Private Set Intersection with Linear Communication from General Assumptions

Brett Hemenway Falk
University of Pennsylvania
fbrett@cis.upenn.edu

Daniel Noble
University of Pennsylvania
dgnoble@cis.upenn.edu

Rafail Ostrovsky
UCLA
rafail@cs.ucla.edu

ABSTRACT

This work presents a 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.

Our protocol builds on the hashing-based PSI protocol of Pinkas et al. (USENIX 2014, USENIX 2015), but we replace one of the subprotocols (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 (10-15% reduction in communication costs) over Kolesnikov et al. (CCS 2016) under real-world parameter choices.

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 et al. (CCS 2016).

CCS CONCEPTS

• **Security and privacy → Cryptography**;

KEYWORDS

Private Set Intersection, PSI, Secure Multiparty Computation, Cryptographic Protocols

ACM Reference Format:
Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. 2019. Private Set Intersection with Linear Communication from General Assumptions. In *18th Workshop on Privacy in the Electronic Society (WPES'19), November 11, 2019, London, United Kingdom.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3338498.3358645

1 INTRODUCTION

Private Set Intersection (PSI) is a secure Multi-Party Computation (MPC) protocol that allows two parties, who each hold a private set of elements from some universe, $\mathcal{U}$, to compute the intersection of their (private) sets, 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 pairwise 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 $n\,(\log n/\log\log n)^2$ 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 [35, 43, 47]. In this modification, the players choose two hash functions, and Alice hashes each of her elements into two bins, and Bob uses the two hash functions to hash his elements into a cuckoo-hash table with a stash [32]. 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 [35, 43, 47]. 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 give a novel PSI protocol that builds on the cuckoo-hashing based approach of [35, 43, 47] by incorporating an efficient protocol for unbalanced PSI (e.g. [4, 5, 33, 50]) to reduce the communication complexity of comparing Bob's stash to Alice's set. This provides both asymptotic and practical improvements in communication complexity. 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 $O(m\lambda)$. These asymptotic efficiency gains do not

WPES'19, November 11, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6830-8/19/11…$15.00
https://doi.org/10.1145/3338498.3358645

rely on concrete number-theoretic hardness assumptions, but can be realized from OT alone.

In addition to the asymptotic improvements in communication complexity, this protocol provides concrete improvements in communication complexity in set sizes of about 4,000 elements. Many practical PSI applications involve sets of thousands or millions of elements. For instance the High Country Bandits case used the intersection of three cell tower dumps, of total size about 150,000, to find a phone that had been active near three different bank robberies [53]. Another example are the FBI-maintained NCIC hotlists, which combined contain over 12 million records such as license plates of stolen vehicles [14]. [1] These are routinely interesected with police agencies who record tens of thousands of license plates per day. At present the agencies are given permission to store the lists locally to perform the intersection, however, since the data is restricted to authorized users, the FBI may prefer for intersections with certain lists to be computed privately.

In Section 5.5 we calculate the actual communication cost of our protocol using different implementations of the unbalanced PSI subprotocol, and compare these with the PSI protocol of Kolesnikov et al. (CCS 16) [35].

We also observe that in the cuckoo-hashing schemes of [35, 43, 47] 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 guarantee that an optimal allocation is found. This is described in Section 4.2.

2 PREVIOUS WORK

There have been many approaches to the problem of private set intersection (PSI), and early works focused on building custom protocols to perform PSI (some examples include [9–11, 16, 20, 28, 29, 34]). 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 her elements, and then Alice and Bob work together to privately evaluate this polynomial at points given by Bob's (private) elements. The private evaluation can be done using any additively homomorphic cryptosystem. Some PSI protocols that fall into this framework include [9, 16, 21, 34]. These protocols typically have good communication complexity (each side sends $O(m)$ ciphertexts), but high computational cost (each side must do $O(m)$ public-key operations). See [11, Appendix A] for an overview of many of the special-purpose PSI protocols.

Another class of PSI protocols use Oblivious Pseudo-Random Functions (OPRFs) [15]. 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 $y \in Y$, they will use the OPRF protocol to give Bob $\mathrm{PRF}_\kappa(y)$. Then Alice will locally evaluate $\mathrm{PRF}_\kappa(x)$ for all $x \in X$, 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 [44] 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 [20] 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 [10, 11] use the one-more RSA assumption in Random Oracle Model to build an OPRF-based linear-time PSI protocol, and the work of [29] 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 [28] builds an OPRF secure in the standard model under the Decisional Composite Residuosity assumption.

In [33] it was observed that the OPRF-based PSI protocols are well-suited to applications where the parties hold sets of 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 [5] uses leveled fully homomorphic encryption to create a PSI protocol for unbalanced set sizes, and this work was later extended to the malicious setting [4]. The work of [50] uses cuckoo filters and a pairing-based public-key cryptosystem design and implements an efficient PSI protocol for unbalanced sets.

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 [10, 11, 29] achieve linear complexity and practical efficiency, but under strong and non-standard cryptographic assumptions.

In the face of the plethora of custom PSI protocols, [23] proposed the idea, that circuit-based PSI protocols built on general-purpose MPC have many advantages, most notably that they were easy to implement and integrate into other, more complex secure computation protocols. The work of [23] 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 merge their sorted sets (using the bitonic

[1] Hotlist contents are not public, so while we here provide an estimate on the total number of records, we do not know the sizes of individual hotlists.

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 [54]) to hide information carried by the position of the intersected elements. The bitonic sorting network[2] requires $O(m \log m)$ comparisons to merge two sorted lists of length m, 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 $O(m \log m)$.

Another class of protocols uses hashing. If the players agree on 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 [43, 47, 48] outline an optimization of this approach, where Alice uses a k-out-of-k hashing, while Bob 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 in Bob's cuckoo hash table to the b elements Alice has in her corresponding 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 $nb + ms$ 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 [43, 47] 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 [43] 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 4.1. The performance of [43] can be further improved by viewing the OT-based solution as a special OPRF-based solution, and instantiating it with novel, special-purpose OPRFs [35]. It appears that the communication of these OT-masking protocols can be improved by the use of "silent-OT" extension [3], but this would come at a cost in terms of computation, and to the best of our knowledge this modification has never been implemented. We remark, however, that our techniques could still be applied to further improve the communication cost if the traditional OT extension were replaced by silent OT extension.

The recent work of [42] improves the general cuckoo-hashing technique of [43, 47, 48], by replacing the OPRF with a *multi-point* OPRF. In [43, 47, 48] Bob uses cuckoo hashing to ensure that each of his hash buckets contains at most one element. This allows them to use a "one-time" OPRF for each bucket, where Bob learns the OPRF value calculated on the single element in his bucket, while Alice learns the OPRF key, and compute the values on all elements in her bucket. In [42], Alice and Bob use a one-time multi-point OPRF, which allows Bob to hash to buckets that can hold more than one element, and this drives down both the asymptotic communication cost (to $\Theta(m\lambda)$) and the concrete cost.

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 [39]. 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 [12] introduces the notion of a garbled Bloom filter. In a traditional Bloom filter, an empty set is represented by the all-zero vector. An element x is inserted by setting each of the k locations defined by the hash functions $h_1, \ldots, h_k$ to 1. 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 [51].

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 [12] is that this resulting garbled Bloom filter

[2] More precisely, this only uses the final "layer" of a bitonic sorting network, which merges two sorted lists using $O(m \log(m))$ comparisons. A full bitonic sorting network sorts an arbitrarily permuted list and requires $O(m \log^2(m))$ comparisons.

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 $\epsilon = O(m^{-1})$, $n = O(m \log m)$.

Several recent works [6, 45, 46] 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. Both these works build on the hashing-based PSI protocols of [43, 47, 48]. 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 [43, 47, 48], 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 [6] 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 allows 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 [43, 47, 48]. Asymptotically, however, the protocol still requires $\omega(m\lambda)$ communication.

The work of [46] 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 each of his elements 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 [46] is to introduce a new type of hashing scheme, where there are two tables and Alice hashes each of her elements into *one* of the two tables using a 2-out-of-2 hashing scheme, and Bob hashes each of his elements into *both* of the two tables using a 1-out-of-2 (cuckoo) hashing scheme. Then a bucket-by-bucket equality test can be instantiated using any generic MPC protocol. The overall communication complexity of this approach is then $\omega(m\lambda)$. As in the cuckoo hashing protocols of [43, 47, 48] the protocol is $\omega(m\lambda)$ instead of $O(m\lambda)$ because asymptotically Bob's "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. Note, however, that this "MPC-friendly" PSI protocol is significantly more communication intensive than existing "cleartext" PSI protocols (e.g., [43, 47, 48]).

The recent work of [45] also computes *secret shares* of the intersection, but has lower communication cost than [6, 46] both asymptotically ($\Theta(m)$ cost) and concretely. The protocol of [45] uses the same hashing techniques as [43, 47, 48], where Alice hashes her set into buckets using k-out-of-k hashing, and Bob uses 1-out-of-k (cuckoo) hashing to hash his elements into buckets, where each bucket holds a single element. Then, instead of doing one OPRF evaluation per bucket, they use a *programmable* OPRF, where, for each bucket, Alice programs the PRF to take the same value on all elements in her bucket. This programming can be easily achieved XOR-ing a traditional PRF output with a carefully interpolated polynomial. Then, they use a traditional MPC to do a single secure equality test for each bin (comparing only the PRF outputs). In this setting, where the results are secret shared, the stash can be handled, by simply reversing the roles of Alice and Bob, and having Bob input only the elements he previously mapped to the stash.

3 PRELIMINARIES

In this section, we review notation and some of the basic functionalities needed for our constructions. Our primitives are standard; we assume the reader has some familiarity with them, and thus we only provide brief reviews. Formal definitions can be found in cryptographic textbooks, e.g., [18, 19].

3.1 Common Notation

Below we show notation that is used repeatedly in the paper.

- $\mathcal{U}$ - The universe of possible set elements
- m - Number of elements in each set
- n - Number of bins
- b - Size of bins
- s - Size of stash
- h - Hash function mapping elements to bins, $h : \mathcal{U} \rightarrow [n]$
- λ - A security parameter
- σ - A parameter determining the failure probability
- k - Number of hash functions used
- κ- Key of a PRF, OPRF, or some variant thereof
- X - Alice's set $|X| = m$, $X \subseteq \mathcal{U}$
- Y - Bob's set $|Y| = m$, $Y \subseteq \mathcal{U}$
- F - A PRF, OPRF, or some variant thereof
- $A[i][j]$ - The j^{th} element in Alice's i^{th} bin
- $B[i]$ - The (unique) element in Bob's i^{th} bin
- $\mathsf{Stash}_B[i]$ - The i^{th} element of Bob's stash
- d - Cost of an OPRF (bits)

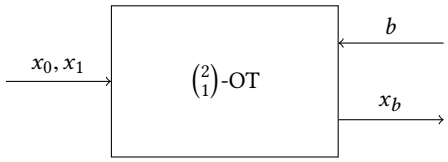

Figure 1: Oblivious Transfer. The sender provides two strings, x_0, x_1, and the receiver has a selection bit, b. The receiver receives a string x_b. The sender receives nothing.

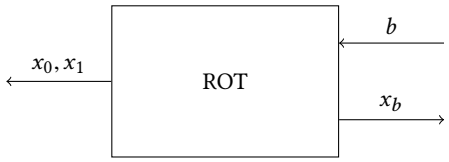

Figure 2: Random Oblivious Transfer. In the ROT functionality, the *sender* provides no input to the protocol, and instead the values x_0, x_1 are generated uniformly at random by the protocol itself.

d' - Cost of a one-time OPRF (bits, amortized)
t - Length of element bit-representation, typically $t = \lceil \log_2 |\mathcal{U}| \rceil$

3.2 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 3.1 (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.

3.3 OT

Oblivious Transfer (OT) [13, 49] 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 3.2 (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 strings x_0, x_1 are not provided by Alice, but instead are randomly generated by the protocol itself. Although ROT seems to be a weaker primitive than OT, they are known to be equivalent [7].

It is known that OT is symmetric (i.e., reversible) [56] and that OT is sufficient ("complete") for general secure multiparty computation [27, 31]. OT can be constructed generically from many different cryptographic primitives, including PIR [8], projective hash proofs [22, 30], dual-mode encryption [41] and noisy channels [26]. On the other hand, a black-box construction of OT from one-way permutations would imply $P \neq NP$ [24], (perfect) OT cannot be constructed from quantum mechanical processes [38], and quantum mechanics doesn't even allow OT extension [52, 55].

One of the key features of OT is that a small number of "base" or "seed" OTs can be extended into a huge number of OTs with low overhead in terms of computation and communication [25]. Thus, although OT is inherently a public-key primitive (OT implies public-key encryption, but not vice-versa [17]), protocols that require a large number of OTs (e.g., [43, 47, 48]) do not require a large number of public-key operations.

3.4 OPRFs

An Oblivious pseudorandom function (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 [15] 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 3.3 (OPRF). An Oblivious Pseudo-Random 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 $F_\kappa(x)$.

3.5 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

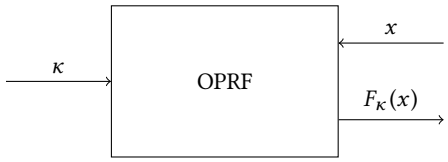

Figure 3: The Oblivious PRF (OPRF) functionality. The sender provides a PRF key, κ, and the receiver provides an input, x. The receiver learns $F_\kappa(x)$, and the sender learns nothing.

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.

each additional run of the one-time OPRF will evaluate the PRF at a different key.

Definition 3.4 (one-time OPRF). A one-time Oblivious Pseudo-Random 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 $F_\kappa(x)$.

3.6 Multiple-choice hashing and Cuckoo hashing

Multiple-choice hashing is a technique for representing a set S using an array A, that has efficient space utilization and allows constant look-up time. First, a set of hash functions $h_1, \ldots, h_k$ is chosen, where $h_i : \mathcal{U} \to [n]$. An element $x_j \in S$, is stored in an index $h_i(x_j)$ of A, for some $i \in [k]$. Look-ups of an element, x, simply require checking whether $A[h_i(x)] = x$ for any $i \in [k]$. Multiple choice hashing assumes that the entire set is known in advance, so items in the set can be allocated indices in A using a 2D matching protocol.

A related technique, Cuckoo hashing, handles situations where the entire set is not known in advance, or may be modified dynamically. Like in Multiple-choice hashing, an element $x \in S$ will be stored in $h_i(x)$ for some $i \in [k]$. The name Cuckoo hashing refers to the method used for insertion: a new element x will be placed in one of its allowed indices $h_i(x)$ and if the spot is already occupied it will, like a Cuckoo bird taking over a nest, evict the existing element from that location. The evicted element will then seek a new home from its allowed indices, possibly evicting yet another element in the process. This process continues until an element can move to a space that was previously empty, or a pre-determined threshold in the number of evictions is reached, in which case the insertion fails.

Many previous PSI works refer to Cuckoo hashing. However, as outlined in more detail in Section 4.2 in the PSI case the items *are* known in advance, so static multiple-choice hashing can be used instead of dynamic Cuckoo hashing. This means that an allocation can always be found if one exists.

However, there is the chance that no allocation exists. In the case where $k = 2$, if the table is of size $2(1+\epsilon)n$, for any $\epsilon > 0$, the probability that multiple-choice hashing fails is $\Theta(1/n)$, where the constant in the Theta-notation depends on ϵ [36]. In many situations, a failure probability of $\Theta(1/n)$ is not acceptable. A solution to this is to have a "stash", of maximum size s, in which to store any elements that were unable to be placed in the table. The probability of failure becomes $\Theta(1/n^{s+1})$ [40]. To acheive negligible failure probability, $s = \omega(1)$ is needed. To acheive a fixed statistical failure probability $2^{-\sigma}$, it is required that $s \geq \frac{\sigma}{\log(n)} + \frac{\log(c)}{\log(n)} - 1$ where c is the constant from the Theta-notation above.

4 TWO GENERIC OPTIMIZATIONS FOR HASHING-BASED PSI PROTOCOLS

Here we present two improvements which can be used for hashing-based PSI protocols in general. Both are orthogonal to the primary contributions of this paper and the rest of the paper should still make sense if this section is skipped.

4.1 Permutation-based hashing [2, 43]

When hashing elements into buckets, the size of the representation of each element can be reduced using the notion of permutation-based hashing [2, 43]. 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. First, we choose a public random permutation $\pi : \mathcal{U} \to \mathcal{U}$ and, for each element x in a set, compute $\hat{x} = \pi(x)$. Suppose $\hat{x}$ has bit-representation $\hat{x} = \hat{x_1}||\hat{x_2}$, where $\hat{x_1}$ has length $\log n$, and $\hat{x_2}$ has length $\log|\mathcal{U}| - \log n$. Let $f : \{0,1\}^{\log|\mathcal{U}|-\log n} \to \{0,1\}^{\log n}$ be a uniform hash function. Then we define $h(\hat{x}) = \hat{x_1} \oplus f(\hat{x_2})$, and we store $\hat{x_2}$ in the bin defined by $h(\hat{x})$. The crucial observation here is that if $\hat{x}$ and $\hat{y}$ are in the same bin, and the stored values are the same (*i.e.*, $\hat{x_2} = \hat{y_2}$), then that means $f(\hat{x_2}) = f(\hat{y_2})$, and since $h(\hat{x}) = h(\hat{y})$, we conclude that $\hat{x_1} = \hat{y_1}$, which means $\hat{x} = \hat{y}$ and therefore $x = y$.

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 [43]). The random permutation, $\pi(\cdot)$, is needed because $h(x)$ may no longer be a uniform hash function, even though $f(x)$ is. If the

input to h is correlated, this can increase the probability of a high number of hash-table collisions. The permutation guarantees that the inputs to h is uncorrelated. The permutation-based hashing trick can be used in all hashing-based PSI protocols, including ours. 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 [37].

4.2 Static (offline) hashing

In many hashing-based PSI functions, Bob will allocate each element one of k possible hash functions, and Bob wishes to choose an optimal allocation, namely one that minimizes the number of elements that need to be stored in his stash.

Existing PSI protocols [35, 43, 47] have used *online* Cuckoo hashing to compute Bob's allocation. Cuckoo hashing is designed to allow *dynamic* insertion of elements, but we observe that in the PSI setting, both parties know their sets in advance. In particular, Bob can allocate hash functions to elements based on knowledge of his entire set.

Instead of using dynamic cuckoo hashing, Bob can statically calculate an optimal allocation. He does this by treating the problem as a 2D matching problem (matching elements to potential bins). For dynamic cuckoo hashing, however, it remains an open problem of whether an optimal allocation can be found [40]. As such this simple modification provides a guarantee that hash functions are allocated optimally. It also eases reasoning about the probability of failure for a given stash size.

5 LINEAR COMMUNICATION PSI VIA HASHING AND OPRFS

Here we present a PSI protocol with linear communication cost. In short, it does this by combining a hashing-based PSI protocol with an unbalanced PSI scheme for handling the stash. Previous PSI protocols with linear communication tended to rely on oblivious polynomial evaluation, and instantiated the idea with an additively homomorphic cryptosystem [9, 16, 21, 34] or require concrete number-theoretic assumptions [10, 11, 20, 28, 29]. By contrast, our protocol uses a hashing-based PSI protocol (like [35, 43, 47]) which can be instantiated from OT, combined with an OPRF (which can also be instantiated with OT).

5.1 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 bins, and Bob uses multiple-choice hashing to hash each of his elements into 1 (out of k possible) bins. Then they perform pairwise comparisons on the bins. For constant k 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 bin.

Then, for each of Bob's bins we use secure comparison protocol (like those described in [35, 43, 47]) to compare the element in Bob's bin to the elements in Alice's corresponding bin. 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 [4, 5, 33, 50]. For concreteness, we use the OPRF-based unbalanced PSI protocol described in [33] because its use of OPRFs is conceptually closest to the one-time OPRFs used in [35], and because many implementations of OPRFs were available for our benchmarks.

Remark 1. *Our construction is also compatible with silent-OT extension [3] that could be used to reduce the communication complexity of [35, 43, 47]. This reduction in communication cost would come at an increase in computation cost, and currently silent-OT extension is only known to be secure under a variant of the LPN assumption.*

(1) Set $k \geq 2$, 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 bins using a 1-out-of-k hashing scheme, such that each bin obtains at most one element. Elements that cannot be allocated to a single bin will be put in a "stash", Stash_B, of size $s = \omega(1)$. Bob can compute this allocation efficiently. Let $B[i]$ denote the element stored in Bob's ith bin. Let $\mathsf{Stash}_B[i]$ for $i \in [s]$ denote the ith element of the stash.
(3) Alice will hash her elements into n bins, using a k-out-of-k hashing scheme. Let $C[i]$ denote the number of elements in Alice's ith bin, and let $A[i][j]$ denote the element in the bin 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 $S_B^i \stackrel{\text{def}}{=} \mathrm{PRF}_{\kappa_i}(B[i])$. For each $i \in [n]$, Alice will locally compute $S_{A,j}^i = \mathrm{PRF}_{\kappa_i}(A[i,j])$ for $j = 1, \ldots, C[i]$.
(5) Alice will shuffle $\left\{S_{A,j}^i\right\}_{i\in[n], j\in[C[i]]}$ 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_B^i 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 s executions of an OPRF protocol, where Bob learns $R_B^i \stackrel{\text{def}}{=} \mathrm{PRF}_\kappa(\mathsf{Stash}_B[i])$ for $i \in [s]$.
(8) Alice will compute $R_A^i \stackrel{\text{def}}{=} \mathrm{PRF}_\kappa(A[i][j])$ for $i \in [n]$, $j \in [C[i]]$ shuffle the set, and send it Bob who will locally compare these values to $\left\{R_B^i\right\}_{i\in s}$ 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 Bob km evaluations of her inputs under the one-time OPRF keys.

- $s = \omega(1)$ secure computations of the Stash OPRF. [3]
- Alice will send Bob m evaluations of her inputs under the Stash OPRF key.

The exact communication cost will depend on how the one-time OPRF and OPRF are instantiated, but standard constructions allow the entire protocol to run with communication linear in m.
In Sections 5.2 and 5.3, we review existing methods for implementing the one-time OPRF using methods from [43] and [35], and give the communication complexity of each approach. Then we describe how to implement the OPRF using methods from [33]. Finally, in Section 5.5 we compare the concrete communication complexity of our modified protocol with the best existing protocol [35].

5.2 An OT-masking-based protocol

In this section, we review the OT-masking-based OPRF protocol from [48]. 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 (Section 4.1), 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 [25], these can be generated using λ base OTs.
The outline above follows the construction of [48], which fixed an error in earlier OT-masking approaches [43, 47].

5.3 Batching OPRFs

In this section, we describe how to use our hashing scheme with the Batched, Related-Key OPRFs (BaRK-OPRFs) introduced in [35]. At a high level, this protocol is very similar to the OT-based protocol described in Section 5.2. 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 [35], 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) $\tilde{F}_\kappa(x)$ does not improve the adversary's distinguishing advantage in the PRF security experiment. In other words, given query access to $\tilde{F}_\kappa(\cdot)$, $F_\kappa(\cdot)$ appears pseudo-random on all unqueried points.
The original OT-extension protocol of [25] relied on a *correlation-robust* hash function. In [35] the notion of correlation-robustness is extended as follows.

Definition 5.1 (k-Hamming Correlation Robustness). 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 [25] 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 5.2 (Pseudo-Random Code [35]). A family of functions, C, is called a (d, ϵ) pseudorandom code (PRC) if for all strings $x \neq x'$,

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

The key security definition of [35] 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 5.3 (m-related key PRFs [35]). 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 $\kappa^*, \kappa_1, \ldots, \kappa_n$ and a challenge bit $b \in \{0,1\}$.
   - If $b = 0$, the challenger sends $\left\{\tilde{F}_{(\kappa^*,\kappa_j)}(x_j)\right\}_j$ and $\left\{F_{(\kappa^*,\kappa_{j_i})}(y_i)\right\}_i$ to the adversary.
   - If $b = 1$, the challenger generates m random strings $z_i \leftarrow \{0,1\}^v$, and sends $\left\{\tilde{F}_{(\kappa^*,\kappa_j)}(x_j)\right\}_j$ 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

[3] If the one-time OPRF and the OPRF compute the same PRF, a small optimization is possible. Rather than Alice generating the OPRF key, it could be generated by a one-time OPRF on the first element of the stash. The protocol would then use $(n+1)$ parallel one-time OPRFs and $(s-1)$ OPRFs. The OPRF and one-time OPRF we used represent different PRFs, so we could not use this.

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 5.4 (Lemma 5 in [35]). *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*

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

is an m-related key PRF as in Definition 5.3.

(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 [35].

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

$$F_{(\kappa^*,\kappa_j)}(r) = F_{((C,s),(j,q_j))}(r) = H(j||q_j \oplus [C(r) \cdot s])$$

for any r, and Bob has the relaxed PRF outputs

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

which allows him to compute $F_{(\kappa^*,\kappa_j)}(r_j)$.

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 $F_{(\kappa^*,\kappa_j)}(B[i])$. Alice will learn the PRF keys, $\kappa^*, \kappa_1, \ldots, \kappa_n$.

(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:
   $$\kappa^*, \kappa_1, \ldots, \kappa_n$$
   and Bob has relaxed PRF outputs $\tilde{F}_{(\kappa^*,\kappa_i)}(B[i])$
(4) From the relaxed outputs, Bob can compute $S_B^i \stackrel{\text{def}}{=} F_{(\kappa^*,\kappa_i)}(B[i])$.
(5) From the PRF keys $\kappa^*, \kappa_1, \ldots, \kappa_n$, Alice can compute $F_{(\kappa^*,\kappa_i)}(x)$ 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 5.5.

5.4 Security

The PSI protocols outlined in Section 5.1 are formed by taking existing one-time OPRF-based PSI protocols (e.g. [35, 43, 47, 48]) and combining them (in parallel) with an OPRF-based PSI protocol (e.g. [33]) 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 s applications 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, s PRF outputs provided by the OPRF protocol for the stash, and m PRF outputs provided by Alice to compare with the stash. Given the intersection of Alice and Bob's sets, Bob's view can be easily simulated by providing $km + m$ random strings (corresponding to the one-time OPRF round) with the correct intersection pattern, and $s + m$ random strings (corresponding to the OPRF round) with the correct intersection pattern.

5.5 Concrete communication benchmarks

In [32] 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 $1 - O\left(n^{-(s+1)}\right)$. The protocols of [35, 43, 47, 48] 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 [43, 47], they set security threshold to be a constant $2^{-\sigma}$, with $\sigma = 40$, independent of the set sizes. The asymptotic failure rate is known to be $\Theta(n^{-(s+1)})$ [32], which implies $s \geq \frac{\sigma}{\log(n)} + \frac{\log(c)}{\log n} - 1$ provides failure probability below $2^{-\sigma}$ for some constant c. In [43], 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 sizes to be $s \sim 40/\log(n)$ (see [43] Figure 2). This is consistent with the analysis above, assuming the constant c is not very large. The scheme of [35] 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 that the necessary stash size *decreases* as the set sizes increase, whereas an asymptotic analysis (which assumes that the failure probability should be negligible in m) requires that the stash size *increases* 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 [35] 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 ms PRF outputs in the stash phase.

We replace 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 m 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 [35, 43].

Let d denote the number of bits required for an OPRF application, and $d' < d$ be the number of bits required for a one-time OPRF evaluation. Then our protocol obtains a concrete performance improvement whenever $sd + mv < sd' + msv$.

We therefore obtain a concrete improvement whenever

$$\frac{s(d-d')}{v(s-1)} < m$$

Table 1: This table shows the communication cost of a single OPRF evaluation, and the minimum number of elements for which replacing the OPRF-based comparison of the stash in [35] would be improved by switching to our protocol. The AES Obliv-C and AES EMP benchmarks were obtained via our internal tests. The ABY benchmarks were taken from [1], the Yao benchmarks were taken from [33], and the LowMC benchmarks were taken from [1].

OPRF	Comm. Cost (d)	m at which our protocol outperforms [35]
AES (Yao - Obliv-C)	8 Mb	2^{21}
AES (Yao - EMP)	212 Kb	2^{16}
AES (GMW - ABY) [1]	170 Kb	2^{15}
AES (Yao - ObliVM) [33]	177 Kb	2^{15}
LowMC [1]	23 Kb	2^{12}

In [1], 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 ([1] Table 6). Similarly, 1024 garbled AES representations were computed using 185 Mb of communication ([33] Table 5]), which reduces to about 177 kb per AES circuit (including pregenerating the OTs).

Although our protocol outperforms [35] asymptotically, the exact point at which it starts to outperform [35] concretely depends on the exact implementation of the OPRF. In Table 1 we show the communication costs of four different standard OPRF implementations, and give the number of elements (m) at which our protocol starts to outperform that of [35].

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 [35] with a true OPRF, whenever the sets being compared have more than $2^{15} = 32768$ elements. Using the "MPC-friendly" PRF, LowMC [1], we start to see concrete efficiency improvements for sets of size $2^{12} = 4096$.

Figure 9 shows the overall reduction in communication when the BaRK protocol [35] is replaced with our protocol instantiated with LowMC. Figure 10 shows the overall reduction in communication (over [35]) 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 [35].

Note that these results are conservative, in that we use the same stash sizes as [35] which assumes that the probability of failure is constant and therefore the stash sizes decrease as m increases. The step-down points of non-differentiability in Figures 9 and 10 correspond to these decreases in the stash size. From a theoretical perspective to maintain negligible failure rates, the stash must be *increasing*in m, and in that setting we would see corresponding step-up points in the graphs at each point where the stash size increased.

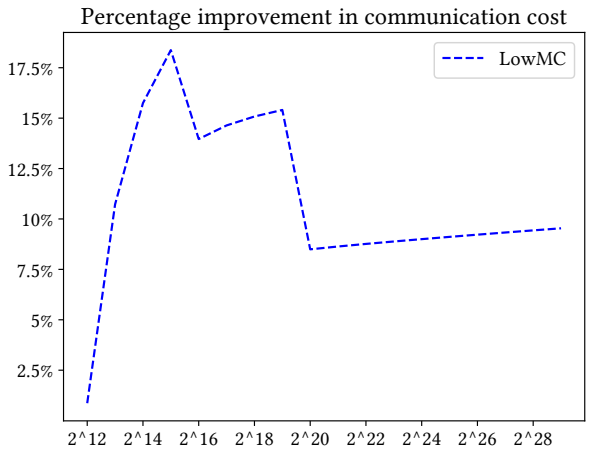

Figure 9: The percentage improvement in overall communication cost when modifying the BaRK protocol to use the LowMC-based PRF to compare the stash. For most values of m, our modification reduces the overall communication cost of the protocol by 10 to 15%. The points of non-differentiability in the graph correspond to the places where the stash sizes drop (we use the same stash sizes as [35, 43])

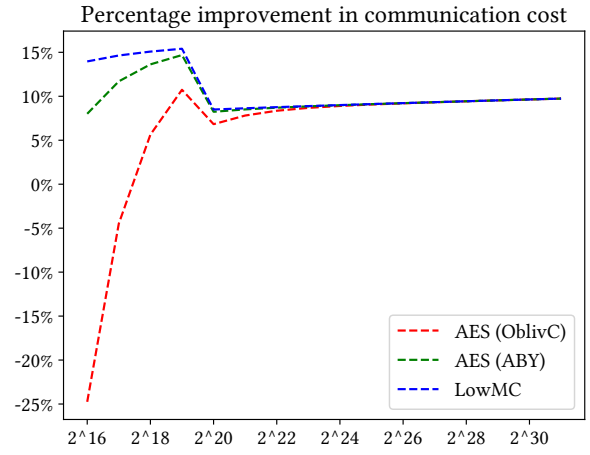

Figure 10: As m increases, and the stash size stays at 2, all three OPRF protocols tend towards a 10% improvement in communication cost over the BaRK protocol of [35]

6 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. Specifically we show that the one-time OPRF protocols of [35, 43, 47] can be improved both asymptotically (from super-linear to linear) and, for $n \geq 2^{12}$, concretely by replacing the stash-comparison step with an unbalanced PSI protocol [33].

REFERENCES

[1] Martin R Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. 2015. Ciphers for MPC and FHE. In *EUROCRYPT*. Springer, 430–454.

[2] Yuriy Arbitman, Moni Naor, and Gil Segev. 2010. Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In *FOCS*. IEEE, 787–796.

[3] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. 2019. Efficient Pseudorandom Correlation Generators: Silent OT Extension and More. IACR ePrint 2019/448. (2019).

[4] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. 2018. Labeled PSI from Fully Homomorphic Encryption with Malicious Security. In *CCS*. ACM, 1223–1237.

[5] Hao Chen, Kim Laine, and Peter Rindal. 2017. Fast Private Set Intersection from Homomorphic Encryption. In *CCS*. 1243–1255.

[6] Michele Ciampi and Claudio Orlandi. 2018. Combining Private Set-Intersection with Secure Two-Party Computation. In *SCN*.

[7] Claude Crépeau. 1987. Equivalence between two flavours of oblivious transfers. In *CRYPTO*. 350–354.

[8] Giovanni D. Crescenzo, Tal Malkin, and Rafail Ostrovsky. 2000. Single Database Private Information Retrieval Implies Oblivious Transfer. In *Eurocrypt*, Vol. 1807. 122–138. https://doi.org/10.1007/3-540-45539-6_10

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

[10] Emiliano De Cristofaro and Gene Tsudik. 2010. Practical Private Set Intersection Protocols with Linear Complexity.. In *FC*, Vol. 10. Springer, 143–159.

[11] Emiliano De Cristofaro and Gene Tsudik. 2012. Experimenting with Fast Private Set Intersection. *Trust* 7344 (2012), 55–73.

[12] Changyu Dong, Liqun Chen, and Zikai Wen. 2013. When private set intersection meets big data: an efficient and scalable protocol. In *CCS*. 789–800.

[13] Shimon Even, Oded Goldreich, and Abraham Lempel. 1985. A randomized protocol for signing contracts. *Commun. ACM* 28, 6 (1985), 637–647.

[14] FBI. 2019. National Crime Information Center (NCIC). https://www.fbi.gov/services/cjis/ncic. (July 2019).

[15] Michael J Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. 2005. Keyword Search and Oblivious Pseudorandom Functions.. In *TCC*, Vol. 3378. 303–324.

[16] Michael J Freedman, Kobbi Nissim, and Benny Pinkas. 2004. Efficient private matching and set intersection. In *EUROCRYPT*. 1–19.

[17] Yael Gertner, Sampath Kannan, Tal Malkin, Omer Reingold, and Mahesh Viswanathan. 2000. The relationship between public key encryption and oblivious transfer. In *FOCS*. 325.

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

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

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

[21] Carmit Hazay and Kobbi Nissim. 2010. Efficient Set Operations in the Presence of Malicious Adversaries.. In *PKC*, Vol. 6056. 312–331.

[22] Dennis Hofheinz and Eike Kiltz. 2007. Secure hybrid encryption from weakened key encapsulation. In *CRYPTO*. Berlin, Heidelberg. http://eprint.iacr.org/2007/288

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

[24] Russell Impagliazzo and Steven Rudich. 1989. Limits on the Provable Consequences of One-Way Permutations. In *STOC*. 44–61.

[25] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending Oblivious Transfers Efficiently.. In *CRYPTO*, Vol. 2729. Springer, 145–161.

[26] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and Jürg Wullschleger. 2011. Constant-Rate Oblivious Transfer from Noisy Channels. In *CRYPTO*. Vol. 6841. Chapter 38, 667–684. https://doi.org/10.1007/978-3-642-22792-9_38

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

[28] Stanislaw Jarecki and Xiaomin Liu. 2009. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection.. In *TCC*, Vol. 5444. Springer, 577–594.

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

[30] Yael T. Kalai. 2005. Smooth Projective Hashing and Two-Message Oblivious Transfer. In *EUROCRYPT*. 78–95. https://doi.org/10.1007/11426639_5

[31] Joe Kilian. 1988. Founding cryptography on oblivious transfer. In *STOC (STOC '88)*. 20–31.

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

[33] Ágnes Kiss, Jian Liu, Thomas Schneider, N Asokan, and Benny Pinkas. 2017. Private set intersection for unequal set sizes with mobile applications. *PoPETs* 4 (2017), 97–117.

[34] Lea Kissner and Dawn Song. 2005. Privacy-preserving set operations. In *CRYPTO*, Vol. 3621. 241–257.

[35] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. 2016. Efficient batched oblivious PRF with applications to private set intersection. In *CCS*. 818–829.

[36] Reinhard Kutzelnigg. 2006. Bipartite Random Graphs and Cuckoo Hashing. In *Fourth Colloquium on Mathematics and Computer Science Algorithms, Trees, Combinatorics and Probabilities (DMTCS Proceedings)*, Philippe Chassaing et al. (Eds.), Vol. DMTCS Proceedings vol. AG, Fourth Colloquium on Mathematics and Computer Science Algorithms, Trees, Combinatorics and Probabilities. Discrete Mathematics and Theoretical Computer Science, Nancy, France, 403–406. https://hal.inria.fr/hal-01184689

[37] Mikkel Lambæk. 2016. Breaking and Fixing Private Set Intersection Protocols. IACR ePrint 2016/665. (2016).

[38] Hoi K. Lo. 1997. Insecurity of quantum secure computations. *Physical Review A* 56 (1997), 1154–1162.

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

[40] Michael Mitzenmacher. 2009. Some Open Questions Related to Cuckoo Hashing. In *ESA*. 1–10.

[41] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. 2008. A Framework for Efficient and Composable Oblivious Transfer.. In *CRYPTO*, David Wagner (Ed.), Vol. 5157. 554–571. https://doi.org/10.1007/978-3-540-85174-5_31

[42] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2019. SpOT-Light: Lightweight Private Set Intersection from Sparse OT Extension. (2019).

[43] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. 2015. Phasing: Private Set Intersection Using Permutation-based Hashing. In *USENIX Security Symposium*. 515–530.

[44] Benny Pinkas, Thomas Schneider, Nigel P Smart, and Stephen C Williams. 2009. Secure Two-Party Computation Is Practical.. In *ASIACRYPT*, Vol. 9. 250–267.

[45] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. 2019. Efficient Circuit-Based PSI with Linear Communication. In *EUROCRYPTO*. 122–153.

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

[47] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2014. Faster Private Set Intersection Based on OT Extension. In *USENIX*. 797–812.

[48] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2016. Scalable Private Set Intersection Based on OT Extension. IACR Cryptology ePrint Archive. (2016).

[49] Michael O. Rabin. 1981. *How To Exchange Secrets with Oblivious Transfer*. Technical Report TR-81. Harvard University.

[50] Amanda C Davi Resende and Diego F Aranha. 2018. Faster unbalanced private set intersection. *FC 2018* (2018).

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

[52] Louis Salvail, Christian Schaffner, and Miroslava Sotáková. 2009. On the Power of Two-Party Quantum Cryptography. In *ASIACRYPT*. 70–87.

[53] Aaron Segal, Bryan Ford, and Joan Feigenbaum. 2014. Catching Bandits and Only Bandits: Privacy-Preserving Intersection Warrants for Lawful Surveillance. In *4th USENIX Workshop on Free and Open Communications on the Internet (FOCI 14)*. USENIX Association, San Diego, CA. https://www.usenix.org/conference/foci14/workshop-program/presentation/segal

[54] Abraham Waksman. 1968. A permutation network. *Journal of the ACM (JACM)* 15, 1 (1968), 159–163.

[55] Severin Winkler and Jürg Wullschleger. 2010. On the Efficiency of Classical and Quantum Oblivious Transfer Reductions. In *CRYPTO*, Vol. 6223. Berlin, Heidelberg, 707–723.

[56] Stefan Wolf and Jürg Wullschleger. 2006. Oblivious transfer is symmetric. In *EUROCRYPT*. 222–232.

ACKNOWLEDGMENTS

This research was sponsored in part by ONR grant (N00014-15-1-2750) "SynCrypt: Automated Synthesis of Cryptographic Constructions", NSF grant CNS-1513671, DARPA/SPAWAR contract N66001-15-C-4065 and ODNI/IARPA contract 2019-1902070008. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the DoD, DARPA, ODNI, IARPA, or the U.S. Government.