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 a two-party Circuit-based Private Set Intersection (PSI), P_0 and P_1 hold sets X and Y respectively and wish to securely compute a function f over the set $X \cap Y$ (e.g., cardinality, sum over associated attributes, and threshold intersection). Following a long line of work, Pinkas *et al.* (PSTY, Eurocrypt 2019) showed how to construct such a Circuit-PSI protocol with linear communication complexity. However, their protocol has super-linear computational complexity.
In this work, we construct Circuit-PSI protocols with linear computational and communication cost. Further, our protocols are concretely more efficient than PSTY – we are $\approx 2.3\times$ more communication efficient and are up to $2.8\times$ faster in LAN and WAN network settings. 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 in PSTY. While using Batch OPPRFs, in the context of Circuit-PSI results, in protocols with super-linear computational complexity, we construct RB-OPPRFs that can be used to build linear cost and concretely efficient Circuit-PSI protocols. We believe that the RB-OPPRF primitive could be of independent interest. As another contribution, we provide more communication efficient protocols (than prior works) for the task of *private set membership* – a primitive used in many PSI protocols including ours.

1 Introduction

Private Set Intersection. Consider a party P_0 that has a set X and a party P_1 that has a set Y. Private set intersection [43,30] allows them to compute the intersection $X \cap Y$ of these 2 sets without revealing anything else to each other. This problem has received much attention [9,36,26,39,35,6] and practical solutions to this problem are now known. However, in most applications, typically P_0 and P_1 would like to compute $f(X \cap Y)$, where f is a *symmetric* function. That is, f operates only on $X \cap Y$ and is oblivious to the order of the elements in $X \cap Y$. Examples of symmetric functions, that one may want to compute over the set intersection, include its cardinality, set intersection sum [45,28] (where every element in the set has an attribute associated with it and one would like to compute the sum of these attributes for all elements in the intersection), and threshold intersection [20] (where one would like to reveal the intersection only if it is larger than a threshold).

Circuit-PSI. To compute such applications securely, one cannot reveal the set intersection in the clear to either party and hence Huang et al. [21] 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. More specifically, for every element $z \in$ (say) X, P_0 and P_1 receive random bits u_0 and u_1 respectively as output where $u = u_0 \oplus u_1 = 1$ if $z \in X \cap Y$ (and is 0 otherwise). Since Circuit-PSI protocols have a stronger security requirement, their efficiency is poorer than that of regular PSI protocols. Following a sequence of works [36,38,7,14], the work of Pinkas *et al.* [37] somewhat surprisingly showed how to construct a Circuit-PSI protocol with *linear communication complexity* in n, the size of input sets.

1.1 Our Contributions

Computational Cost of Circuit-PSI protocols. While the protocol of [37] has linear communication complexity, unfortunately its computational complexity is super-linear (and specifically $\mathcal{O}\left(n(\log n)^2\right)$).

Hence, as noted by them, one of the main bottlenecks in their protocols is indeed its computational cost. In this work, we construct the first Circuit-PSI protocols whose communication as well as computational costs are linear in n. Furthermore, we experimentally validate our protocols and show them to be concretely more efficient than all prior works – as an example, our protocol is $\approx 2.3\times$ more communication efficient than [37] and between $2.3 - 2.8\times$ faster (in LAN/WAN settings) than [37], when the P_0 and P_1's sets comprise of 2^{22} elements.

Main Technical Contribution. At the heart of our technical contributions, we introduce the notion of *Relaxed-Batch Oblivious Programmable Pseudorandom Functions.* (RB-OPPRF), which can be seen as a strict generalization of Batch Oblivious Programmable Pseudorandom Functions (B-OPPRF), used in [37]. The linear communication construction for B-OPPRF in [37] required expensive polynomial interpolation for large degree polynomials, and hence was the source of the main computational (super-linear) inefficiency. As our technical contribution, we construct RB-OPPRF using cheap operations such as Cuckoo hashing. By coupling with a new more efficient protocols for the task of *Private Set Membership* (where one party has a list B of d elements and the other has a single element a and they wish to test if $a \in B$), we obtain Circuit-PSI protocols with linear communication and computation that are also concretely efficient.

A recent work by Karakoç and Küpçü [23] provides a Circuit-PSI protocol with linear communication and computational cost using completely different techniques from ours; however, since their focus was not on concrete efficiency, their protocol has significantly higher ($\approx 4\times$) communication than even [37] and is 5–12$\times$ slower than [37] in the LAN setting [23].

1.2 Technical Overview

The protocol from [37]. To describe our construction in more detail, first consider the protocol from [37] that works as follows. First, P_0 uses Cuckoo hashing (with d hash functions, $\{h_i\}_{i=1}^{d}$) to hash the elements from set X into a hash table HT_0 with β bins (β 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 case where such a stash does not exist ([37] also 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 Y into a hash table HT_1 with the same number of bins. That is, each element $y \in Y$ 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 stashless setting, observe that if some element $z \in X \cap Y$ 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 P_0's bin to the corresponding elements in P_1's bin. Since comparing each of the $\mathcal{O}(\log n)$ elements in P_1's bin with an element in P_0's, for a linear number of bins would result in super-linear communication cost, [37] introduced a primitive known as Batch Oblivious Programmable Pseudorandom Functions (B-OPPRF) to reduce the number of comparisons for each bin from $\mathcal{O}(\log n)$ to 1.

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. Additionally, when the sender can "program" the PRF to output certain values on certain (private) input points, such a primitive is known as an Oblivious Programmable Pseudorandom Function (OPPRF) [27]. This is done by the functionality through additionally providing a "hint" hint to both sender and receiver, where the size of the hint is linear in the number of points being programmed. When invoking β independent instances of OPPRF, where the number of programmed points in each instance could vary, but the total programmed points, N, across all instances is fixed, [37], showed how to provide a single hint that is linear in N and further hide even the number of programmed points in every instance. Such a primitive is known as a B-OPPRF. Creation of such a hint is computationally the most expensive part of the protocol [37] and leads to super-linear complexity since it requires polynomial interpolation. [37] then showed that P_1 can play the role of the sender in β instances of B-OPPRF protocols setting the programmed points in the j^{th}

instance to be the set of elements in $\mathsf{HT}_1[j]$, where *all the programmed outputs are a single random value t_j*. P_0 will obliviously evaluate the j^{th} OPPRF with the element in $\mathsf{HT}_0[j]$. Now, P_0 and P_1 will each hold a single value that is the same if $\mathsf{HT}_0[j] \in \mathsf{HT}_1[j]$ and different otherwise. They can then employ a standard equality protocol (which is a special case of private set membership) to compare these 2 elements. To summarize, using $\mathsf{B\text{-}OPPRF}$, they reduce the number of comparisons per bin from $\mathcal{O}(\log n)$ to 1, at the cost of super-linear computation.

Relaxed $\mathsf{B\text{-}OPPRF}$ and Our Protocol. In our construction, we replace the expensive $\mathsf{B\text{-}OPPRF}$ primitive with a generalization of it called *Relaxed* $\mathsf{B\text{-}OPPRF}$ (or $\mathsf{RB\text{-}OPPRF}$). This primitive reduces the number of comparisons per bin from $\mathcal{O}(\log n)$ to 3 while incurring only *linear computation*. In an $\mathsf{RB\text{-}OPPRF}$, the functionality provides a set of 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 $\mathsf{B\text{-}OPPRF}$ primitive. Surprisingly, such a simple generalization makes constructing it much more efficient. In particular, we show how to construct an $\mathsf{RB\text{-}OPPRF}$ 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 [37]. Applying a $\mathsf{RB\text{-}OPPRF}$ to the blueprint of [37] gives us the following guarantee for every bin: P_0 will hold 3 values b_1, b_2, b_3 and P_1 will hold a single value a such that $a = b_i$ for some $i \in [3]$ 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* protocol [15] with a set size of only 3. While many protocols for this task exist [21,39,7], we construct 2 new protocols, $\mathsf{PSM1}$ and $\mathsf{PSM2}$ that have $4.2\times$ and $6.4\times$ lower communication than prior works (see Table 1, Section 4.3). Protocol $\mathsf{PSM1}$ has better computation but worse communication than $\mathsf{PSM2}$. Protocol $\mathsf{PSM1}$ uses techniques from tree-based comparison protocols [17,11,41], and $\mathsf{PSM2}$ uses the table-based OPPRF construction [27] on small sets.

Finally, we remark that our circuit-PSI protocols make use of OPRF protocol [26] and OT-Extension protocols [22,25,1]. The concrete communication complexity of these primitives can be further improved by making use of the recent line of work on silent-OT and its extensions [3,44] as discussed in [3,14]. We leave the integration of such improvements into our protocols for future work.

Circuit-PSI with stash. As mentioned earlier, the above discussion assumed that no stash is created during the cuckoo hashing phase. In [37], they employed a novel dual-execution technique to compare stash elements with hash table elements with linear cost. We show how to apply a similar dual-execution technique in our context to handle stash. Crucial to our construction is the observation that when the construction in [37] 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 linear in n_0 but super-linear in n_1. This imbalance allows us to use their protocol within our framework to obtain an overall linear computation and communication protocol even with stash.

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 Relaxed Batch Programmable Pseudorandom Functions and the corresponding oblivious 2-party functionality. We then provide efficient constructions for these primitives. Section 4 is devoted to our two new protocols for private set membership. We describe our circuit-PSI protocol in Section 5. Finally, in Section 6, we experimentally validate our circuit-PSI protocol and show that it outperforms prior works in various 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 $X(i)$ denote the i^{th} element of set X. $x \leftarrow y$ 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 X and Y, 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 $f(X \cap Y)$. For this, the two parties agree on a circuit C that computes f. There are two kinds of functions that one can consider here. 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. Similar to prior works on Circuit-PSI [21,36,38,14,37,23] , we provide security against static probabilistic polynomial time (PPT) semi-honest adversaries in real/ideal simulation paradigm [19,29]. 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 presence of $\mathcal{A}$ and environment $\mathcal{Z}$. Let $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}$ denote the binary distribution ensemble describing $\mathcal{Z}$'s output in 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 [5] 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. Our hashing schemes use 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 the hash table 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 [34] uses $d > 1$ universal hash functions $h_1, \ldots, h_d : \{0,1\}^* \mapsto [\alpha]$ to map n elements to α bins in hash table HT, where $\alpha = O(n)$. 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 one of the empty bins. (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*. Hence, after cuckoo hashing, an element e can be found in one of the following bins: $h_1(x)$, …, $h_d(x)$ or the stash. Observe that in cuckoo hashing each bin 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, also known as failure probability, is a function of the number of hash functions d and the number of bins α. In [24], it was shown that Cuckoo hashing of n elements into $(1+\varepsilon)n$ bins with $\varepsilon \in (0,1)$ for any $d \geqslant 2(1+\varepsilon)\ln(\frac{1}{\varepsilon})$ and $s \geqslant 0$ fails with probability $O(n^{1-c(s+1)})$, for constant $c > 0$ as $n \mapsto \infty$. In application of cuckoo hashing to the problem of PSI and Circuit-PSI [36,26,39,27,38,37], 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 [39] 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 $s = 0$, α= $1.27n$, $1.09n$ and $1.05n$ bins are required for $d = 3, 4$ and 5 respectively. In our experiments, similar to [39,37], we use this result to set our hashing parameters for no stash setting, which achieves statistical security of 2^{-40}.

Secret Sharing Schemes. We use 2-out-of-2 additive secret sharing scheme [42] 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 sometimes refer shares over $\mathbb{Z}_2$ as boolean shares.

Oblivious Transfer. We consider 1-out-N Oblivious Transfer (OT) functionality [40,4] $\binom{N}{1}$-OT_ℓ that takes as input N messages from sender $m_1, \ldots, m_N \in \{0,1\}^\ell$ and index $i \in [N]$ 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 [25,22]. These protocols allow to extend λ 'base' OTs to large number of oblivious transfers OTs (polynomial many in λ) using symmetric key primitives only. The protocols $\binom{N}{1}$-OT_ℓ [25] and $\binom{2}{1}$-OT_ℓ [22] execute in two rounds and have total communication of $2\lambda + N\ell$ and $\lambda + 2\ell$ bits respectively.

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 the parties. $\mathcal{F}_{\mathsf{AND}}$ can be realized using bit-triples [2], which are of the form $(\langle a\rangle_s, \langle b\rangle_s, \langle c\rangle_s)$, where $s \in \{0,1\}$ and $a \wedge b = c$. We make use of protocol given in [11,41] for triple generation which has an amortized communication cost of 144 bits per triple.

3 Relaxed Batch PPRF and OPPRF

Batch PPRF and OPPRF. Informally, a pseudorandom function (PRF) [18], sampled with a key k from a function family, is guaranteed to be computationally indistinguishable from a uniformly random function, to an adversary who does not have k, but is given oracle access to the function. A programmable PRF (PPRF in short), introduced by [27] 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. The hint leaks no information about the programmed inputs/outputs, but is not guaranteed to keep the number of programmed points private. When β independent instances of a PPRF are used, the β different hints can be combined into a single hint that still does not guarantee to keep the total number of programmed points across all instances private, but leaks no information about the number of programmed points of every instance. This notion was introduced and formalized as Batch PPRF (BPPRF in short) by [37].

The 2-party Oblivious PRF (OPRF) functionality was defined by [15] 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 BPPRF), 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 BPPRF) on its evaluation points. The corresponding functionalities are then known as Oblivious PPRF (OPPRF in short) and Batch Oblivious PPRF (B-OPPRF in short) respectively.

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

The functionality works as follows:

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

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

Relaxed Notions. While the size of the hint in the BPPRF (and hence B-OPPRF) scheme of [37] 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 required interpolating an m-degree polynomial (where m is a parameter that is 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, [37] proposed an optimization that brings down the cost to $\omega(m\text{log}m)$ using Lagrange interpolation or $\omega(m(\text{loglog}m)^2)$ using FFT. Even with this optimization, it was noted in [37] that the computational complexity of this polynomial interpolation step is one of the major bottlenecks. For completeness, we recall the polynomial based BPPRF construction of [37] in Fig. 9 of Appendix A.

With the computational cost in mind, we generalize the notions of BPPRF and B-OPPRF in the following way. While the BPPRF 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 the programmed output is one 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 have hint size linear in total number of programmed points as in [37] and also linear computational cost (and in fact are concretely efficient too). 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-art [37] (see Section 6).

3.1 Definitions of RB-PPRF and RB-OPPRF

We first present the definition of Relaxed Batch Programmable PRF (RB-PPRF). As discussed earlier, this is a generalization of BPPRF 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 BPPRF presented in [37]. As was in the case of BPPRF, we will only require that the programmed outputs come from a distribution $\mathcal{T}$ 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., $F' : \{0,1\}^\lambda \times \{0,1\}^\ell \to \{0,1\}^{d\ell}$.

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

- $Hint(K, X, T) \to \mathsf{hint}$*: 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 said to be a (ℓ, β, d)-$\mathsf{RB\text{-}PPRF}$ *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* $\mathsf{RB\text{-}PPRF}$ *oracle over* $\hat{F}$ *if, when interacting with a "caller"* $\mathcal{A}$*, it works as follows:*
  1. *M is given disjoint sets* $X = X_1, \ldots, X_\beta$ *from* $\mathcal{A}$.
  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* $W_1, \ldots, W_\beta$ *where* $W_j = F(k_j, \mathsf{hint}, x_j)$.
  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. In the definition of (ℓ, β, d)-$\mathsf{RB\text{-}PPRF}$, we considered that input length is the same as the output length, i.e., ℓ. One can also consider (ℓ, β, d)-$\mathsf{RB\text{-}PPRF}$ schemes with input length ℓ', different from the output length.

Property 1. This is an additional correctness property that we want a (ℓ, β, d)-$\mathsf{RB\text{-}PPRF}$ primitive to satisfy when used as a building primitive in broader constructions. The property establishes that for all $j \in [\beta]$ and non-programmed points x, the probability that the output of (ℓ, β, d)-$\mathsf{RB\text{-}PPRF}$ scheme is one of the elements in target multi-set T_j is negligible. Formally, for all $j \in [\beta]$ and $x \notin X_j$, $\Pr[F(k_j, \mathsf{hint}, x) \cap T_j \neq \emptyset] = 2^{-p(\ell, d)}$, for a polynomial function $p()$.

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

3.2 Constructions for RB-PPRF and RB-OPPRF

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 γ bins. As described in Section 2.2, cuckoo hashing is performed into the hash table along with a stash of size s and the probability that the hashing process will fail to insert an element either into the hash table or the stash drops exponentially in s. However, for efficiency reasons in applications, one would ideally require $s = 0$. For this setting, the work of [39] empirically determined that, when $d = 3$, for a concrete failure probability of 2^{-40}, setting $\gamma = 1.27N$ (for various values of N), suffices. At a very high level, our idea to construct the $\mathsf{RB\text{-}PPRF}$ is 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. We know

Parameters. An (ℓ, β, d)-$\mathsf{RB\text{-}PPRF}$ $\hat{F} = (Hint, F)$.
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)-$\mathcal{F}_{\mathsf{RB\text{-}OPPRF}}$

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. We take a PRF F' that outputs $f_1\|f_2\|f_3$, each f_b is of length ℓ bits and for every programmed input $X_j(i)$, we store $T_j(i) \oplus f_{\mathsf{idx}}$ at position $h_{\mathsf{idx}}(X_j(i))$. 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. Cuckoo hashing based $(\ell, \beta, 3)$-$\mathsf{RB\text{-}PPRF}$ construction can be extended for different values of d such as $d = 4$ and 5 to obtain (ℓ, β, d)-$\mathsf{RB\text{-}PPRF}$ construction by considering d-many hashing functions in the Cuckoo Hashing scheme and setting the parameters appropriately as discussed in [39] for these values of d to obtain concrete failure probability of 2^{-40}.

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 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. First, consider an adversary $\mathcal{B}$ who is given oracle access to a set of β functions $F'(k_j, \cdot)$ for $j \in [\beta]$, with every k_j being an independently uniform key to the PRF F'. Now, from the PRF security of F', it holds that $\mathcal{B}$ cannot distinguish between being given oracle access to these β functions and β truly random functions $R_1, \cdots, R_\beta$ (with appropriate domain and range), except with negligible probability. Call such an adversary $\mathcal{B}^{\mathcal{O}}$ (that receives oracle access to one of these sets of β functions).

Now, to show the security of our construction, as in Definition 1, let M be an $(\ell, \beta, 3) -$ $\mathsf{RB\text{-}PPRF}$ oracle over F, where F is as defined in the construction of Fig. 3. Let $\mathcal{S}$ be the simulator described in Definition 1. That is, when a caller $\mathcal{A}$ provides M with disjoint sets $X_1, \cdots, X_\beta$, $\mathcal{S}$ outputs a random string in $\{0,1\}^{c \cdot \ell \cdot N}$ as the hint. Next when it receives queries $x_1, \cdots, x_\beta$ from $\mathcal{A}$, it responds with β sets of values, each comprising of d uniformly random values in $\{0,1\}^\ell$. Now, assume for the sake of contradiction, that there exists a distinguisher $\mathcal{D}$ that distinguishes between the output of M and the output of $\mathcal{S}$ after interacting with $\mathcal{A}$, with noticeable probability. Then, we construct a distinguisher $\mathcal{B}^{\mathcal{O}}$ below.

Reduction. First, $\mathcal{B}^{\mathcal{O}}$ samples $T_1, \ldots, T_\beta$ from $\mathcal{T}$ and the universal hash functions h_1, h_2, and h_3. Next, $\mathcal{B}^{\mathcal{O}}$ prepares a garbled hash table GT in a similar manner as the Hint algorithm in our construction.

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 $\gamma = (1+\varepsilon)N$.

$\underline{\mathsf{Hint}(K, X, T)}$: Given the keys for F', $K = k_1, \dots, k_\beta \in \{0,1\}^\lambda$, disjoint input sets $X = X_1, \dots, X_\beta$ (with total elements N) and target multi-sets $T = T_1, \dots, T_\beta$, 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 $X_1, \dots, X_\beta$ in a hash
   table $\mathsf{HT}$ with $\gamma$ bins.
2. Let $\mathsf{E}$ be a mapping that maps elements in input sets to the hash function that was eventually used to
   insert that element into $\mathsf{HT}$, i.e. $\mathsf{E}(X_j(i)) = \mathsf{idx}$ if $\mathsf{HT}[h_{\mathsf{idx}}(X_j(i))] = X_j(i)$.
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 $\mathsf{GT}$, pick $r_i \overset{\$}{\leftarrow} \{0,1\}^\ell$ and set $\mathsf{GT}[i] \leftarrow r_i$.
10. Return $\mathsf{GT}$ as $\mathsf{hint}$.
```

$\underline{F(k, \mathsf{hint}, x)}$: 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 $\mathsf{hint}$ as a garbled hash table $\mathsf{GT}$.
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 $(\ell, \beta, 3) - \mathsf{RB\text{-}PPRF}$

That is for all $j \in [\beta]$ and $i \in [|X_j|]$, $\mathcal{B}^{\mathcal{O}}$ does the following: (1) $\mathcal{B}^{\mathcal{O}}$ determines the index $b_{j,i} \in [3]$, such that $\mathsf{HT}[h_{b_{j,i}}(X_j(i))] = X_j(i)$. (2) Next, $\mathcal{B}^{\mathcal{O}}$ queries $\mathcal{O}$ on $(j, X_j(i))$ - i.e., for the output of the j^{th} function on input $X_j(i)$ and parses the oracle response as $f_{j,i,1} \| f_{j,i,2} \| f_{j,i,3}$. (3) $\mathcal{B}^{\mathcal{O}}$ sets $\mathsf{GT}[h_{b_{j,i}}(X_j(i))] = f_{j,i,b_{j,i}} \oplus T_j(i)$. (4) Finally, $\mathcal{B}^{\mathcal{O}}$ pads the empty bins in GT with random values. It sends GT to $\mathcal{A}$ as the hint. Upon receiving a query x_j from $\mathcal{A}$, $\mathcal{B}^{\mathcal{O}}$ 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 whatever $\mathcal{D}$ outputs.

First, we show that if $\mathcal{O}$ is a set of truly random functions, then $\mathcal{B}^{\mathcal{O}}$'s responses are identical to that of $\mathcal{S}$. To see this, first observe that since every element of GT is either chosen by $\mathcal{B}^{\mathcal{O}}$ uniformly at random (from Step (4) above) or $T_j(i)$ is masked by the output of $\mathcal{O}$ on a unique query point $(j, X_j(i))$ (from Step (3) above), GT is uniformly random. Now, consider the query responses provided by $\mathcal{B}^{\mathcal{O}}$. For the j^{th} query x_j, there are two cases depending on whether $x_j \in X_j$ or not. In the former case, let us assume $x_j = X_j(i)$ for some $i \in |X_j|$. $\mathcal{B}^{\mathcal{O}}$'s response is W_j which is a list of 3 values one of which is $T_j(i)$. Suppose $W_j[b^*] = T_j(i)$ for some $b^* \in [3]$ and let $f_1 \| f_2 \| f_3 \leftarrow R_j(x_j)$. Now, for $b = b^*$, if we argue that apart from $T_j(i)$, the other elements in T_j are unknown, then since every element in T_j is individually guaranteed to be uniformly random, it holds that $W[b] = T_j(i)$ is uniformly random (even given hint). It is indeed the case because the other values in T_j, i.e., for $i' \neq i$, $T_j(i')$ is masked with completely random substring of $R_j(X_j(i'))$ which is unknown to $\mathcal{A}$ because for $j \in [\beta]$, $\mathcal{A}$ can make only a single query x_j which turns out to be $X_j(i)$ in this case. When $b \neq b^*$, then observe that the response is a value that is masked by an f_b value received from $\mathcal{O}$, where f_b has not been used previously anywhere (in particular in hint). Hence, in this case $\mathcal{B}^{\mathcal{O}}$'s response is identical to $\mathcal{S}$. For the case when $x_j \notin X_j, W_j$ is a set of values that has been masked by a response from $\mathcal{O}$ on a completely fresh point and hence is uniformly random.

On the other hand, when $\mathcal{O}$ is a set of PRFs, then it is easy to see that $\mathcal{B}^{\mathcal{O}}$ behaves identically to $(\ell, \beta, 3)$-RB-PPRF M. Hence, the probability that $\mathcal{B}^{\mathcal{O}}$ succeeds is identical to that of $\mathcal{D}$ succeeding.

Property 1. Now, to show that Property 1 holds the argument is similar to the one we made for the case when $x_j \notin X_j$ in the security proof. In entries of W_j, value from hint is masked by PRF output on a completely fresh point which is never considered during the construction of hint. Hence, from security of PRF we conclude that these entries are pseudo-random and computationally independent of values in T_j. Hence, for $x_j \notin X_j$ and $W = F(k_j, \mathsf{hint}, x)$, By union-bound, $\Pr[W \cap T_j \neq \emptyset]$ is less than $3 \cdot 2^{-\ell} = 2^{-(\ell - \log 3)}$.

□

$\mathsf{RB\text{-}OPPRF}$ **Construction.** We provide an $(\ell, \beta, 3)$-$\mathsf{RB\text{-}OPPRF}$ construction in Fig. 4 and prove its security in Theorem 2.

Parameters. Functionality $\mathcal{F}_{\mathsf{OPRF}}$, Cuckoo hashing based $(\ell, \beta, 3)$-$\mathsf{RB\text{-}PPRF}$ scheme $\hat{F} = (Hint, F)$ (see Fig. 3) that uses hash functions h_1, h_2 and h_3. The PRF $F' : \{0,1\}^\lambda \times \{0,1\}^\ell \to \{0,1\}^{3\ell}$ 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. $\mathsf{RB\text{-}OPPRF}$ construction using Cuckoo-hashing based $\mathsf{RB\text{-}PPRF}$ scheme

Theorem 2. *Given construction in Figure 3 is a secure $(\ell, \beta, 3)$-$\mathsf{RB\text{-}PPRF}$ scheme, construction in Figure 4 securely realizes $(\ell, \beta, 3)$-$\mathcal{F}_{\mathsf{RB\text{-}OPPRF}}$ (Figure 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 construction follows from the correctness of $\mathcal{F}_{\mathsf{OPRF}}$ functionality and $(\ell, \beta, 3)$-$\mathsf{RB\text{-}PPRF}$ construction.

Security. To simulate the view of sender, sample random PRF keys $K = k_1, \ldots, k_\beta \in \{0,1\}^\lambda$ and send it to sender.

To simulate view of the receiver, let $\mathcal{S}_1$ be the simulator for $\mathsf{RB\text{-}PPRF}$. 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 $\mathsf{RB\text{-}PPRF}$ as proved in Theorem 1. Finally, to argue overall linear complexity, it suffices to argue linear complexity for hint computation, communication and evaluation. We note that computing the hint using cuckoo hashing and garbled hash table has linear computation in N for the sender. Size of the hint is $\gamma \cdot \{0,1\}^\ell = O(N\ell)$. And it is easy to see that receiver's compute is linear in β. □

4 Private Set Membership

The set membership [15] is the 2-party functionality that takes as input a set B of d elements $\{B(1), \ldots, B(d)\}$, with $B(i) \in \{0,1\}^\ell$, $\forall i \in [d]$, 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 [19,2,1,10] and many specialized protocols [21,39,7]. However, the resultant protocols have high communication cost. In this section, we propose two specialized protocols - $\mathsf{PSM1}$ and $\mathsf{PSM2}$ to realize $\mathcal{F}_{\mathsf{PSM}}$ that both have significantly lower communication overhead than prior approaches. While $\mathsf{PSM2}$ (in Section 4.2) has the lower of the two communication, its concrete computation cost is higher than $\mathsf{PSM1}$ (in Section 4.1). We discuss these trade-offs and also provide comparison with other generic and specialized protocols in Section 4.3.

Inputs of P_0. Input set B of size d, where $\forall i \in [d]$, $B(i) \in \{0,1\}^{\ell}$.
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 $\mathcal{F}_{\mathsf{PSM}}$

4.1 Private Set Membership Protocol 1

Our protocol $\mathsf{PSM1}$ builds on the idea in [17,11,41] 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 [41] for the case of Millionaires', there is a compute vs communication trade-off between large and small values of m and one can do empirical analysis to obtain a 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 $\bigoplus_{i\in[d]} \mathbf{1}\{B(i) = a\}$, that is, involves computing a batch of equalities. We show that for comparing an element a with all elements in a set B of size d, we can do much better than d instances of equality checks. We observe that for the leaf nodes, inputs of P_1 is 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 receiver's inputs are same in all d executions, these OTs can be batched together, and we replace d instances of $\binom{M}{1}$-OT_1 with a single instance of $\binom{M}{1}$-OT_d reducing communication per leaf from $d \times (2\lambda + M)$ to $(2\lambda + Md)$.

For ease of exposition, we describe the scheme formally in Fig. 6 for the special case of $m|\ell$ and $q = \ell/m$ is a power of 2. Using the notation in Fig. 6, the batching of equality computation across d instances for the leaf nodes works as follows: P_0 has input set B of d elements such that each element $B(i) = b_{i,q-1}\| \dots \|b_{i,0}$ for all $i \in [d]$. Similarly, P_1 has input $a = a_{q-1}\| \dots \|a_0$. It holds that a_j, $b_{i,j}$ $\in \{0,1\}^m$, for all $j \in \{0, \dots, q-1\}$ and $i \in [d]$. For each $j \in \{0, \dots, q-1\}$, the task is to compute shares of $\mathsf{eq}_{0,i,j} = \mathbf{1}\{b_{i,j} = a_j\}$ for all $i \in [d]$, and this is where we use our batching technique. We use an instance of $\binom{M}{1}$-OT_d 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 $e_{j,v} = p_1\| \dots \|p_d$, 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 $\{\mathbf{1}\{b_{i,j} = v\}\}_{i\in[d]}$. Once we have the leaf computation of the whole batch of size d, traversing up the tree to the root happens independently for each instance. Finally, we learn the shares corresponding to the roots, i.e. $\{\mathbf{1}\{b_i = a\}\}_{i\in[d]}$. 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 d, where $\forall i \in [d]$, $B(i) \in \{0,1\}^\ell$.
Inputs of P_1. Element $a \in \{0,1\}^\ell$.
Parameters. Radix Parameter m, $M = 2^m$, $\binom{M}{1}$-OT_d 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 $i \in [d]$.
3. for $j \in \{0,\ldots,q-1\}$ do
4.     $P_0$ samples $\langle \mathsf{eq}_{0,i,j}\rangle_0 \xleftarrow{\$} \{0,1\}$, $\forall i \in [d]$.
5.     for $v \in [M]$ do
6.         $P_0$ sets $e_{j,v} \leftarrow \langle \mathsf{eq}_{0,1,j}\rangle_0 \oplus \mathbf{1}\{b_{1,j} = v\}\|\ldots\|\langle \mathsf{eq}_{0,d,j}\rangle_0 \oplus \mathbf{1}\{b_{d,j} = v\}$.
7.     end
8.     $P_0$ & $P_1$ invoke $\binom{M}{1}$-$\mathsf{OT}_d$ 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 $e = \langle \mathsf{eq}_{0,1,j}\rangle_1\|\ldots\|\langle \mathsf{eq}_{0,d,j}\rangle_1$ as output, where $\langle \mathsf{eq}_{0,i,j}\rangle_1 \in \{0,1\}$ for all $i \in [d]$.
9. end
10. for $t = 1$ to $\log q$ do
11.     for $j \in \{0,\ldots,q/2^t-1\}$ do
12.         for $i \in [d]$ do
13.             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.         end
15.     end
16. end
17. For $s \in \{0,1\}$, $P_s$ computes $y_s \leftarrow \bigoplus_{i\in[d]} \langle \mathsf{eq}_{\log q,i,0}\rangle_s$ and outputs $y_s$.
```

Fig. 6. Private Set Membership Protocol, PSM1

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

By correctness of $\binom{M}{1}$-OT_d 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 tth 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 $j \in \{0,\ldots,q_t-1\}$. Next, 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 $i \in [d]$, 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, $y_0 \oplus y_1 = \bigoplus_{i\in[d]} \mathbf{1}\{B(i) = a\}$. If $a \notin B$, we get $y_0 \oplus y_1 = 0$ else $y_0 \oplus y_1 = 1$, since B is a set, i.e., 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 $\in \{0,1\}^d$ as output from $\binom{M}{1}$-OT_d instances. This is correct because each of $\langle \mathsf{eq}_{0,i,j}\rangle_0$ is random. Outputs of $\mathcal{F}_{\mathsf{AND}}$ can be simulated in a similar manner as above. □

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 [41]. For completeness, the main idea is as follows. Let $r = \ell \mod m$ then $a_{q-1} \in \{0,1\}^r$. For the last leaf, we invoke $\binom{2^r}{1}$-OT_d. 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 would have $d(q-1)$ calls to $\mathcal{F}_{\mathsf{AND}}$ (in both special case and general case).

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

4.2 Private Set Membership Protocol 2

Consider the $\mathcal{F}_{\mathsf{OPPRF}}$ functionality [27] 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 d 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 [27], one can realize $\mathcal{F}_{\mathsf{PSM}}$ by first invoking $\mathcal{F}_{\mathsf{OPPRF}}$ (to reduce the d equality checks to a single equality check) and then computing the single check using a call to $\mathcal{F}_{\mathsf{eq}}$ (the functionality obtained in $\mathcal{F}_{\mathsf{PSM}}$ by setting $d = 1$). Kolesnikov et al. [27] proposed three OPPRF constructions, viz., polynomial based OPPRF, garbled bloom filter [12] based OPPRF and table-based OPPRF construction. For small set sizes, it was experimentally confirmed in [27] that table-based OPPRF construction is the fastest in practice. 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 $d = 1$. This protocol is $\approx 2\times$ more communication efficient than the state-of-the-art protocol [8] 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 [27] in Figure 10 of Appendix A). We get the best benefit of using this construction, as in our case we deal with a constant set size, i.e., $d = 3$, whereas Kolesnikov et al. [27] made use of this construction in a scenario where d is sub-linear.

Concrete Complexity. A single call to $\mathsf{PSM2}$ involves interaction between the two parties in call to OPRF functionality realized using construction given in [26], hint transmission and equality check. The amortized communication cost for a single instance of OPRF evaluation using this protocol is 3.5λ bits [26, Table 1]. The hint size is $\lambda + u\ell$, where $u = 2^{\lceil \log d+1 \rceil}$. Finally, we make a call to $\mathsf{PSM1}$ (with $d = 1$, 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 $d = 3$ 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 [7,10] and refer the reader to [7] for more details on prior PSM protocols. Table 1 illustrates the communication complexity of the PSM schemes. In our setting, we will invoke PSM protocols with a set size $d = 3$. The table also provides a comparison of communication

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

1. P_0 samples a random target value t and prepares a set T such that it has d, 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, $\mathsf{PSM2}$

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. Finally, we remark that the computation overhead incurred in $\mathsf{PSM1}$, CO [7] and ABY [10] are nearly the same, while the computation cost of $\mathsf{PSM2}$ is higher due to its use of the OPPRF scheme.

	Communication	Concrete Example
CO [7]	$2\ell d\lambda + d\lambda$	49536
ABY [10]	$2d\lambda(\ell - 1)$	48384
$\mathsf{PSM1}$ $(m = 4)$	$\ell d\lambda/4 + \ell\lambda/2 + 8\ell d$	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, d denotes the number of elements and $u = 2^{\lceil \log(d+1) \rceil}$. For the concrete example, we consider: $\ell = 64$, $d = 3$, $\lambda = 128$.

5 Circuit-PSI with Linear Communication and Computation

In circuit PSI [21,36,38,14,37,23], we are interested in computing a function on the intersection of two sets, X and Y. Formally, the circuit-PSI functionality, $\mathcal{F}_{\mathsf{PSI},f}$, takes X from P_0 and Y from P_1 and outputs $f(X \cap Y)$ 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 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 $\beta = O(n)$. 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 first describe our construction in the stashless setting using the parameter settings based on the empirical analysis in [39]. Later, we describe in Section 5.2 how our ideas easily extend to the setting with stash by building on dual execution idea from [37]. We emphasize that both our constructions (with and without stash) have linear communication and linear computation complexity.

5.1 Circuit-PSI via stashless Cuckoo Hashing

Construction Overview. Our construction follows a similar high-level blueprint as [37]. 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 X into hash table HT_0. We pick ε, used in setting the number of bins β, from parameters suggested in [39] to ensure that stash is not required in Cuckoo hashing. On the other side, P_1 does simple hashing of Y 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 $z \in X \cap Y$, such that $\mathsf{HT}_0[j] = z$, then $z \in \mathsf{HT}_1[j]$ as well. Hence, it suffices to compare the elements in HT_0 and HT_1 per bin. For this step, [37] used their batch OPPRF construction. In this work, we use the computationally more efficient RB-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 $\{Y_j\}_{j\in[\beta]}$ as $Y_j = \{(y\|j) \mid y \in \mathsf{HT}_1[j]\}$. Next, for $j \in [\beta]$, P_1 samples $t_j \in \{0,1\}^\ell$ independently and uniformly, and constructs T_j with $|Y_j|$ 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 $x_j \in Y$, where Y 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.

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 X of size n, where $X(i) \in \{0,1\}^\mu$, for all $i \in [n]$.
Inputs of P_1. Input set Y of size n, where $Y(i) \in \{0,1\}^\mu$, 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 X 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 Y to hash table HT_1 with β bins.

Computing relaxed batch OPPRF:

3. P_1 creates input sets $Y_1, \ldots, Y_\beta$ and target sets $T_1, \ldots, T_\beta$ as follows: For all $j \in [\beta]$, $Y_j = \{(y\|j) \mid y \in \mathsf{HT}_1[j]\}$, sample random and independent target value t_j and T_j has $|Y_j|$ elements all equal to t_j.
4. P_0 creates queries $x_1, \ldots, x_\beta$ such that $x_j = (\mathsf{HT}_0[j]||j)$ for all $j \in [\beta]$.
5. P_0 & P_1 invoke (ℓ, β, d)-$\mathcal{F}_{\mathsf{RB\text{-}OPPRF}}$ with P_1 as sender with input sets $\{Y_j\}_{j\in[\beta]}$ 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 RB-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}

Instantiating the protocol in Fig. 8 . We can realize the $(\ell, \beta, 3)$-RB-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 PSM1 (see Section 4.1) or PSM2 (see Section 4.2). This gives us two protocols for circuit-PSI problem, that we call C-PSI$_1$ and C-PSI$_2$ respectively. These protocols have 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 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 $z \in X \cap Y$, 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 $x_j \in Y_j$ in our construction. By correctness of $\mathcal{F}_{\mathsf{RB\text{-}OPPRF}}$, if $x_j \in Y_j$, then $t_j \in W_j$. Moreover, if $\mathsf{HT}_0[j] \notin \mathsf{HT}_1[j]$, then $x_j \notin Y_j$. By property 1, it holds that if $x_j \notin Y_j$, then $t_j \in W_j$ with probability at most $2^{-p(\ell,d)}$, for a polynomial function $p(\cdot)$ and $d = 3$. Taking a union bound over β bins, total probability of false positives is upper bound by $\mathsf{fail} = \beta \cdot 2^{-p(\ell,d)}$. We pick ℓ such that $\mathsf{fail} < 2^{-40}$. 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. The 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 $X \cap Y$. This follows from immediately from the following: 1) Our construction of $(\ell, \beta, 3)$-$\mathsf{RB\text{-}OPPRF}$ 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 $\mathsf{PSM1}/\mathsf{PSM2}$ is invoked independently for each j, step 6 has linear complexity. 3) Inputs to $C_{\beta,\mu}$ are linear in size and hence the computation until the step of computing $X \cap Y$ has linear complexity.

PSI with Associated Payload. From Section 2.1, recall that we consider two types of function to be computed on the intersection of the input sets. In the first, the function f takes only the elements as input; while in the second, there is a payload data associated with every input element and the function f takes both the elements and their associated payloads as input. While the protocol described in Figure 8 handles functions of the first type, we can adapt this protocol in a similar way as was done in [37] to handle a function f of the second type. We briefly describe this below.

Let $U(x)$ and $V(y)$ respectively denote the payloads associated with element $x \in X$ and $y \in Y$ and let all the payloads have the same length δ. Parties P_0 and P_1 on respective input sets X and Y execute steps 1 to 5 of Π_{PSI} protocol (see Figure 8). We will use $(\ell, \beta, 3)$-$\mathsf{RB\text{-}OPPRF}$ protocol instantiated with our $(\ell, \beta, 3)$-$\mathsf{RB\text{-}PPRF}$ protocol given in Figure 3 in step 5 of Π_{PSI} protocol and let h'_1, h'_2 and h'_3 be the set of universal hash functions used in $(\ell, \beta, 3)$-$\mathsf{RB\text{-}PPRF}$ protocol. 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 $Y_1, \ldots, Y_\beta$, target values $t_1, \ldots, t_\beta$ and hash table HT (step 1 in Figure 8).

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 $|Y_j|$, 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 $i \in |Y_j|$. P_0 and P_1 then invoke another instance of $(\delta, \beta, 3)$-$\mathsf{RB\text{-}OPPRF}$ protocol instantiated with $(\delta, \beta, 3)$-$\mathsf{RB\text{-}PPRF}$ protocol given in Figure 3 with P_1 as sender with input sets $\{Y_j\}_{j\in[\beta]}$ and target sets $\{\widetilde{T}_j\}_{j\in[\beta]}$ and P_0 as receiver with input queries $\{x_j\}_{j\in[\beta]}$. Morever, P_1 uses the same hash table HT in creation of hint in this instance of $(\delta, \beta, 3)$-$\mathsf{RB\text{-}PPRF}$ protocol. This is a valid cuckoo hash table in this $\mathsf{RB\text{-}PPRF}$ instance too as the input sets and the universal hash functions used are the same across both these $\mathsf{RB\text{-}OPPRF}$ instances. Let $\widetilde{W}_1, \ldots, \widetilde{W}_\beta$ be the output lists received by P_0 from execution of $(\delta, \beta, 3)$-$\mathsf{RB\text{-}OPPRF}$ protocol.

Finally, the parties securely compute a circuit C_{PL} that is described as follows. The circuit C_{PL} takes as input HT_0, PU, $\{W_j\}_{j\in[\beta]}$ 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 $PU(j)$ 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 $x_j \in Y_j$ and in the second $(\delta, \beta, 3)$-$\mathsf{RB\text{-}OPPRF}$ 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 of this construction follows from the security of $\mathsf{RB\text{-}OPPRF}$ protocol.

5.2 Circuit-PSI via Dual Execution

In this section, we describe our linear complexity protocol for the scenario when Cuckoo hashing results in a stash[1]. Our overall idea is inspired by the dual execution idea from Pinkas et al. [37]. Our construction also makes use of the protocol from [37] in the "unbalanced set-size" setting - i.e., when P_0 and P_1 have unequal set sizes. Here, 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 will only make use of this protocol in a setting where one party, say P_0 has a very small set of size $\mathcal{O}(\log n)$ and other party, P_1, has a set of size n. We give the theorem and its corollary that we use in our construction.

Theorem 5 ([37]). *Consider parties Q_0 and Q_1 with input sets X and Y of size n_0 and n_1, respectively such that $n_1 \leqslant n_0$. Then, there exists a circuit-PSI protocol ([37, 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. The communication complexity of the protocol is $\mathcal{O}(n_0 + sn_1)$.*

Corollary 1 ([37]). *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 computational complexity $\mathcal{O}(n)$ and communication $\mathcal{O}(n)$.*

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 make use of Cuckoo hashing twice. First, P_0 uses Cuckoo hashing to map its elements in X into HT_0 and this can result in a stash. Let us denote the elements that fit in main table of HT_0 as X_T and elements in stash by X_S such that $|X_S| = \mathcal{O}(\log n / \log\log n)$. Second, 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 Y using h_1, h_2, h_3. Hence, each element in Y occurs at most thrice in HT_1 and also in sets $Y_1, \ldots, Y_\beta$, concatenated with different bin number. Now, P_1 acts as the sender in RB-OPPRF, and hashes the $n' = 3n$ elements in $Y_1, \ldots, Y_\beta$ using Cuckoo hashing that can lead to a stash. Let us denote the elements that fit in main table as Y'_T and stash elements by Y'_S. Let Y_S contain elements $y \in Y$, such that there exists $j \in [\beta]$ with $(y \| j) \in Y'_S$. It holds that $|Y_S| = \mathcal{O}(\log n / \log\log n)$. Let $Y_T = Y \setminus Y_S$.

Now we run four instances of circuit-PSI (in parallel) as follows: (1) Use our protocol described in Fig. 8 between elements in X_T and Y_T. (By the above construction it is guaranteed that there would be no stash when invoking this protocol[2].) (2) Use protocol given by Corollary 1 between elements in X_S and Y_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 X_T and Y_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 X_S and Y_S. This protocol has complexity, $|X_S| \cdot |Y_S|$, that is sub-linear in n.

It is easy to see the correctness and security of the above protocol. Moreover, it has linear computation and communication using Theorem 4 and Corollary 1.

6 Experimental Results

In this section, we compare the performance of our Circuit-PSI schemes $\mathsf{C\text{-}PSI}_1$ and $\mathsf{C\text{-}PSI}_2$ with the state-of-the-art Circuit-PSI scheme in literature [37], referred to as the PSTY scheme. For a comparison of PSTY with other prior circuit-PSI schemes [21,36,38,14], we refer the reader to [37, Section 7].

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. For $d = 3$ hash functions in the Cuckoo hashing, and for a

[1] Even though in all practical setting, if parameters are picked based on empirical analysis of [39] , no stash is observed.

[2] We take the mapping into the Cuckoo Table that resulted from above description.

hash table of size $\beta = 1.27n$ (i.e. $\varepsilon = 0.27$), see also Section 2.2, [39] empiricially determined that the failure probability (of not inserting an element into the hash main table) would be $< 2^{-40}$. Similar to PSTY, we use this empirical analysis to instantiate the parameters of Cuckoo hashing scheme. As discussed in Section 5.1, we set the output length ℓ of RB-OPPRF scheme (see Section 3.2) as $\ell = 40 + \lceil \log(3\beta) \rceil$, to achieve failure probability $< 2^{-40}$ for the circuit-PSI protocol. Table 2 summarizes the minimum output bitlength ℓ for our RB-OPPRF scheme with varying set sizes. Finally, we use the computational security parameter, $\lambda = 128$, and the statistical correctness parameter, $\sigma = 40$.

Size of Input Set (n)	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}
RB-OPPRF output length ℓ	56	58	60	62	64

Table 2. The minimum output bit-length ℓ required for input set of size n in our RB-OPPRF construction.

Implementation Details. The underlying OPRF construction in our schemes $\mathsf{C\text{-}PSI}_1$, $\mathsf{C\text{-}PSI}_2$ and Table based OPPRF construction [27] (in PSM2) is instantiated using the protocol of Kolesnikov et al. [26], whose implementation is available at [33]. For initial hashing as well as in the implementation of our RB-OPPRF construction, we make use of the hash tables library from [32]. For implementing our protocol PSM1 and the protocol for $\mathcal{F}_{\mathsf{eq}}$ functionality in our PSM2 scheme, we make use of the implementation of OT-Extension protocols [22,25] and Bit-Triple generation protocol [11,41] available at [31]. We compare with the implementation of PSTY scheme available at [13].

Experimental Setup. 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. As is the case with PSTY, both 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 3 for varying input set sizes n and for inputs of arbitrary bitlength. Similar to PSTY, and unlike circuit-PSI protocols proposed in [21,36,38,14,23], 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). 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 RB-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 in the protocol.

Finally, unlike PSTY and our protocols, the recent linear computation protocol of [23] has a communication cost that varies with bit-length. For $\ell = 32$ and 64, $\mathsf{C\text{-}PSI}_2$ is $\approx 8.8 - 12.5\times$ more communication efficient than their protocol; see [23, Table 3]. For instance, they communicate 836 MB for input sets of size 2^{16} and element bitlength of 64, while we communicate only 65.4 MB.

6.2 Performance Comparison

In this section, we compare the run-times of our circuit-PSI protocols $\mathsf{C\text{-}PSI}_1$ and $\mathsf{C\text{-}PSI}_2$ with PSTY for varying input set sizes [3] and in both network settings (see Table 4). For each entry in Table 4, we report the median value across 10 executions.

[3] In PSTY implementation [13], polynomial interpolation is implemented in a prime field where the prime is the Mersenne prime $2^{61} - 1$. The Mersenne prime $2^{61} - 1$ ensures statistical security of atleast 40-bits for input sets

n	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}
PSTY [37]	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 3. Concrete 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. The total communication cost for scheme C-PSI$_i$ can be obtained by adding the communication cost of these common components to the communication incurred in the corresponding private set membership protocol PSMi.

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 4).

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. 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$_2$.

Network Setting	LAN			WAN		
n	PSTY [37]	C-PSI$_1$	C-PSI$_2$	PSTY [37]	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 4. Comparison of total run-time in seconds of our Circuit-PSI schemes C-PSI$_1$ and C-PSI$_2$ to [37] for n elements. The best values are marked in bold.

Breakdown of individual components. Next, we present a breakdown of the overall execution times in the PSTY and C-PSI$_2$ protocols (see Table 5). 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. 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 [10]) with the protocol realizing $\mathcal{F}_{\mathsf{eq}}$ in PSM2 protocol (see

of size upto 2^{20}. Using Mersenne prime $2^{61}-1$ in PSTY construction for input sets of size 2^{22}, one only obtains a 38-bit statistical security guarantee. 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}.

Figure 7). Such a protocol would indeed be the most frugal in terms of communication complexity (but not by much – only $\approx 1.2\times$ more communication efficient than $\mathsf{C\text{-}PSI}_2$). 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 54.52 s (i.e., $2.7\times$ slower than $\mathsf{C\text{-}PSI}_2$ protocol) in the LAN setting and 72 s ($1.7\times$ slower than $\mathsf{C\text{-}PSI}_2$ protocol) in the WAN setting.

LAN Setting						
Scheme	PSTY			$\mathsf{C\text{-}PSI}_2$		
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			$\mathsf{C\text{-}PSI}_2$		
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 5. Breakdown of runtimes in seconds (s) of PSTY [37] and $\mathsf{C\text{-}PSI}_2$ circuit-PSI schemes for input sets of size 2^{16}, 2^{20} and 2^{22}. Approximate percentage of the total run-time is provided in parenthesis for each component.

Threshold Private Set Intersection. In threshold PSI [16,20], the intersection is revealed only if the size of the intersection set is bigger/smaller than a certain threshold. The protocol obtained on extending the circuit-PSI protocol PSTY [37] to the threshold setting is the state-of-the-art threshold PSI protocol. Since our circuit-PSI protocols have a similar structure to PSTY, they can be extended in the same way as PSTY to compute threshold PSI, with similar improvements expected in the threshold setting as well.

References

1. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 535–548. ACM, 2013.
2. Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991.
3. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 489–518. Springer, 2019.
4. Gilles Brassard, Claude Crépeau, and Jean-Marc Robert. All-or-nothing disclosure of secrets. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 234–238. Springer, 1986.
5. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.

6. Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 34–63. Springer, 2020.
7. Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. In Dario Catalano and Roberto De Prisco, editors, *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*, volume 11035 of *Lecture Notes in Computer Science*, pages 464–482. Springer, 2018.
8. Geoffroy Couteau. New protocols for secure equality test and comparison. In Bart Preneel and Frederik Vercauteren, editors, *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, volume 10892 of *Lecture Notes in Computer Science*, pages 303–320. Springer, 2018.
9. Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 213–231. Springer, 2010.
10. Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015.
11. 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 *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017.
12. Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 789–800. ACM, 2013.
13. Encrypto Group. OPPRF-PSI. `https://github.com/encryptogroup/OPPRF-PSI`. Accessed: 2020-10-07.
14. Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. Private set intersection with linear communication from general assumptions. In Lorenzo Cavallaro, Johannes Kinder, and Josep Domingo-Ferrer, editors, *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society, WPES@CCS 2019, London, UK, November 11, 2019*, pages 14–25. ACM, 2019.
15. Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324. Springer, 2005.
16. Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2004.
17. Juan A. Garay, Berry Schoenmakers, and José Villegas. Practical and secure solutions for integer comparison. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*, volume 4450 of *Lecture Notes in Computer Science*, pages 330–342. Springer, 2007.
18. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*, pages 464–479. IEEE Computer Society, 1984.
19. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987.
20. Per A. Hallgren, Claudio Orlandi, and Andrei Sabelfeld. Privatepool: Privacy-preserving ridesharing. In *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*, pages 276–291. IEEE Computer Society, 2017.
21. Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *19th Annual Network and Distributed System Security Symposium, NDSS, 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society, 2012.
22. Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference,*

Santa Barbara, California, USA, August 17-21, 2003, Proceedings, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161. Springer, 2003.

23. Ferhat Karakoç and Alptekin Küpçü. Linear complexity private set intersection for secure two-party protocols. Cryptology ePrint Archive, Report 2020/864, 2020. `https://eprint.iacr.org/2020/864`.
24. Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. *SIAM J. Comput.*, 39(4):1543–1561, 2009.
25. Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 54–70. Springer, 2013.
26. Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 818–829. ACM, 2016.
27. Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1257–1272. ACM, 2017.
28. B. Kreuter. Secure multiparty computation at google. In *RWC*, 2017.
29. Yehuda Lindell. How to simulate it - a tutorial on the simulation proof technique. Cryptology ePrint Archive, Report 2016/046, 2016. `https://eprint.iacr.org/2016/046`.
30. Catherine A. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *Proceedings of the 1986 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 7-9, 1986*, pages 134–137. IEEE Computer Society, 1986.
31. mpc-msri. EzPC. `https://github.com/mpc-msri/EzPC`. Accessed: 2020-10-07.
32. Oleksandr-Tkachenko. HashingTables. `https://github.com/Oleksandr-Tkachenko/HashingTables`. Accessed: 2020-10-07.
33. osu-crypto. libOTe. `https://github.com/osu-crypto/libOTe`. Accessed: 2020-10-07.
34. Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In Friedhelm Meyer auf der Heide, editor, *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, volume 2161 of *Lecture Notes in Computer Science*, pages 121–133. Springer, 2001.
35. Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: Lightweight private set intersection from sparse OT extension. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 401–431. Springer, 2019.
36. Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In Jaeyeon Jung and Thorsten Holz, editors, *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, pages 515–530. USENIX Association, 2015.
37. Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 122–153. Springer, 2019.
38. Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 125–157. Springer, 2018.
39. Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.*, 21(2):7:1–7:35, 2018.
40. Michael O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 2005. `https://eprint.iacr.org/2005/187`.
41. Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. Cryptology ePrint Archive, Report 2020/1002, 2020. `https://eprint.iacr.org/2020/1002`.
42. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

43. Adi Shamir. On the power of commutativity in cryptography. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 582–595. Springer, 1980.
44. Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 1607–1626. ACM, 2020.
45. Moti Yung. From mental poker to core business: Why and how to deploy secure computation protocols? In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1–2. ACM, 2015.

A Prior BPPRF and OPPRF constructions

In Fig. 9, we present the polynomial based Batch Programmable Pseudorandom Function construction of [37].

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 BPPRF Construction [37].

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 [27]