{A}$ corrupts either P_0 or P_1 at the beginning of the protocol and tries to learn as much as possible from the protocol execution while following the protocol specifications honestly.

Security is modeled using *real* and *ideal* interactions. In the real interaction, the parties execute the protocol Π in the presence of $\mathcal{A}$ and the environment $\mathcal{Z}$. Let $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}$ denote the binary distribution ensemble describing $\mathcal{Z}$'s output in the real interaction. In the ideal execution, the parties send their inputs to a trusted functionality $\mathcal{F}$ that performs the computation faithfully. Let $\mathcal{S}$ (the simulator) denote the adversary in this idealized execution and $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the binary distribution ensemble describing $\mathcal{Z}$'s output in this interaction. A protocol Π is said to securely realize a functionality $\mathcal{F}$ if for every adversary $\mathcal{A}$ in the real interaction, there exists an adversary $\mathcal{S}$ in the ideal interaction, such that no environment $\mathcal{Z}$ can tell apart the real and ideal interactions, except with negligible probability. That is, $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}} \approx_c \mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, where $\approx_c$ denotes computational indistinguishability. The universal composability (UC) framework [7] allows one to guarantee the security of arbitrary composition of different protocols. Hence, we can prove security of individual sub-protocols and the security of the full protocol follows from the composition.

2.2 Building Blocks

Simple Hashing. Consider a hash table HT consisting of α bins. Simple hashing uses a hash function $h : \{0,1\}^* \mapsto [\alpha]$ sampled from a universal hash function family $\mathcal{H}$ to map elements to bins in HT. An element e is inserted to HT by simply appending it to the bin $h(e)$. Evidently, a hash table built using simple hashing can have more than one element per bin. A variant of simple hashing utilizes d many universal hash functions, say, $h_1, \ldots, h_d : \{0,1\}^* \mapsto [\alpha]$ and an element e is inserted into all the bins $h_1(e), \ldots, h_d(e)$.

Cuckoo Hashing. Cuckoo hashing [39] uses $d > 1$ universal hash functions $h_1, \ldots, h_d : \{0,1\}^* \mapsto [\alpha]$ to map n_h elements to α bins in hash table HT, where $\alpha = O(n_h)$. To insert an element e into HT do the following: (1) If one of $\mathsf{HT}[h_1(e)], \ldots, \mathsf{HT}[h_d(e)]$ bins is empty, insert e in the lexicographically first empty bin. (2) Else, sample $i \in [d]$ uniformly at random, evict the element present in $\mathsf{HT}[h_i(e)]$, place e in bin $\mathsf{HT}[h_i(e)]$, and recursively try to insert the evicted element. If a threshold number of evictions are reached, the final evicted element is placed in a special bin called the *stash* that can hold multiple elements. Hence, after cuckoo hashing, an element e can be found in one of the following bins: $h_1(x), \ldots, h_d(x)$ or the stash. Observe that in cuckoo hashing each bin except the stash is restricted to accommodate at most one element.

For a stash of size s, insertion of $s+1$ elements into stash leads to stash overflow, and this event is termed as a hashing failure. The probability (over the sampling of hash functions) that a stash of size s overflows is known as the failure probability. In [29], it was shown that Cuckoo hashing of n_h elements into $(1+\varepsilon)n_h$ bins with $\varepsilon \in (0,1)$ for any $d \geqslant 2(1+\varepsilon)\ln(\frac{1}{\varepsilon})$ and $s \geqslant 0$ has failure probability $O({n_h}^{1-c(s+1)})$, for constant $c > 0$ as $n_h \mapsto \infty$. In the application of cuckoo hashing to the problem of PSI and Circuit-PSI [42,31,45,32,44,43], large stash is quite detrimental to performance, and hence, it preferable to use hashing parameters that lead to a very small stash or no stash at all. Concrete parameter analysis of Cuckoo hashing that balances security and efficiency in the context of PSI was performed in [45] by empirically determining the failure probability given the stash size s, the number of hash functions d, and the number of bins α. Through the analysis, they determined that for achieving a concrete failure probability of less than 2^{-40} for stash size $s = 0$, $\alpha = 1.27n_h$, $1.09n_h$ and $1.05n_h$ bins are required for $d = 3, 4$ and 5 respectively. In our experiments, similar to [45,43], we use this result to set our hashing parameters for no stash setting, which achieves statistical correctness of 2^{-40}.

Oblivious Transfer. We consider 1-out-M Oblivious Transfer (OT) functionality [46,6] $\binom{M}{1}$-OT_ℓ that takes as input M messages from sender $m_1, \ldots, m_M \in \{0,1\}^\ell$ and index $i \in [M]$ from the receiver. The functionality outputs m_i to the receiver, while the sender receives no output. We use the OT extension protocols proposed in [30,26]. These protocols allow to extend λ 'base' OTs to large (polynomial in λ) number of OTs using symmetric key primitives only. The protocols $\binom{M}{1}$-OT_ℓ [30] and $\binom{2}{1}$-OT_ℓ [26] execute in two rounds and have total communication of $2\lambda + M\ell$ and $\lambda + 2\ell$ bits respectively.

Secret Sharing Schemes. We use 2-out-of-2 additive secret sharing scheme [49] over field $\mathbb{Z}_2$ and use $\langle x \rangle_0$ and $\langle x \rangle_1$ to denote shares of an element $x \in \mathbb{Z}_2$. Shares are generated by sampling random elements $\langle x \rangle_0$ and $\langle x \rangle_1$ from $\mathbb{Z}_2$ with the constraint that $\langle x \rangle_0 \oplus \langle x \rangle_1 = x$. To reconstruct a value x using shares x_0 and x_1, compute $x \leftarrow x_0 \oplus x_1$. We refer shares over $\mathbb{Z}_2$ as boolean shares.

AND Functionality. The AND Functionality $\mathcal{F}_{\mathsf{AND}}$ takes as input shares of bits b_0 and b_1 from the two parties and outputs shares of b_0 AND b_1 to both parties. $\mathcal{F}_{\mathsf{AND}}$ can be realized using bit-triples [3], which are of the form $(\langle p \rangle_s, \langle q \rangle_s, \langle r \rangle_s)$, where $s \in \{0,1\}$ and $p \wedge q = r$. We use the protocol in [47] for triple generation which has an amortized communication cost of 144 bits/triple.

3 Relaxed Batch OPPRF

Batch PPRF and OPPRF. Informally, a pseudorandom function (PRF) [22], sampled with a key k from a function family, is computationally indistinguishable from a uniformly random function, to any adversary that only has oracle access to the function. A programmable PRF (PPRF in short), introduced by [32] is similar to a PRF, except that the function instead outputs "programmed" values on a set of "programmed" input points. A "hint", also given to the adversary, enables encoding such programmed inputs and outputs. Although the size of the hint grows with the number of programmed points, it leaks no other information about programmed inputs or outputs. When β independent instances of a PPRF are used, the β different hints can be combined into a single hint that only grows with the total number of programmed points but leaks no information about the number of programmed points in every instance. This notion was introduced and formalized as Batch PPRF (B-PPRF) by *Pinkas et al.* [43].

Parameters. A PRF F: $\{0,1\}^\lambda \times \{0,1\}^\ell \to \{0,1\}^\ell$.
Sender's Inputs. No input.
Receiver's Inputs. Query $q \in \{0,1\}^\ell$.

The functionality works as follows:

1. Sample $k \xleftarrow{\$} \{0,1\}^\lambda$.
2. Output k to sender and $F(k,q)$ to receiver.

Fig. 1. OPRF Functionality $\mathcal{F}_{\mathsf{OPRF}}$

The 2-party Oblivious PRF (OPRF) functionality was defined by [17] and provides a PRF key to the sender and gives the receiver the evaluation of the PRF on a point chosen by the receiver (see Fig. 1). Such a functionality can also be defined with the notions of PPRF (respectively B-PPRF), where the sender specifies the programmed inputs/outputs, and the receiver specifies the evaluation points. The functionality gives the sender the key k and hint, while the receiver obtains the hint as well as the output of the PPRF (respectively B-PPRF) on its evaluation points. The corresponding functionalities are then known as Oblivious PPRF (OPPRF) and Batch Oblivious PPRF (B-OPPRF) respectively.

Constructions in [43]. While the size of the hint in the B-PPRF (and hence B-OPPRF) scheme of [43] is linear in the total number of programmed points, the computational complexity of generating it is super-linear. This is because generation of the hint requires interpolating an m-degree polynomial (where m is a parameter linear in the number of programmed points), which can be done in $\mathcal{O}(m^2)$ using Lagrange interpolation or $\mathcal{O}(m \log^2 m)$ using FFT. For its application to circuit-PSI problem, [43] proposed an optimization that brings down the cost to $\omega(m\mathrm{log}m)$ using Lagrange interpolation or $\omega(m(\mathrm{loglog}m)^2)$ using FFT. Even with this optimization, it was noted in [43] that the computational complexity of this polynomial interpolation step is a major bottleneck. We recall the polynomial based B-PPRF construction of [43] in Fig. 9 of Appendix A.

New Relaxed Notions. With the computational cost in mind, we generalize the notions of B-PPRF and B-OPPRF in the following way. While the B-PPRF primitive outputs a single pseudorandom value on every input point, we allow the primitive to output a set of d pseudorandom values, with the only constraint that for a programmed input, the programmed output is one out of these d elements. We call this notion Relaxed Batch Programmable PRF (RB-PPRF in short) and also define the corresponding 2-party Relaxed Batch Oblivious Programmable PRF (RB-OPPRF). We present the definitions of these notions in Section 3.1 and show how to construct them in Section 3.2 for $d = 3$. Our constructions are concretely efficient and have hint size linear in total number of programmed points as in [43] and unlike [43] only requires linear compute. Further, in Section 5 we show how to make use of this relaxed variant to construct an efficient Circuit-PSI protocol that outperforms the state-of-the-art [43] (see Section 6).

3.1 Defining RB-OPPRF

We first present the definition of Relaxed Batch Programmable PRF (RB-PPRF). As discussed earlier, this is a generalization of B-PPRF such that the programmed PRF outputs d pseudorandom values (instead of 1). On programmed inputs, one of these outputs is guaranteed to be the programmed output. We present our definition in such a way that setting $d = 1$, we obtain the same definition of B-PPRF presented in [43]. Let $\mathcal{T}$ be a distribution of multi-sets whose each element is uniformly random but where the elements can be correlated. Let F' be a PRF with keys of length λ and mapping ℓ bits to $d\ell$ bits, i.e., $F' : \{0,1\}^\lambda \times \{0,1\}^\ell \to \{0,1\}^{d\ell}$.

Parameters. An (ℓ, β, d)-RB-PPRF $\hat{F} = (Hint, F)$.
Sender's Inputs. Input sets $X_1, \ldots, X_\beta$ and target sets $T_1, \ldots, T_\beta$, where $|X_j| = |T_j|$ for every $j \in [\beta]$ and $X_j(i) \in \{0,1\}^\ell$ and $T_j(i) \in \{0,1\}^\ell$ for every $j \in [\beta]$ and $i \in [|X_j|]$. Let the total number of elements across the input sets be $N = \sum_{j=1}^{\beta} |X_j|$. The target sets are sampled independently from $\mathcal{T}$.
Receiver's Inputs. The queries $x_1, \ldots, x_\beta \in \{0,1\}^\ell$.

The functionality does the following:
1. Sample random and independent PRF keys for F', $K = k_1, \ldots, k_\beta$.
2. Invoke $\mathsf{hint} \leftarrow Hint(K, X, T)$.
3. Output keys K to sender.
4. Output hint and for all $j \in [\beta]$, $F(k_j, \mathsf{hint}, x_j)$ to receiver.

Fig. 2. Relaxed Batch OPPRF Functionality (ℓ, β, d)-$\mathcal{F}_{\mathsf{RB\text{-}OPPRF}}$

Definition 1 (Relaxed Batch Programmable PRF). *An $(\ell,\ \beta,\ d)$ Relaxed Batch Programmable PRF (or (ℓ, β, d)-*RB-PPRF*) is a pair of algorithms $\hat{F} = (Hint, F)$ described below:*

- $Hint(K, X, T) \rightarrow \mathsf{hint}$*: Given a set of uniformly random and independent PRF keys for F', $K = k_1, \ldots, k_\beta \in \{0,1\}^\lambda$, the disjoint input sets $X = X_1, \ldots, X_\beta$ and target multi-sets $T = T_1, \ldots, T_\beta$ such that for all $j \in [\beta]$, $|T_j| = |X_j|$ and for all $i \in [|X_j|]$, $X_j(i) \in \{0,1\}^\ell$ and $T_j(i) \in \{0,1\}^\ell$. Moreover, all sets T_j are sampled independently from $\mathcal{T}$. Output the hint* $\mathsf{hint} \in \{0,1\}^{c \cdot \ell \cdot N}$ *where $N = \sum_{j=1}^{\beta} |X_j|$ and $c \geqslant 1$ is a constant.*
- $F(k, \mathsf{hint}, x) \rightarrow W$*: Given a key $k \in \{0,1\}^\lambda$ and a hint* $\mathsf{hint} \in \{0,1\}^{c \cdot \ell \cdot N}$ *and an input query $x \in \{0,1\}^\ell$, outputs a list W of d elements of length ℓ, i.e., for all $i \in [d]$, $W[i] \in \{0,1\}^\ell$.*

  *A scheme is a (ℓ, β, d)-*RB-PPRF *if it satisfies:*

- ***Correctness****. For every $K = k_1, \ldots, k_\beta$, $T = T_1, \ldots, T_\beta$, $X = X_1, \ldots, X_\beta$ and* $\mathsf{hint} \leftarrow Hint(K, X, T)$*, we have for every $j \in [\beta]$ and $i \in [|X_j|]$, $T_j(i) \in F(k_j, \mathsf{hint}, X_j(i))$.*
- ***Security****. We say that an interactive machine M is a* RB-PPRF *oracle over $\hat{F}$ if, when interacting with a "caller" $\mathcal{A}$, it works as follows:*
  1. *$\mathcal{A}$ gives disjoint sets $X = X_1, \ldots, X_\beta$ to M.*
  2. *M samples uniform PRF keys $K = k_1, \ldots, k_\beta$ and target multi-sets $T = T_1, \ldots, T_\beta$ from $\mathcal{T}$. M sends* $\mathsf{hint} \leftarrow Hint(K, X, T)$ *to $\mathcal{A}$.*
  3. *M receives β queries $x_1, \ldots, x_\beta$ from $\mathcal{A}$ and responds with $W_1, \ldots, W_\beta$; $W_j = F(k_j, \mathsf{hint}, x_j)$.*
  4. *M halts.*

  The scheme $\hat{F}$ is said to be secure if for every disjoint sets $X_1, \ldots, X_\beta$ (where $N = \sum_{j=1}^{\beta} |X_j|$) input by a PPT *machine $\mathcal{A}$, the output of M is computationally indistinguishable from the output of $\mathcal{S}(1^\lambda, N)$, such that $\mathcal{S}$ outputs a uniformly random* $\mathsf{hint} \in \{0,1\}^{c \cdot \ell \cdot N}$ *and set of β lists each comprising of d uniformly random values from $\{0,1\}^\ell$.*

Remark. Above definition of (ℓ, β, d)-RB-PPRF sets the input length to be the same as the output length, i.e., ℓ. The definition easily generalizes to (ℓ, β, d)-RB-PPRF with input length ℓ', different from the output length, ℓ.

Relaxed Batch Oblivious Programmable PRF (RB-OPPRF). This 2-party functionality takes as input the set of programmed input sets $\{X_j\}_{j=1}^{\beta}$ and target sets $\{T_j\}_{j=1}^{\beta}$ from the sender. It takes as input a set of queries $x_1, \cdots, x_\beta$ from the receiver. It samples the set of β keys and hint for an (ℓ, β, d)-RB-PPRF scheme and provides the sender with the keys and the hint. It gives the hint and the output of the RB-PPRF for all points $x_j, j \in [\beta]$ to the receiver. We define the functionality for RB-OPPRF formally in Fig. 2.

Parameters. Let $F' : \{0,1\}^\lambda \times \{0,1\}^\ell \to \{0,1\}^{3\ell}$ be a PRF, let h_1, h_2 and h_3: $\{0,1\}^\ell \mapsto [\gamma]$ be three universal hash functions, where $\gamma = (1+\varepsilon)N$ and $\varepsilon \in (0,1)$.

$\underline{\mathsf{Hint}(K, X, T)}$: Given β keys for F', $K = k_1, \ldots, k_\beta \in \{0,1\}^\lambda$, disjoint input sets $X = X_1, \ldots, X_\beta$ (with total elements N) and target multi-sets $T = T_1, \ldots, T_\beta$, prepare a garbled hash table GT with γ bins as described below.

1. Do cuckoo hashing using h_1, h_2 and h_3 to store all the elements in input sets $X_1, \ldots, X_\beta$ in a hash table HT with γ bins.
2. Let E be a mapping that maps elements to the index of the hash function that was eventually used to insert that element into HT, i.e. $\mathsf{E}(X_j(i)) = \mathsf{idx}$ such that $\mathsf{HT}[h_{\mathsf{idx}}(X_j(i))] = X_j(i)$.
3. **for** $j \in [\beta]$ **do**
4.   **for** $i \in [|X_j|]$ **do**
5.   Compute $f_1 \| f_2 \| f_3 \leftarrow F'(k_j, X_j(i))$, where $f_b \in \{0,1\}^\ell$ for all $b \in [3]$.
6.   For $\mathsf{idx} \leftarrow \mathsf{E}(X_j(i))$ and $\mathsf{pos} \leftarrow h_{\mathsf{idx}}(X_j(i))$, set $\mathsf{GT}[\mathsf{pos}] \leftarrow f_{\mathsf{idx}} \oplus T_j(i)$.
7.   **end**
8. **end**
9. For every empty bin i in GT, pick $r_i \xleftarrow{\$} \{0,1\}^\ell$ and set $\mathsf{GT}[i] \leftarrow r_i$.
10. Return GT as hint.

$\underline{F(k, \mathsf{hint}, x)}$: Given $k \in \{0,1\}^\lambda$, $\mathsf{hint} \in \{0,1\}^{\gamma \cdot \ell}$ and input query $x \in \{0,1\}^\ell$, compute list W as follows:

1. Interpret hint as a garbled hash table GT.
2. Compute $\mathsf{pos}_b \leftarrow h_b(x)$ for all $b \in [3]$.
3. Compute $f_1 \| f_2 \| f_3 \leftarrow F'(k, x)$, where $f_b \in \{0,1\}^\ell$ for all $b \in [3]$.
4. Return list $W = [f_b \oplus \mathsf{GT}[\mathsf{pos}_b]]_{b \in [3]}$.

Fig. 3. Construction of $(\ell, \beta, 3) - \mathsf{RB\text{-}PPRF}$

3.2 RB-PPRF Construction

In this section, we present our construction of an $(\ell, \beta, 3) - \mathsf{RB\text{-}PPRF}$ that has linear computational complexity (in N, the total number of programmed points). Our construction makes use of cuckoo hashing, instantiated with 3 hash functions, to hash elements from β input sets (with a total of N elements) into $\gamma = (1+\varepsilon)N$ bins, where $\varepsilon \in (0,1)$. The construction assumes the stashless setting in cuckoo hashing. If indeed a stash is created, it is handled by the circuit-PSI protocol that uses $\mathsf{RB\text{-}PPRF}$ (see Section 5.2).

The construction is formally described in Fig. 3. Let $F' : \{0,1\}^\lambda \times \{0,1\}^\ell \to \{0,1\}^{3\ell}$ be a PRF. At a very high level, we construct an $\mathsf{RB\text{-}PPRF}$ as follows. First, using cuckoo hashing with 3 hash functions (h_1, h_2, h_3), we hash the N elements from the β sets into γ bins of a hash table. Now, we know that every element $X_j(i)$ is present in one of 3 locations $h_1(X_j(i)), h_2(X_j(i))$, or $h_3(X_j(i))$ of the hash table. For every element that was hashed, we identify which of the 3 hash functions was used to insert that element - i.e., the index idx such that $X_j(i)$ was stored at location $h_{\mathsf{idx}}(X_j(i))$. Now, a garbled hash table GT with γ bins is created as follows. For every programmed input $X_j(i)$, we compute $f_1 \| f_2 \| f_3 \leftarrow F'(k_j, X_j(i))$, where each f_b is of length ℓ bits and store $T_j(i) \oplus f_{\mathsf{idx}}$ at position $h_{\mathsf{idx}}(X_j(i))$ in GT (see steps 5 & 6). We fill empty bins of GT with random values. This GT now serves as the hint to evaluate the $\mathsf{RB\text{-}PPRF}$. Evaluation of the $\mathsf{RB\text{-}PPRF}$ on an element x works by computing $\mathsf{pos}_b = h_b(x)$ for all $b \in [3]$, $f_1 || f_2 || f_3 \leftarrow F'(k, x)$ and outputting the 3 elements $f_b \oplus \mathsf{GT}[\mathsf{pos}_b]$ for $b \in [3]$. It is now quite easy to see that on programmed inputs, one of these values would indeed be the programmed output. The formal construction is described in Fig. 3 and we prove its correctness and security in Theorem 1.

Remark. It is easy to see that our $(\ell, \beta, 3)$-$\mathsf{RB\text{-}PPRF}$ can be extended to different values of d by varying the number of hash functions in the cuckoo hashing.

Theorem 1. *The construction described in Fig. 3 is a secure construction of an* $(\ell, \beta, 3)$-$\mathsf{RB\text{-}PPRF}$.

Proof. **Correctness.** For correctness, we need to show that for programmed points, we get values from the target set as output. That is, for every $j \in [\beta]$ and $i \in [|X_j|]$, $T_j(i) \in W = F(k_j, \mathsf{hint}, X_j(i))$. In particular, we show the following: Let $\mathsf{idx} \leftarrow \mathsf{E}(X_j(i))$, that is, $X_j(i)$ is inserted into Cuckoo hash table HT using

h_{idx}. Then, $W[\mathsf{idx}] = T_j(i)$. Let $f_1\|f_2\|f_3 \leftarrow F'(k_j, X_j(i))$ and $\mathsf{pos}_{\mathsf{idx}} = h_{\mathsf{idx}}(X_j(i))$. From the construction, it holds that $\mathsf{GT}[\mathsf{pos}_{\mathsf{idx}}] = f_{\mathsf{idx}} \oplus T_j(i)$. Hence, $W[\mathsf{idx}] = f_{\mathsf{idx}} \oplus \mathsf{GT}[\mathsf{pos}_{\mathsf{idx}}] = f_{\mathsf{idx}} \oplus f_{\mathsf{idx}} \oplus T_j(i) = T_j(i)$.

Security. Let M be $(\ell, \beta, 3) - \mathsf{RB\text{-}PPRF}$ oracle described in Definition 1 for $\hat{F}$, where $\hat{F}$ is as defined in the construction of Fig. 3 and let $\mathcal{S}$ be the simulator described in Definition 1. We show that if there exists a PPT distinguisher $\mathcal{D}$ that breaks security of $\hat{F}$ in Fig. 3 with noticeable probability then there exists a PPT adversary $\mathcal{B}^{\mathcal{O}}$ that breaks PRF security of F' with noticeable probability. More concretely, adversary $\mathcal{B}^{\mathcal{O}}$ successfully distinguishes between oracle access to β pseudorandom functions $\{F'(k_j,\cdot)\}_{j\in[\beta]}$ and β random functions $\{R_j\}_{j\in[\beta]}$. With this, proof follows by contradiction using PRF security against PPT adversaries.

Reduction. $\mathcal{B}^{\mathcal{O}}$ receives input sets $X = X_1, \ldots, X_\beta$ from $\mathcal{A}$ and sets $N = \sum_{j=1}^{\beta} |X_j|$. $\mathcal{B}^{\mathcal{O}}$ then samples $T = T_1, \ldots, T_\beta$ from $\mathcal{T}$. Let h_1, h_2, and h_3: $\{0,1\}^\ell \mapsto [\gamma]$ be three universal hash functions, where $\gamma = (1+\varepsilon)N$ and $\varepsilon \in (0,1)$. $\mathcal{B}^{\mathcal{O}}$ prepares a garbled hash table GT with γ bins as described below. (1) $\mathcal{B}^{\mathcal{O}}$, using cuckoo hashing with h_1, h_2, and h_3 stores all the elements in the input sets $X_1, \ldots, X_\beta$ into a hash table HT with γ bins. Let E be the mapping with $\mathsf{E}(X_j(i)) = \mathsf{idx}$ such that $\mathsf{HT}[h_{\mathsf{idx}}(X_j(i))] = X_j(i)$. (2) For all $j \in [\beta]$ and $i \in [|X_j|]$, $\mathcal{B}^{\mathcal{O}}$ does the following: (a) Query $\mathcal{O}$ on input $(j, X_j(i))$ - i.e., for the output of the j^{th} function on input $X_j(i)$ and parse the oracle response as $f_1\|f_2\|f_3$, where $f_b \in \{0,1\}^\ell$ for all $b \in [3]$. (b) For $\mathsf{idx} \leftarrow \mathsf{E}(X_j(i))$ and $\mathsf{pos} \leftarrow h_{\mathsf{idx}}(X_j(i))$, set $\mathsf{GT}[\mathsf{pos}] \leftarrow f_{\mathsf{idx}} \oplus T_j(i)$. (3) For every empty bin i in GT, $\mathcal{B}^{\mathcal{O}}$ picks $r_i \xleftarrow{\$} \{0,1\}^\ell$ and sets $\mathsf{GT}[i] \leftarrow r_i$. $\mathcal{B}^{\mathcal{O}}$ sends GT as hint to the adversary $\mathcal{A}$. $\mathcal{B}^{\mathcal{O}}$ receives β queries $x = x_1, \ldots, x_\beta$ from $\mathcal{A}$. For all $j \in [\beta]$, $\mathcal{B}^{\mathcal{O}}$ responds with $W_j = [f_b \oplus \mathsf{GT}[h_b(x_j)]]_{b\in[3]}$, where $f_1\|f_2\|f_3 \leftarrow \mathcal{O}(j, x_j)$. Finally, $\mathcal{B}^{\mathcal{O}}$ outputs $\mathcal{D}$'s output.

First, when $\mathcal{O}$ is $\{F'(k_j,\cdot)\}_{j\in[\beta]}$, then it is easy to see that $\mathcal{B}^{\mathcal{O}}$ behaves identically to M in security game of $(\ell,\beta,3)$-$\mathsf{RB\text{-}PPRF}$.

For the other case, we argue that if $\mathcal{O}$ is $R_1, \cdots, R_\beta$, then $\mathcal{B}^{\mathcal{O}}$ is identical to $\mathcal{S}$. First, GT is uniformly random. This is because each element of GT is either chosen uniformly at random (from Step (3) above) or it is some $T_j(i)$ masked by the output of $\mathcal{O}$ on a unique query point $(j, X_j(i))$ (from Step (2.b) above). Second, from the above it is clear that GT does not leak any information about any of the target sets T_j for all $j \in [\beta]$. Now, consider the query responses provided by $\mathcal{B}^{\mathcal{O}}$. For the query x_j, there are two cases depending on whether $x_j \in X_j$ or not. In the former case, $x_j = X_j(i)$ for some $i \in |X_j|$. Let $\mathsf{E}(X_j(i)) = \mathsf{idx}$, that is, $X_j(i)$ is inserted into Cuckoo hash table HT using h_{idx}, and let $f_1\|f_2\|f_3 \leftarrow R_j(X_j(i))$ and $\mathsf{pos}_{\mathsf{idx}} = h_{\mathsf{idx}}(X_j(i))$. From the construction of $\mathcal{B}^{\mathcal{O}}$, it holds that $\mathsf{GT}[\mathsf{pos}_{\mathsf{idx}}] = f_{\mathsf{idx}} \oplus T_j(i)$. Hence, $W[\mathsf{idx}] = f_{\mathsf{idx}} \oplus \mathsf{GT}[\mathsf{pos}_{\mathsf{idx}}] = f_{\mathsf{idx}} \oplus f_{\mathsf{idx}} \oplus T_j(i) = T_j(i)$. It holds that $W[\mathsf{idx}]$ is uniformly random because (a) GT leaks no information about target sets, (b) T_j are sampled uniformly from $\mathcal{T}$ (c) Each $T_j(i)$ is uniformly random, and (d) there is a single query per $j \in [\beta]$. Also, $W[b] = f_b \oplus \mathsf{GT}[\mathsf{pos}_b]$, $\mathsf{pos}_b = h_b(x_j)$ for $b \neq \mathsf{idx}$ is uniformly random because given the view of the adversary so far, f_b is uniformly random. Hence, in the case where $x_j \in X_j$, $\mathcal{B}^{\mathcal{O}}$'s response is identical to $\mathcal{S}$. For the case when $x_j \notin X_j$, W_j is a set of values that have been masked by a response from $\mathcal{O}$ on a fresh input x_j and hence is uniformly random.

Hence, probability that $\mathcal{B}^{\mathcal{O}}$ succeeds in distinguishing a set of PRFs from random functions is identical to that of $\mathcal{D}$ succeeding in security game of $\mathsf{RB\text{-}PPRF}$.

Correctness Property for Non-programmed Points. When using the $\mathsf{RB\text{-}PPRF}$ construction in a circuit-PSI protocol, similar to [43], in order for the probability of false positives to be negligible in σ, we require the $\mathsf{RB\text{-}PPRF}$ construction to satisfy the following property. For every $K = k_1, \ldots, k_\beta$, $T = T_1, \ldots, T_\beta$, $X = X_1, \ldots, X_\beta$ and $\mathsf{hint} \leftarrow Hint(K, X, T)$ in Definition 1, we require that for every $j \in [\beta]$ and $x \notin X_j$, $\Pr[F(k_j, \mathsf{hint}, x) \cap T_j \neq \emptyset]$ be negligible in σ. To see that this property holds, observe that for all $j \in [\beta]$ and $x \notin X_j$, the entries in $W = F(k_j, \mathsf{hint}, x)$ in our $(\ell,\beta,3)$-$\mathsf{RB\text{-}PPRF}$ given in Fig. 3 comprise of a value from hint that is masked with PRF output on a completely fresh point which is never considered during the construction of hint. From PRF security it follows that these entries are pseudorandom and independent of values in T_j; hence, by a union bound, $\Pr[W_j \cap T_j \neq \emptyset] < 3\cdot 2^{-\ell} = 2^{-(\ell-\log 3)}$. It is easy to see that a similar property also holds for our $(\ell,\beta,3)$-$\mathsf{RB\text{-}OPPRF}$ construction described below when instantiated with $\mathsf{RB\text{-}PPRF}$ construction from Fig. 3.

3.3 RB-OPPRF Construction

We provide an $(\ell, \beta, 3)$-RB-OPPRF construction in Fig. 4 that uses the $(\ell, \beta, 3)$-RB-PPRF described above.

Theorem 2. *Given construction in Fig. 3 is a secure $(\ell, \beta, 3)$-RB-PPRF scheme, the construction in Fig. 4 securely realizes $(\ell, \beta, 3)$-$\mathcal{F}_{\mathsf{RB\text{-}OPPRF}}$ (Fig. 2) in the $\mathcal{F}_{\mathsf{OPRF}}$-hybrid model. Moreover, the scheme has linear communication and linear computational complexity in N.*

Proof. **Correctness.** The correctness of the construction follows easily from the correctness of $\mathcal{F}_{\mathsf{OPRF}}$ functionality and $(\ell, \beta, 3)$-RB-PPRF construction.

Security. To simulate the view of the sender, sample random PRF keys $K = k_1, \ldots, k_\beta \in \{0,1\}^\lambda$ and send it to sender. To simulate the view of the receiver, let $\mathcal{S}_1$ be the simulator for RB-PPRF. Let inputs of receiver be $x_1, \ldots, x_\beta$. Invoke $\mathcal{S}_1(1^\lambda, N)$ to learn hint and $\{W_j\}_{j\in[\beta]}$ where $W_j \in \{0,1\}^{3\ell}$. Next, parse hint as garbled hash table GT of size γ. Compute $\mathsf{pos}_b \leftarrow h_b(x_j)$ for all $b \in [3]$. For all $j \in [\beta], b \in [3]$, set $z_{j,b} = W_{j,b} \oplus \mathsf{GT}[\mathsf{pos}_b]$. Set $z_j = z_{j,1}||z_{j,2}||z_{j,3}$. Send $\{z_j\}_{j\in[\beta]}$ and hint to the receiver. It is easy to see that the security follows from security of RB-PPRF as proved in Theorem 1.

Linear Complexity. First, note that parties make β calls to $\mathcal{F}_{\mathsf{OPRF}}$ and $\beta \leqslant N$. To argue overall linear complexity for the sender, it suffices to argue linear complexity for hint computation and communication. We note that computing the hint using cuckoo hashing and garbled hash table has linear computation in N for the sender. Also, the size of the hint is $\gamma \cdot \ell = (1+\epsilon)N\ell$ for a constant $\epsilon \in (0,1)$. Finally, given linear size of hint, it is easy to see that receiver's compute is linear in N.

Parameters. Functionality $\mathcal{F}_{\mathsf{OPRF}}$, $(\ell, \beta, 3)$-RB-PPRF scheme $\hat{F} = (Hint, F)$ described in Fig. 3 that uses hash functions h_1, h_2 and h_3. Let $F' : \{0,1\}^\lambda \times \{0,1\}^\ell \rightarrow \{0,1\}^{3\ell}$ be the PRF used in both $\mathcal{F}_{\mathsf{OPRF}}$ and $\hat{F}$.
Sender's Inputs. Input sets $X_1, \ldots, X_\beta$ with total elements N and target multi-sets $T_1, \ldots, T_\beta$ that are sampled independently from $\mathcal{T}$. For all $j \in [\beta]$, $|X_j| = |T_j|$ and for all $i \in [|X_j|]$, $X_j(i) \in \{0,1\}^\ell$ and $T_j(i) \in \{0,1\}^\ell$.
Receiver's Inputs. Queries $x_1, \ldots, x_\beta \in \{0,1\}^\ell$.

The construction proceeds in the following manner:

1. For every $j \in [\beta]$, the parties invoke an instance of $\mathcal{F}_{\mathsf{OPRF}}$ where receiver inputs x_j. The sender gets a key k_j and receiver gets output $z_j \in \{0,1\}^{3\ell}$.
2. Sender computes $\mathsf{GT} \leftarrow Hint(K, X, T)$, where $K = k_1, \ldots, k_\beta$ and sends GT to the receiver.
3. Receiver does the following for each $j \in [\beta]$.
   - Parse $z_j = z_{j,1}||z_{j,2}||z_{j,3}$ s.t. $z_{j,b} \in \{0,1\}^\ell$ for all $b \in [3]$.
   - Compute $\mathsf{pos}_b \leftarrow h_b(x_j)$ for all $b \in [3]$.
   - Output list $\{W_j\}_{j\in[\beta]}$, where $W_j = [z_{j,b} \oplus \mathsf{GT}[\mathsf{pos}_b]]_{b\in[3]}$.

Fig. 4. RB-OPPRF construction using Cuckoo-hashing based RB-PPRF scheme

4 Private Set Membership

The set membership [17] is the 2-party functionality that takes as input a set B of n_p elements $\{B(1), \ldots, B(n_p)\}$, with $B(i) \in \{0,1\}^\ell$, $\forall i \in [n_p]$, from P_0 and an element $a \in \{0,1\}^\ell$ from P_1. Define $y = \mathbf{1}\{a \in B\}$. The functionality outputs boolean shares of y to P_0 and P_1; i.e., random bits y_0 and y_1 to P_0 and P_1 respectively such that $y_0 \oplus y_1 = y$. This functionality, $\mathcal{F}_{\mathsf{PSM}}$, is formally defined in Fig. 5.

$\mathcal{F}_{\mathsf{PSM}}$ can be realized using one of several standard protocols [23,3,1,12] and many specialized protocols [25,45,9]. However, the resultant protocols have high communication cost. In this section, we propose two specialized protocols - PSM1 and PSM2 to realize $\mathcal{F}_{\mathsf{PSM}}$ that both have significantly lower

Inputs of P_0**.** Input set B of size n_p, where $\forall i \in [n_p]$, $B(i) \in \{0,1\}^\ell$.
Inputs of P_1**.** $a \in \{0,1\}^\ell$.
The functionality outputs random y_b to party P_b, where y_0 and y_1 are boolean shares of $y = \mathbf{1}\{a \in B\}$.

Fig. 5. Private Set Membership Functionality $\mathcal{F}_{\mathsf{PSM}}$

communication overhead than prior approaches. While PSM2 (in Section 4.2) has the lower of the two communication, its concrete computation cost is higher than PSM1 (in Section 4.1). Trade-offs and comparison with other generic/specialized protocols are in Section 4.3.

4.1 Private Set Membership Protocol 1

Our protocol PSM1 builds on the idea in [19,13,47] used for the Millionaires' problem, where parties have secret inputs a and b respectively and want to compute shares of $\mathbf{1}\{a < b\}$. At a high level, the protocol uses recursion to reduce the problem of inequality on large strings to computing both equalities and inequalities on smaller substrings. We describe how we build on these ideas to realize $\mathcal{F}_{\mathsf{PSM}}$.

First, consider the case of single equality, i.e., computing $\mathbf{1}\{a = b\}$, $a, b \in \{0,1\}^\ell$. Let $a = a_1 \| a_0$ and $b = b_1 \| b_0$, where $a_0, b_0, a_1, b_1 \in \{0,1\}^{\ell/2}$. The problem of computing equality on ℓ bits strings can be reduced to equalities on $\ell/2$ length strings as follows:

$$\mathbf{1}\{a = b\} = \mathbf{1}\{a_1 = b_1\} \wedge \mathbf{1}\{a_0 = b_0\}, \tag{1}$$

We can then follow this approach recursively, and go to even smaller instances. Overall, we can build a tree, all the way upto the m-bit leaves that can be computed using $\binom{M}{1}$-OT_1 for $M = 2^m$. Then, we can traverse the tree bottom up using $\mathcal{F}_{\mathsf{AND}}$ functionality. As was also observed in [47] for the case of Millionaires', there is a compute vs communication trade-off between large and small values of m and one can empirically determine the value of m that gives best performance. More concretely, larger values of m result in lower communication but the compute grows super-polynomially in m.

The problem of set membership, i.e. $\mathbf{1}\{a \in B\}$ can be alternatively written as $\bigoplus_{i \in [n_p]} \mathbf{1}\{B(i) = a\}$, that is, it involves computing a batch of equalities. We show that for comparing an element a with all elements in a set B of size n_p, we can do much better than n_p instances of equality checks. We observe that for the leaf nodes, the inputs of P_1 are the same in all executions. Now, we run the OTs needed for the leaves with P_0 as the sender and P_1 as the receiver. Since the receiver's inputs are same in all n_p executions, these OTs can be batched together, and we replace n_p instances of $\binom{M}{1}$-OT_1 with a single instance of $\binom{M}{1}$-OT_{n_p} reducing communication per leaf from $n_p \times (2\lambda + M)$ to $(2\lambda + M n_p)$.

For ease of exposition, we describe the scheme formally in Fig. 6 for the special case when $m | \ell$ and when $q = \ell/m$ is a power of 2. Using the notation in Fig. 6, the batching of equality computation across n_p instances for the leaf nodes works as follows: P_0 has input set B of n_p elements such that each element $B(i) = b_{i,q-1} \| \ldots \| b_{i,0}$ for all $i \in [n_p]$. Similarly, P_1 has input $a = a_{q-1} \| \ldots \| a_0$. It holds that a_j, $b_{i,j} \in \{0,1\}^m$, for all

$j \in \{0, \ldots, q-1\}$ and $i \in [n_p]$. For each $j \in \{0, \ldots, q-1\}$, the task is to compute shares of $\mathsf{eq}_{0,i,j} = \mathbf{1}\{b_{i,j} = a_j\}$ for all $i \in [n_p]$, and this is where we use our batching technique. We use an instance of $\binom{M}{1}$-OT_{n_p} where P_0 is the sender and P_1 is the receiver with input a_j. Sender's input are $\{e_{j,v}\}_{v \in [M]}$, where $e_{j,v} = p_1 || \ldots || p_{n_p}$, with $p_i = \langle \mathsf{eq}_{0,i,j} \rangle_0 \oplus \mathbf{1}\{b_{i,j} = v\}$, with a uniformly random $\langle \mathsf{eq}_{0,i,j} \rangle_0$. Essentially, sender's v^{th} input are boolean shares of values $\{\mathbf{1}\{b_{i,j} = v\}\}_{i \in [n_p]}$. Once we have the leaf computation of the whole batch of size n_p, traversing up the tree to the root happens independently for each instance. Finally, we learn the shares corresponding to the roots, i.e. $\{\mathbf{1}\{b_i = a\}\}_{i \in [n_p]}$. For final output, parties locally XOR these shares.

Next, we prove correctness and security of our protocol. Finally we describe how the scheme can be naturally extended to the general case.

Inputs of P_0**.** Input set B of size n_p, where $\forall i \in [n_p]$, $B(i) \in \{0,1\}^\ell$.
Inputs of P_1**.** Element $a \in \{0,1\}^\ell$.
Parameters. Radix Parameter m, $M = 2^m$, $\binom{M}{1}\text{-}\mathsf{OT}_{n_p}$ and $\mathcal{F}_{\mathsf{AND}}$ functionality.

1. Set $q \leftarrow \ell/m$.
2. P_0 parses each of its input element as $B(i) = b_{i,q-1}\|\ldots\|b_{i,0}$ and P_1 parses its input as $a = a_{q-1}\|\ldots\|a_0$, where $a_j,\, b_{i,j} \in \{0,1\}^m$, for all $j \in \{0,\ldots,q-1\}$ and $i \in [n_p]$.
3. **for** $j \in \{0,\ldots,q-1\}$ **do**
4. $\quad$ P_0 samples $\langle \mathsf{eq}_{0,i,j}\rangle_0 \xleftarrow{\$} \{0,1\}$, $\forall i \in [n_p]$.
5. $\quad$ **for** $v \in [M]$ **do**
6. $\quad\quad$ P_0 sets $e_{j,v} \leftarrow \langle \mathsf{eq}_{0,1,j}\rangle_0 \oplus \mathbf{1}\{b_{1,j} = v\}\|\ldots\|\langle \mathsf{eq}_{0,n_p,j}\rangle_0 \oplus \mathbf{1}\{b_{n_p,j} = v\}$.
7. $\quad$ **end**
8. $\quad$ P_0 & P_1 invoke $\binom{M}{1}\text{-}\mathsf{OT}_{n_p}$ with P_0 as sender and inputs $\{e_{j,v}\}_{v\in[M]}$ and P_1 as receiver and input a_j. P_1 receives $e = \langle \mathsf{eq}_{0,1,j}\rangle_1\|\ldots\|\langle \mathsf{eq}_{0,n_p,j}\rangle_1$ as output, where $\langle \mathsf{eq}_{0,i,j}\rangle_1 \in \{0,1\}$ for all $i \in [n_p]$.
9. **end**
10. **for** $t = 1$ to $\log q$ **do**
11. $\quad$ **for** $j \in \{0,\ldots,q/2^t - 1\}$ **do**
12. $\quad\quad$ **for** $i \in [n_p]$ **do**
13. $\quad\quad\quad$ For $s \in \{0,1\}$, P_s invokes $\mathcal{F}_{\mathsf{AND}}$ with inputs $\langle \mathsf{eq}_{t-1,i,2j}\rangle_s$ and $\langle \mathsf{eq}_{t-1,i,2j+1}\rangle_s$ to learn output $\langle \mathsf{eq}_{t,i,j}\rangle_s$.
14. $\quad\quad$ **end**
15. $\quad$ **end**
16. **end**
17. For $s \in \{0,1\}$, P_s computes $y_s \leftarrow \bigoplus_{i\in[n_p]} \langle \mathsf{eq}_{\log q,i,0}\rangle_s$ and outputs y_s.

Fig. 6. Private Set Membership Protocol, $\mathsf{PSM1}$

Theorem 3. *Construction in Fig. 6 securely realizes Functionality* $\mathcal{F}_{\mathsf{PSM}}$ *(see Fig. 5) in the* $(\binom{M}{1}\text{-}\mathsf{OT}_d, \mathcal{F}_{\mathsf{AND}})$*-hybrid model.*

Proof. **Correctness.** We first prove that $\langle \mathsf{eq}_{\log q,i,0}\rangle_0, \langle \mathsf{eq}_{\log q,i,0}\rangle_1$ are correct boolean shares of $\mathbf{1}\{a = B(i)\}$ for all $i \in [n_p]$.

The proof is by induction on the level of the tree. By correctness of $\binom{M}{1}\text{-}\mathsf{OT}_{n_p}$ in line 8 of construction, it follows that $\langle \mathsf{eq}_{0,i,j}\rangle_1 \leftarrow \langle \mathsf{eq}_{0,i,j}\rangle_0 \oplus \mathbf{1}\{b_{i,j} = a_j\}$, for $j \in \{0,\ldots,q-1\}$ which proves the base case of induction. Let $q_t = q/2^t$. At the t^{th} level of tree, parse $B(i) = b^{(t)}_{i,q_t-1}\|\ldots\|b^{(t)}_{i,0}$ and parse $a = a^{(t)}_{q_t-1}\|\ldots\|a^{(t)}_0$. Let us assume that the correctness holds true for level t, i.e., $\mathsf{eq}_{t,i,j} = \langle \mathsf{eq}_{t,i,j}\rangle_1 \oplus \langle \mathsf{eq}_{t,i,j}\rangle_0 = \mathbf{1}\{b^{(t)}_{i,j} = a^{(t)}_j\}$ for $j \in \{0,\ldots,q_t - 1\}$. We prove that the same holds true for level $t+1$. By correctness of $\mathcal{F}_{\mathsf{AND}}$, for $j \in \{0,\ldots,q_{t+1}-1\}$, $\langle \mathsf{eq}_{t+1,i,j}\rangle_0 \oplus \langle \mathsf{eq}_{t+1,i,j}\rangle_1 = \mathsf{eq}_{t,i,2j} \wedge \mathsf{eq}_{t,i,2j+1} = \mathbf{1}\{b^t_{i,2j} = a^t_{2j}\} \wedge \mathbf{1}\{b^t_{i,2j+1} = a^t_{2j+1}\} = \mathbf{1}\{b^{t+1}_{i,j} = a^{t+1}_j\}$. Hence, for all $i \in [n_p]$, it holds that $\langle \mathsf{eq}_{\log q,i,0}\rangle_0 \oplus \langle \mathsf{eq}_{\log q,i,0}\rangle_1 = \mathbf{1}\{B(i) = a\}$. Now, $y_0 \oplus y_1 = \bigoplus_{i\in[n_p]} \mathbf{1}\{B(i) = a\}$. If $a \notin B$, $y_0 \oplus y_1 = 0$ else $y_0 \oplus y_1 = 1$, since B has distinct elements.

Security. To simulate the view of corrupt P_0, the simulator sends random bits as outputs from $\mathcal{F}_{\mathsf{AND}}$ functionality under the constraint that when plugged into the protocol, they result in the final share received from $\mathcal{F}_{\mathsf{PSM}}$. To simulate view of corrupt P_1, the simulator sends random message $\in \{0,1\}^{n_p}$ as output from $\binom{M}{1}\text{-}\mathsf{OT}_{n_p}$ instances. This is correct since each $\langle \mathsf{eq}_{0,i,j}\rangle_0$ is random. Outputs of $\mathcal{F}_{\mathsf{AND}}$ are simulated similarly.

General Case. The case when m does not divide ℓ and $q = \lceil \ell/m \rceil$ is not a power of 2 can be handled similar to the Millionaires's problem in [47]. For completeness, the main idea is as follows. Let $r = \ell \bmod m$ then $a_{q-1} \in \{0,1\}^r$. For the last leaf, we invoke $\binom{2^r}{1}\text{-}\mathsf{OT}_{n_p}$. Finally, when q is not a power of 2, we construct maximal possible perfect binary trees and connect the roots of these trees using Equation (1). The final tree has $n_p(q-1)$ calls to $\mathcal{F}_{\mathsf{AND}}$ (in both special/general cases).

Inputs of P_0**.** Input set B of size n_p, where $\forall i \in [n_p]$, $B(i) \in \{0,1\}^\ell$.
Inputs of P_1**.** Element $a \in \{0,1\}^\ell$.
Parameters. $\mathcal{F}_{\mathsf{OPPRF}}$ instantiated with table-based OPPRF construction (see Fig. 10) and $\mathcal{F}_{\mathsf{eq}}$ instantiated with construction from Fig. 6 (with $n_p = 1$).

1. P_0 samples a random target value t and prepares a set T such that it has n_p, elements all equal to t.
2. P_0 and P_1 invoke $\mathcal{F}_{\mathsf{OPPRF}}$ in which P_0 plays the role of sender with input set B and target multi-set T and P_1 plays the role of receiver with a as the input query. P_0 receives key $k \in \{0,1\}^\lambda$ and P_1 receives hint $\mathsf{hint} \in \{0,1\}^{u\ell}$ and PPRF output $w \in \{0,1\}^\ell$.
3. P_0 and P_1 call $\mathcal{F}_{\mathsf{eq}}$ with inputs t and w and receive bits y_0 and y_1 respectively.

Fig. 7. Private Set Membership Protocol, $\mathsf{PSM2}$

Concrete Complexity. In the special case, we invoke $q = \ell/m$ instances of $\binom{M}{1}$-OT_{n_p} and $n_p(q-1)$ instances of $\mathcal{F}_{\mathsf{AND}}$. Hence, total communication is $q(2\lambda + Mn_p) + n_p(q-1)(\lambda+16)$. For the general case, let $q = \lceil \ell/m \rceil$. Then, we use $q-1$ instances of $\binom{M}{1}$-OT_{n_p} and 1 instance of $\binom{2^r}{1}$-OT_{n_p} to compute the leaves where $r = \ell \mod m$. Then, we make $n_p(q-1)$ invocations of $\mathcal{F}_{\mathsf{AND}}$ in $\log q$ rounds. Total communication is $(q-1)(2\lambda + Mn_p) + (2\lambda + 2^r n_p) + n_p(q-1)(\lambda+16)$. E.g., for $\ell = 64, \lambda = 128, m = 4$, and $n_p = 3$ communication required is 11344 bits.

	Communication	Concrete Example
CO [9]	$2\ell n_p\lambda + n_p\lambda$	49536
ABY [12]	$2n_p\lambda(\ell-1)$	48384
$\mathsf{PSM1}$ $(m=4)$	$\ell n_p\lambda/4 + \ell\lambda/2 + 8\ell n_p$	11344
$\mathsf{PSM2}$ $(m=4)$	$(8+u)\ell + 3\ell\lambda/4 + 9\lambda/2$	7488

Table 1. Communication Complexity for Private Set Membership Protocols in bits. ℓ denotes the bit-length of elements, n_p denotes the number of elements and $u = 2^{\lceil \log(n_p+1) \rceil}$. For the concrete example, we consider: $\ell = 64$, $n_p = 3$, $\lambda = 128$.

4.2 Private Set Membership Protocol 2

Consider the $\mathcal{F}_{\mathsf{OPPRF}}$ functionality [32] that is obtained by setting $d = 1$ and $\beta = 1$ in $\mathcal{F}_{\mathsf{RB\text{-}OPPRF}}$. First at a high level, observe that $\mathcal{F}_{\mathsf{PSM}}$, must compute n_p equality checks, while $\mathcal{F}_{\mathsf{OPPRF}}$ is a functionality that allows reducing multiple equality checks to a single check. Hence, similar to the construction of [32], one can realize $\mathcal{F}_{\mathsf{PSM}}$ by first invoking $\mathcal{F}_{\mathsf{OPPRF}}$ (to reduce the n_p equality checks to a single equality check) and then computing the single check using a call to $\mathcal{F}_{\mathsf{eq}}$ [12,10,28] (the functionality obtained in $\mathcal{F}_{\mathsf{PSM}}$ by setting $n_p = 1$). Kolesnikov et al. [32] proposed three OPPRF constructions, viz., polynomial based OPPRF, garbled bloom filter [14] based OPPRF and table-based OPPRF constructions. For small set sizes, it was experimentally confirmed in [32] that table-based OPPRF construction is the fastest. Since the set size is 3 in $\mathcal{F}_{\mathsf{PSM}}$ when used in our final circuit-PSI construction, $\mathcal{F}_{\mathsf{OPPRF}}$ can most efficiently be realized by using this table-based construction. $\mathcal{F}_{\mathsf{eq}}$ can be realized using our protocol from Fig. 6 with $n_p = 1$. This protocol is $\approx 2\times$ more communication efficient than the state-of-the-art protocol [10] for equality testing. It is then easy to see that the final protocol $\mathsf{PSM2}$ obtained realizes $\mathcal{F}_{\mathsf{PSM}}$ (this follows trivially from the fact that the underlying protocols realize $\mathcal{F}_{\mathsf{OPPRF}}$ and $\mathcal{F}_{\mathsf{eq}}$ respectively). For completeness, we describe this protocol in Fig. 7 (and the table-based OPPRF [32] in Fig. 10 of Appendix A).

Concrete Complexity. A single call to $\mathsf{PSM2}$ involves interaction between the two parties in call to OPRF functionality realized using the construction given in [31], followed by hint transmission and equality check. The amortized communication cost for a single instance of OPRF evaluation using this protocol is 3.5λ bits [31, Table 1]. The hint size is $\lambda + u\ell$, where $u = 2^{\lceil \log n_p + 1 \rceil}$. Finally, we make a call to $\mathsf{PSM1}$ (with $n_p = 1$, see Fig. 6) and from its concrete communication analysis we get, $(q-1)(2\lambda + M) + (2\lambda + 2^r) + (q-1)(\lambda+16)$ to be the communication cost incurred in this call. Now, for simplicity let us set $m = 4$ and consider the special case in analysis of communication cost of $\mathsf{PSM1}$ construction.

Thus, the total communication cost is $(8+u)\ell+0.75\ell\lambda+4.5\lambda$. As an example, for $\ell=64, \lambda=128, m=4$ and $n_p=3$ communication required is 7488 bits.

4.3 Comparison of PSM Protocols

In this section, we discuss the communication complexity of our protocols, $\mathsf{PSM1}$ and $\mathsf{PSM2}$. We compare with the state-of-the-art protocols [9,12] and refer the reader to [9] for more details on prior PSM protocols. Table 1 illustrates the communication complexity of the PSM schemes[1]. In our setting, we will invoke PSM protocols with a set size $n_p=3$. The table also provides a comparison of communication costs for this setting – observe that our protocols $\mathsf{PSM1}$ and $\mathsf{PSM2}$ are $4.2\times$ and $\approx 6.5\times$ communication efficient than prior works respectively. From the communication complexity of the protocols given in the table, note that the performance gains of our protocols over existing protocols improve with increase in n_p. Finally, we remark that the computation overhead incurred in $\mathsf{PSM1}$, CO [9] and ABY [12] are nearly the same, while the computation cost of $\mathsf{PSM2}$ is higher due to its use of the OPPRF scheme.

5 Linear Circuit-PSI

In circuit PSI [25,42,44,16,43,27], the task is to compute a function f on the intersection of two sets, S_0 and S_1. Formally, the circuit-PSI functionality, $\mathcal{F}_{\mathsf{PSI},f}$, takes S_0 from P_0 and S_1 from P_1 and outputs $f(\mathsf{S}_0 \cap \mathsf{S}_1)$ to both parties. Similar to prior works, we consider symmetric functions whose output does not depend on the order of elements in the intersection.

We consider a standard two party computation functionality $\mathcal{F}_{\mathsf{2PC}}$ that is parameterized by a circuit C. It takes as input I_0 and I_1 from parties P_0 and P_1 respectively. The functionality then computes the circuit C on the inputs of the parties and returns $C(I_0, I_1)$ as output to the parties. In our construction, to evaluate a symmetric function f, we consider the following circuit: $C_{\beta,\mu}$ takes as input $a_1^{(0)}, \ldots, a_\beta^{(0)}$ and $z_1, \ldots, z_\beta$ from P_0 and $a_1^{(1)}, \ldots, a_\beta^{(1)}$ from P_1, where $a_j^{(0)}, a_j^{(1)} \in \{0,1\}$ and $z_j \in \{0,1\}^\mu$ for all $j \in [\beta]$ and $\beta = O(n)$, where n is the set size of both parties. The circuit first computes $a_j = a_j^{(0)} \oplus a_j^{(1)}$, for all $j \in [\beta]$. It then computes $f(Z)$ where $Z = \{z_j \mid a_j = 1\}_{j \in [\beta]}$.

Below, in Section 5.1 we describe our construction in the stashless setting of cuckoo hashing using the parameter settings based on the empirical analysis from [45] (see Section 2.2). We first discuss a circuit-PSI protocol for functions that take only the elements in the intersection as input. This protocol can be trivially extended to the case where the function takes as input both the elements as well as their associated payloads as long as only one party has a payload associated with its elements (e.g. set intersection sum). Later, we describe the protocol for the case where the function operates on payloads provided by both parties. Finally, we describe in Section 5.2 how our ideas easily extend to the setting with stash by building on the dual execution technique from [43]. We emphasize that all our constructions (with and without stash) have linear communication and linear computation complexity.

5.1 Circuit-PSI via Stashless Hashing

Construction Overview. Our construction follows a similar high-level blueprint as [43]. The parties start by hashing their input sets as follows: Consider three universal hash functions $h_1, h_2, h_3 : \{0,1\}^\mu \to [\beta]$, where $\beta = (1+\varepsilon)n$ and n is size of input sets. Now, P_0 does Cuckoo hashing using h_1, h_2, h_3 of input set S_0 into hash table HT_0. The ε, used in setting the number of bins β, is picked using parameters calculated in [45] for the stashless setting. On the other side, P_1 does simple hashing of S_1 into HT_1 using the same hash functions h_1, h_2, h_3 (where every bin in HT_1 can have multiple elements). Now, the following property holds: Consider $z \in \mathsf{S}_0 \cap \mathsf{S}_1$, such that $\mathsf{HT}_0[j] = z$, then $z \in \mathsf{HT}_1[j]$ as well.

[1] Though the communication complexity of our protocols is minimized for $m=7$, the computational complexity grows super-polynomiall with m. Similar to [47], we observe best performance for lower values of m ($m=4,5$).

Parameters. Functionalities Relaxed batch OPPRF, $(\ell, \beta, 3)$-$\mathcal{F}_{\mathsf{RB\text{-}OPPRF}}$ (Fig. 2), Private Set Membership, $\mathcal{F}_{\mathsf{PSM}}$ (Fig. 5), Two-party computation, $\mathcal{F}_{\mathsf{2PC}}$.
Inputs of P_0. Input set S_0 of size n, where $\mathsf{S}_0(i) \in \{0,1\}^\mu$, for all $i \in [n]$.
Inputs of P_1. Input set S_1 of size n, where $\mathsf{S}_1(i) \in \{0,1\}^\mu$, for all $i \in [n]$.

Hashing: Parties agree on universal hash functions $h_1, h_2, h_3 : \{0,1\}^\mu \to [\beta]$, where $\beta = (1+\varepsilon)n$, which are used as follows:

1. P_0 does Cuckoo hashing using h_1, h_2, h_3, to map elements in S_0 to hash table HT_0 with β bins. Since $\beta > n$, P_0 fills the empty bins in HT_0 with a uniformly random value.
2. P_1 does Simple hashing using h_1, h_2, h_3 to map elements in S_1 to hash table HT_1 with β bins.

Computing Relaxed Batch OPPRF:

3. P_0 creates queries $x_1, \ldots, x_\beta$ such that $x_j = (\mathsf{HT}_0[j]||j)$ for all $j \in [\beta]$.
4. P_1 creates input sets $X_1, \ldots, X_\beta$ and target sets $T_1, \ldots, T_\beta$ as follows: For all $j \in [\beta]$, $X_j = \{(y\|j) \mid y \in \mathsf{HT}_1[j]\}$, sample random and independent target value t_j and T_j has $|X_j|$ elements all equal to t_j.
5. P_0 & P_1 invoke $(\ell, \beta, 3)$-$\mathcal{F}_{\mathsf{RB\text{-}OPPRF}}$ with P_1 as sender with input sets $\{X_j\}_{j\in[\beta]}$ and target sets $\{T_j\}_{j\in[\beta]}$ and P_0 as receiver with queries $\{x_j\}_{j\in[\beta]}$. P_1 gets keys set K and P_0 gets output lists $W_1, \ldots, W_\beta$.

Comparing the $\mathsf{RB\text{-}OPPRF}$ Outputs and Target Values:

6. For each j in $[\beta]$, P_0 and P_1 invoke $\mathcal{F}_{\mathsf{PSM}}$ with inputs W_j and t_j, resp. P_0 and P_1 get as output $\langle a_j\rangle_0, \langle a_j\rangle_1 \in \{0,1\}$, respectively.

Computing the Circuit $C_{\beta,\mu}$:

7. The parties call $\mathcal{F}_{\mathsf{2PC}}$ parameterized by circuit $C_{\beta,\mu}$ with inputs $\langle a_1\rangle_0, \ldots, \langle a_\beta\rangle_0$ and HT_0 from P_0 and $\langle a_1\rangle_1, \ldots, \langle a_\beta\rangle_1$ from P_1. Both parties receive output y.

Fig. 8. Circuit-PSI Protocol Π_{PSI}

Hence, it suffices to compare the elements in HT_0 and HT_1 per bin. For this step, [43] used their batch OPPRF construction. In this work, we use the computationally more efficient $\mathsf{RB\text{-}OPPRF}$. For this, P_0 plays the role of the receiver and the queries are $\{x_j\}_{j\in[\beta]}$ such that $x_j = \mathsf{HT}_0[j]||j$. P_1 plays the role of the sender and constructs input sets $\{X_j\}_{j\in[\beta]}$ as $X_j = \{(y\|j) \mid y \in \mathsf{HT}_1[j]\}$. Next, for $j \in [\beta]$, P_1 samples $t_j \in \{0,1\}^\ell$ independently and uniformly, and constructs T_j with $|X_j|$ elements, all equal to t_j. From the (ℓ, β, d)-$\mathcal{F}_{\mathsf{RB\text{-}OPPRF}}$ functionality, P_0 receives lists $\{W_j\}_{j\in[\beta]}$. At a high level, we argue below that $t_j \in W_j$ if and only if $x_j \in \mathsf{S}_1$, where S_1 is the input set of P_1. To check this set membership, i.e., whether t_j lies in W_j, parties invoke instances of $\mathcal{F}_{\mathsf{PSM}}$ and learn boolean shares of membership. These shares along with HT_0 are finally sent to $\mathcal{F}_{\mathsf{2PC}}$ that computes the circuit $C_{\beta,\mu}$, i.e., reconstructs these shares, picks elements from HT_0 corresponding to shares of 1 and computes f on them. We describe the construction formally in Fig. 8.

Instantiating the Protocol in Fig. 8. We can realize the $(\ell, \beta, 3)$-$\mathsf{RB\text{-}OPPRF}$ functionality using our scheme in Section 3.2. Note that our scheme uses Cuckoo hashing and here again, we pick our parameters that ensure no stash. We set $\ell = \sigma + \log(3\beta)$ required by the correctness proof below, to bound the probability of false positives by $2^{-\sigma}$. Later, we use $\sigma = 40$. The functionality $\mathcal{F}_{\mathsf{PSM}}$ can be realized either using $\mathsf{PSM1}$ (see Section 4.1) or $\mathsf{PSM2}$ (see Section 4.2). This gives us two protocols for circuit-PSI, that we call $\mathsf{C\text{-}PSI}_1$ and $\mathsf{C\text{-}PSI}_2$ respectively. These protocols have a similar communication vs compute trade-off as discussed in Section 4.3 and we compare them empirically in Section 6.

Theorem 4. *Construction in Fig. 8 securely realizes $\mathcal{F}_{\mathsf{PSI},f}$ functionality with $\mathcal{O}(n)$ communication and computational complexity.*

Correctness. By correctness of hashing and use of same hash functions by both P_0 and P_1, it holds that for any element $z \in \mathsf{S}_0 \cap \mathsf{S}_1$, there exists a unique $j \in [\beta]$ such that $\mathsf{HT}_0[j] = z$ and $z \in \mathsf{HT}_1[j]$. Hence, it suffices to compare $\mathsf{HT}_0[j]$ with all elements in $\mathsf{HT}_1[j]$, for all $j \in [\beta]$. So consider a bin $j \in [\beta]$. If $\mathsf{HT}_0[j] \in \mathsf{HT}_1[j]$, then $x_j \in X_j$ in our construction. By correctness of $\mathcal{F}_{\mathsf{RB\text{-}OPPRF}}$, if $x_j \in X_j$, then $t_j \in W_j$. Moreover, if $\mathsf{HT}_0[j] \notin \mathsf{HT}_1[j]$, then $x_j \notin X_j$. From the correctness property for non-programmed points property (Section 3.2), it holds that if $x_j \notin X_j$, then $t_j \in W_j$ with probability at most $3 \cdot 2^{-\ell}$. Taking a union bound over β bins, total probability of false positives is upper bound by $\mathsf{fail} = 3\beta \cdot 2^{-\ell}$.

We pick $\ell > \log(3\beta) + \sigma$ such that $\mathsf{fail} < 2^{-\sigma}$. Next, by correctness of $\mathcal{F}_{\mathsf{PSM}}$, $\langle a_j \rangle_0 \oplus \langle a_j \rangle_1 = 1$ if and only if $t_j \in W_j$. Finally, correctness of final output y follows from correctness of $\mathcal{F}_{\mathsf{2PC}}$.

Security. Security of the protocol follows immediately from security of $\mathcal{F}_{\mathsf{RB\text{-}OPPRF}}, \mathcal{F}_{\mathsf{PSM}}, \mathcal{F}_{\mathsf{2PC}}$ functionalities.

Communication and Computational Complexity of $\mathsf{C\text{-}PSI}_1/\mathsf{C\text{-}PSI}_2$. We argue that the protocol has linear complexity, i.e., $\mathcal{O}(n)$ in both communication and compute ignoring the complexity of function f being computed on $\mathsf{S}_0 \cap \mathsf{S}_1$. This follows immediately from the following: 1) Our construction of $(\ell, \beta, 3)$-RB-OPPRF in Section 3.2 has linear complexity (see Theorem 2). 2) Since each of W_j has a constant number of elements, i.e., $d = 3$, and protocol PSM1/PSM2 is invoked independently for each j, step 6 has linear complexity. 3) Inputs to $C_{\beta,\mu}$ are linear in size; hence computation until step of computing $\mathsf{S}_0 \cap \mathsf{S}_1$ has linear complexity.

PSI With Associated Payload. The above protocol can be trivially extended to the case when P_0 alone has a payload associated with its elements; this is done by simply appending the payload to Z. IWe now discuss how we can adapt the protocol in Fig. 8 (similar to [43]) to handle the case when elements of both P_0 and P_1 have associated payloads, and we wish to compute a function of both the elements as well as payloads in the intersection.

Let $U(x) \in \{0,1\}^\delta$ and $V(y) \in \{0,1\}^\delta$ respectively denote the payloads associated with element $x \in \mathsf{S}_0$ and $y \in \mathsf{S}_1$. We significantly build on our protocol for circuit-PSI in Fig. 8. At a high level, parties engage in one more instance of RB-OPPRF (consistent with the previous one) such that for elements in the intersection, they hold shares of payload of P_1 in one of 3 locations.

The protocol is as follows: Parties P_0 and P_1 on respective input sets S_0 and S_1 execute steps 1 to 5 of Π_{PSI} protocol (see Fig. 8). After the execution of the above steps, P_0 has hash table HT_0, query elements $x_1, \ldots, x_\beta$ and output lists $W_1, \ldots, W_\beta$. P_1 has hash table HT_1, input sets $X_1, \ldots, X_\beta$, target values $t_1, \ldots, t_\beta$. Note that step 5 in Fig. 8 uses an $(\ell, \beta, 3)$-RB-OPPRF protocol instantiated with our $(\ell, \beta, 3)$-RB-PPRF protocol given in Fig. 3. Let h_1', h_2' and h_3' be the set of universal hash functions used in $(\ell, \beta, 3)$-RB-PPRF protocol. In step 5 of Fig. 3, P_1 acts as sender and within RB-PPRF uses these functions to hash elements in input sets using cuckoo hashing. HT (step 1, Fig. 3) denotes this hash table.

Now, P_1 samples random and independent target values $\widetilde{t}_1, \ldots, \widetilde{t}_\beta$ and prepares target sets $\widetilde{T}_j$ of size $|X_j|$, for all $j \in [\beta]$ as follows: Set $\widetilde{T}_j(i) \leftarrow \widetilde{t}_j \oplus V(\mathsf{HT}_1[j](i))$, for all $i \in |X_j|$. P_0 and P_1 then invoke another instance of $(\delta, \beta, 3)$-RB-OPPRF protocol with P_1 as the sender and input sets $\{X_j\}_{j \in [\beta]}$ and target sets $\{\widetilde{T}_j\}_{j \in [\beta]}$ such that it uses the same hash table HT inside RB-PPRF as was used before. Party P_0 acts as receiver with input queries $\{x_j\}_{j \in [\beta]}$. Let $\widetilde{W}_1, \ldots, \widetilde{W}_\beta$ be the output lists received by P_0 from execution of $(\delta, \beta, 3)$-RB-OPPRF protocol.

Let P_0 define Q of length β as follows: For all $j \in [\beta]$ if $z = \mathsf{HT}_0[j] \in \mathsf{S}_0$, then $Q[j] = U(z)$, else some dummy value. Finally, the parties invoke $\mathcal{F}_{\mathsf{2PC}}$ with circuit C_{PL} that is described as follows. The circuit C_{PL} takes as input HT_0, Q, $\{W_j\}_{j \in [\beta]}$ and $\{\widetilde{W}_j\}_{j \in [\beta]}$ from P_0 and $\{t_j\}_{j \in [\beta]}$ and $\{\widetilde{t}_j\}_{j \in [\beta]}$ from P_1. For $j \in [\beta]$ and $i \in [3]$, the circuit C_{PL} sets $b_{j,i} = 1$ if $t_j = W_j[i]$ and 0 otherwise. If $b_{j,i} = 1$ then C_{PL} forwards $\mathsf{HT}_0[j]$, P_0's payload $Q(j)$ and $\widetilde{t}_j \oplus \widetilde{W}_j[i]$ to an internal sub-circuit that computes f. From Theorem 4, it follows that $\exists i_j \in [3]$ such that $b_{j,i_j} = 1$ iff $x_j \in X_j$[2] and in the second $(\delta, \beta, 3)$-RB-OPPRF instance, since P_1 makes use of the same hash table HT in hint computation, it follows that $\widetilde{W}_j[i_j] = \widetilde{t}_j \oplus V(\mathsf{HT}_0[j])$, for all $j \in [\beta]$. Hence, $\widetilde{t}_j \oplus \widetilde{W}_j[i_j] = V(\mathsf{HT}_0[j])$ which corresponds to the payload of P_1 associated with element $\mathsf{HT}_0[j]$. The security follows from the security of RB-OPPRF protocol.

[2] To show that exactly one such i_j exists, we can use a correctness property similar to that used for non-programmed points.

5.2 Circuit-PSI via Dual Execution

We describe our linear complexity protocol for the scenario when Cuckoo hashing results in a stash[3]. Our idea is inspired by the dual execution idea of Pinkas *et al.* [43]. It uses the protocol from [43] in the "unbalanced set-size" setting - i.e., when P_0 and P_1 have unequal set sizes. We observe that their protocol can be made to have linear (in the larger set) computation cost, while being super-linear in the smaller set. We make use of this protocol in a setting where one party, say P_0 has a small set of size $\mathcal{O}(\log n)$ and other party, P_1, has a set of size n.

Theorem 5 ([43]). *Consider parties Q_0 and Q_1 with input sets S_0 and S_1 of size n_0 and n_1, respectively such that $n_1 \leqslant n_0$. Then, there exists a circuit-PSI protocol ([43, Protocol 9]) with computational complexity $\mathcal{O}(n_1 \log n_1 + n_0 + sn_1)$, where s is the stash size in cuckoo hashing of n_0 elements, and can be set to $\mathcal{O}(\log n_0 / \log\log n_0)$, for negligible failure probability. Communication of the protocol is $\mathcal{O}(n_0 + sn_1)$.*

Corollary 1 ([43]). *For circuit-PSI between sets of sizes $n_0 = n$ and $n_1 = \mathcal{O}(\log n / \log\log n)$, there exists a protocol with complexity $\mathcal{O}(n)$.*

As discussed in Section 2.2, for Cuckoo hashing with 3 hash functions, it holds that failure probability is negligible in n for stash size $s = \mathcal{O}(\log n / \log\log n)$. In our construction in Fig. 8, we use Cuckoo hashing twice. First, P_0 uses Cuckoo hashing to map its elements in S_0 into HT_0 and this can result in a stash. Denote the elements that fit in main table of HT_0 as $\mathsf{S}_{0,T}$ and elements in stash by $\mathsf{S}_{0,S}$ s.t. $|\mathsf{S}_{0,S}| = \mathcal{O}(\log n / \log\log n)$. Our RB-OPPRF construction can lead to a stash at P_1 as follows: Earlier, in Fig. 8, P_1 does simple hashing on elements in S_1 using h_1, h_2, h_3. Hence, each element in S_1 occurs at most thrice in HT_1 and also in sets $X_1, \ldots, X_\beta$, concatenated with different bin number. Now, P_1 acts as the sender in RB-OPPRF, and hashes the $n' = 3n$ elements in $X_1, \ldots, X_\beta$ using Cuckoo hashing that can lead to a stash. Denote the elements that fit in main table as $\mathsf{S}'_{1,T}$ and stash elements by $\mathsf{S}'_{1,S}$. Let $\mathsf{S}_{1,S}$ contain elements $y \in \mathsf{S}_1$, s.t. there exists $j \in [\beta]$ with $(y \| j) \in \mathsf{S}'_{1,S}$. Now, $|\mathsf{S}_{1,S}| = \mathcal{O}(\log n / \log\log n)$. Let $\mathsf{S}_{1,T} = \mathsf{S}_1 \setminus \mathsf{S}_{1,S}$.

Now we run 4 instances of circuit-PSI (in parallel) as follows: (1) Use our protocol described in Fig. 8 between elements in $\mathsf{S}_{0,T}$ and $\mathsf{S}_{1,T}$. (By the above construction it is guaranteed that there would be no stash when invoking this protocol.) (2) Use protocol given by Corollary 1 between elements in $\mathsf{S}_{0,S}$ and $\mathsf{S}_{1,T}$, where P_0 plays the role of Q_1 and P_1 plays the role of Q_0. (3) We do a role reversal, and run protocol from Corollary 1 between elements $\mathsf{S}_{0,T}$ and $\mathsf{S}_{1,S}$ where P_0 plays role of Q_0 and P_1 plays role of Q_1. (4) Run a protocol to do exhaustive comparison between $\mathsf{S}_{0,S}$ and $\mathsf{S}_{1,S}$. This protocol has complexity, $|\mathsf{S}_{0,S}| \cdot |\mathsf{S}_{1,S}|$, (sub-linear in n).

6 Implementation and Evaluation

Our protocols are implemented[4] in C++ and we compare the performance of our Circuit-PSI protocols $\mathsf{C\text{-}PSI}_1$ and $\mathsf{C\text{-}PSI}_2$ with the state-of-the-art protocol [43], referred to as the PSTY. For a comparison of PSTY with other prior circuit-PSI schemes [25,42,44,16], we refer the reader to [43, Section 7]. We set computational security parameter, $\lambda = 128$, and the statistical security parameter, $\sigma = 40$.

Protocol Parameters. It was shown in PSTY that the Circuit-PSI protocol for the stashless setting was most performant. Hence, we consider the stashless setting, and compare our performance with the corresponding stashless protocol from PSTY. Similar to PSTY, we use $d = 3$ hash functions to hash n_h elements into $\beta = 1.27n_h$ (i.e. $\varepsilon = 0.27$) bins using the analysis from [45] (see Section 2.2). As in Section 5.1, we set output length ℓ of RB-OPPRF scheme as $\ell = \sigma + \lceil \log(3\beta) \rceil$, where $\beta = 1.27n$ and n

[3] Even though in all practical setting, if parameters are picked based on empirical analysis of [45] , no stash is observed.

[4] Code available at `https://aka.ms/2PC-Circuit-PSI`.

is input set size.

Implementation Details. The underlying OPRF in our $\mathsf{RB\text{-}OPPRF}$ and Table based OPPRF construction [32] (in $\mathsf{PSM2}$) is instantiated using the implementation [38] of Kolesnikov *et al.* [31]. In our $\mathsf{RB\text{-}OPPRF}$ construction (see Fig. 4), the output length of the underlying PRF in $\mathcal{F}_{\mathsf{OPRF}}$ is 3ℓ. For the input set sizes that we consider, the maximum output length is 192 bits. We configure the output length of the underlying PRF in the OPRF construction [31] to 128 bits and then use a PRG to expand this PRF output based on the input setting (upto 192 bits). PSTY protocol requires the output length of the underlying PRF in $\mathcal{F}_{\mathsf{OPRF}}$ to be atmost 64 bits. However, our circuit-PSI protocols do not incur additional communication cost over PSTY protocol in the OPRF phase because the communication cost of OPRF construction [31] is the same for the underlying PRF with output length 64 and 128 bits [31, Table 1]. For initial hashing as well as in the implementation of our $\mathsf{RB\text{-}OPPRF}$ construction, we make use of the hash tables library from [37]. For implementing our protocol $\mathsf{PSM1}$ and the protocol for $\mathcal{F}_{\mathsf{eq}}$ functionality in our $\mathsf{PSM2}$ scheme, we make use of the implementation of OT-Extension protocols [26,30] and Bit-Triple generation protocol [13,47] available at [36]. We compare with the implementation of PSTY scheme available at [15].

System Details. We ran our experiments in both the LAN and WAN network settings. In the LAN setting, we observed a network bandwidth of 375 MBps with an echo latency of 0.3 ms, while the corresponding numbers in the WAN setting were 34 MBps and 80 ms. The machines we used were commodity class hardware: 3.7 GHz Intel Xeon processors with 16GBs of RAM. To be fair in our comparison with PSTY (whose code is single threaded), we also restricted our code to execute in this setting. Similar to PSTY, our protocols can benefit from parallelization through multi-threading.

6.1 Concrete Communication Cost

In this section, we discuss the concrete communication cost of our circuit-PSI schemes. We summarize the communication cost of PSTY and compare them with our schemes in Table 2 for varying input set sizes n and for inputs of arbitrary bitlength. Similar to PSTY, and unlike circuit-PSI protocols proposed in [25,42,44,16,27], the communication cost of our protocols is independent of the bitlength of elements in the input sets and depends only on the size of the input sets. As can be observed from the table, our protocol $\mathsf{C\text{-}PSI}_2$ is $\approx 2.3\times$ more communication efficient than PSTY (while $\mathsf{C\text{-}PSI}_1$ is $\approx 1.5\times$ more efficient).

Next, we discuss the breakdown of communication. We provide a breakdown of our communication explicitly for both $\mathsf{C\text{-}PSI}_1$ and $\mathsf{C\text{-}PSI}_2$ protocols. The communication cost of OPRF phase and hint transmission of PSTY protocol is essentially the same as our circuit-PSI protocols (refer Table 2 and [43, Table 3]). The communication incurred in equality checks in PSTY protocol can, thus, be obtained by subtracting the communication cost of OPRF phase and hint transmission from the total communication. This gives a breakdown of communication cost of PSTY protocol. Also, since the communication incurred in OPRF and hint transmission constitute the communication of $\mathsf{B\text{-}OPPRF}$ and $\mathsf{RB\text{-}OPPRF}$ constructions, the communication cost of polynomial-based $\mathsf{B\text{-}OPPRF}$ construction of [43] and our $\mathsf{RB\text{-}OPPRF}$ construction is almost the same. Similar to PSTY scheme, the bulk of our communication is incurred in the final phase of our protocol, where we need to compare the outputs received from our $\mathsf{RB\text{-}OPPRF}$ scheme. For $\mathsf{C\text{-}PSI}_1$ and $\mathsf{C\text{-}PSI}_2$, PSM accounts for around 93% and 90% of the total communication cost respectively. In PSTY, circuit component accounts for 96% of the overall communication.

Finally, unlike PSTY and our protocols, the recent linear computation protocol of [27] has a communication cost that varies with bit-length. For $\ell = 32$ and 64, $\mathsf{C\text{-}PSI}_2$ is $\approx 8.8 - 12.7\times$ more communication efficient than their protocol; see [27, Table 3]. For instance, they communicate 836 MB for input sets of size 2^{16} and element bitlength of 64, while we communicate 65.4 MB.

n	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}
PSTY [43]	40.6	162	650	2600	10397
C-PSI$_1$ (Ours)	24.1	96.9	387	1661	6667
C-PSI$_2$ (Ours)	**16.8**	**65.4**	**261**	**1107**	**4435**
Breakdown					
OPRF	1.12	4.46	17.9	71.4	286
Hint transmission	0.48	2	7.6	30	122
PSM1	22.4	90.4	362.1	1560	6259
PSM2	15.2	58.9	235	1005	4027

Table 2. Communication in MB of circuit-PSI schemes for sets of size n and elements of arbitrary length. The best values are marked in bold. The first two costs, viz., OPRF and Hint Transmission are common to both our schemes. Total communication for scheme C-PSI$_i$ can be obtained by adding the communication of these components to the communication of the corresponding PSMi.

6.2 Performance Comparison

In Table 3, we compare the run-times of our circuit-PSI protocols C-PSI$_1$ and C-PSI$_2$ with PSTY for input set sizes upto 2^{22} elements[5] in both the LAN/WAN settings. Table 3 entries are median across 10 executions.

Network Setting	LAN			WAN		
n	PSTY [43]	C-PSI$_1$	C-PSI$_2$	PSTY [43]	C-PSI$_1$	C-PSI$_2$
2^{14}	1.32	**0.86**	1.27	4.92	**4.7**	6.5
2^{16}	3.80	**1.83**	2.1	9.14	**6.69**	8.07
2^{18}	13.87	6.03	**5.54**	25.19	15.08	**14.78**
2^{20}	54.91	23.4	**20.21**	90.03	49.1	**43.37**
2^{22}	220.86	93.03	**77.89**	353.75	184.33	**155.02**

Table 3. Comparison of total run-time in seconds of our Circuit-PSI schemes C-PSI$_1$ and C-PSI$_2$ to [43] for n elements. The best values are marked in bold.

End-to-end Execution Times. Overall, our protocols are up to 2.8× faster than PSTY and outperform PSTY in all network settings and set sizes (Table 3).

In both LAN and WAN settings, on small input sets (e.g. of size 2^{14} and 2^{16}), C-PSI$_1$ has the best overall run-time whereas for larger input sets of size $2^{18}, 2^{20}$ and 2^{22}, C-PSI$_2$ is the most performant due to its lower communication. Recall that, C-PSI$_2$ makes use of the computationally more expensive (due to the table-based OPPRF) PSM2 protocol for private set membership. Even though C-PSI$_2$ incurs lesser communication than C-PSI$_1$ in all cases, for input sets of size 2^{14} and 2^{16}, the respective communication difference of 7.1 MB and 31.5 MB is not significant enough to compensate for the additional computational cost introduced by the OPPRF construction even in the WAN setting. Hence, in these cases, C-PSI$_1$ out-performs C-PSI$_2$.

Breakdown of Individual Components. Next, we present a breakdown of the overall execution times in the PSTY and C-PSI$_2$ protocols (see Table 4). Since, C-PSI$_1$ and C-PSI$_2$ only differ at the usage of PSM protocol, the breakdown of run-time C-PSI$_2$ except for the PSM component is same for C-PSI$_1$. An important point to note is that the bulk of the cost in the LAN setting in PSTY comes from the hint creation cost that made use of an OPPRF protocol. For example, for a set size of 2^{22} this cost is 155.1 s and about 71% of the entire cost. In contrast, hint creation in C-PSI$_2$, through the use RB-OPPRF, for the same setting is about 8× faster – it executes in < 20 s, representing only 26% of the total cost. While using an RB-OPPRF protocol does increase the total number of comparisons by a factor of 3, since we have a more communication efficient protocol for PSM, the cost of this phase is only marginally more than the corresponding phase in PSTY, thus leading to an overall faster protocol.

[5] In PSTY implementation [15], polynomial interpolation is implemented in a prime field using Mersenne prime $2^{61}-1$. While Mersenne prime $2^{61}-1$ ensures statistical security of atleast 40-bits for input sets of size upto 2^{20}, it only provides statistical security of 38-bits for set size 2^{22}. In contrast, implementations of C-PSI$_1$ and C-PSI$_2$ provide statistical security of atleast 40-bits even for input sets of size 2^{22}.

LAN Setting						
Scheme	PSTY			C-PSI$_2$		
n	2^{16}	2^{20}	2^{22}	2^{16}	2^{20}	2^{22}
Hashing	0.07 (2%)	1.49 (3%)	6.51 (3%)	0.07 (4%)	1.5 (8%)	6.6 (9%)
OPRF	0.43 (12%)	2.01 (4%)	6.53 (3%)	0.51 (26%)	1.97 (11%)	6.51 (9%)
Hint Creation	2.28 (63%)	37.84 (70%)	155.1 (71%)	0.22 (11%)	5.34 (28%)	19.98 (26%)
Hint Transmission	0.02 (1%)	0.09 (0%)	0.36 (0%)	0.02 (1%)	0.11 (1%)	0.4 (0%)
Hint Evaluation	0.27 (8%)	4.46 (8%)	17.65 (8%)	0.04 (2%)	0.62 (3%)	2.49 (3%)
Circuit/PSM	0.51 (14%)	7.75 (15%)	30.51 (15%)	1.09 (56%)	9.27 (49%)	41.33 (53%)
Total	3.8	54.91	220.86	2.1	20.21	77.89
WAN Setting						
Scheme	PSTY			C-PSI$_2$		
n	2^{16}	2^{20}	2^{22}	2^{16}	2^{20}	2^{22}
Hashing	0.07 (1%)	1.49 (2%)	6.56 (2%)	0.07 (1%)	1.5 (4%)	6.67 (4%)
OPRF	1.37 (16%)	4.9 (6%)	16.3 (5%)	1.38 (20%)	4.9 (11%)	16.29 (11%)
Hint Creation	2.39 (29%)	38.06 (42%)	154.69 (45%)	0.23 (3%)	5.31 (12%)	20.11 (13%)
Hint Transmission	0.5 (6%)	1.59 (2%)	4.1 (1%)	0.42 (6%)	1.5 (4%)	3.96 (3%)
Hint Evaluation	0.28 (3%)	4.39 (5%)	17.78 (5%)	0.04 (1%)	0.63 (1%)	2.53 (2%)
Circuit/PSM	3.73 (45%)	38.13 (43%)	146.73 (42%)	4.81 (69%)	28.96 (68%)	104.46 (67%)
Total	9.14	90.03	353.75	8.07	43.37	155.02

Table 4. Breakdown of runtimes in seconds (s) of PSTY [43] and C-PSI$_2$ circuit-PSI schemes for set size n. Approximate percentage of the total run-time is provided in parenthesis for each component.

Finally, we note that one could construct a circuit-PSI protocol by using the original PSTY protocol but replacing their circuit protocol (based on the ABY protocol [12]) with the protocol realizing $\mathcal{F}_{\mathsf{eq}}$ in PSM2 protocol (see Fig. 7). Such a protocol would indeed be most frugal in terms of communication complexity (but not by much – only $\approx 1.2\times$ more communication efficient than C-PSI$_2$). This protocol would however still have a high concrete computational cost and perhaps more importantly would not have linear computational complexity. For the modified protocol to outperform our proposed constructions, based on experimental run-times, we estimate that for input sets of size 2^{20}, the network bandwidth has to be poorer than 5MBps. For this modified protocol, we ran experiments for input sets of size 2^{20} and observed that the run-time of this protocol is 54.52s (i.e., $2.7\times$ slower than C-PSI$_2$ protocol) in the LAN setting and 72s ($1.7\times$ slower than C-PSI$_2$ protocol) in the WAN setting.

Applications of Circuit-PSI. It was argued in [44,43] that the circuits for well-studied problems of PSI-CAT & threshold PSI [18,24,53,54,44,43,20,21,2] and PSI-Sum [33,44,43] (see Section 1.1 for problem descriptions) are only slightly larger than the circuit of circuit-PSI protocol. Hence, the overall runtimes for circuit-PSI reported in Table 3 is a good estimate of performance for these problems. Moreover, for all of these problems, as shown in [43], the protocols obtained using circuit-PSI are most performant and beats prior state-of-the-art by huge margins. Since we improve on both communication and computation (asymptotically as well as concretely) over [43], we improve the state-of-the-art for all these problems. To summarize, for all of these problems, our new protocols provide $> 2\times$ improvement in performance over the prior best [43].

7 Conclusion

We provide concretely efficient protocols for 2-party circuit-PSI with linear computational and communication complexity that are up to $2.8\times$ more performant than the state-of-the-art [43]. Both [43] and our protocols make use of IKNP style OT-Extension protocols [26,30,1,31], the concrete communication of which can be improved with the recent work on silent-OT extensions [5,51] as discussed in [5,16,48]. While the communication of these protocols is significantly lower, they are computationally more involved and hence their concrete performance depends on the network parameters. Based on empirical analysis, we expect the use of silent-OTs to improve the run-time of circuit-PSI protocols in our WAN setting (34 MBps) but not impact our LAN setting (375MBps). We leave silent-OT extension integration into our implementation for future work. Finally, our protocols are secure only against semi-honest adversaries, and we leave the exploration of building protocols that are secure against malicious adversaries who deviate arbitrarily from the protocol to future work. As also noted in [43], one of the main challenges in making our protocols maliciously secure is in ensuring that the hashing is performed correctly by the parties.

References

1. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *CCS*, 2013.
2. Saikrishna Badrinarayanan, Peihan Miao, Srinivasan Raghuraman, and Peter Rindal. Multi-party threshold private set intersection with sublinear communication. In *PKC*, 2021.
3. Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991.
4. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, 1988.
5. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO*, 2019.
6. Gilles Brassard, Claude Crépeau, and Jean-Marc Robert. All-or-nothing disclosure of secrets. In *CRYPTO*, 1986.
7. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.
8. Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In *CRYPTO*, 2020.
9. Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. In *SCN*, 2018.
10. Geoffroy Couteau. New protocols for secure equality test and comparison. In *ACNS*, 2018.
11. Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *ASIACRYPT*, 2010.
12. Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.
13. Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, Shaza Zeitouni, and Michael Zohner. Pushing the communication barrier in secure computation using lookup tables. In *NDSS*, 2017.
14. Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *CCS*, 2013.
15. Encrypto Group. OPPRF-PSI. `https://github.com/encryptogroup/OPPRF-PSI`. Accessed: 2020-10-07.
16. Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. Private set intersection with linear communication from general assumptions. In *ACM WPES@CCS*, 2019.
17. Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, 2005.
18. Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, 2004.
19. Juan A. Garay, Berry Schoenmakers, and José Villegas. Practical and secure solutions for integer comparison. In *PKC*, 2007.
20. Satrajit Ghosh and Tobias Nilges. An algebraic approach to maliciously secure private set intersection. In *EUROCRYPT*, 2019.
21. Satrajit Ghosh and Mark Simkin. The communication complexity of threshold private set intersection. In *CRYPTO*, 2019.
22. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, 1984.
23. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, 1987.
24. Per A. Hallgren, Claudio Orlandi, and Andrei Sabelfeld. Privatepool: Privacy-preserving ridesharing. In *CSF*, 2017.
25. Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.
26. Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *CRYPTO*, 2003.
27. Ferhat Karakoç and Alptekin Küpçü. Linear complexity private set intersection for secure two-party protocols. In *CANS*, 2020.
28. Ferhat Karakoç, Majid Nateghizad, and Zekeriya Erkin. SET-OT: A secure equality testing protocol based on oblivious transfer. In *ARES*, 2019.

29. Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. *SIAM J. Comput.*, 39(4), 2009.
30. Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In *CRYPTO*, 2013.
31. Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *CCS*, 2016.
32. Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *CCS*, 2017.
33. B. Kreuter. Secure multiparty computation at google. In *RWC*, 2017.
34. Yehuda Lindell. How to simulate it - a tutorial on the simulation proof technique. Cryptology ePrint Archive, Report 2016/046, 2016.
35. Catherine A. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *IEEE S & P*, 1986.
36. mpc-msri. EzPC. `https://github.com/mpc-msri/EzPC`. Accessed: 2020-10-07.
37. Oleksandr-Tkachenko. HashingTables. `https://github.com/Oleksandr-Tkachenko/HashingTables`. Accessed: 2020-10-07.
38. osu-crypto. libOTe. `https://github.com/osu-crypto/libOTe`. Accessed: 2020-10-07.
39. Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In *Algorithms - ESA*, 2001.
40. Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: Lightweight private set intersection from sparse OT extension. In *CRYPTO*, 2019.
41. Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from paxos: Fast, malicious private set intersection. In *EUROCRYPT*, 2020.
42. Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security*, 2015.
43. Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In *EUROCRYPT*, 2019.
44. Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In *EUROCRYPT*, 2018.
45. Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.*, 21(2), 2018.
46. Michael O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 2005.
47. Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In *CCS*, 2020.
48. Peter Rindal and Phillipp Schoppmann. VOLE-PSI: fast OPRF and circuit-psi from vector-ole. In *EUROCRYPT*, 2021.
49. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11), 1979.
50. Adi Shamir. On the power of commutativity in cryptography. In *ICALP*, 1980.
51. Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In *CCS*, 2020.
52. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, 1986.
53. Yongjun Zhao and Sherman S. M. Chow. Are you the one to share? secret transfer with access structure. *PoPETs*, 2017(1):149–169, 2017.
54. Yongjun Zhao and Sherman S. M. Chow. Can you find the one for me? In *WPES@CCS*, 2018.

A Prior B-PPRF and OPPRF Constructions

Pinkas et al. [43] proposed a polynomial based Batch Programmable Pseudorandom Function (B-PPRF) construction that builts on the polynomial based PPRF construction of [32] by combining hints corresponding to individual input sets in order to obtain a single hint. This is achieved by interpolating a single polynomial for all the elements across the input sets. We present this polynomial based B-PPRF construction in Fig. 9.

Fig. 10 describes the table-based OPPRF construction of [32]. The sender prepares a hash table by hashing elements in input set X using random oracle $\mathcal{O} : \{0,1\}^\ell \to \{0,1\}^u$, where $u = 2^{\lceil \log(|X|+1) \rceil}$. In step 2, the sender samples a nonce ν until all elements in X hash to distinct positions in hash table HT. The expected number of times that the nonce ν has to be sampled is $1/\mathrm{Pr}_{\mathsf{unique}}$, where $\mathrm{Pr}_{\mathsf{unique}}$ is

Parameters. A PRF $G : \{0,1\}^\lambda \times \{0,1\}^\ell \to \{0,1\}^\ell$.

$\underline{Hint(k, X, T)}$. Given the keys $k = k_0, \ldots, k_{\beta-1}$, the sets $X = X_0, \ldots, X_{\beta-1}$ and target multi-sets $T = T_0, \ldots, T_{\beta-1}$, interpolate the polynomial p using points $\{X_j(i), G(k_j, X_j(i)) \oplus T_j(i)\}_{j\in[\beta], i\in[|X_j|]}$. Return p as the hint.

$\underline{F(k_i, \mathsf{hint}, x)}$. Interpolate hint as polynomial p. Return $G(k_i, x) \oplus p(x)$.

Fig. 9. Polynomial-based B-PPRF Construction [43].

Sender's Inputs. Set X where $X(i) \in \{0,1\}^\ell$ for all $i \in [|X|]$ and set T sampled from $\mathcal{T}$ (recall from Section 3 that $\mathcal{T}$ is a distribution of multi-sets whose each element is uniformly random but the elements can be correlated) such that $|X| = |T|$ and $T(i) \in \{0,1\}^\ell$ for all $i \in [|T|]$.
Receiver's Inputs. The query $x \in \{0,1\}^\ell$.
Parameters. Random Oracle $\mathcal{O} : \{0,1\}^\ell \to \{0,1\}^u$, where $u = 2^{\lceil \log(|X|+1) \rceil}$. The underlying PRF in OPRF functionality is denoted by $F' : \{0,1\}^\lambda \times \{0,1\}^\ell \to \{0,1\}^\ell$.

1. The parties invoke an instance of $\mathcal{F}_{\mathsf{OPRF}}$ where the receiver inputs x. The sender gets a key k and receiver gets output $z \in \{0,1\}^\ell$.
2. Sender samples $\nu \xleftarrow{\$} \{0,1\}^\lambda$ until $\{\mathcal{O}(F'(k, X(i)) \| \nu) \mid i \in [|X|]\}$ are all distinct.
3. For $i \in [|X|]$, sender computes $\mathsf{pos}_i = \mathcal{O}(F'(k, X(i)) \| \nu)$, and sets $\mathsf{HT}[\mathsf{pos}_i] = F'(k, X(i)) \oplus T(i)$.
4. For $j \in \{0,1\}^m \setminus \{\mathsf{pos}_i | i \in [|X|]\}$, sender sets $\mathsf{HT}[j] \xleftarrow{\$} \{0,1\}^\ell$.
5. Sender sends HT and ν to receiver.
6. Receiver computes $\mathsf{pos} = \mathcal{O}(z \| \nu)$, and outputs $\mathsf{HT}[\mathsf{pos}] \oplus z$.

Fig. 10. Table-based OPPRF construction from [32]

$\prod_{i=1}^{|X|} \left(1 - \frac{i}{2^u}\right)$. This expectation is low for only small sized input sets. Hence, table-based OPPRF is a suitable OPPRF candidate when input set X is small. Let $F'(k, \cdot)$ be the PRF in OPRF functionality. For every programmed element $X(i)$ and its respective target value $T(i)$, the sender uses $F'(k, X(i))$ to mask the target value $T(i)$ and stores this masked value at index $\mathcal{O}(F'(k, X(i)) \| \nu)$ in HT. The empty bins in HT are filled with random values. The hash table HT along with the nonce ν serves as the hint of this OPRF. To evaluate the OPPRF at element x, receiver computes $\mathsf{pos} = \mathcal{O}(z \| \nu)$ and outputs $\mathsf{HT}[\mathsf{pos}] \oplus z$, where z is the output it receives from OPRF functionality. From the correctness of OPRF functionality, it follows that $z = F'(k, x)$. If $x = X(i)$ for some i, then $\mathsf{HT}[\mathcal{O}(z \| \nu)] \oplus z = T(i)$. Thus, the correctness of OPPRF construction immediately follows. The security of the construction relies on random oracle assumption, security of PRF F' and OPRF functionality $\mathcal{F}_{\mathsf{OPRF}}$.