Private Set Intersection with Linear Communication from General Assumptions

Brett Hemenway Falk
University of Pennsylvania

Daniel Noble
University of Pennsylvania

Rafail Ostrovsky
UCLA

May 14, 2018

Abstract

This work presents an improved hashing-based algorithm for Private Set Intersection (PSI) in the honest-but-curious setting. The protocol is generic, modular and provides both asymptotic and concrete efficiency improvements over existing PSI protocols.

If each player has m elements, our scheme requires only $O(m\lambda)$ communication between the parties, where λ is a security parameter. This is the first protocol to achieve PSI using only asymptotically linear communication under standard cryptographic assumptions and without Random Oracles. Our protocol also provides 10-15% reduction in communication costs under real-world parameter choices.

Our protocol builds on the hashing-based PSI protocol of Pinkas et al. (USENIX 2014, USENIX 2015), but we replace one of the sub-protocols (handling the cuckoo "stash") with a special-purpose PSI protocol that is optimized for comparing sets of unbalanced size. This brings the asymptotic communication complexity of the overall protocol down from $\omega(m\lambda)$ to $O(m\lambda)$, and provides concrete performance improvements over the most efficient existing PSI protocols.

Our protocol is simple, generic and benefits from the permutation-hashing optimizations of Pinkas et al. (USENIX 2015) and the Batched, Relaxed Oblivious Pseudo Random Functions of Kolesnikov (CCS 2016).

1 Introduction

Private Set Intersection (PSI) is a secure computation protocol that allows two parties, who each hold a private list of elements from some universe $\mathcal{U}$, to compute the intersection of their (private) lists, without revealing information about the elements outside the intersection. PSI is an important cryptographic tool, and is a building block for many more complex functionalities and thus has received a lot of attention from the cryptographic community. In this work, we focus on PSI in the honest-but-curious setting, and thus we focus on comparing our protocol to other protocols that target the honest-but-curious security model.

A simple, generic method for privately computing set intersection would be to securely compute all pair-wise comparisons between the elements. If each player has m elements, this would require m^2 comparisons. The number of comparisons can be reduced by first hashing the elements into bins. The players would agree on a hash function $h : \mathcal{U} \to [n]$, and then hash their elements into bins, where each bin is of size b. Then for each bin, the players can perform a secure comparison of all the elements. This basic hashing protocol requires nb^2 secure comparisons. If $n = O(m)$, then with high probability, the maximum bin size is $\log n / \log\log n$, so this would require $n\left(\log n/\log\log n\right)^2$ secure comparisons. If, instead of performing all pair-wise comparisons in each bin, we recursed, hashing each bin into sub-bins, we obtain a protocol that requires $O(n \log n)$ secure comparisons.

This scheme can be improved by replacing one player's hash table with a cuckoo hash table [PSZ14, PSSZ15, KKRT16]. In this modification, the players choose two hash functions, and Alice hashes each of her elements into two bins, and Bob uses uses the two hash functions to hash his elements into a cuckoo-hash table with a stash. Then, for each location in Bob's cuckoo hash table, the players compute a secure comparison between the single element in that bucket and every element in Alice's corresponding bucket. Finally, they compare every element in Bob's stash against every element in Alice's table. Since Bob only has a single

element in each of his buckets, this method only requires $O(m)$ comparisons to compare Bob's cuckoo table against Alice's table. Unfortunately, comparing Bob's stash (of size s) against Alice's table still requires $O(ms\lambda)$ communication in the protocols of [PSZ14, PSSZ15, KKRT16]. The entire protocol then requires $O(ms\lambda)$ communication, and somewhat counterintuitively, the asymptotic communication complexity of the protocol is dominated by computing the intersection with the small ($\omega(1)$-sized) stash.

In this work, we make three improvements to this hashing-based PSI protocol

1. We observe that Bob's hashing protocol does not need to support dynamic insertions and deletions, and thus we can replace the cuckoo-hashing scheme with a 1-out-of-k hashing scheme, and compute the optimal allocation of his elements in an offline pre-processing phase. Although the size of the hash table, n, remains $n = O(m)$, this allows us to shrink the constants, and remove some of the heuristics from the failure probability analysis. This is described in Section 4.

2. We show how to use efficient protocols for unbalanced PSI [KLS$^+$17, CLR17] to reduce the communication complexity of comparing Bob's stash to Alice's set. This provides both asymptotic and practical improvements in communication complexity of existing PSI protocols. Our protocol reduces the communication cost of the stash-comparison step from $\omega(m\lambda)$ to $O(m\lambda)$, thus reducing the communication complexity of the entire protocol from $\omega(m\lambda)$ to $O(m\lambda)$.

   We give details of our construction in Section 5, and concrete performance numbers in Section 9.

3. Finally, we show how to modify the protocol so that the participants learn only a secret-sharing of the intersection, rather than the intersection itself. This is necessary for many secure computations that use PSI as a building block (e.g. secure computation of cross-tabs and secure database joins). Our protocols exhibit better asymptotic communication complexity than existing generic PSI protocols for computing secret sharings of an intersection. We give details of this protocol in Section 8.

Together, these modifications yield an extremely efficient PSI protocol based on general assumptions that achieves linear communication complexity. In addition to the asymptotic improvements in communication complexity outlined above, our protocols also provide concrete improvements in communication complexity for real-world set sizes. In Section 9 we calculate the actual communication cost of our protocol, and identify the set sizes at which our modifications begin to offer concrete improvements in communication cost.

2 Previous work

There have been many approaches to the problem private set intersection (PSI), and early works focused on building custom protocols to perform PSI (some examples include [FNP04, KS05, HL10, JL09, JL10, DSMRY09, DCT10, DCT12]). Over the years, many special-purpose PSI protocols have been proposed and implemented. Many of the early protocols were designed around Oblivious Polynomial Evaluation (OPE). In this framework, Alice interpolates a polynomial with roots at their elements, and then Alice and Bob work together to privately evaluate this polynomial at Bob's private elements. The private evaluation can be done using any additively homomorphic cryptosystem. Some PSI protocols that fall into this framework include [FNP04, HN10, KS05, DSMRY09]. Appendix A in [DCT12] provides a nice overview of many of the special-purpose PSI protocols.

In the face of the plethora of custom PSI protocols, [HEK12] proposed the idea, that *generic* (circuit-based) PSI protocols had many advantages, most notably that they were easy to implement and integrate into other, more complex secure computation protocols.

One natural approach is to use Oblivious Pseudo-Random Functions (OPRFs) [FIPR05]. An oblivious PRF is a protocol where Alice holds a PRF key κ, and Bob holds an input x, and the OPRF protocol allows Bob to learn $\mathrm{PRF}(\kappa, x)$ while Alice learns nothing about x.

OPRFs provide a natural method for a linear-communication PSI protocol in the semi-honest model. If Alice has a set X and Bob has a set Y, Alice will generate a key, κ, for an Oblivious PRF, and for every $x \in X$, they will use the OPRF protocol to give Bob $\mathrm{PRF}(\kappa, x)$. Then Alice will locally evaluate $\mathrm{PRF}(\kappa, y)$ for all $y \in Y$, and send these evaluations to Bob. Bob can then locally compute the intersection by comparing his evaluations to those received from Alice.

OPRFs can be implemented generically, using secure Multiparty Computation to compute an ordinary PRF, where Alice's private input is the PRF key, and Bob's private input is his evaluation point. This approach was taken in [PSSW09] where they used garbled circuits to obliviously compute the AES-based PRF. There have also been many special-purpose constructions of Oblivious PRFs designed specifically for set intersection protocols. The work of [HL10] shows how to instantiate an oblivious version of the Naor-Reingold PRF (based on the DDH assumption), and make it secure against malicious adversaries. The works [DCT10, DCT12] use the one-more RSA assumption in Random Oracle Model to build an OPRF-based linear-time PSI protocol, and the work of [JL10] uses an OPRF-based PSI protocol to provide security against malicious adversaries based on the one-more gap Diffie-Hellman problem in the Random Oracle Model. The work of [JL09] builds an OPRF secure in the standard model under the Decisional Composite Residuosity assumption.

In [KLS+17] it was observed that the OPRF-based PSI protocols are well-suited to applications where the parties hold sets unequal size. In particular, if Bob's set Y is much smaller than Alice's set X, then using an OPRF-based PSI protocol, they only need $|Y|$ OPRF calls, followed by $|X|$ communication. In particular, they show that the natural approach of using garbled circuits to implement an AES-based PRF is extremely efficient when Bob's set is sufficiently small. The recent work of [CLR17] uses leveled fully homomorphic encryption to create a PSI protocol for unbalanced set sizes.

Unfortunately, when the set sizes are roughly balanced, the generic OPRF protocols that use general-purpose MPC machinery to obliviously evaluate a PRF are not as efficient as the custom-PSI protocols, whereas the custom OPRF-based PSI protocols like [DCT10, DCT12, JL10] achieve linear complexity and practical efficiency, but under strong and non-standard cryptographic assumptions.

The work of [HEK12] identified three natural PSI protocols that could easily be implemented by an off-the-shelf MPC protocol. If the universe $\mathcal{U}$ of elements is known in advance, and is not too large, the players can simply encode their sets as characteristic vectors in $\{0,1\}^{|\mathcal{U}|}$, and then perform $|\mathcal{U}|$ secure bit-wise AND operations to compute their intersection (called the Bit-Wise And (BWA) protocol). When the universe is not known in advance (or is too large), the players can securely perform all pairwise comparisons (this requires m^2 secure comparisons to intersect two sets of size m), this is called the Pair-Wise Comparison (PWC) protocol, and is included only as a baseline, or straw-man protocol. Finally, they introduce the Sort-Compare-Shuffle (SCS) paradigm, where each player locally sorts their sets, then they engage in a secure computation to securely sort their joint set (using the bitonic sorting network). After the joint multi-set is sorted, all elements in the intersection will occur twice in two adjacent positions. Thus the intersection can be computed using $2m-1$ secure comparisons (comparing element i and $i+1$ for $i = 1, \ldots, 2m-1$). Finally, before the intersection can be revealed, it must be randomly shuffled (using the Waksman permutation network) to hide information carried by the position of the intersected elements. The bitonic sorting network requires $O(m \log m)$ comparisons to sort m elements, the Waksman permutation network requires $O(m \log m)$ gates (each of which could be implemented using a comparison) to randomly permute m elements and thus the total number of comparisons required by the SCS approach is $O(m \log m)$.

If the players agree a hash function (or hash functions), they can use the hash functions to locally sort their elements into bins, and then perform pairwise comparisons on the bins. In its simplest form, the players agree on a hash function, $h : \mathcal{U} \to [n]$, and some bucket size b. Then they locally hash their m elements into n buckets of size b. If any bucket receives more than b elements, the protocol will fail, so b must be chosen to be large enough so that this probability is sufficiently small. Then for each bucket, the players will engage in a secure computation to compare all elements within that bucket. If they use brute-force comparison within the bucket, this requires b^2 comparisons, and the entire protocol requires nb^2 comparisons to compute the intersection. If the players use the SCS method (described above) within each bucket, the number of comparisons drops to $O(nb \log b)$. If $n = O(m)$, then b must be $O(\log m / \log \log m)$, and the entire protocol is $O(m \log m)$.

The works of [PSZ14, PSSZ15, PSZ16] outline an optimization of this approach, where one player uses a traditional hash, while the other uses cuckoo hashing. To do this, Alice and Bob agree on k hash functions $h_1, \ldots, h_k$, and Alice hashes each of her elements into the k buckets defined by these hash functions. Alice's buckets will be sized to store as many elements as necessary. Bob, on the other hand, uses $h_1, \ldots, h_k$ to build a cuckoo hash table (with a stash), and hashes each element into this cuckoo hash table. For each of the n bins, Alice and Bob engage in a secure computation to compare the single element in that bin Bob's cuckoo hash table to the b elements Alice has in her bin. Finally, they compare each element in Bob's stash to

every one of Alice's elements. If the stash has size s, this requires $nb + ns$ secure comparisons. Using cuckoo hashing, we can set $n = O(m)$, $s = \omega(1)$, and as above $b = O(\log(m))$, and thus the protocol uses $O(m \log m)$ secure comparisons. The protocols of [PSZ14, PSSZ15] use an OT-masking protocol to replace the $(b + s)n$ secure comparisons to simply sending $(1 + s)n$ pseudo random masks. This reduces the communication complexity of these protocols to $\omega(n\lambda) = \omega(m\lambda)$. The primary improvement introduced in [PSSZ15] is the notion of *permutation-based hashing* which reduces the complexity of each secure comparison (but not the number of secure comparisons). Permutation-based hashing can be used to improve the performance of all the hashing-based PSI protocols (including ours), and we review the details of permutation-based hashing in Section 3.5. The performance of [PSSZ15] can be further improved by viewing the OT-based solution as a special OPRF-based solution, and instantiating it with novel, special-purpose OPRFs [KKRT16].

Bloom filters provide a natural, generic method for computing set intersections. If each participant inserts their m elements into a Bloom filter of size n, then they can use a secure bitwise-AND calculation to calculate the intersection of their Bloom filters. The players can locally query this "intersected" Bloom filter on each of their elements to find the intersection. This approach was taken in [MBD12]. It is straightforward to check that if an element appears in the intersection, it will also show up in the intersected Bloom filter. It is not too hard to see that the "intersected" Bloom filter created in this way may have extra ones that would not appear in a fresh Bloom filter created by inserting only the elements in the intersection of the two private sets. These extra ones, have the potential to leak information about the underlying sets, and thus this simple method of computing a set intersection using Bloom filters cannot be made to meet the security definitions of PSI.

Nevertheless, Bloom filters can be used to perform PSI. The protocol of [DCW13] introduces the notion of a garbled Bloom filter. In a traditional Bloom filter, an element x is inserted by ORing a 1-bit into the k locations defined by the hash functions $h_1, \ldots, h_k$. In a garbled Bloom filter, each entry holds a string (rather than a single bit), and an element x is inserted by secret-sharing x using a k-out-of-k secret sharing scheme, $(x = s_1 + \cdots + s_k)$ and inserting the shares s_i into the location determined by h_i. If the slot $h_i(x)$ is occupied, we *re-use* the existing share in that location. As long as one of the k slots is unoccupied, there will be enough freedom to make the s_i sum to x. If all k slots are occupied, then the insertion fails (just as in a regular Bloom filter). This approach can also be made secure against malicious adversaries [RR17].

The garbled Bloom filter can be made into a PSI protocol as follows. Alice will create a standard Bloom filter encoding her set, while Bob will create a garbled Bloom filter encoding his set. Denote these Bloom filters $A \in \{0,1\}^n$ and $B \in \left(\{0,1\}^\lambda\right)^n$. Then for each entry $i \in [n]$, if $A[i] = 0$, then set $C[i] \xleftarrow{\$} \{0,1\}^\lambda$. If $A[i] = 1$, then $C[i] = B[i]$. It is not hard to check that this is a garbled Bloom filter that encodes all elements in the intersection. The somewhat surprising result from [DCW13] is that this resulting garbled Bloom filter has exactly the same distribution as a garbled Bloom filter created by the intersection, and thus it leaks no information beyond the intersection. Implementing this PSI protocol using MPC requires n secure (single-bit) comparisons. Note that in the semi-honest setting, Alice can generate all the random values (for when $A[i] = 0$) and then Bob can receive the garbled Bloom filter, compute the intersection, and send the intersection to Alice, and thus there is no need to generate the random values within the MPC protocol. For a false-positive rate ϵ, a Bloom filter holding m elements needs to be of size $O(m \log(1/\epsilon))$, thus when $\epsilon = O(m^{-1})$, $n = O(m \log m)$.

Two concurrent, independent works [CO18, PSWW18] considered the problem of designing PSI protocols that do not reveal the intersection itself, but instead allow the players to compute *secret shares* of the intersection which could then be used in future secure computation protocols (we also consider this application in Section 8). Both these works build on the hashing-based PSI protocols of [PSZ14, PSSZ15, PSZ16]. In these hashing-based schemes, Bob uses cuckoo-hashing to hash each of his elements into one of k possible buckets, whereas Alice hashes each of her elements into all k potential buckets. Then, for each bucket, they must compare whether Bob's element matches one of Alice's elements in the corresponding bucket. Thus, for each bucket the players need a method for securely computing a *Private Set Membership* (PSM), where one party has a single element, and the other party has a large set. Prior works used OT to compute a one-time OPRF for each bucket [PSZ14, PSSZ15, PSZ16], but the hashing-based PSI protocols can thus be instantiated with any secure PSM protocol in place of the one-time OPRF. If the PSM protocol can be made to output secret-shares, rather than membership status in the clear, then the entire hashing-based PSI protocol can be made to output secret-shares, rather than the intersection in the clear. A generic MPC

calculation of set membership would require a number of equality tests equal to the size of the set, and would thus result in a PSI protocol that is fairly inefficient. The primary technical contribution of [CO18] is the development of a novel, efficient PSM protocol that outputs secret shares (or encryptions) of the membership query. At a high-level, their PSM protocol works as follows: the sender constructs a binary tree, each node of which contains a key for a symmetric key cryptosystem. If the tree has depth t (i.e., the sender's elements are bit-strings of length t), then the sender and receiver engage in t $\binom{2}{1}$-OTs, where the receiver's inputs are the bits of her element. This allow the receiver to traverse the tree obliviously, and learn a single value, corresponding to whether her element was in the sender's set. Using OT-extension, this PSM protocol has comparable efficiency to the one-time OPRF-based schemes of [PSZ14, PSSZ15, PSZ16]. Asymptotically, however, the protocol still requires $\omega(m\lambda)$ communication.

The work of [PSWW18] takes a different approach, and instead modifies the hashing structure, introducing the notion of two-dimensional cuckoo hashing. In the basic cuckoo-hashing scheme, Bob maps his element into one of k buckets, and Alice maps her elements to k-out-of-k buckets, thus ensuring that if there is an overlap in their sets, it will result in an overlap in exactly one of the buckets. The reason this approach is not amenable to a generic circuit-based protocol is that Alice's buckets may have many (about $\log m / \log \log m$) elements. The primary contribution of [PSWW18] is to introduce a new type of hashing scheme, where Bob maps his elements to 4-out-of-8 buckets, and Alice maps her elements to 2-out-of-8 buckets in such a way that: 1) both Alice and Bob have one element per bucket and 2) whenever Alice and Bob have elements that overlap, they will overlap in exactly one bucket. Then a bucket-by-bucket equality test can be instantiated using any generic MPC protocol. The overall complexity of this approach is then $\omega(m\sigma)$. As in the cuckoo hashing protocols of [PSZ14, PSSZ15, PSZ16] the protocol is $\omega(m\sigma)$ instead of $O(m\sigma)$ because the "stash" needs to be of size $\omega(1)$. In practice, however, they make do with a constant-sized stash, and this results in an extremely efficient circuit-based PSI that can then be implemented using any generic circuit-based MPC protocol.

3 Preliminaries

In this section, we review some of the basic functionalities needed for our constructions. These are all standard cryptographic primitives, and we assume the reader has some familiarity with them, and thus we only provide brief reviews. Formal definitions can be found in most cryptographic textbooks, e.g. [Gol01, Gol04].

3.1 Pseudorandom Functions

A pseudorandom function (PRF) is an efficiently computable, deterministic, keyed function with the property that for any adversary without the key, the outputs of the function $F_\kappa(\cdot)$ are indistinguishable from independent uniformly random values.

Definition 1 (PRF)**.** A deterministic function $F : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^m$, is a called a pseudorandom function (PRF) if it satisfies the security properties captured by the game below.

- The challenger generates a key, $\kappa \xleftarrow{\$} \{0,1\}^\lambda$
- The challenger uniformly chooses a bit $b \xleftarrow{\$} \{0,1\}$, and initializes a set $X = \emptyset$.
- The challenger and adversary engage in the following protocol where the adversary sends the challenger an input $x_i \in \{0,1\}^n$, and receives a response $y_i \in \{0,1\}^m$ until the adversary outputs a guess b'. The challenger generates its responses as follows
  - If $b = 0$, the challenger sets $y_i = F\kappa(x_i)$
  - If $b = 1$, the challenger checks if there is a pair $(x_i, y) \in X$, if so the challenger responds with the value y. If not, the challenger uniformly selects $y \in \{0,1\}^m$, and sets $X = X \cup (x_i, y)$, and returns y to the adversary.

The adversary is said to win the game if the adversary's guess b' is equal to the challenger's bit b. A PRF is said to be secure if any probabilistic polynomial-time adversary has a negligible (as a function of λ) probability of winning the above game.

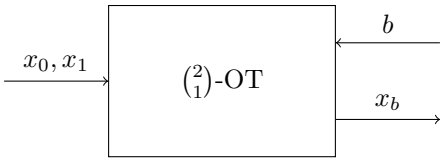

Figure 1: Oblivious Transfer. The sender provides two strings, x_0, x_1, and the receiver has a selection bit, b. The receiver receives a string x_b. The sender receives nothing.

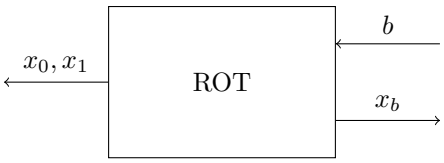

Figure 2: Random Oblivious Transfer. In the ROT functionality, the *sender* provides no input to the protocol, and instead the values x_0, x_1 are generated uniformly at random by the protocol itself.

3.2 OT

Oblivious Transfer (OT) [Rab81, EGL85] is a two party protocol that allows one party (Bob) to privately select one of two input strings held by the other party (Alice).

Definition 2 (OT)**.** Oblivious Transfer (OT) is a two party protocol that securely realizes the following functionality

inputs

- Alice inputs strings x_0, x_1
- Bob inputs a choice bit b

outputs

- Alice receives nothing
- Bob receives x_b

Sender Random-OT (ROT) is similar to oblivious transfer, except the random strings, x_0, x_1 are not provided by Alice, but instead is randomly generated by the protocol itself. Although ROT seems to be a weaker primitive than OT, they are known to be equivalent [Cré87].

It is known that OT is equivalent to ROT [Cré87], OT is symmetric (i.e., reversible) [WW06] and that OT is sufficient ("complete") for general secure multiparty computation [Kil88, IPS08]. OT can be constructed generically from many different cyrptographic primitives, including PIR [CMO00], projective hash proofs [Kal05, HK07], Dual-mode encryption [PVW08] and noisy channels [IKO$^+$11]. On the other hand, a black-box construction of OT from one-way permutations would imply $P \neq NP$ [IR89], (perfect) OT cannot be constructed from quantum mechanical processes [Lo97], and quantum mechanics doesn't even allow OT extension [SSS09, WW10].

One of the key features of OT is that it can be efficiently "extended." OT extension allows a small number of "base" or "seed" OTs to be extended into a huge number of OTs with low overhead in terms of computation and communication [IKNP03]. Thus, although OT is inherently a public-key primitive (OT implies public-key encryption, but not vice-versa [GKM$^+$00]), protocols that require a large number of OTs (e.g. [PSZ14, PSSZ15, PSZ16]) do not require a large number of public-key operations.

Figure 3: The Oblivious-PRF (OPRF) functionality. The sender provides a PRF key, κ, and the receiver provides an input, x. The receiver learns $F(\kappa, x)$, and the sender learns nothing.

3.3 OPRFs

An Oblivious pseudorandom functions (OPRF) is a two-party protocol for securely computing a PRF. The OPRF protocol securely realizes the functionality where one party (Alice) provides a PRF key, κ, and the other party (Bob) provides an input value, x. In the ideal functionality, Alice learns nothing, while Bob learns $F_\kappa(x)$. See Figure 3.

OPRFs were introduced in [FIPR05] as a means of achieving private keyword search, and since that time, many OPRF protocols have been introduced. A generic method for realizing the OPRF functionality is to use a general-purpose MPC protocol (e.g. garbled circuits) to implement a PRF.

Definition 3 (OPRF)**.** An oblivious pseudorandom function (OPRF) is a two party protocol that securely realizes the following functionality, where F is a cryptographically secure pseudorandom function (PRF).

inputs

- Alice inputs a key κ
- Bob inputs a value x

outputs

- Alice receives nothing
- Bob receives $F_\kappa(x)$

3.4 One-time OPRFs

A one-time OPRF is essentially an OPRF where the PRF key is randomly generated by the protocol (instead of specified by one of the players). In this sense, the relationship between a one-time OPRF and an OPRF is similar to the relationship between Random OT and OT. The qualifier one-time indicates that the one-time OPRF protocol can only be used to securely evaluate $F_\kappa(\cdot)$ once, since each additional run of the one-time OPRF will evaluate the PRF at a different key.

Definition 4 (one-time OPRF)**.** A one-time oblivious pseudorandom function (one-time OPRF) is a two party protocol that securely realizes the following functionality, where F is a cryptographically secure pseudorandom function (PRF).

inputs

- Alice inputs nothing
- Bob inputs a value x

outputs

- Alice receives a uniformly chosen key, κ
- Bob receives $F_\kappa(x)$

Figure 4: The one-time OPRF functionality. In the one-time OPRF functionality, the PRF key is not specified by the sender, but instead is generated by the protocol itself.

3.5 Permutation-based hashing

When hashing elements into buckets, the size of the representation of each element can be reduced using the notion of permutation-based hashing [ANS10, PSSZ15]. In general, it takes $\log |\mathcal{U}|$ bits to store an element $x \in \mathcal{U}$. In a traditional hash table, a hash function, $h : \mathcal{U} \to [n]$ is chosen, and the element x is stored in the location indexed by $h(x)$. Notice, however, that the bucket index, $h(x)$, carries $\log n$ bits of information, so it should be possible to reduce the information stored in the bucket from $\log |\mathcal{U}|$ bits to $\log |\mathcal{U}| - \log n$ bits, while still retaining the ability to uniquely recover an element x. This reduction in storage is possible, if we choose our hash function carefully. Permutation-based hashing provides a method for doing this, using a Feistel-style trick. Suppose an element x has bit-representation $x = x_1||x_2$, where x_1 has length $\log n$, and x_2 has length $\log \mathcal{U} - \log n$. If $f : \{0,1\}^{\log \mathcal{U} - \log n} \to \{0,1\}^{\log n}$, then we define $h(x) = x_1 \oplus f(x_2)$, and we store x_2 in the bin defined by $h(x)$. The crucial observation here is that if x and y are in the same bin, and the stored values are the same (i.e., $x_2 = y_2$), then that means $f(x_2) = f(y_2)$, and since $h(x) = h(y)$, we conclude that $x_1 = y_1$, which means $x = y$.

This trick allows us to reduce the size of the representation of elements within the bins, while still providing the property that if two elements have the same representation and are in the same bin, they must be equal. In the context of PSI, this means that secure computations are done on elements with smaller representations, and this can result in noticeable efficiency improvements (quantified in [PSSZ15]). The permutation-based hashing trick can be used in all hashing-based PSI protocols, including this one. In the situation where there are multiple hash functions, the ID of the hash function must also be stored in the bin to allow equality tests [Lam16].

4 Static (offline) hashing

Instead of cuckoo hashing, we propose using multiple-choice hashing [RMS01], where there are k hash functions, and each element is hashed into d buckets (using some choice of d of the hash functions) such that each bucket contains at most one element. A simple observation is that if player 1 uses a d_1-out-of-k hashing scheme, and player 2 uses a d_2-out-of-k scheme with $d_1 + d_2 > k$, then any element in the intersection must appear in at least one overlapping bucket. Then the intersection can be computed efficiently by simply comparing the buckets. The schemes of [PSZ14, PSSZ15, KKRT16] which use cuckoo hashing, essentially do this (in the online setting) with $d_1 = k$, and $d_2 = 1$. Our schemes will also focus on the case where $d_1 = k$, and $d_2 = 1$.

A simple variant of this idea is when $d_1 = d_2 = 2$, and $k = 3$, *i.e.*, both players use two-out-of-three hashing scheme, which guarantees that every common element will collide in at least one of its three buckets. Two-out-of-three hashing has been used to build data structures that support efficient set intersection queries in the setting where privacy is not a concern [AP11, EGMT17]. In [AP11], two-out-of-three hashing is used to build dynamic data structures (supporting insertions and deletions) that allow efficient set intersection queries, but they do not consider the notion of *private* set intersection. In [EGMT17], they consider the notion of two-out-of-three cuckoo hashing, which improves performance over standard two-out-of-three hashing.

Cuckoo hashing is a specific type of multiple-choice hashing procedure that is designed for *dynamically* adding elements to the hash table. In the PSI setting, however, both parties know their sets in advance, and can find the optimal static allocation. In fact, it remains an open question whether the dynamic cuckoo hashing insertion algorithm can obtain the optimum load threshold achievable by a static allocation [Mit09].

We review the bounds for offline hashing in Appendix A.1, and review an efficient algorithm for finding the optimal allocation in Section A.2.

5 Construction

At a high level, our construction is as follows: Alice and Bob choose k hash functions, $h_i : \{0,1\}^* \to [n]$, Then Alice hashes each of her elements into k buckets, and Bob uses multiple-choice hashing to hash each of his elements into 1 (out of k possible) buckets. Then they perform pairwise comparisons on the buckets. If k is constant (relative to m) we can set n to be $O(m)$ and the hashing will succeed with probability $1 - o(n)$. To make this failure probability negligible, we allow Bob to keep a super-constant sized "stash" of elements that could not be allocated to a single bucket. See Appendix A.1 for a more detailed analysis of the failure probability.

Then, we use secure comparison protocol (like those described in [PSZ14, PSSZ15, KKRT16]) to compare the element in each of Bob's buckets to the elements in Alice's corresponding bucket. Finally, we need to compare Bob's stash (of size $\omega(1)$) to Alice's elements (of size $O(m)$). To do this, we use an unbalanced PSI protocol like those described in [KLS+17].

1. Set $k \geq 1$, and $n = O(m)$, and the players choose k hash functions $h_i : \mathcal{U} \to [n]$.
2. Bob will hash his elements into n buckets using a 1-out-of-k hashing scheme, such that each bucket obtains at most one element. Elements that cannot be allocated to a single bucket will be put in a "stash", Stash_B, of size $m' = \omega(1)$. Bob can compute this allocation efficiently as described in Appendix A.2. Let $B[i]$ denote the element stored in Bob's ith bucket. Let $\mathsf{Stash}_B[i]$ for $i \in m'$ denote the ith element of the stash.
3. Alice will hash her elements into n buckets, using a k-out-of-k hashing scheme. Thus with high probability some of Alice's buckets will have $O(\log m / \log \log m)$ elements. Let $C[i]$ denote the number of elements in Alice's ith bucket, and let $A[i][j]$ denote the element in the bucket for $1 \leq i \leq n$, $1 \leq j \leq C[i]$.
4. Alice and Bob will engage in n parallel executions of a one-time-OPRF, where for $i \in [n]$, Alice learns a key κ_i, and Bob learns $S^i_B \stackrel{\text{def}}{=} \text{PRF}(\kappa_i, B[i])$. For each $i \in [n]$, Alice will locally compute $S^i_{A,j} = \text{PRF}(\kappa_i, A[i,j])$ for $j = 1, \ldots, C[i]$.
5. Alice will shuffle $\left\{S^i_{A,j}\right\}_{i \in [n], j \in [C[i]]}$ and send the shuffled set to Bob. Note that this set will have exactly km elements.
6. Bob will locally compute the intersection of his non-stash set with Alice's set by finding which of the S^i_B are in the set received from Alice.
7. To handle the stash, Alice will generate a key, κ, for an OPRF, and Alice and Bob will engage in m' executions of an OPRF protocol, where Bob learns $R^i_B \stackrel{\text{def}}{=} \text{PRF}(\kappa, \mathsf{Stash}_B[i])$ for $i \in m'$.
8. Alice will compute $R^i_A \stackrel{\text{def}}{=} \text{PRF}(\kappa, A[i][j])$ for $i \in [n]$, $j \in [C[i]]$ shuffle the set, and send it Bob who will locally compare these values to $\left\{R^i_B\right\}_{i \in m'}$ to find the intersection of Alice's set with the stash. Note that the set Alice sends will have exactly m elements.

Figure 5: The high-level outline of our algorithm

The communication cost of the protocol is the sum of the costs of the following:

- n parallel executions of a one-time-OPRF. The cost of this will depend on how the one-time-OPRF is instantiated.

- Alice will send km outputs of the one-time OPRF to Bob.
- $m' = O(\log n)$ secure computations of the Stash OPRF.
- Alice will send m outputs of the Stash OPRF to Bob.

In the next sections, we describe alternative methods for implementing the one-time OPRF using methods from [PSSZ15] and [KKRT16], and how to implement the OPRF using methods from [KLS+17].

5.1 An OT-masking-based protocol

In this section, we review the OT-masking-based OPRF protocol from [PSZ16]. When instantiated with OT, this protocol implements a standard (multi-time) OPRF, when instantiated with ROT, this protocol implements a one-time OPRF, which is sufficient for the PSI applications.

1. Bob will represent his element, $B[i]$, (the contents of the ith bucket) as a bit vector of length t, including a tag for which of the k hash functions was used. Using permutation-based hashing, this can be done with $t = \log_2 |\mathcal{U}| - \log_2 n + \log_2 k$.
2. Alice constructs a vector of length 2^t of random masks, M. This vector is the OPRF key (in fact, actually a truth-table for a truly random function). For input $j \in 2^t$, M_j is the evaluation of the OPRF on input j. Since Alice can choose the key, this is a normal OPRF rather than a one-time OPRF. If Alice and Bob are using ROT (instantiating a one-time OPRF) they can skip this step.
3. To evaluate the (one-time) OPRF on his input, $B[i]$, Alice and Bob engage in a $\binom{2^t}{1}$ string-OT (respectively $\binom{2^t}{1}$ string-ROT). Bob uses his input $B[i] \in 2^t$ as the input to the OT. From this, Bob learns $M_{B[i]}$, which is the evaluation of the PRF on $B[i]$.

Figure 6: Implementing OPRF using Oblivious Transfer

This protocol requires a single $\binom{2^t}{1}$-string OTs (for random strings) for each bucket, and thus a total of n $\binom{2^t}{1}$-string OTs. Using OT extension [IKNP03], these can be generated using λ base OTs.
The OT-based OPRF instantiation of [PSZ16] (outlined above) differs from the (flawed) original OT-masking approach outlined in [PSZ14, PSSZ15]. See Appendix B for a review of the original approach.

6 Batching OPRFs

In this section, we describe how to use our hashing scheme with the Batched, related-key OPRFs (BaRK-OPRFs) introduced in [KKRT16]. At a high level, this protocol is very similar to the OT-based protocol described in Section 5.1. As this is the most efficient instantiation of the one-time OPRF used in our scheme we provide a description in more detail below.
Before outlining the actual construction of [KKRT16], we review some of the necessary terminology. First, a *relaxed*-PRF is a pair $(F, \tilde{F})$ where F is a PRF such that 1) $F(\kappa, x)$ can be computed from $\tilde{F}(\kappa, x)$ and 2) $\tilde{F}(k, x)$ does not improve the adversary's distinguishing advantage in the PRF security experiment. In other words, given query access to $\tilde{F}(k, \cdot)$, $F(\kappa, \cdot)$ appears pseudo-random on all unqueried points.
The original OT-extension protocol of [IKNP03] relied on a *correlation-robust* hash function. In [KKRT16] the notion of correlation-robustness is extended as follows.

Definition 5 (k-Hamming Correlation Robustness [KKRT16])**.** Let $H : \{0,1\}^* \to \{0,1\}^v$ be a hash function

such that for all $z_i \in \{0,1\}^*$, and $a_i, b_i \in \{0,1\}^n$ for $i = 1, \ldots, m$, with $\mathrm{wt}(b_i) \geq k$, we have

$$\left\{ \{H(z_i || a_i \oplus [b_i \cdot s])\}_{i \in [m]} \;\middle|\; s \xleftarrow{\$} \{0,1\}^n \right\}$$
$$\approx$$
$$\left\{ \{r_i\}_{i \in m} \;\middle|\; r_i \xleftarrow{\$} \{0,1\}^v \right\}$$

The definition of correlation robustness in [IKNP03] required $k = n$, i.e., $b_i \cdot s = s$.
A pseudo-random code is a relaxation of an error-correcting code such that for any two distinct messages, their encodings have a large minimum distance with high probability over the choice of a specific code from the family.

Definition 6 (Pseudo-Random Code [KKRT16])**.** A family of functions, $\mathcal{C}$, is called a (d, ϵ) pseudorandom code (PRC) if for all strings $x \neq x'$,

$$\Pr_{C \leftarrow \mathcal{C}} \left[\mathrm{wt}\left(C(x) \oplus C(x')\right) < d \right] \leq 2^{-\epsilon}$$

The key security definition of [KKRT16] is the notion of related-key, relaxed PRFs. These are PRFs that remain secure when the challenger chooses n keys, $\kappa_1, \ldots, \kappa_n$, and the adversary sees the *relaxed* output for each key, then any m additional outputs (of the PRF, F) corresponding to any of the keys are indistinguishable from random.

Definition 7 (m-related key PRFs [KKRT16])**.** An m-related key PRF is a pair of functions $F, \tilde{F}$, and security is defined relative to the following game:

1. The adversary chooses input strings $\{x_j\}_{j \in [n]}$ and pairs $\{(j_i, y_i)\}_{i \in [m]}$, with $j_i \in [n]$ and $y_i \neq x_{j_i}$ for $i \in [m]$.
2. The challenger chooses PRF keys $k^*, k_1, \ldots, k_n$ and a challenge bit $b \in \{0,1\}$.
   - If $b = 0$, the challenger sends $\left\{ \tilde{F}((k^*, k_j), x_j) \right\}_j$ and $\{F((k^*, k_{j_i}), y_i)\}_i$ to the adversary.
   - If $b = 1$, the challenger generates m random strings $z_i \leftarrow \{0,1\}^v$, and sends $\left\{ \tilde{F}((k^*, k_j), x_j) \right\}_j$ and $\{z_i\}_i$ to the adversary.
3. The adversary outputs a guess b', and the adversary wins if $b' = b$. We say the adversary's advantage is $\Pr[b' = b] - 1/2$.

We say the pair $F, \tilde{F}$ is secure if the adversary's advantage is negligible. Intuitively, the pair $F, \tilde{F}$ is an m-related key PRF if the relaxed output, $\tilde{F}$, on n distinct keys, does not reveal information that would allow an adversary to distinguish F from a true random function, even if the inputs are arbitrarily correlated across keys.

Related-key PRFs can be efficiently instantiated using a pseudo random code, and a correlation robust hash function as follows.

Lemma 1 (Lemma 5 in [KKRT16])**.** If C is a $(d, \epsilon + \log m)$-pseudo random code, and $2^{-\epsilon}$ is negligible and H is a d-Hamming correlation robust hash function, then

$$F\left(((C, s), (q, j)), r\right) = H(j || q \oplus [C(r) \cdot s])$$
$$\tilde{F}\left(((C, s), (q, j)), r\right) = (j, C, q \oplus [C(r) \cdot s])$$

is an m-related key PRF as in Definition 7.

1. Alice chooses a random PRC $C \xleftarrow{\$} \mathcal{C}$ and sends the code to Bob.
2. Alice chooses a key $s \xleftarrow{\$} \{0,1\}^k$.
3. Bob generates two matrices $T_0, T_1 \in \{0,1\}^{m \times k}$ as follows. For $j = 1, \ldots, m$,
   (a) The jth row of T_0 is generated uniformly at random $t_{0,j} \xleftarrow{\$} \{0,1\}^k$.
   (b) The jth row of T_1 is defined as $t_{1,j} = C(r_j) \oplus t_{0,j}$.

   The ith columns of T_0 and T_1 are denoted t_0^i, t_1^i respectively.
4. Alice and Bob engage in k parallel instances of $\binom{2}{1}$-OT for strings of length m as follows:
   - Bob acts as sender and his inputs are the k columns of T_0 and T_1, i.e., Bob's inputs to the OT are $\left\{t_0^i, t_1^i\right\}_{i \in k}$
   - Alice acts as receiver and her inputs are the k bits of s, i.e., $\{s_i\}_{i \in k}$
   - Alice receives k outputs (each of length m) denoted $\left\{q^i\right\}_{i \in k}$

   Alice creates the $m \times k$ matrix Q whose columns are the received vectors $\left\{q^i\right\}$. Thus the ith column of Q is $t_{s_i}^i$. Let q_j denote the jth row of Q. Then
   $$q_j = ((t_{0,j} \oplus t_{1,j}) \cdot s) \oplus t_{0,j} = t_{0,j} \oplus (C(r_j) \cdot s)$$
5. For $j \in [m]$, Alice keeps the PRF seed $((C, s), (j, q_j))$
6. For $j \in [m]$, Bob keeps the relaxed PRF output $(j, C, t_{0,j})$.

Figure 7: Instantiating a BaRK-OPRF using OT [KKRT16].

At the end of the protocol in Figure 7, Alice has the keys $k^* = (C, s)$, and $k_j = (j, q_j)$, which allows her to evaluate the BaRK-OPRF

$$F((k^*, k_j), r) = F((C, s), (j, q_j)), r) = H(j || q_j \oplus [C(r) \cdot s])$$

for any r, and Bob has the relaxed PRF outputs

$$(j, C, t_{0,j}) = (j, C, q_j \oplus [C(r_j) \cdot s]) = \tilde{F}((k^*, k_j), r_j)$$

which allows him to compute $F((k^*, k_j), r_j)$.

The BaRK-OPRF protocol allows Alice and Bob to securely compute n parallel, one-time OPRFs in a very efficient manner. Since the protocol is inherently "batched," it does not fit exactly into the one-time OPRF paradigm described in Figure 5. Conceptually, the idea is essentially the same. Alice and Bob will engage in a BaRK-OPRF protocol, where Bob learns $\tilde{F}((k^*, k_j), B[i])$ for $i = 1, \ldots, n$, which allows him to learn the PRF outputs $F((k^*, k_j), B[i])$. Alice will learn the PRF keys, $k^*, k_1, \ldots, k_n$.

1. Bob has inputs $B[i]$, for $i = 1, \ldots, n$
2. Alice has no inputs
3. Alice and Bob will use the BaRK-OPRF protocol (Figure 7) with Bob providing the contents of his n buckets as his inputs. At the end of the protocol, Alice has keys:
   $$k^*, k_1, \ldots, k_n$$

and Bob has relaxed PRF outputs $\tilde{F}((k^*, k_i), B[i])$

4. From the relaxed outputs, Bob can comput $S_B^i \stackrel{\text{def}}{=} F((k^*, k_i), B[i])$.
5. From the PRF keys $k^*, k_1, \ldots, k_n$, Alice can compute $F((k^*, k_i), x)$ for any x of her choosing.

Figure 8: Using BaRK-OPRFs to instantiate n parallel one-time OPRFs.

Concretely, our implementation will use the BaRK-OPRF protocol to compute the n parallel one-time OPRFs, and then use a different (multi-use) PRF for the stash comparison. See section 9.

7 Security

The PSI protocols outlined in Section 5 are formed by taking existing one-time OPRF-based PSI protocols (e.g. [PSZ14, PSSZ15, PSZ16, KKRT16]) and combining them (in parallel) with an OPRF-based PSI protocol (e.g. [KLS$^+$17]) to handle the unbalanced stash. The proof of security (against honest-but-curious) adversaries follows in a straightforward manner from the security of the two underlying protocols.

For completeness, we outline the security of our protocol here. We define security in the standard, simulation paradigm, i.e., a PSI protocol is secure if there exists an efficient (probabilistic polynomial-time) simulator that can simulate the view of each player given the output of the protocol alone.

In our basic protocol (Figure 5), Alice's view consists of n parallel executions of a one-time-OPRF (from which she receives n random keys), and one application of an OPRF (from which she receives nothing). Thus her view is completely independent of Bob's input and can be trivially simulated since her view consists solely of random strings. Bob's view consists of m PRF-outputs (provided by the one-time OPRF protocol), km PRF outputs (provided by Alice), then for the stash, Bob receives m' PRF outputs (provided by the OPRF protocol), and m PRF outputs (provided by Alice). Given the intersection of Alice and Bob's sets, Bob's view can be easily simulated by provided by provided $km+m$ random strings (corresponding to the one-time OPRF round) with the correct intersection pattern, and $m'+m$ random strings (corresponding to the OPRF round) with the correct intersection pattern.

8 Generic PSI protocols for computing sharings

In this section, we outline a generic PSI protocol with $O(mt \log\log m)$ communication for t-bit values, by combining a basic hashing scheme with the Sort-Compare-Shuffle (SCS) protocol [HEK12] to compare elements within each bucket. This provides asymptotic communication improvements over existing schemes, using only generic MPC techniques, and is naturally composable with larger secure computations.

The generic one-time OPRF protocol (Section 5) and its instantiation with BaRKs (Section 6) are very efficient, but they reveal the intersection to the players. In many situations, however, PSI is used as a sub-protocol in a larger secure computation, and the intersection itself should never be revealed (e.g. in secure database-joins). In these situations, the protocols outlined in Section 5 are not appropriate, and a different solution is needed.

Two concurrent, independent works also addressed the question of building PSI protocols that don't output the intersection in the clear, and thus are suitable for combination within larger MPC protocols. The works of [CO18, PSWW18] both build on the hashing-based PSI protocols of [PSZ14, PSSZ15, PSZ16] to obtain an "MPC-friendly" PSI protocol with $\omega(m\lambda)$ communication. Their solutions differ from each other, and from our solution which achieves $O(mt \log\log m)$ communication (where t is the bit-length of the strings).

8.1 Improving SCS via hashing

The hashing-based PSI protocols of [PSZ14, PSSZ15, KKRT16] all require the players transmit $(s+1)n$ PRF evaluations, where $n = O(m)$ is the number of hash buckets and s is the size of the stash. To achieve

negligible error probability, it must be that $s = \omega(1)$, however, in practice a small constant-sized stash is used (in fact, [KKRT16] *decreases* the stash size as n increases).
No matter how small the stash is, however, the protocols require communicating at least $O(mv)$ bits, where v is the output size of a PRF. To keep the probability of spurious collisions in the PRF to below $2^{-\sigma}$, v is set to $v = \sigma + 2\log n$.
In this section, we outline a generic PSI protocol with $O(mt \log\log m)$ communication for t-bit values, by combining a basic hashing scheme with the SCS protocol to compare elements within each bucket. This provides asymptotic communication improvements over existing schemes, using only generic MPC techniques, and this naturally composes with larger secure computations.
Our scheme is a simple hashing-based scheme, but we use a small number of hash buckets. In particular, instead of setting $n > m$ (the number of buckets is larger than the number of elements being hashed) we set $n = m/b$, for some $b > 1$, and allow each bucket to have $(1+\delta)b$ elements.

1. Set $n = m/b$, and let $h : \mathcal{U} \to [n]$ be a hash function
2. Alice and Bob will each hash all of their elements to n buckets, using the hash function, h. If any bucket contains more than $(1+\delta)b$ elements, Alice and Bob abort the protocol. Since the abort leaks information, we will show that this happens with probability that is negligible (in m).
3. For each bucket, Alice and Bob will engage in a 2-party secure computation, implementing the sort-compare-shuffle (SCS) protocol of [HEK12].

Figure 9: Combining Sort-Compare-Shuffle and hashing to reduce the asymptotic communication cost.

First, we examine the failure probability, *i.e.*, the probability that a bucket contains more than $(1+\delta)b$ elements. Fix a player, and a bucket, and let X_i denote the random variable that is 1 if the player's ith element lands in the chosen bucket, and 0 otherwise. Let $X = \sum_{i=1}^{m} X_i$ denote the number of elements that land in the given bucket. If we model h as a truly random hash function, each X_i is an independent random variable with $\Pr[X_i = 1] = E[X_i] = \frac{1}{n}$. Then, $E[X] = b$, and by a Chernoff Bound,

$$\Pr\left[X > (1+\delta)b\right] \leq e^{-\frac{\delta^2 b}{3}}$$

Taking a union bound over the 2 players and the n buckets, we find that the probability of abort (and hence information leakage) is upper bounded by

$$\frac{2me^{-\frac{\delta^2 b}{3}}}{b} < 2^{-\frac{\delta^2 b}{3} + \log m}$$

Asymptotically, setting $\delta = 1$, and $b = \log^2 m$, we have that the failure probability is negligible in m. Since the SCS protocol requires $O(tb \log b)$ AND gates to compare each bucket of size $O(b)$, the total number of AND gates is $O(mt \log\log m)$. Using the GMW protocol [GMW87], each AND gate can be implemented using 2 OTs. Using OT-extensions, the entire secure two-party computation can be implemented using computation and communication that is linear in the size of the circuit [IKOS08].
Concretely, to keep the overall failure probability below $2^{-\sigma}$, we set

$$\begin{aligned}
\frac{2me^{-\frac{\delta^2 b}{3}}}{b} &< 2^{-\frac{\delta^2 b}{3} + \log m} < 2^{-\sigma} \\
&\Rightarrow \log m - \frac{\delta^2 b}{3} < -\sigma \\
&\Rightarrow \frac{3(\log m + \sigma)}{\delta^2} < b
\end{aligned}$$

Next, we look at the communication cost. We assume that Alice and Bob's sets can be represented using t bits. If we use permutation-based hashing (as in [PSSZ15]), the elements can be represented in buckets

$\log(m)$	t	δ	b	Number of AND gates per element
8	12	0.88	130	370
8	16	0.62	263	499
8	20	0.52	378	621
12	16	0.85	150	378
12	20	0.61	295	507
12	24	0.51	421	631
16	20	0.83	169	386
16	24	0.60	327	515
16	28	0.50	465	640
20	24	0.81	190	392
20	28	0.59	361	523
24	28	0.80	211	399
24	32	0.58	395	530
28	32	0.78	233	405
28	36	0.58	430	537
32	36	0.77	255	411

Table 1: Number of AND gates required per element in the hashing-SCS protocol. m is the total number of elements, t is the bit length of each element, b is the expected bucket size, and $(1+\delta)b$ is the maximum alotted bucket size. Thus the overall computation requires m/b SCS sub-computations, each on sets of size $(1+\delta)b$.

using $t - \log n$ bits. Using the SCS protocol of [HEK12], to compare ℓ elements of length τ, requires

$$\begin{aligned}
&\frac{5}{3}\tau\ell\log\ell + 2\tau\ell + ((3\ell-1)\tau - \ell) - \frac{\tau\ell+\tau}{3} \\
&\qquad = \frac{5}{3}\tau\ell\log\ell + \frac{14}{3}\tau\ell - \frac{4}{3}\tau - \ell \\
&\qquad < \frac{5}{3}\tau\ell\log\ell + \frac{14}{3}\tau\ell \\
&\qquad = \frac{\tau\ell}{3}(5\log\ell + 14)
\end{aligned}$$

AND gates. In our situation, each bucket has $\ell = (1+\delta)b$ elements and each element is of length $\tau = t - \log n$ bits. Thus the calculation requires

$$\begin{aligned}
&\frac{\tau\ell}{3}(5\log\ell + 14) \\
&\quad = \frac{(t-\log n)(1+\delta)b}{3}(5\log((1+\delta)b) + 14) \\
&\quad = \frac{(t-\log n)(1+\delta)b}{3}(5\log((1+\delta)b) + 14) \\
&\quad = \frac{(t-\log m+\log b)(1+\delta)b}{3}(5\log(1+\delta) + 5\log b + 14)
\end{aligned}$$

AND gates. For given values of number of elements, m, bit-length, t, and a given error threshold (e.g. $\sigma = 40$), we can find the value of δ to minimize the total number of AND gates required. Table 1 summarizes the number of AND gates per element required to implement our hashing-SCS scheme (with $\sigma = 40$, and optimal choices of δ and b).

Thus for real-world parameter choices, the entire PSI protocol can be computed by a circuit using approximately $500m$ AND gates, and this circuit can be evaluated with constant overhead (independent of the

security paramter) using the generic techniques of [IKOS08].

9 Concrete communication benchmarks

In [KMW09] it was shown that under the assumption that the dynamic (online) cuckoo hashing algorithm achieves the optimal load, allocating m elements to $n = O(m)$ buckets with a stash of size s will succeed with probability at least $1 - O\left(n^{-s}\right)$. The protocols of [PSZ14, PSSZ15, KKRT16], set $n = 1.2m$. Thus, under the assumption that online cuckoo hashing achieves the optimal offline load, to achieve a negligible probability of failure, the stash size, s, must be $s = \omega(1)$ (e.g. $s = O(\log n)$).

In the context of PSI, a cuckoo-hashing failure reveals information about the players' sets, so the failure probability should be set below the security threshold. In the work of [PSZ14, PSSZ15], they set security threshold to be a constant 2^{-40} independent of the set sizes. In [PSSZ15], they empirically tested the failure probability with 2-way cuckoo hashing, and different stash sizes for m up to about 2^{14}. Then they extrapolated these failure probabilities up to larger set sizes, and used these values to choose the stash size (See [PSSZ15] Figure 2). The empirical values they found were consistent with the asymptotic failure rate of $O(n^{-s})$ found [KMW09], with an implicit constant of about 1. Thus to achieve a failure probability of 2^{-40}, they could set the stash size s to be about $s > 40/\log(n)$. The scheme of [KKRT16] uses similar stash parameters even though they are hashing with three hash functions instead of two.

In practice, fixing a concrete security parameter that does not increase with m, means the necessary stash size that *decreases* as the set sizes increases, whereas an asymptotic analysis (which assumes that the failure probability should be negligible in m) requires that the stash size *increase* as the set sizes increase. Thus this choice of a concrete security parameter means that the concrete and asymptotic *performance* metrics diverge as the set sizes increase.

For a hash table of size n, and a stash of size s, the hashing protocol of [KKRT16] requires n applications of a one-time OPRF, and the communication of n (truncated) PRF outputs in the main phase, and s one-time OPRF evaluations and the communication of ns PRF outputs in the stash phase.

Replacing the stash computation with an unbalanced PSI protocol based on a standard (reusable) OPRF, reducing the communication in the stash phase to s evaluations of an OPRF followed by sending n PRF outputs.

In practice, because equality testing is done by comparing the pseudorandom masks (PRF outputs), there is no need to transmit the entire mask. Instead, to achieve error probability less than $2^{-\sigma}$, it is sufficient to transmit only about $v = \sigma + 2\log m$ bits of the mask, and in this case, by the birthday bound, the probability of a spurious collision between masks is negligible in m. This is what is done in practice by [PSSZ15, KKRT16].

Let d denote the number of bits required to for a one-time OPRF application, and $d' > d$ be the number of bits required for an OPRF evaluation. Then our protocol obtains a concrete performance improvement whenever $sd' + n\lambda < sd + ns\lambda$. Ignoring the sd term, we obtain a concrete improvement whenever

$$\frac{sd'}{v(s-1)} < n$$

In [ARS+15], they report that a single (GMW-based) AES evaluation using ABY can be computed using only 170 kb of communication. Using their custom, "MPC-friendly" PRF, the garbled circuit can be computed using only 23 kb of communication ([ARS+15] Table 6). Similarly, 1024 garbled AES representations were computed using 185 Mb of communication ([KLS+17] Table 5]), which reduces to about 177 kb per AES circuit (including pregenerating the OTs). Table 2 summarizes the communication costs of these different OPRFs, and the set sizes at which our protocol starts to improve over the [KKRT16] protocol.

Thus using an off-the-shelf AES128 implementation, we start to see concrete improvements in communication costs by replacing the one-time OPRF protocol of [KKRT16] with a true OPRF, whenever the sets being compared have more than $2^{16} \approx 40,0000$ elements. Using the "MPC-friendly" PRF, LowMC [ARS+15], we start to see concrete efficiency improvements for sets of size $2^{12} \approx 6,000$. Note that these are very conservative estimates, since we are assuming the stash size, s, is minimal ($s = 2$), and we are ignoring the cost of s one-time OPRFs.

Figure 10 shows the overall reduction in communication when the BaRK protocol [KKRT16] is replaced with our protocol instantiated with LowMC. Figure 11 shows the overall reduction in communication (over

OPRF	Comm. Cost	Break-even point
AES (Obliv-C)	8 Mb	2^{24}
AES (GMW - ABY)	170 Kb	2^{16}
AES (Yao)	177 Kb	2^{16}
LowMC	23 Kb	2^{12}

Table 2: The minimum valueof n for which the OPRF instantiation has a lower communication cost than the one-time OPRF instantiation in [KKRT16] This table shows the communication cost of a single OPRF evaluation, and the minimum number of buckets, for which replacing the OPRF-based comparison of the stash in [KKRT16] would be improved by switching to this protocol. The AES Obliv-C benchmarks were obtained via our internal tests. The ABY benchmarks were taken from [ARS+15], the Yao benchmarks were taken from [KLS+17], and the LowMC benchmarks were taken from [ARS+15].

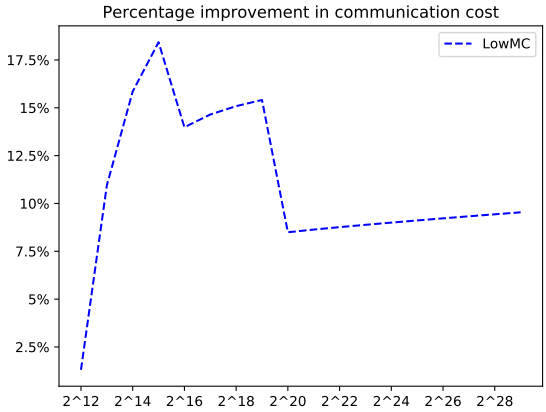

Figure 10: The percentage improvement in overall communication cost when modifying the BaRK protocol to use the LowMC-based PRF to compare the stash. For most values of m, our modification reduces the overall communication cost of the protocol by 10 to 15%. The points of non-differentiability in the graph correspond to the places where the stash sizes drop (we use the same stash sizes as [PSSZ15, KKRT16])

[KKRT16]) when our protocol is instantiated using different OPRF instantiations. Since the OPRF cost does not increase with m, all three instantiations of our protocols approach the same asymptotic improvement (about 10%) over the protocol of [KKRT16].

10 Conclusion

PSI is one of the most basic and fundamental types of secure computation, and numerous diverse types of PSI protocols have been proposed and implemented. Existing PSI protocols are highly optimized and extremely efficient.

In this work, we show how simple, modular composition of existing PSI protocols leads to both asymptotic and concrete improvements in efficiency. Our main result is showing that the one-time OPRF protocols of [PSZ14, PSSZ15, KKRT16] can be improved by replacing the stash-comparison step with an unbalanced PSI protocol [KLS+17].

We also show how combining a naive hashing-based PSI protocol with the Sort-Compare-Shuffle (SCS) protocol yields a generic (circuit-based) PSI protocol with better asymptotic efficiency over either solution. This solution is generic, can easily be adapted to support computing only the *secret-sharing* of the intersection set, and provides asymptotic efficiency comparable to the best honest-but-curious PSI protocols.

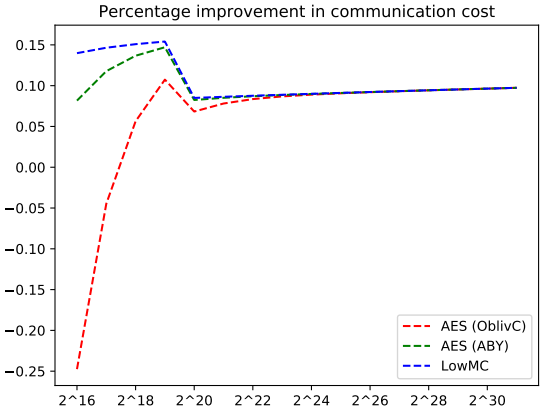

Figure 11: As m increases, and the stash size stays at 2, all three OPRF protocols tend towards a 10% improvement in communication cost over the BaRK protocol of [KKRT16]

References

[ANS10] Yuriy Arbitman, Moni Naor, and Gil Segev. Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 787–796. IEEE, 2010.

[AP11] Rasmus Resen Amossen and Rasmus Pagh. A new data layout for set intersection on gpus. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 698–708. IEEE, 2011.

[ARS+15] Martin R Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for mpc and fhe. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 430–454. Springer, 2015.

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. IACR ePrint 2017/299, 2017.

[CMO00] Giovanni D. Crescenzo, Tal Malkin, and Rafail Ostrovsky. Single Database Private Information Retrieval Implies Oblivious Transfer. In *Eurocrypt '00*, volume 1807 of *Lecture Notes in Computer Science*, pages 122–138. Springer Berlin / Heidelberg, 2000.

[CO18] Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. IACR ePrint 2018/105, 2018.

[Cré87] Claude Crépeau. Equivalence between two flavours of oblivious transfers. In *CRYPTO*, pages 350–354. Springer, 1987.

[DCT10] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography*, volume 10, pages 143–159. Springer, 2010.

[DCT12] Emiliano De Cristofaro and Gene Tsudik. Experimenting with fast private set intersection. *Trust*, 7344:55–73, 2012.

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 789–800. ACM, 2013.

[DSMRY09] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In *Applied Cryptography and Network Security*, pages 125–142. Springer, 2009.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[EGMT17] David Eppstein, Michael T Goodrich, Michael Mitzenmacher, and Manuel R Torres. 2-3 cuckoo filters for faster triangle listing and set intersection. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 247–260. ACM, 2017.

[FIPR05] Michael J Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, volume 3378, pages 303–324. Springer, 2005.

[FNP04] Michael J Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *International conference on the theory and applications of cryptographic techniques*, pages 1–19. Springer, 2004.

[GKM+00] Yael Gertner, Sampath Kannan, Tal Malkin, Omer Reingold, and Mahesh Viswanathan. The relationship between public key encryption and oblivious transfer. In *FOCS '00*, page 325, Washington, DC, USA, 2000. IEEE Computer Society.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *STOC*, pages 218–229. ACM, 1987.

[Gol01] Oded Goldreich. *Foundations of cryptography: volume 1*. Cambridge university press, 2001.

[Gol04] Oded Goldreich. *Foundations of cryptography: volume 2*. Cambridge university press, 2004.

[GW10] Pu Gao and Nicholas C Wormald. Load balancing and orientability thresholds for random hypergraphs. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 97–104. ACM, 2010.

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

[HK07] Dennis Hofheinz and Eike Kiltz. Secure hybrid encryption from weakened key encapsulation. In *Proceedings of the 27th annual international cryptology conference on Advances in cryptology*, CRYPTO'07, pages 553–571, Berlin, Heidelberg, 2007. Springer-Verlag.

[HL10] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. *Journal of cryptology*, 23(3):422–456, 2010.

[HN10] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. In *Public Key Cryptography*, volume 6056, pages 312–331. Springer, 2010.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Crypto*, volume 2729, pages 145–161. Springer, 2003.

[IKO+11] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and Jürg Wullschleger. Constant-Rate Oblivious Transfer from Noisy Channels. In Phillip Rogaway, editor, *CRYPTO '11*, volume 6841 of *Lecture Notes in Computer Science*, chapter 38, pages 667–684. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[IKOS08] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 433–442. ACM, 2008.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding Cryptography on Oblivious Transfer - Efficiently. In *CRYPTO '08*, pages 572–591, 2008.

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the Provable Consequences of One-Way Permutations. In *STOC '89*, pages 44–61. ACM, 1989.

[JL09] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In *TCC*, volume 5444, pages 577–594. Springer, 2009.

[JL10] Stanisław Jarecki and Xiaomin Liu. Fast secure computation of set intersection. *Security and Cryptography for Networks*, pages 418–435, 2010.

[Kal05] Yael T. Kalai. Smooth Projective Hashing and Two-Message Oblivious Transfer. In *EUROCRYPT '05*, pages 78–95, 2005.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, pages 20–31, New York, NY, USA, 1988. ACM.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 818–829. ACM, 2016.

[KŁ02] Michał Karoński and Tomasz Łuczak. The phase transition in a random hypergraph. *Journal of Computational and Applied Mathematics*, 142(1):125–135, 2002.

[KLS+17] Ágnes Kiss, Jian Liu, Thomas Schneider, N Asokan, and Benny Pinkas. Private set intersection for unequal set sizes with mobile applications. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 4:97–117, 2017.

[KMW09] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. *SIAM Journal on Computing*, 39(4):1543–1561, 2009.

[KS05] Lea Kissner and Dawn Song. Privacy-preserving set operations. In *Crypto*, volume 3621, pages 241–257. Springer, 2005.

[Lam16] Mikkel Lambæk. Breaking and fixing private set intersection protocols. IACR ePrint 2016/665, 2016.

[Lel12] Marc Lelarge. A new approach to the orientation of random hypergraphs. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 251–264. SIAM, 2012.

[Lo97] Hoi K. Lo. Insecurity of quantum secure computations. *Physical Review A*, 56:1154–1162, August 1997.

[LP14] Po-Shen Loh and Rasmus Pagh. Thresholds for extreme orientability. *Algorithmica*, 69(3):522–539, 2014.

[MBD12] Dilip Many, Martin Burkhart, and Xenofontas Dimitropoulos. Fast private set operations with SEPIA. Technical Report 345, ETH Zurich, march 2012.

[Mit09] Michael Mitzenmacher. Some open questions related to cuckoo hashing. In *ESA*, pages 1–10. Springer, 2009.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P Smart, and Stephen C Williams. Secure two-party computation is practical. In *Asiacrypt*, volume 9, pages 250–267. Springer, 2009.

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security Symposium*, pages 515–530, 2015.

[PSWW18] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based psi via cuckoo hashing. IACR ePrint 2018/120, 2018.

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

[PSZ16] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on OT extension. IACR Cryptology ePrint Archive, 2016.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A Framework for Efficient and Composable Oblivious Transfer. In David Wagner, editor, *Crypto '08*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2008.

[Rab81] Michael O Rabin. How to exchange secrets with oblivious transfer. Technical Report TR-81, Harvard University, 1981.

[RMS01] Andrea W Richa, M Mitzenmacher, and R Sitaraman. The power of two random choices: A survey of techniques and results. *Combinatorial Optimization*, 9:255–304, 2001.

[RR17] Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. In *EUROCRYPT*, pages 235–259. Springer, 2017.

[SSS09] Louis Salvail, Christian Schaffner, and Miroslava Sotáková. On the Power of Two-Party Quantum Cryptography. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '09, pages 70–87, Berlin, Heidelberg, 2009. Springer-Verlag.

[WW06] Stefan Wolf and Jürg Wullschleger. Oblivious transfer is symmetric. In *In Advances in Cryptology - EUROCRYPT '06*, pages 222–232, 2006.

[WW10] Severin Winkler and Jürg Wullschleger. On the Efficiency of Classical and Quantum Oblivious Transfer Reductions. In Tal Rabin, editor, *CRYPTO '10*, volume 6223 of *Lecture Notes in Computer Science*, pages 707–723, Berlin, Heidelberg, 2010. Springer Berlin / Heidelberg.

A Bounds for static (offline) multiple-choice hashing

In our protocol Bob uses a multiple-choice hashing scheme to hash his m elements into n buckets. In general, each element will be hashed into k out of d possible locations. In previous work, the [PSZ14, PSSZ15, KKRT16], Bob uses 1-out-of-k cuckoo hashing with a stash. Cuckoo hashing is designed for dynamic (online) hashing, whereas Bob is assumed to know his entire set at the time he makes the hash allocation.
Since the communication complexity of the resulting protocols depends on n (the number of hash buckets), we seek the minimum value of n that results in an acceptably low failure probability. Error bounds for multiple-choice hashing schemes have been well studied, and most results are obtained by analyzing the orientability of certain hypergraphs. In the following sections, we review some of the pertinant results.

A.1 Orienting hypergraphs

Given k hash functions, $h_i : \mathcal{U}[n]$, the problem of hashing m elements into n bins such that each bin contains at most b elements, and each element is hashed using d functions can be analyzed by translating the problem into one of orienting hypergraphs.
Formally, we have,

Definition 8 (Orientability of hypergraphs)**.** A k-uniform hypergraph $H = (V, E)$ is called (d, b)-orientable if there exists an assignment of each hyperedge $e \in E$ to exactly d of its vertices $v \in e$ such that no vertex is assigned more than b hyperedges.

Translating from our original context, hash-bucket will correspond to a vertex in the hypergraph (thus there are n vertices), and each element will correspond to an hyper-edge (thus there are m hyper-edges). The hyper-edge corresponding to an element $x \in \mathcal{U}$ will contain the k vertices corresponding to $h_1(x), \ldots, h_k(x)$. An (d, b) orientation of this hypergraph then corresponds to choosing a set of d hash buckets (vertices) for each element (hyper-edge) such that each vertex (bucket) is chosen at most b times.

k	d	b	$c_{k,d,b}$
3	2	1	.1666
5	3	1	.2119
7	4	1	.1747
9	5	1	.1453
11	6	1	.1236
13	7	1	.1074
15	8	1	.0948

Table 3: The threshold for orientability in random hypergraphs. If m is the number of elements, and n is the number of buckets, a random hashing scheme that hashes each element into d out of k buckets, where each bucket has size b will succeed with probability approaching one if and only if $m/n < c_{k,d,b}$. In particular, if we hash each element into 3 out of 5 buckets, then (with high probability) there will be no collisions as long as $n > 5m$.

The problem of orienting random hypergraphs is well-studied, and many deep results are known [KŁ02, GW10, Lel12, LP14].
[GW10] give asymptotic thresholds for (d, b)-orientability in terms of n and m, but their results only hold when b is "sufficiently large."
[AP11] show how a similar 2-out-of-3 hash structure can be used to compute set intersections in GPUs. There work focuses on the online setting, and they show the naive insertion algorithm has expected constant running time when $n = O(m)$.
[Lel12] gives asymptotic thresholds for (d, b)-orientability in terms of n and m, for almost all values of d, b. The main result of [Lel12] is the following: for any positive integers, k, d, b there is an explicit (although complicated) $c^* = c_{k,d,b}$ such that if $m > c^* n$ then the probability a random n, m, k hypergraph is (d, b)-orientable tends to 0 as $n \to \infty$, and if $m < c^* n$, this probability tends to 1.

Theorem 1 (Theorem 1 [Lel12])**.** For integers $k > d \geq 1$, and $b \geq 1$, then if ξ is the unique solution to

$$kd = \xi \frac{E\left[\max(d - \mathrm{Bin}(k, 1 - Q(\xi, b)), 0)\right]}{Q(\xi, b+1)\Pr\left[\mathrm{Bin}(k-1, 1-Q(\xi,b)) < d\right]}$$

and

$$c_{k,d,b} = \frac{\xi}{k \Pr\left[\mathrm{Bin}(k-1, 1-Q(\xi,b)) < \ell\right]}$$

where

$$Q(x, y) \stackrel{\text{def}}{=} e^{-x} \sum_{j=y}^{\infty} \frac{x^j}{j!}$$

and $\mathrm{Bin}(n, p)$ is the binomial distribution with $\Pr\left[\mathrm{Bin}(n,p) = k\right] = \binom{n}{k} p^k (1-p)^{n-k}$ then

$$\lim_{n\to\infty} \Pr\left[H_{n,m,k} \text{ is } (d,b)\text{-orientable }\right] = \begin{cases} 0 \text{ if } m > c_{k,d,b} n \\ 1 \text{ if } m < c_{k,d,b} n \end{cases}$$

In the case $d = k - 1$ and $b = 1$, we have $c_{k,k-1,1} = \frac{1}{k(k-1)}$.

A.2 Finding an optimal allocation

Given $2k-1$ hash functions, $h_i : \mathcal{U} \to [n]$, and m, we would like to find an allocation such that each of the m values is hashed into k of its possible locations, and each bucket contains only a single value. As described above, this corresponds to finding a $(k, 1)$ orientation on a $(2k-1)$-regular hypergraph (V, E).
If such an allocation exists, it can be found efficiently using a max-flow algorithm.

k	d	b	$c_{k,d,b}$
2	1	1	.4999
3	1	1	.9179
4	1	1	.9768
5	1	1	.9924
6	1	1	.9973

Table 4: The threshold for 1-out-of-k orientability in random hypergraphs. If m is the number of elements, and n is the number of buckets, a random hashing scheme that hashes each element into d out of k buckets, where each bucket has size b will succeed with probability approaching one if and only if $m/n < c_{k,d,b}$. In particular, if we hash each element into 1 out of 3 buckets, then (with high probability) there will be no collisions as long as $n > 1.09 \cdot m$.

Theorem 2. Given $2k-1$ hash functions, $h_i : \mathcal{U} \to [n]$, and $\{x_i\}_{i=1}^m \subset \mathcal{U}$, it is possible to find an allocation $A : [m] \to \mathcal{P}([2k-1])$ that allocates each element k hash functions, satisfying

1. Each element is allocated exactly k hash functions, i.e., $|A(i)| = k$ for all $i \in [m]$.
2. Each bucket has at most one element, i.e., for all $i \neq i' \in [m]$, $\{h_j(x_i)\}_{j \in A(i)}$ and $\{h_j(x_{i'})\}_{j \in A(i')}$ are disjoint.

and the allocation can be found in time $O(nk)$.

Proof. We begin by translating the problem to finding a $(k,1)$-orientation in an $(2k-1)$-regular hypergraph $G_{\mathsf{hyper}} = (V_{\mathsf{hyper}}, H_{\mathsf{hyper}})$, with n vertices and m edges. As described above, each bucket corresponds to an vertex in the hypergraph, and each element $x \in \mathcal{U}$ corresponds to a hyperedge, containing the $2k-1$ vertices $\{h_1(x), \ldots, h_{2k-1}(x)\}$.
Finding a $(k,1)$-orientation of this hypergraph corresponds to finding an orientation, where each hyper-edge "points" to k of its vertices, and each vertex has in-degree at most 1.
This problem can be translated to a problem on a flow network as follows.
Consider a network with node set with nodes for each vertex and each edge in the hypergraph, thus the set of nodes is $H_{\mathsf{hyper}} \cup V_{\mathsf{hyper}} \cup \{s,t\}$. For each edge in the hypergraph, put a capacity 1 edge connected the node corresponding to that edge with the $2k-1$ nodes corresponding to the vertices in V_{hyper} that it contains. Add capacity k edges from the source s to each vertex corresponding to an edge in H_{hyper}, and capacity 1 edges from each vertex corresponding to a vertex in V_{hyper} to the sink, t.
Now, note that any $|H_{\mathsf{hyper}}|(k)$ flow from s to t in the flow network induces a $(k,1)$ orientation in the hypergraph G_{hyper}. Conversely, any $(k,1)$ orientation of the hypergraph corresponds exactly to an $|H_{\mathsf{hyper}}|(k)$ flow from s to t in the induced flow network.
Using the Ford-Fulkerson algorithm, the max flow can be computed in time $O((|V_{\mathsf{hyper}}| + |H_{\mathsf{hyper}}|)(k)) = O((m+n)k) = O(nk)$. □

B OPRFs based on OT-masking

The works of [PSZ14, PSSZ15] outlined a method for constructing a Private Set Membership (PSM) protocol using an OPRF, and then instantiate the OPRF using OT-based masking.
If Bob has an input $x \in [N]^t$, and Alice has set of inputs $Y \subset [N]^t$), OT-masking protocol works as follows:

- Alice and Bob engage in t parallel invocations of $\binom{N}{1}$-ROT, with the digits of Bob's input acting as the selection string.
- Thus Alice receives $2t$ random "masks", $\left\{M_b^i\right\}$ for $i = 1, \ldots, t$, and $b \in \{0,1\}$.
- Bob receives t random masks $\left\{M_{x_i}^i\right\}$ for $i = 1, \ldots, t$.

- Bob computes his "PRF output"
$$S_B \stackrel{\text{def}}{=} \oplus_{i=1}^{t} M_{x_i}^i$$
as the XOR of his t masks.
- For each y in Alice's set, Alice computes the "PRF"
$$f(y) \stackrel{\text{def}}{=} \oplus_{i=1}^{t} M_{y_i}^i$$
i.e., Alice uses the t digits of y to select t of the masks, and XORs them together.
- Alice sends these m composite masks to Bob.
- Bob checks if his mask S_B is in the set received from Alice.

Although this "OT-masking" is secure as a "private equality test" (where Alice has only 1 input), it does not securely realize the one-time OPRF functionality (or the PSM functionality). In particular, the masks on Alice's elements are not independent, and thus given masks for y and y', Bob can check whether $y \oplus y'$ is in Alice's set.
This error appears to have been fixed in the full-version [PSZ16], where the outline a true OT-based one-time OPRF by having Alice and Bob engage in a single $\binom{N^t}{1}$-ROT, where Bob's input is his value $x \in [N]^t$, and Alice essentially receives the "truth-table" of a truly random function (with domain $[N]^t$).