Linear Complexity Private Set Intersection for Secure Two-Party Protocols

Ferhat Karakoç[1] and Alptekin Küpçü[2]

[1] Ericsson Research, İstanbul, Turkey
ferhat.karakoc@ericsson.com
[2] Koç University, İstanbul, Turkey
akupcu@ku.edu.tr

Abstract. In this paper, we propose a new private set intersection (PSI) protocol that computes the following functionality. The two parties (P_1 and P_2) input two sets of items (X and Y, respectively) and one of the parties outputs a *function of the intersection* ($f(X \cap Y)$). This functionality is generally required when the PSI protocol is used as a part of a larger secure two-party secure computation. Pinkas et al. presented a PSI protocol at Eurocrypt 2019 for this functionality, which has linear complexity only in communication. While there are PSI protocols with linear computation and communication complexities in the classical PSI setting where the intersection itself is revealed to one party, to the best of our knowledge, there is no PSI protocol, which outputs a function of the intersection and satisfies linear complexity in both communication and computation. We present the first PSI protocol that outputs only a function of the intersection with linear communication and computation complexities. While creating the protocol, as a side contribution, we provide a one-time oblivious programmable pseudo-random function based on garbled Bloom filters. We also implemented our protocol and provide performance results.

Keywords: Private set intersection, two-party computation, Bloom filters, oblivious transfer, cuckoo hashing

1 Introduction

Private set intersection (PSI) protocols are one of the commonly used two party secure communication primitives where two parties, P_1 and P_2, have their own respective private sets, X and Y, and at least one of the parties learn the intersection $X \cap Y$ but nothing more. Although this functionality can be realized using generic secure two-party computation protocols, the main drawback of such realizations is the high communication complexity, which is an important parameter when deploying such a solution for real world applications. Because of that, in the last decade, considerable amount of custom PSI protocols have been proposed in the literature. However, most of the proposed solutions reveal the intersection to at least one of the parties, which makes the protocols not usable as

a building block in a larger secure computation protocol, because in that larger protocol, intermediate information would leak due to the nature of the employed PSI protocol. In this work, we focus on designing a PSI protocol which outputs a *function f of the intersection* ($f(X \cap Y)$) in the semi-honest security model. Such a function can be a homomorphic encryption of the membership result of the items under the public key of the sender party to make the result usable in a larger homomorphic encryption based computation protocol, or it can be secret shares of the result for each item or labels for the corresponding input wires for circuit based secure computation protocols.

Related Work: To the best of our knowledge, the existing protocols that output a function of the intersection result were proposed by Ciampi and Orlandi [5], Pinkas et al. [25], and Falk et al. [11] in addition to the circuit based solutions of [13, 28]. In [5], a custom private set membership protocol (PSM) (where one of the parties has only one item instead of a set) based on oblivious navigation of a graph was introduced and this PSM protocol was converted to a PSI protocol with $O(n \log n / \log \log n)$ communication and computation complexities using the hashing techniques proposed in [27, 24, 28]. [11] has a communication complexity of $O(n \log \log n)$ when the output of the protocol is a function of the intersection. In [25], Pinkas et al. proposed a PSI protocol with $O(n)$ communication and $\omega(n(\log \log n)^2)$ computation complexities using the oblivious programmable pseudo-random function (OPPRF) introduced in [19]. That protocol is similar to the protocols introduced in [24] in terms of the usage of cuckoo hashing, and it additionally uses OPPRF to check the private set membership relation in the hashed bins, where the result is not output in clear text, and then deploys a comparison circuit for the output of the membership result that can be given to a function as the input. Also in literature, there have been special purpose PSI protocols such as [31, 21, 6, 18, 9, 8, 32, 14, 15], which output a specific function of the intersection such as cardinality of the set, intersection-sum or a threshold function.

In our solution, we follow the idea of Pinkas et al. [25] in that we first run a PSM protocol for each bin in the cuckoo hash table and then execute a comparison protocol. We diverge from their idea in the following ways. The first one is that we construct a Bloom-filter (BF) based PSM protocol by modifying Dong et al. PSI solution [10] to reduce the computation complexity. The second point is that, instead of using a comparison circuit, we execute Ciampi-Orlandi PSM protocol as a secure equality testing protocol such as the one used in [17], which makes the equality testing free by using the base oblivious transfer already executed in the BF-based PSM protocol. Following these two methods along the idea of Pinkas et al., we are able construct the first custom PSI protocol having linear computation and communication complexities for the functionality we consider (outputting not the result set, but a function of the intersection), to the best of our knowledge. Note that there have been PSI solutions with linear complexities such as the protocols in [7, 10] but in these protocols the intersection is revealed to at least one party while in our protocol no party learns the intersection in cleartext. We implemented our PSM and PSI protocols and the

Ciampi-Orlandi PSM protocol to make a fair comparison. Experimental performance results, which validate our performance analysis, are given in Section 6.

We organized the paper as follows. In Section 2, we introduce the notation used throughout the paper and mention the primitives related to our proposal. We present our PSM and PSI protocols in Section 3 and 4, respectively. Security definitions and proofs are given in Section 5. After presenting performance results of our protocols in Section 6, we conclude the paper with Section 7.

2 Preliminaries and Similar Protocols

2.1 Notation

P_1 and P_2 are the parties who run the protocol, X and Y are the corresponding item sets of the parties, and f is the function to be applied on the set intersection result. P_1 and P_2 respectively play the sender and receiver roles, and at the end of the PSI protocol, P_2 learns $f(X \cap Y)$. The remaining notation we use throughout the paper is as follows:

- ℓ : The length of the items in the sets
- κ : Security parameter
- η : Statistical correctness parameter
- n : The number of items in the sets
- m : Bloom filter size
- k : Number of hash functions used in Bloom filter
- H_i : Set of k hash functions used in the construction of Bloom filters for i-th bin in the cuckoo table where $H_i = \{h_{i,1}, ..., h_{i,k}\}$
- β : The number of bins in cuckoo table

2.2 Oblivious Transfer

A 1-out-of-2 oblivious transfer (OT) [29] is a secure two-party protocol that realizes Functionality 1. In the 1-out-of-2 random OT, the messages are chosen by the functionality instead of P_1. P_1 inputs nothing, P_2 inputs the choice bit b, P_1 outputs the message pair (m_0 and m_1) and P_2 outputs only m_b [1, 22]. While OT is one of the commonly used primitives in secure protocols, the main drawback of this primitive is the need of asymmetric key operation executions. With the help of OT extension (OTE) method introduced in [16], to execute 1-out-of-2 OT for m pairs of length ℓ (OT_ℓ^m) it is enough to run OT_κ^κ, called as

Functionality 1 Oblivious Transfer

Inputs. The sender inputs a pair (x^0, x^1), the receiver inputs a choice bit $b \in \{0, 1\}$.

Outputs. The functionality returns the message x^b to the receiver and returns nothing to the sender.

base OTs, where κ is the security parameter, which keeps the number of heavy public key operations as a constant independent from the number of pairs m and item lengths ℓ.

In recent works, it was shown that the number of rounds can be 2 instead of 3 for an OT extension protocol by executing some of the computations in the offline phase of the protocol [3, 4]. In our solution, we don't consider the preprocessing operations and so we don't use these constructions in our protocols.

2.3 Cuckoo Hashing

Cuckoo hashing [23] is a hashing primitive that allows to map items of a set to the bins, where there is at most one item in each bin. This primitive employs two hash functions h_0 and h_1 and maps n items to a table T of $(1+\epsilon)n$ bins. An item x_i is inserted into bin $T[h_b(x_i)]$. If this bin already accommodates a previous item x_j, then x_j is relocated to bin $T[h_{1-b}(x_j)]$. If in that bin there is another item, then this procedure is repeated until there is no need or a replacement threshold is reached. If a threshold is employed, then a stash is used to store the items that are not located into the bins.

2.4 Bloom Filter Based PSI

A Bloom filter (BF) [2] is a representation of a set $X = x_1, ..., x_n$ of n elements using an m-bit string BF. BF is constructed with the help of a set of k independent and uniform hash functions $(H = h_1, ..., h_k)$ where $h_i : \{0,1\}^\ell \rightarrow \{1, 2, ..., m\}$ as follows: BF is first set to 0^m. Then, for each item in X, $BF[h_i(x_j)]$ is set to 1 where $1 \leq i \leq k$ and $1 \leq j \leq n$. To check whether an item x is in the set X, one checks $BF[h_i(x)]$ is equal to 1 or not for each i $(1 \leq i \leq k)$. If for all i $(1 \leq i \leq k)$ the corresponding bit in BF is equal to 1, then it means that the item is probably in the set. Otherwise (for some i the corresponding bit is 0), the item is not in the set.

A Bloom filter based PSI was proposed by Dong et al. [10]. In that solution, a variant of BF called as Garbled Bloom Filter (GBF) was used. A GBF of a set X, GBF, is similar to BF except that while for each hash function h_i in H we have $BF[h_i(x)] = 1$, $GBF[h_i(x)]$ is a secret share of x: that is, $\bigoplus_{i=1}^{k} GBF[h_i(x)] = x$ and other cells are random values instead of simple zeros. In the first step of the protocol, P_1 and P_2 construct a GBF (GBF_X) and a BF (BF_Y), respectively. Then, P_1 and P_2 run m-pair oblivious transfer of ℓ-bit strings (OT_ℓ^m) where P_1's input is $(0^\ell, GBF_X[i])$ and P_2's input is $BF_Y[i]$ for the i-th OT, and the output of P_2 is $GBF_Y[i]$. In this way, P_2 learns $GBF_X[i]$ if $BF_Y[i] = 1$. P_2 checks, for each item $y_j \in Y$, whether it is in X or not, by comparing $\bigoplus_{i=1}^{k} GBF_Y[h_i(y_j)] \stackrel{?}{=} y_j$.

Pinkas et al. improved the solution using random 1-out-of-2 OT [27]. In that protocol, P_1 and P_2 run m times random 1-out-of-2 OT where P_2's input is the choice bit $BF[i]$, P_2's output is $GBF_Y[i] = m^i_{BF[i]}$, P_1's output is (m^i_0, m^i_1) and P_1 sets $GBF_X[i] = m^i_1$ if there is at least one item in his set such that $h_l(x_j) = i$ for $(1 \leq l \leq k)$. Then, P_1 sends $K_{x_j} = \bigoplus_{l=1}^{k} GBF_B[h_l(x_j)]$ to P_2 and P_2 learns

the membership result for her item y by checking $\bigoplus_{l=1}^{k} GBF_Y[h_l(y)]$ equals to one of the key strings K_{x_j} sent by P_1. Rindal and Rosulek [30] proposed to send the hash of concatenation of x_j and K_{x_j} ($H(x_j \| K_{x_j})$) instead of K_{x_j} because of the collision probability of $h_i(x) = h_{i'}(x)$.

2.5 Oblivious Pseudo-Random Function Based PSM

An oblivious pseudo-random function (OPRF), introduced in [12], is a two-party protocol where party P_1 holds a key K, party P_2 holds a string x, and at the end of the protocol P_1 learns nothing, while P_2 learns $F_K(x)$ where F is a pseudo-random function family that gets a κ-bit key K and an ℓ-bit input string x and outputs an ℓ-bit random-looking result. An oblivious programmable pseudo-random function (OPPRF) [19] is similar to an OPRF except that in OPPRF, the protocol outputs predefined values for some of the programmed inputs. In this protocol P_2 should not be able to distinguish which inputs are programmed.

The basic idea in OPRF based PSM protocols are as follows. P_1 holds a key K to compute a pseudo-random function F_K, P_2 learns $F_K(y)$ for his item y obliviously, and P_1 sends $F_K(x_i)$ for her items $x_i \in X$ to P_2. P_2 checks if $F_K(y)$ is in the set $\{F_K(x_i)\}$. An example PSI protocol can be found in [28]. In the OPRF solution, P_2 learns whether or not his item is in the set of P_1. This solution cannot be used in our setting where nobody learns the result in cleartext and the parties only learn a function result of the intersection. Pinkas et al. [25] converted the OPRF solution to the setting we consider using an oblivious programmable pseudo-random function. In that solution, P_1 sends the same (random) output r for the items in her set. Otherwise, she sends some random output to P_2. Then P_1 and P_2 run a circuit to check the equality of r and the outputs P_1 sent to P_2. At the end of this equality check circuit, one party obtains a function based on the result of the equality, i.e, of the membership.

2.6 Usage of Ciampi-Orlandi PSM Protocol to Test Equality of Two Strings

The private set membership (PSM) protocol proposed by Ciampi and Orlandi [5] works on the setting that P_1 and P_2's inputs are a set of items X and an item y, respectively, and at the end of the protocol, P_2 learns a function of the membership relation and P_1 learns nothing. The protocol is based on oblivious graph tracing and uses oblivious transfer. In our construction, we use that protocol for the case that P_1's input is just one item instead of a set, as considered in [17]. In this case, the PSM protocol becomes a secure equality testing outputting a function (we call functional equality testing - FEQT) protocol that realizes Functionality 2. This simplification also greatly increases efficiency, helping us achieve linear costs. The reader can refer to [5] and [17] for a full description of the protocol and the FEQT version of the protocol, respectively.

Functionality 2 Functional Secure Equality Testing

Parameters. A function f to be computed on the equality relation result.

Inputs. P_1 inputs x, P_2 inputs y. *Outputs.* The functionality checks the equality of x and y and returns $f(x = y)$ or $f(x \neq y)$ according to the equality relation to P_2.

3 Our Private Set Membership Protocol

In this section, we propose a new PSM protocol that realizes Functionality 3. As discussed in the introduction, our protocol does not output the membership result, but instead outputs some function of it, so that it can be directly integrated into a larger secure computation protocol. After this section, we show how to extend our protocol to set intersection as well.

In the construction of the protocol, we use the following idea of [25]: If $y \in X$, then both parties learn the same random value. Otherwise, they learn different random values. Then, the parties run a comparison protocol that outputs a function of the equality instead of the equality itself (Functionality 2). Our solution diverges from the solution of [25] in two folds. To realize the first part, [25] makes use of an OPPRF construction based on polynomials. We propose a new OPPRF construction based on Bloom filters. The selection of Bloom filters enables us to reduce the computation complexity of the protocol to a linear complexity. The other difference is that we utilize Ciampi-Orlandi PSM protocol [5] for secure equality testing as done in [17] for Functionality 2, instead of running a comparison circuit. Thus, our overall construction is not a circuit-based construction.

Functionality 3 Private Set Membership

Parameters. A function f to be computed on the membership relation result.

Inputs. P_1 inputs $X = \{x_1, ..., x_n\}$, P_2 inputs y

Outputs. The functionality checks the membership of y in X and returns $f(y \stackrel{?}{\in} X))$ according to the membership relation.

3.1 Bloom Filter Based OPPRF Construction

We introduce a one-time OPPRF construction by modifying Dong et al. PSI protocol [10]. While in their protocol secret shares of items are stored in a garbled Bloom filter, in our construction, secret shares of a random value chosen by the sender are stored. The OPPRF functionality we use in our PSM protocol is given in Functionality 4 and our construction that implements the functionality is presented in Protocol 1.

Note that we allow the programmed values (t_i) to be correlated. Because of that, the functionality is secure only if the receiver makes only one query. For

the purposes of PSM, we notice that one query is enough. In our PSM solution the programmed values will be the same; that is, all the t_i values will be equal.

Functionality 4 (One-Time) Oblivious Programmable Pseudo Random Function

Inputs. P_1 inputs predefined items $X = \{x_1, ..., x_n\}$ and corresponding programmed values $T = \{t_1, ..., t_n\}$, P_2 inputs y

Outputs. The functionality checks the membership $y \in X$ and returns t_i to P_2 if $\exists x_i$ s.t. $y = x_i$ $(1 \leq i \leq n)$; returns a random value otherwise to P_2, and returns nothing to P_1.

3.2 Our Full PSM Protocol

To achieve private set membership, the parties first run the one-time OPPRF protocol that is based on garbled Bloom filters. At the end, P_1 outputs r (a random value chosen by P_1), whereas P_2 learns some random value that may be r or something different. The value P_2 learns is always random and indistinguishable; but, this random value is equal to r if and only if $y \in X$.[3] Following this part, the parties run a secure functional equality testing protocol, where at the end of the protocol P_2 learns the function result of the equality relation, which is also the function result of the membership relation. We make use of the PSM protocol of Ciampi-Orlandi [5] for secure functional equality testing by reducing the number of items of the sender set to one. We present our semi-honest secure PSM solution in Protocol 2.

4 Our Private Set Intersection Protocol

4.1 Batch One-Time OPPRF

Our PSM protocol can be used to build an efficient PSI protocol using the hashing techniques introduced in [27, 24]. In this technique, one party constructs a cuckoo table as mentioned in Section 2.3 using two hash functions and the other party maps her items into bins in a hash table using the two hash functions that are applied on each item. Then, a private set membership protocol is applied on each bin where the party who constructs the cuckoo table inputs the (single) item in the i-th bin, and the other party inputs the set of items in the i-th bin of its hash table, for the i-th execution of the PSM protocol. If one were to directly employ our PSM construction to obtain a PSI protocol using this hashing technique, the computation and communication complexities of the full PSI protocol would be $O(n \log n / \log \log n)$, since the number of items in each hash table bin is $O(\log n / \log \log n)$ and the number of bins is $O(n)$. Note that with this usage,

[3] With a randomly constructed garbled Bloom filter, the probability of obtaining the same r with a different item is negligible, as shown in [10].

Protocol 1 Our One-Time OPPRF Protocol

Parameters. A set of hash functions $H = \{h_1, ..., h_k\}$

Inputs. P_1 inputs a set of items $X = \{x_1, ..., x_n\}$ and corresponding programmed values $T = \{t_1, ..., t_n\}$, P_2 inputs an item y.
Outputs. P_1 outputs nothing and P_2 outputs t_i if $\exists x_i$ s.t. $y = x_i$ $(1 \leq i \leq n)$, otherwise outputs a random value.
The protocol steps:

1. P_1 constructs a garbled Bloom filter GBF_X such that

$$\bigoplus_{i=1}^{k} GBF_X[h_i(x_j)] = t_j$$

   for $1 \leq j \leq n$.
2. P_2 constructs a (standard) Bloom filter BF_y for the item y.
3. P_1 and P_2 run m oblivious transfers where P_1's input is $(0, GBF_X[i])$ and P_2's input is $BF_y[i]$ for the i-th oblivious transfer, and the output of P_2 is 0 if $BF_y[i] = 0$ or $GBF_X[i]$ if $BF_y[i] = 1$. Call the output of P_2 as $GBF_y[i]$.
4. P_1 outputs nothing and P_2 outputs $\bigoplus_{i=1}^{k} GBF_y[h_i(y)]$.

Protocol 2 Our Private Set Membership Protocol

Parameters. A set of hash functions $H = \{h_1, ..., h_k\}$

Inputs. P_1 inputs a set of items $X = \{x_1, ..., x_n\}$, P_2 inputs an item y.

Outputs. P_2 outputs $f(y \stackrel{?}{\in} X)$.

The protocol steps:

1. P_1 picks an η-bit random value r and sets $T = \{t_1 = r, ..., t_n = r\}$.
2. P_1 and P_2 run Protocol 1 for one-time OPPRF with the respective inputs (X,T) and y. Denote the output of P_2 as r'.
3. P_1 and P_2 run Ciampi-Orlandi PSM protocol for functional equality testing with the respective inputs r and r'. The output of the PSM protocol is the output of the FEQT protocol.

for each bin, P_2 and P_1 run $O(n)$ parallel OPPRF protocols and then apply $O(n)$ parallel FEQT protocols. Instead of following this straightforward way, we show that it is possible to make the communication and computation complexities linear while extending our PSM solution to a PSI solution by using a batch OPPRF protocol. We propose a new batch one-time OPPRF construction in Protocol 3 that implements Functionality 5.

Note that, for each set, different sets of hash functions (set H_i for the i-th set) must be used, since we allow the items in the sets to be correlated. Otherwise,

Functionality 5 Batch One-Time Oblivious Programmable Pseudo Random Function

Inputs. P_1 inputs a predefined set of item sets $X = \{X_1, ..., X_\beta\}$, where $X_i = \{x_{i,1}, ..., x_{i,n}\}$, and corresponding programmed value sets $T = \{T_1, ..., T_\beta\}$, where $T_i = \{t_{i,1}, ..., t_{i,n}\}$, and P_2 inputs a set of items $Y = \{y_1, ..., y_\beta\}$
Outputs. The functionality checks the membership relations $y_i \in X_i$ and returns $r'_i = t_{i,j}$ if $\exists x_{i,j}$ s.t. $y_i = x_{i,j}$ $(1 \leq j \leq n)$; returns a random r'_i otherwise, for each i where $1 \leq i \leq \beta$.

Protocol 3 Bloom Filter Based Batch One-Time OPPRF Protocol

Parameters. A set of hash function sets $H = \{H_1, ..., H_\beta\}$ where $H_i = \{h_{i,0}, ...h_{i,k}\}$

Inputs. P_1 inputs a set of item sets $X = \{X_1, ..., X_\beta\}$, where $X_i = \{x_{i,1}, ..., x_{i,n}\}$, and corresponding programmed value sets $T = \{T_1, ..., T_\beta\}$, where $T_i = \{t_{i,1}, ..., t_{i,n}\}$, and P_2 inputs a set of items $Y = \{y_1, ...y_\beta\}$.
Outputs. P_2 outputs a set of random values $R' = \{r'_1, ..., r'_\beta\}$, where $r'_i = t_{i,j}$ if $\exists x_{i,j}$ s.t. $y_i = x_{i,j}$ $(1 \leq j \leq n)$; otherwise r'_i is a random value; for $1 \leq i \leq \beta$.

The protocol steps:

1. P_1 constructs a garbled Bloom filter GBF_X such that

$$\bigoplus_{j=1}^{k} GBF_X[h_{i,j}(x_{i,l})] = t_{i,l}$$

   for $1 \leq i \leq \beta$ and $1 \leq j \leq k$.
2. P_2 constructs a Bloom filter BF_Y for the items in Y.
3. P_1 and P_2 run m oblivious transfers where P_1's input is $(0, GBF_X[i])$ and P_2's input is $BF_Y[i]$ for the i-th oblivious transfer, and the output of P_2 is 0 if $BF_y[i] = 0$ or $GBF_X[i]$ if $BF_Y[i] = 1$. Call the OT output P_2 obtains as $GBF_Y[i]$.
4. P_2 outputs $R' = \{r'_1, ..., r'_\beta\}$ where $r'_i = \bigoplus_{j=1}^{k} GBF_Y[h_{i,j}(y_i)]$.

there will be collisions while constructing the garbled Bloom filter because some items will exist in more than one set.

4.2 Our Full PSI Protocol

Our full PSI protocol that realizes Functionality 6 is introduced in Protocol 4. Note that when we use two hash functions for cuckoo hashing, then there will be some items in Y which cannot be placed into the table and have to be moved to a stash. For each of these items in the stash, a PSM protocol also has to be executed. When we consider the number of these items as $\omega(1)$, then the complexity of our PSI protocol becomes bigger than $O(n)$. To make the complexity linear, Pinkas et al. proposed to use dual execution or a stash-less cuckoo hashing [25]. In dual execution, after the first run of the PSI protocol,

P_2 learns the membership result for its items except the ones in the stash. Then the parties run the PSI protocol swapping their roles, that is, P_1 constructs a cuckoo table for X and P_2 constructs a hash table for the items in the stash. Since there may be some items of P_1 which have not been placed in the cuckoo table and moved to a stash, P_1 and P_2 should run the PSM protocol for their items in the stashes. However, this usage does not realize the Functionality 6 that we consider, since in the second run, P_2 learns the function of the membership result between its items in the stash and the set X, and in the final PSM protocols run for the items in the stashes, P_2 again learns the function of the membership result between its items in the stash and P_1's items in the stash. That is, P_2 learns two different results for its items in the stash that makes the protocol diverge from Functionality 6. Because of these two reasons, we make use of the second method of Pinkas et al, which is the usage of stash-less cuckoo hashing.

Functionality 6 Private Set Intersection

Parameters. A function f to be computed on the intersection. *Inputs.* P_1 inputs $X = \{x_1, ..., x_n\}$, P_2 inputs $Y = \{y_1, ..., y_n\}$ *Outputs.* The functionality checks the membership of each y_i in X and returns $f(y_i \overset{?}{\in} X)$ to P_2.

Protocol 4 Bloom Filter Based Private Set Intersection Protocol

Parameters. A set of hash function sets $H = \{H_1, ..., h_\beta\}$ where $H_i = \{h_{i,1}, ..., h_{i,k}\}$ for $1 \leq i \leq \beta$.

Inputs. P_1 inputs a set of items $X = \{x_1, ..., x_n\}$, P_2 inputs a set of items $Y = \{y_1, ..., y_n\}$.

Outputs. P_2 outputs $f(y_i \overset{?}{\in} X)$ for $1 \leq i \leq n$.

The protocol steps:

1. P_1 constructs a hash table for the set X.
2. P_2 constructs a cuckoo table for the set Y.
3. P_1 picks a set of β η-bit random values $R = \{r_1, ..., r_\beta\}$.
4. P_1 and P_2 run Protocol 3 with their respective inputs: (hash table, R) and cuckoo table. Let the output of P_2 be $R' = \{r'_1, ..., r'_\beta\}$.
5. P_1 and P_2 run β parallel Ciampi-Orlandi PSM protocols for functional equality testing, where for the i-th run, the inputs of P_1 and P_2 are r_i and r'_i. For each item y_j in Y, P_2 outputs the i-th Ciampi-Orlandi PSM protocol output where y_j is placed into i-th bin in the cuckoo table.

5 Security

5.1 Definitions

Since there are two parties who run the protocol, it is enough to prove that the protocol is secure when one of the parties is corrupted. There are two possible cases: either P_1 or P_2 is corrupted.

We follow the simulation-based security proof paradigm. Since we only consider honest-but-curious adversaries, the existence of a simulation in the "ideal world" whose protocol transcript is computationally indistinguishable from the adversary's view in the protocol execution in the "real world" (together with the parties' outputs in both worlds) proves that the protocol is secure. The basic idea in this proof paradigm is that if it is possible for the simulator to create a protocol transcript indistinguishable from the real execution transcript, then the transcript doesn't reveal any piece of information about the private input of the honest party. This security proof paradigm was formalized in [20] as follows. Protocol π implements the functionality $\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2)$ where the output of P_1 and P_2 are $\mathcal{F}_1(x, y)$ and $\mathcal{F}_2(x, y)$, respectively, and x and y are the inputs of the parties. The view of P_i for $i \in \{1, 2\}$ (denoted as $view_i^{\pi}(x, y)$) in the execution of the protocol π is the input of P_i, the internal random number coin tosses, the messages received from the other party in the execution of the protocol, and the outputs. The existence of probabilistic polynomial-time (PPT) algorithms S_i (the simulators) that takes the input of P_i and the output of P_i such that

$$\{S_i(w_i, \mathcal{F}_i(x, y))\}_{x,y} \approx \{view_i^{\pi}(x, y)\}_{x,y}$$

for $i \in \{1, 2\}$ where $w_1 = x$ and $w_2 = y$ proves that the protocol π realizes the functionality $\mathcal{F}$ securely.

As for the underlying primitives, namely OT, OPPRF, and FEQT, whose functionalities were presented as Functionalities 1, 4, and 2, respectively, there exists simulators who can simulate the view for both parties. These simulators take the input and output of the corresponding party as input, and produce indistinguishable views as output. In our proofs, we make use of these simulators for the underlying primitives.

Lastly, in our proofs, we provide the simulators for semi-honest adversaries. Note that the simulated view (including the outputs) must be indistinguishable from the real view. In all our proofs, this is either obvious (directly comes from the security of the underlying primitive, or comes from the fact that the simulated values are picked from the same distribution as the original ones), or were proven by others (in which case we also cite those papers). Thus, we do not delve deep into the indistinguishability discussions, considering also the page limits.

5.2 Security of our One-Time OPPRF Construction

Theorem 1. *Protocol 1 securely realizes Functionality 4 when P_1 is corrupted by a semi-honest adversary $\mathcal{A}$, assuming that the OT protocol is semi-honest secure.*

Proof. The input set X and the programmed values T are given to the simulator S. The simulator computes a garbled Bloom filter GBF_X using its random tape such that $\bigoplus_i^k GBF_X[h_i(x_j)] = t_j$ for $1 \leq j \leq n$. S runs the simulator of OT as the sender m times, where for the i-th run, the input of the simulator is $((0, GBF_X[i]), \perp)$. Here, $(0, GBF_X[i])$ is the input of the sender in the OT protocol and there in no output of the sender. Thus, the simulated view and output of the parties, and the view of the adversary in the real execution of the protocol and the output of the parties are indistinguishable.

Theorem 2. *Protocol 1 securely realizes Functionality 4 when P_2 is corrupted by a semi-honest adversary $\mathcal{A}$, assuming that the OT protocol is semi-honest secure.*

Proof. The input item y and the output $\bigoplus_{i=1}^k GBF_y[h_i(y)]$ are given to the simulator S. The simulator constructs the Bloom filter using y regularly, and creates GBF'_y by running the following steps:

1. Set random values to $GBF'_y[h_i(y)]$ for $1 \leq i < k$.
2. Set $GBF'_y[h_k(y)] = \bigoplus_{i=1}^k GBF_y[h_i(y)] \oplus \bigoplus_{i=1}^{k-1} GBF'_y[h_i(y)]$.
3. Set $GBF'_y[i] = 0$ if $BF_y[i] = 0$.

Finally, S runs the OT simulator as the receiver m times, where in the i-th, run the receiver's input is $BF_y[i]$ and the receiver's output is $GBF'_y[i]$. The proof concludes when we show that GBF'_y is indistinguishable from GBF_y. While it is rather obvious that they are perfectly indistinguishable, for the interested reader, we refer to the proofs of Theorems 4 and 6 in [10] for the details.

5.3 Security of our PSM protocol

Theorem 3. *Protocol 2 securely realizes Functionality 3 when P_1 is corrupted by a semi-honest adversary $\mathcal{A}$, assuming that the OPPRF and FEQT protocols are semi-honest secure.*

Proof. The simulator S is given the input set X. S picks a random value r using its random tape and sets $T = \{t_1 = r, ..., t_n = r\}$. The simulator S runs the simulator of OPPRF protocol with the input $((X, T), \perp)$. Then, S runs the simulator of FEQT protocol with the input $(r, \perp)$. This completes the whole simulation, and indistinguishability is a direct result of the underlying simulators.

Theorem 4. *Protocol 2 securely realizes Functionality 3 when P_2 is corrupted by a semi-honest adversary $\mathcal{A}$, assuming that the OPPRF and FEQT protocols are semi-honest secure.*

Proof. The simulator S is given the input item y and the output $f(y \stackrel{?}{\in} X)$. The simulator picks a η-bit random value r''. S runs the simulator of OPPRF with the input (y, r'') and the simulator of FEQT with the input $(r'', f(y \stackrel{?}{\in} X))$. S does not know the uniform random value r' used in the real execution, but it follows the same distribution as r'', and therefore they are perfectly indistinguishable. The computational indistinguishability comes from the FEQT and OPPRF simulations, which are based on OT simulations.

5.4 Security of our Batch One-Time OPPRF Construction

Theorem 5. *Protocol 3 securely realizes Functionality 5 when P_1 is corrupted by a semi-honest adversary $\mathcal{A}$, assuming that the OT protocol is semi-honest secure.*

Proof. The simulator S is given the input set of sets X and the programmed values set T. The simulator computes a garbled Bloom filter using its random tape such that $\bigoplus_{j=1}^{k} GBF_X[h_{i,j(x_{i,l})}] = t_{i,l}$. S runs the simulator of the OT protocol as the sender with the input $(GBF_X, \perp)$. This concludes the simulation. Indistinguishability directly comes from the garbled Bloom filter construction following the protocol, and the OT simulator.

Theorem 6. *Protocol 3 securely realizes Functionality 5 when P_2 is corrupted by a semi-honest adversary $\mathcal{A}$, assuming that the OT protocol is semi-honest secure.*

Proof. The input set Y and the output R' iare given to the simulator S. The simulator constructs a Bloom filter for Y and a garbled Bloom filter GBF'_Y following the steps:

1. Sets random values to $GBF'_Y[h_{i,j}(y_i)]$ for $1 \leq i \leq \beta$ and $1 \leq j < k$.
2. Sets $GBF'_Y[h_{i,k}(y_i)] = \bigoplus_{j=1}^{k-1} GBF'_Y[h_{i,j}(y_i)] \oplus r'_i$ for $1 \leq i \leq \beta$.
3. Sets $GBF'_Y[i] = 0$ if $BF_Y[i] = 0$.
4. Sets random values to the empty cells in GBF'_Y.

Then, S runs the simulator of the OT protocol as the receiver with the input (BF_Y, GBF'_Y). Note that the garbled bloom filters GBF'_Y and GBF_Y are indistinguishable from the real ones (see also Theorems 4 and 6 in [10]).

5.5 Security of our PSI protocol

Theorem 7. *Protocol 4 securely realizes Functionality 6 when P_1 is corrupted by a semi-honest adversary $\mathcal{A}$, assuming that the batch OPPRF and FEQT protocols are semi-honest secure.*

Proof. The input set X is given to the simulator S. The simulator computes the hash table for X and picks β η-bit random values $R'' = \{r''_1, ..., r''_\beta\}$ using its random tape. Then S runs the simulator of batch OPPRF protocol with the input $((\text{hash table}, R), \perp)$. Finally, S runs the simulator of FEQT protocol β times, where the input in the i-th run is $(r_i, \perp)$. Since r''_i and r_i are random numbers from a uniform distribution, they are indistinguishable. Hence, indistinguishability of S follows the indistinguishability of the underlying batch OPPRF and FEQT simulators.

Theorem 8. *Protocol 4 securely realizes Functionality 6 when P_2 is corrupted by a semi-honest adversary $\mathcal{A}$, assuming that the batch OPPRF and FEQT protocols are semi-honest secure.*

Proof. The simulator S is given the input set Y and the output $F(y_i \overset{?}{\in} X)$ for $1 \leq i \leq n$. S computes a cuckoo table for the set Y and picks β η-bit random values $R'' = \{r''_1, ..., r''_\beta\}$. S runs the simulator of batch OPPRF protocol with the input (cuckoo table, R'') and the simulator of FEQT protocol β times, where the input in the i-th run is $(r''_i, F(y_j \overset{?}{\in} X))$ where y_j is assigned to the i-th bin. Since R' in the real execution and R'' in the ideal world are uniformly selected sets of random numbers, they are indistinguishable. Hence, indistinguishability of S follows the indistinguishability of the underlying batch OPPRF and FEQT simulators.

6 Performance Evaluation

6.1 Complexity Analysis

Parameter Choices. We take the number of hash functions used in the construction of Bloom filters as $k = \eta$ and follow the choice of [10] to set the size of the Bloom filter as taking $m = 1.44kn$. Note that taking $k = \eta$ doesn't reduce the security level to statistical correctness parameter because the result of BF-based OPPRF protocol are random numbers which then be inputs of the equality testing protocol. Following the parameters in [25], we choose the number of bins as $1.27n$ and the number of cuckoo hashes as 3, which makes the probability of having at least one item in the stash 2^{-40}, consistent with our preferred statistical correctness parameter η.

Asymptotic Complexity of our PSM protocol. Protocol 1 requires $O(n)$ hash function computations for the Bloom filters and $O(n)$ hash function computations to perform the oblivious transfer extension. FEQT employs $O(\eta)$ operations for η oblivious transfers in the Ciampi-Orlandi PSM protocol. Thus, the asymptotic computation complexity of our PSM protocol becomes $O(n)$. The communication complexity comes from the oblivious transfers. Considering the oblivious transfer extension communication complexity as linear in the number of OTs, the communication complexity of Protocol 2 is $O(n)$.

Concrete Complexity of our PSM protocol. The concrete complexity can be computed as follows. For the Bloom filters, P_1 and P_2 compute nk and k hash functions, respectively. For the OT-extension in the OPPRF part, they run m oblivious transfer whose total computation complexity is approximately equal to $3m$ symmetric key operations thanks to the oblivious transfer extension [16]. Finally, the parties execute Ciampi-Orlandi PSM protocol where the number of items in the set of P_1 is one, which makes the computation complexity 6η symmetric key operations at P_1 and 5η symmetric key operations at P_2 (the reader can refer to [17] for the complexity calculation for the FEQT protocol). Thus the computation complexity of the protocol at the party where majority of workload is done is $nk + 3m + 6\eta$. Since we choose $m = 1.44kn$ and $k = \eta$ then the complexity becomes $5.32n\eta + 6\eta$. For the parameter $\eta = 40$ the

complexity will be $212.8n + 240$ symmetric key operations. The communication complexity comes from the oblivious transfers. In the OPPRF step, the message lengths in the oblivious transfer is η bits, while for the FEQT part, it is $2(\kappa + \eta)$ bits. Considering that the total number of bits transferred in the OT extension equals to 2 times the items' length times number of pairs, the communication complexity of the protocol becomes $2 \times m \times \eta + 2 \times \eta \times 2 \times (\kappa + \eta) = 2 \times (1.44 \times n \times \eta) \times \eta + 2\eta \times 2 \times (\kappa + \eta) = 2.88n\eta^2 + 4\kappa\eta + 4\eta^2$.

Asymptotic Complexity of our PSI protocol. While it seems that there are $O(n \log n / \log \log n)$ items in the hash table of P_1, which makes the length of the Bloom filters $O(n \log n / \log \log n)$, the actual number of items is $O(3n) = O(n)$ since the other items are random values padded to the bins to make the number of items in the bins $O(\log n / \log \log n)$. Thus, the length of the bloom filters can be $O(m) = O(1.44 \times (3n) \times k) = O(4.32n\eta) = O(n)$, which requires $O(n)$ computation and communication for $O(4.32n\eta) = O(n)$ oblivious transfer and $O(1.27n) = O(n)$ equality testing protocol executions, where each equality test protocol run requires $O(1)$ computation and communication.

Concrete Complexity of our PSI protocol. The concrete complexities can be computed as follows. To construct the cuckoo hash table, P_2 and P_1 perform at most $3n$ hash operations. Then, they construct BF performing kn and $3kn$ hash computations, respectively. They execute $4.32n\eta$ OTs using OT extension, which costs $3 \times 4.32n\eta$ hash computations. In the last step, P_1 and P_2 perform $1.27n \times (5\eta)$ and $1.27n \times (6\eta)$ hash computations, respectively. Thus the total computation cost on the party who has the maximum overhead is $3n + 3nk + 3 \times 4.32n\eta + 1.27n \times (6\eta) = 26.58n\eta + 3n$. The communication cost comes from the OT executions for Bloom filter and equality test. Since for the Bloom filter, $4.32n\eta$ η-bit message pairs are obliviously sent, the dominant cost is $2 \times 4.32n\eta \times \eta = 8.64n\eta^2$. For the equality test, the length of the pairs is $2 \times (\kappa + \eta)$ and the number of pairs is $1.27n$; hence, the dominant part is $2 \times 1.27n \times 2(\kappa + \eta)$. Thus, the total communication cost is approximately is $8.64n\eta^2 + 5.08n(\kappa + \eta)$.

6.2 Experimental Verification

Setup. We implemented Ciampi-Orlandi and our PSM protocol using C programming language and GMP library. In our experiment setup, P_1 and P_2 run on the same machine as different processes and communicate with each other over a TCP channel. We run the protocols for different size of sets and item lengths on a single CPU core of a computer that has 2.1 Ghz 16-core Intel Xeon CPU with 64 GB RAM. In the experiments, we chose RSA 2048 as asymmetric encryption algorithm in base OT, the statistical correctness parameter η as 40 bits, AES as the encryption algorithm, SHA-256 with different initialization vectors as the hash functions. We take the f function such that it outputs 128-bit wire labels. We take the number of hash functions in the construction of Bloom filters in our protocols as $k = 40$. The results are the averages over 10 executions of the protocols.

PSM. Table 1 shows the total amount of data transmitted between P_1 and P_2 during the execution of the protocols and the run-times in LAN and WAN setting. As can be seen from the table, our BF-based semi-honest PSM protocol has linear complexity both on computation and communication, and we provide comparable performance. Our asymptotic advantage becomes visible with larger ℓ values.

Table 1. Performance results of Ciampi-Orlandi and our PSM protocols. Run-time estimates are done for LAN and WAN under the assumption that the bandwidth in LAN (respectively in WAN) is 1 Gbps (100 Mbps) and RTT is 1 ms (100 ms).

Protocol		Ciampi-Orlandi PSM			Our PSM		
Set size n		$n = 2^{12}$	$n = 2^{14}$	$n = 2^{16}$	$n = 2^{12}$	$n = 2^{14}$	$n = 2^{16}$
Comm. [MB]	$\ell = 32$	5.4	21.3	84.5	6.0	23.6	93.8
	$\ell = 48$	8.1	31.8	126.5	6.5	25.4	101.0
	$\ell = 64$	10.7	42.3	168.5	7.4	29.0	115.4
LAN [ms]	$\ell = 32$	2045	4195	11717	2583	6508	22031
	$\ell = 48$	2444	5759	18115	2655	6564	22337
	$\ell = 64$	2793	7361	24396	2656	6599	22542
WAN [ms]	$\ell = 32$	10207	36389	139436	11651	42118	163745
	$\ell = 48$	14686	53824	209315	12419	44895	174973
	$\ell = 64$	18966	71296	279050	13781	50371	196904

PSI. We also implemented our PSI protocol to validate our performance analysis and compare the efficiency of our protocol with the existing solutions. We choose the number of hash functions in cuckoo hashing as 3, the number of bins as $1.27n$, and the size of the BF as $n \times 1.44 \times 3 \times k$ where k is the number of hash functions used in BF and 3 comes from the number of hash functions in cuckoo hashing. We evaluated the effect of k on the performance of our PSI protocol running the protocol for different k values which is related to the correctness of our protocol. The results are given in Table 2.

Table 2. Effect of number of hash functions in Bloom filter on the performance of our PSI protocol.

	Comm [MB]			LAN [ms]			WAN [ms]		
	$n = 2^{10}$	$n = 2^{12}$	$n = 2^{14}$	$n = 2^{10}$	$n = 2^{12}$	$n = 2^{14}$	$n = 2^{10}$	$n = 2^{12}$	$n = 2^{14}$
$k = 40$	9.4	36.9	147.4	2604	6804	22488	16812	62577	245277
$k = 60$	11.5	45.7	182.5	3018	8586	28725	20400	77660	304566
$k = 80$	13.8	54.5	217.6	3545	10278	35086	24403	92652	363980

We run our PSI protocol for different item bit lengths and set sizes choosing $k = 40$, which satisfies enough correctness in practical applications, and obtained

the results in Table 3. The table verifies our complexity claims and shows that our PSI protocol has linear communication and computation complexities. We also present the linear trend in computation complexity of our protocol in Figure 1 where the numbers are taken from Table 3 for $\ell = 32$ and LAN setting.

Table 3. Performance results of our PSI protocol.

	Comm [MB]			LAN [ms]			WAN [ms]		
	$\ell = 32$	$\ell = 48$	$\ell = 64$	$\ell = 32$	$\ell = 48$	$\ell = 64$	$\ell = 32$	$\ell = 48$	$\ell = 64$
$n = 2^8$	2.4	2.8	3.5	1407	1498	1556	5034	5730	6847
$n = 2^{10}$	9.4	10.6	13.2	2604	2710	3024	16812	18731	22976
$n = 2^{12}$	36.9	42.1	52.4	6804	7323	7891	62577	70956	87091
$n = 2^{14}$	147.4	168.0	209.3	22488	24486	27757	245277	278412	344105
$n = 2^{16}$	589.0	671.5	836.6	85134	92271	106268	975384	1107216	1370755

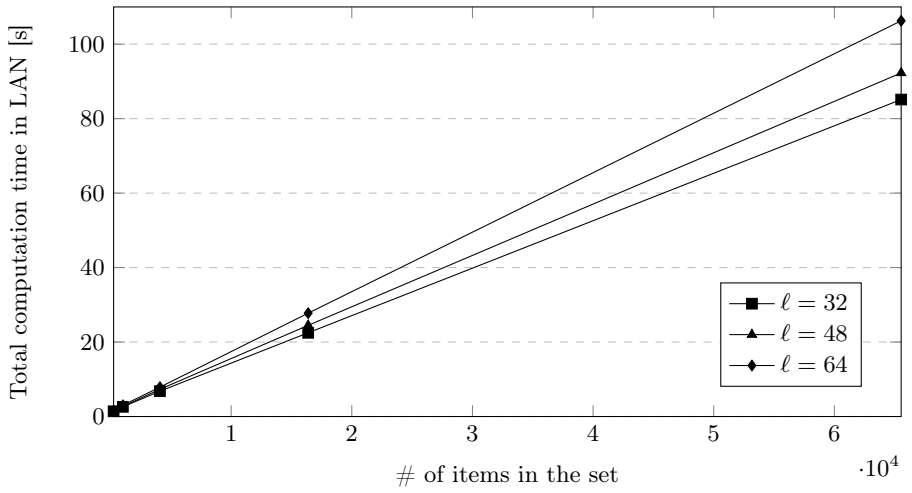

Fig. 1. Computation complexity of our PSI protocol for different set sizes and item bit lengths in the LAN setting.

Table 3 shows that for $n = 2^{12}$ and $\ell = 32$ our protocol's communication and computation complexity is (36.9 MB, 6804 ms in LAN) while the numbers for other circuit based PSI protocols of [25], [26] and [24] respectively are (9 MB, 1199 ms), (51 MB, 5031 ms) and (130 MB, 7825 ms) as given in [25].

With Figure 2, we compare our concrete computation complexity of our PSI protocol with the complexity of no-stash PSI solution of [25] for the case that $\ell = 32$ and the setting is LAN. In practice, circuit-based solutions like [25]

enjoy the benefits of recent advances in the two-party computation techniques. Therefore, we conclude that Bloom filter based solutions and oblivious transfer extension techniques should be investigated further in practice to improve their concrete complexity. It may be that our advantage becomes visible for different parameters, but we could not obtain the implementation of [25], and thus could only compare with the values provided in their paper.

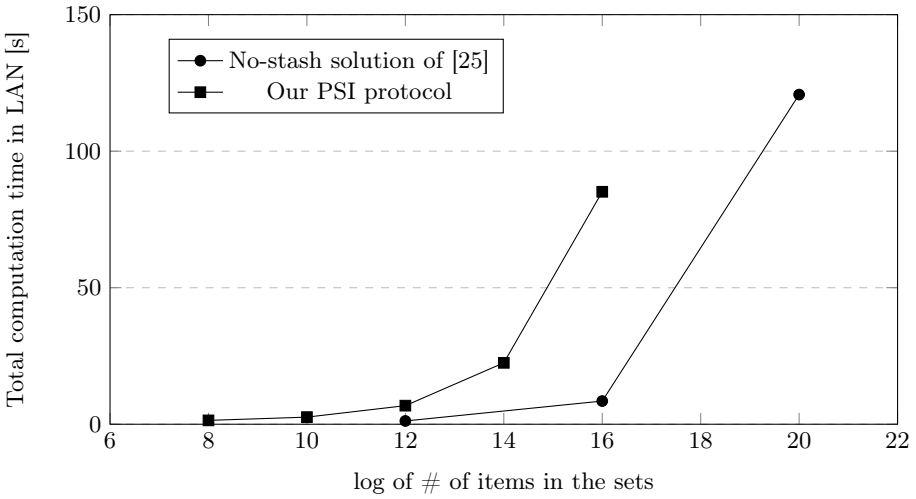

Fig. 2. Comparison of our protocol with no-stash PSI solution of [25].

7 Conclusion

We proposed the first private set intersection (PSI) protocol that outputs a function of the intersection, with linear communication and computation complexities, to the best of our knowledge. To construct such a protocol, we first designed a one-time oblivious programmable pseduo-random function (OPPRF) and then proposed a private set membership (PSM) protocol. To reduce the complexity while converting the PSM solution to a PSI protocol using hashing techniques, we constructed another primitive that is called a batch one-time OPPRF. Finally, using these new constructions, we introduced our PSI protocol with linear communication and computation complexities. We also implemented our protocols to validate our performance analysis and show concrete efficiency of our protocols.

References

1. G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM CCS*, pages 535–548, 2013.
2. B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
3. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl. Efficient two-round OT extension and silent non-interactive secure computation. *IACR Cryptology ePrint Archive*, 2019:1159, 2019.
4. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO*, pages 489–518, 2019.
5. M. Ciampi and C. Orlandi. Combining private set-intersection with secure two-party computation. In *SCN*, pages 464–482, 2018.
6. E. D. Cristofaro, P. Gasti, and G. Tsudik. Fast and private computation of cardinality of set intersection and union. In J. Pieprzyk, A. Sadeghi, and M. Manulis, editors, *Cryptology and Network Security, 11th International Conference, CANS 2012, Darmstadt, Germany, December 12-14, 2012. Proceedings*, volume 7712, pages 218–231. Springer, 2012.
7. E. D. Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In R. Sion, editor, *Financial Cryptography and Data Security, 14th International Conference, FC 2010, Tenerife, Canary Islands, Spain, January 25-28, 2010, Revised Selected Papers*, volume 6052 of *Lecture Notes in Computer Science*, pages 143–159. Springer, 2010.
8. A. Davidson and C. Cid. An efficient toolkit for computing private set operations. In J. Pieprzyk and S. Suriadi, editors, *Information Security and Privacy - 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3-5, 2017, Proceedings, Part II*, volume 10343 of *Lecture Notes in Computer Science*, pages 261–278. Springer, 2017.
9. S. K. Debnath and R. Dutta. Secure and efficient private set intersection cardinality using bloom filter. In J. López and C. J. Mitchell, editors, *Information Security - 18th International Conference, ISC 2015, Trondheim, Norway, September 9-11, 2015, Proceedings*, volume 9290 of *Lecture Notes in Computer Science*, pages 209–226. Springer, 2015.
10. C. Dong, L. Chen, and Z. Wen. When private set intersection meets big data: an efficient and scalable protocol. In *ACM CCS*, pages 789–800, 2013.
11. B. H. Falk, D. Noble, and R. Ostrovsky. Private set intersection with linear communication from general assumptions. In L. Cavallaro, J. Kinder, and J. Domingo-Ferrer, editors, *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society, WPES@CCS 2019, London, UK, November 11, 2019*, pages 14–25. ACM, 2019.
12. M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, pages 303–324, 2005.
13. Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.
14. M. Ion, B. Kreuter, A. E. Nergiz, S. Patel, M. Raykova, S. Saxena, K. Seth, D. Shanahan, and M. Yung. On deploying secure computing commercially: Private intersection-sum protocols and their business applications. *IACR Cryptol. ePrint Arch.*, 2019:723, 2019.

15. M. Ion, B. Kreuter, E. Nergiz, S. Patel, S. Saxena, K. Seth, D. Shanahan, and M. Yung. Private intersection-sum protocol with applications to attributing aggregate ad conversions. *IACR Cryptol. ePrint Arch.*, 2017:738, 2017.
16. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO*, pages 145–161, 2003.
17. F. Karakoç, M. Nateghizad, and Z. Erkin. SET-OT: A secure equality testing protocol based on oblivious transfer. In *ARES*, pages 12:1–12:9, 2019.
18. V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu. Efficient batched oblivious PRF with applications to private set intersection. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 818–829. ACM, 2016.
19. V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu. Practical multiparty private set intersection from symmetric-key techniques. In *ACM CCS*, pages 1257–1272, 2017.
20. Y. Lindell. How to simulate it - A tutorial on the simulation proof technique. In Y. Lindell, editor, *Tutorials on the Foundations of Cryptography.*, pages 277–346. Springer International Publishing, 2017.
21. C. A. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *Proceedings of the 1986 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 7-9, 1986*, pages 134–137. IEEE Computer Society, 1986.
22. J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, pages 681–700, 2012.
23. R. Pagh and F. F. Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.
24. B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: Private set intersection using permutation-based hashing. *IACR Cryptology ePrint Archive*, 2015:634, 2015.
25. B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai. Efficient circuit-based PSI with linear communication. In *EUROCRYPT*, pages 122–153, 2019.
26. B. Pinkas, T. Schneider, C. Weinert, and U. Wieder. Efficient circuit-based PSI via cuckoo hashing. In J. B. Nielsen and V. Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 125–157. Springer, 2018.
27. B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on OT extension. In *USENIX Security*, pages 797–812, 2014.
28. B. Pinkas, T. Schneider, and M. Zohner. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.*, 21(2):7:1–7:35, 2018.
29. M. O. Rabin. How to exchange secrets by oblivious transfer. Technical report, Harvard Aiken Computation Laboratory Technical Report TR-81, 1981.
30. P. Rindal and M. Rosulek. Improved private set intersection against malicious adversaries. In *EUROCRYPT*, pages 235–259, 2017.
31. A. Shamir. On the power of commutativity in cryptography. In J. W. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 582–595. Springer, 1980.
32. Y. Zhao and S. S. M. Chow. Can you find the one for me? privacy-preserving matchmaking via threshold PSI. *IACR Cryptology ePrint Archive*, 2018:184, 2018.