Server Aided PIR-PSI for Unbalanced Set Sizes

Abstract

Private Set Intersection (PSI) allows two parties, each with their own private set of data, to efficiently compute the intersection of their sets without revealing any information about their individual elements. For unbalanced set sizes, the performance of most PSI protocols is unsatisfactory because the computation and communication costs scale with the size of the larger set. We present a new protocol that uses Private Information Retrieval (PIR) to perform server-aided information-theoretic PSI, using the server-aided online/offline model. We manage to reduce each PIR query to $O(\sqrt{N})$ sub-linear time, where $N$ is the larger set size. We introduce two protocol implementations, one using PRFs and achieving the full efficiency in communication and computation, and another one which maintains the same computational cost, and also the whole protocol’s information-theoretic which does not rely on any cryptographic assumptions.

1 Introduction and Motivation

Private set intersection (PSI) is a fundamental problem in cryptography, which allows two parties, each with a private set of data, to compute the intersection of their sets without revealing any information about their own individual elements. This problem has broad applications in diverse areas such as secure multiparty computation, distributed systems, and privacy-preserving data analysis. In recent years, PSI has received increased attention due to the growing need for privacy-preserving solutions in various domains, such as healthcare, finance, and social networks.

The vast majority of PSI protocols today are designed or applied under the setting that the two parties have their sets of similar size. However, when it comes to the circumstances where the set size of two parties differs greatly, the communication and computation cost for most of these PSI protocols will increase in proportion to the size of the larger set. The case where set size of two objects differs significantly has many application scenarios in reality, such as contact discovery, mobile malware detection, and leaked passwords discovery. To solve these unbalanced PSI (or asymmetric PSI) problems, there are several approaches. The first one, which is also the most intuitive, is to simply apply standard PSI to the unbalanced-set-size cases. However, even for some state-of-the-art works that solve balanced PSI problems [KKRT16,PSZ18], the communication cost is at least $O(N + n)$, where $N$ and $n$ are size of two sets, which means that the communication cost is linear in the total size of two parties. This is obviously undesirable for the unbalanced set size setting, since $n$ will be rather small but $N$ is very large, perhaps the size of a large company’s database. Another approach that takes into account the unbalanced size of sets is PSI based on Homomorphic Encryption. Some named works and their follow-ups [CLR17,CHLR18,CMdG+21] that based on Fully Homomorphic Encryption requires communication cost linear in the size of the smaller set, and sub-linear in the size of the larger one. The weakness of FHE based PSI protocols is that, the servers need large amount of heavy homomorphic evaluation, which leads to large server-side computational cost. There are also some other papers [KLS+17,RA18] that introduce
a pre-processing phase, where the server sends some large messages to the client, and then conduct
the main process of the PSI.

Recently another approach [DRRT18] based on a different security model gives researchers
new inspiration. This work combines the techniques of two-server Private Information Retrieval
(PIR) and two-party PSI, taking advantage of the high efficiency of PIR and its good adaptation
to the unbalanced set size setting. They use PIR based on Distributed Point Functions [BGI15,
BGI16, GI14] to achieve sub-linear communication cost of $O(\log(N))$. However, at the same time we
observed that the computational cost of their work is not optimized, since PIR based on distributed
point functions still needs $O(N)$ linear server-side computational cost. Previous PIR protocols and
approaches tend to focus more on the communication side, but sacrifice some computational cost.
In this paper, we propose a new protocol of PIR-PSI, which under the setting of two servers and
offline/online model. We get the inspiration from the PIR work [CGK20, KCG21] which uses a brand
new structure named puncturable sets to achieve high efficiency PIR, in time sub-linear in the size
of the database. More specifically, the asymptotic computation complexity of this puncturable-
sets based PIR is $O(\sqrt{N})$ on the server side. So our protocol managed to reach the asymptotic
computation complexity in terms of the Online Phase to $O(n \cdot \delta \sqrt{N_e})$, where $\delta$ is the statistical
parameter. For the benefit of the online/offline model, most of the heavy work in the offline phase
can be done through night or smartphone’s sleep mode. We also succeeded in constructing a fully
information-theoretic version of our protocol that does not rely on cryptographic assumptions.

2 Related Work

As we have discussed in the introduction section, traditionally, PSI protocols, such as those proposed
in [KKRT16] and [PSZ18] have been optimized for scenarios where the two parties possess sets
of similar sizes. [KKRT16] provides a lightweight protocol aiming for oblivious evaluation of a
pseudorandom function (OPRF), and applies the technique to the PSI problem. On the other
hand, [PSZ18] introduce their optimization on the approach based on OT extension. However,
although their protocols were considered as state-of-the-art when they were published, they did
not take two parties’ set size into consideration, which leads to the communication costs that are
linear in the total size of both parties’ data sets. The intuition is that to solve the challenge of
unbalanced PSI, there would be alternative approaches which specifically targeted on this setting,
instead of directly applying the state-of-the-art standard PSI to it.

A notable direction has been the use of Fully Homomorphic Encryption (FHE) in PSI protocols,
and the inspirational thing is that cryptography scientists began to notice and attach importance to
the real-world applications on unbalanced set size setting. [CLR17] first introduces a PSI protocol
that is constructed by fully homomorphic encryption under the semi-honest adversaries setting. The
performances in terms of the communication is very satisfying in that the communication complexity
is linear in the size of the smaller set, and logarithmic in the larger set. Later, [CHLR18] comes as a
comprehensive collection and improvement of the previous one, which adds the support for arbitrary
length items and also achieves better concrete performance. However, as we have mentioned in
the motivation section, a large amount of overhead cost of FHE comes from the computation
side, as the homomorphic multiplications bring considerable computational burdens on the servers
side. [CMdG+21] fortunately notices this bottleneck and provides their PSI protocol that has an
asymptotically better computation cost than the previous two papers’ protocols, requiring $O(\sqrt{|X|})$
homomorphic multiplications, where $|X|$ is the larger set size. Their approach is considered as the
state-of-the-art in terms of the usage of fully homomorphic encryption, and if we want to keep improving the performance especially for the online time, we need to come up with other methods. Recently, the idea of online-offline model draws lots of attentions, and cryptography scientists start to convey the model to the field of PSI. [KLS+17, RA18] are two representative papers that make the heavy work done at the pre-processing (offline) phase. We notice that in the field of Private Information Retrieval (PIR), there are also new approaches [CGK20, KCG21] that are based on this online-offline model, so we decide to combine the idea and technique of PIR with PSI. There is one work [DRRT18] sharing the similar approaches that also make the PIR-PSI as the basic framework and structure, but the PIR they have used is based on Distributed Point Functions [BGI15, BGI16, GI14]. Our approach is aimed at improving the performance in terms of the computation cost, especially the online computation time, both asymptotically and concretely.

3 Preliminaries

Secure Multi-Party Computation Secure multi-party computation (MPC) [Yao86, GMW87] allows multiple parties, each holding a private input, to jointly compute a function on their private inputs without revealing anything beyond the output of the function.

We say an adversary is semi-honest if it follows the protocol execution honestly, while trying to extract information as much as it can from the execution. In our work, we assume both servers and the client are semi-honest. We follow the Universal Composition (UC) security definition of MPC, and more detail about UC security and proof can be found in [Can01].

Private Set Intersection (PSI) Private set intersection (PSI) is a specific secure two-party computation (2PC) protocol which allows two parties, each with a private set of items or elements, to jointly compute the intersection of their sets without revealing any other information (their own respective elements).

Cuckoo Hashing Cuckoo Hashing is an efficient hashing scheme which used to address and resolve hash collisions, and maintain a constant worst-case lookup time. Instead of using just one hash function, cuckoo hashing uses k hash functions h1, . . . , hk. During the time of insertion of item x, we first examine whether one of the slot of hash table T[h1(x)], T[h2(x)], . . . , T[hk(x)] is occupied or not. If there is one slot that is not occupied, then the item is inserted in that slot; otherwise we randomly choose one of these k locations and evict the existing item x′ and put x into the slot. Afterwards we re-insert the evicted item x′ into the table following the same procedure. The process continues until an empty position is found to insert the item, or the total eviction-insertion procedure reaches a threshold of loop rounds.

Private information retrieval (PIR) Private Information Retrieval (PIR) is defined as the problem of retrieving a specific item stored in the database of the server without revealing which item the client retrieves. More formally, it can be generalized as the client privately retrieving the i-th bit out of an N-bit string from the server. The privacy on the user side makes it an interesting and popular question attracting cryptography scientists for a long time.

Notation We use \( [v] \) to denote an additive secret sharing of a value \( v \in \mathbb{Z}_{2^\ell} \) between two servers A and B. We use \( \overset{\$}{\leftarrow} \) to denote random sampling from a uniform distribution. We use \( [n] \) to
denote the set \{1, 2, \ldots, n\}. For a vector \( \mathbf{v} \), we use \( \mathbf{v}[i] \) to denote the \( i \)-th element of the vector. By \( \text{neg}(\lambda) \) we denote a negligible function, i.e., a function \( f \) such that \( f(\lambda) < 1/p(\lambda) \) holds for any polynomial \( p(\lambda) \) and sufficiently large \( \lambda \). Suppose a cuckoo hashing table \( CT \), we choose \textit{cuckoo table expansion parameter} \( e \) to set the cuckoo table size to be \(|CT| = e \cdot N = N_e \), such that inserting \( N \) elements into cuckoo hash table \( CT \) succeeds without stash with probability \( \geq 1 - 2^{-\lambda} \), here \( \lambda \) is statistical security parameter.

4 Our Protocol

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>statistical security parameter</td>
<td>( \lambda )</td>
</tr>
<tr>
<td>computational security parameter</td>
<td>( \kappa )</td>
</tr>
<tr>
<td>server database</td>
<td>( DB )</td>
</tr>
<tr>
<td>server set size [# of elements]</td>
<td>( N =</td>
</tr>
<tr>
<td>client database</td>
<td>( CD )</td>
</tr>
<tr>
<td>client set size [# of elements]</td>
<td>( n =</td>
</tr>
<tr>
<td>server cuckoo hash table</td>
<td>( CT )</td>
</tr>
<tr>
<td>server cuckoo hash table size</td>
<td>( N_e = e \cdot N )</td>
</tr>
<tr>
<td>number of cuckoo hash functions</td>
<td>( k )</td>
</tr>
</tbody>
</table>

**Protocol:** PSI (Two servers \( A \& B \), the client \( C \))

\( DB \subseteq \{0, 1\}^l \) is the set held by both Server \( A \) and \( B \), \( CD \subseteq \{0, 1\}^l \) is the set held by Client \( C \).

\( N = |DB|, n = |CD| \).

\( \lambda \) and \( \kappa \) are the statistical and computational security parameters respectively.

**[Offline Phase]**

(i) Hashing: Server \( A \), \( B \) and Client \( C \) agree on the same \( k \) hash functions \( h_1, \ldots, h_k : \{0, 1\}^l \rightarrow \{N_e\} \) and cuckoo table size \( N_e = |CT| \), compute their own Cuckoo Hashing Table \( CT \):

\[
\forall y \in DB, \exists i \in \{1, \ldots, k\} \text{ s.t. } CT[h_i(y)] = y
\]

(ii) Sampling: Server \( A \) randomly samples \( m = \delta \sqrt{N_e} \) sets \( \{S_i\}_{i\in[m]} \), each of size \( |S_i| = \sqrt{N_e} \), and all elements in the set are distinct integers from \( 0, 1, \ldots, N_e - 1 \). Here \( \delta \) is the statistical parameter that \( \forall i \in \{0, 1, \ldots, N_e - 1\}, \exists j \in [m], i \in S_j \) with probability \( \geq 1 - 2^{-\lambda} \). Server \( A \) sends all \( m \) sets to Client \( C \).

(iii) Calculating Hints: Server \( A \) computes \( m \) sums: \( \text{sum}_{i\in[m]} := \bigoplus_{j\in S_i} CT[j] \). Server \( A \) also samples \( m \) masks: \( ra_{i\in[m]} \xleftarrow{\$} \{0, 1\}^l \) and sends \( \{\text{sum}_i \oplus ra_i\}_{i\in[m]} \) to Server \( B \). Server \( B \) also samples \( m \) masks: \( rb_{i\in[m]} \xleftarrow{\$} \{0, 1\}^l \) and sends \( \{\text{hint}_i := \text{sum}_i \oplus ra_i \oplus rb_i\}_{i\in[m]} \) to client \( C \).

(iv) QueryMask: Client \( C \) samples \( m \) pairs of masks: \( rca_{i\in[m]} \xleftarrow{\$} \{0, 1\}^l, rcb_{i\in[m]} \xleftarrow{\$} \{0, 1\}^l \). Client \( C \) sends \( \{rca_i\}_{i\in[m]} \) to Server \( B \), and sends \( \{rcb_i\}_{i\in[m]} \) to Server \( A \). Afterwards, Server \( A \) send \( \{rcb_i \oplus ra_i\}_{i\in[m]} \) to Server \( B \); Server \( B \) send \( \{rca_i \oplus rb_i\}_{i\in[m]} \) to Server \( A \).
A. Finally, Server A will hold \( \{ qmaskA_i = rca_i \oplus ra_i \oplus rb_i \}_{i \in [m]} \); Server B will hold \( \{ qmaskB_i = rcb_i \oplus ra_i \oplus rb_i \}_{i \in [m]} \).

Offline Summary:
Server A will keep Cuckoo Hashing Table \( CT, \{ ra_i \}_{i \in [m]} \), and \( \{ qmaskA_i \}_{i \in [m]} \)
Server B will keep Cuckoo Hashing Table \( CT, \{ rb_i \}_{i \in [m]} \), and \( \{ qmaskB_i \}_{i \in [m]} \)
Client C will keep \( m \) sets \( S_{i=1...m} \), hints: \( \{ hint_i \}_{i \in [m]} \), \( \{ rca_i \}_{i \in [m]} \) and \( \{ rcb_i \}_{i \in [m]} \)

Protocol: PSI (Two servers A & B, the client C) Cont.

[Online Phase]
For Client C’s each item \( x \in CD \):
For \( \forall \alpha \in \{1...k\} \), \( \psi = h_\alpha(x) \):
C samples a bit:
\[
b \leftarrow_R \text{Bernoulli}(2(\sqrt{N_e} - 1)/N_e) : \]
--- If \( b = 0 \), process the Common case;
--- If \( b = 1 \), process the Rare case;

(a) Common case:

(i) Query:
Client C randomly select \( \beta \in [m] \) s.t. \( \psi \in S_\beta \). Then C randomly samples a new set \( S_{new} \) with size \( |S_{new}| = \sqrt{N_e} \) and \( \psi \in S_{new} \).
Client C randomly select \( \gamma \in S_{new} \setminus \{ \psi \} \), samples \( r_\gamma \leftarrow \{0,1\}^l \). C sends \( (S_{new} \setminus \{ \psi \}, \gamma) \) and \( r_\gamma \) to Server A.
Client C randomly select \( \tau \in S_\beta \setminus \{ \psi \} \), samples \( r_\tau \leftarrow \{0,1\}^l \). C sends \( (S_\beta \setminus \{ \psi \}, \tau) \) and \( r_\tau \) to Server B.
Client C sets \( q = \{0\}^{\beta-1}||1||0^{m-\beta} \). C samples \( q_0 \leftarrow \{0,1\}^m \) and send to Server A; C sets key \( q_1 = q \oplus q_0 \) and send to Server B.

(ii) Fill up PET vectors:
A computes \( Sum_A := \bigoplus_{i \in S_{new} \setminus \{ \psi \}} CT[i] \) and \( SR_A := \bigoplus_{i \in [m] \text{s.t. } q_0[i]=1} qmaskA_i \), sends \( (Sum_A \oplus CT[\gamma] \oplus r_\gamma) \) and \( SR_A \) to B;
B computes \( Sum_B := \bigoplus_{i \in S_\beta \setminus \{ \psi \}} CT[i] \) and \( SR_B := \bigoplus_{i \in [m] \text{s.t. } q_1[i]=1} qmaskB_i \), sends \( (Sum_B \oplus CT[\gamma] \oplus r_\tau) \) and \( SR_B \) to A;
C computes \( Sum_C := hint_\beta \oplus x \oplus \bigoplus_i \text{s.t. } q_0[i]=1 rca_i \oplus \bigoplus_j \text{s.t. } q_1[j]=1 rcb_j \).

\( c_1, c_2, c_3, c_4 \) and \( a_1, b_2, a_3, b_4 \) are all vectors of size \( k \).
Client C: \( c_1[\alpha] = Sum_C \); Server A: \( a_1[\alpha] = Sum_A \oplus SR_A \oplus SR_B \);
Client C: \( c_2[\alpha] = Sum_C \); Server B: \( b_2[\alpha] = Sum_B \oplus SR_A \oplus SR_B \);
Client C: \( c_3[\alpha] = x \oplus r_\gamma \); Server A: \( a_3[\alpha] = Sum_A \oplus (Sum_B \oplus CT[\gamma] \oplus r_\gamma) \);
Client C: \( c_4[\alpha] = x \oplus r_\gamma \); Server B: \( b_4[\alpha] = Sum_B \oplus (Sum_A \oplus CT[\gamma] \oplus r_\gamma) \);
(iii) Updates ra, rb and hint:

Number of sets: \( m = m + 1 \);
Set \( S_m = S_{new} \);
Server A samples \( ra_m \leftarrow \{0,1\}^l \), sends Server B: \((Sum_A \oplus ra_m)\);
Server B samples \( rb_m \leftarrow \{0,1\}^l \), sends Client C: \((Sum_A \oplus ra_m) \oplus (Sum_B \oplus rb_m)\);
Client C updates \( hint_m := hint_\beta \oplus (Sum_A \oplus r_0) \oplus (Sum_B \oplus r_1) \).

C samples \( rcb_m \leftarrow \{0,1\}^l \) sends to A; samples \( rca_m \leftarrow \{0,1\}^l \) sends to B
Server A sends \((rcb_m \oplus ra_m)\) to B; Server B sends \((rca_m \oplus rb_m)\) to A
Server A updates qmask: \( qmask_A = (rca_m \oplus ra_m \oplus rb_m) \);
Server B updates qmask: \( qmask_B = (rcb_m \oplus ra_m \oplus rb_m) \).

(b) Rare case:

Client C samples a bit \( b' \leftarrow R Bernoulli(1/2) \):
-- If \( b' = 1 \), A and B will be switched for all operations.

w.l.o.g. we show the procedures below under

(i) Query:
Client randomly samples a new set \( S_{new} \) with \( |S_{new}| = \sqrt{N_m} \) and \( \psi \in S_{new} \).
Client C randomly select \( \gamma \in S_{new} \setminus \{\psi\} \), samples \( r_\gamma \leftarrow \{0,1\}^l \). C sends \((S_{new} \setminus \{\psi\}, \gamma)\) and \( r_\gamma \) to Server A.

Client C randomly select \( \tau \in S_{new} \setminus \{\gamma\} \), samples \( r_\tau \leftarrow \{0,1\}^l \). C sends \((S_{new} \setminus \{\gamma\}, \tau)\) and \( r_\tau \) to Server B.

Client C set \( q = \{0\}^m \). C samples \( q_0 \leftarrow \{0,1\}^m \) and send to Server A; C sets key \( q_1 = q \oplus q_0 \) and send to Server B.

(ii) Fill up PET vectors:

A computes \( Sum_A := \bigoplus_{i \in S_{new}} \{\psi\} CT[i] \) and \( SR_A := \bigoplus_{i \in [m], q_0[i] = 1} qmask_A i \), sends \((Sum_A \oplus CT[\gamma] \oplus r_\gamma)\) and \( SR_A\) to B;

B computes \( Sum_B := \bigoplus_{i \in S_{new}} \{\gamma\} CT[i] \) and \( SR_B := \bigoplus_{i \in [m], q_1[i] = 1} qmask_B i \), sends \((Sum_B \oplus CT[\tau] \oplus r_\tau)\) and \( SR_B\) to A;

C computes \( Sum_C := hint_\beta \oplus x \oplus \bigoplus_{i, q_0[i] = 1} rca_i \oplus \bigoplus_{j, q_1[j] = 1} rcb_j \).

\( c_1, c_2, c_3, c_4 \) and \( a_1, b_2, a_3, b_4 \) are all vectors of size \( k \).

Client C: \( c_1[\alpha] = Sum_C \);
Server A: \( a_1[\alpha] = Sum_A \oplus SR_A \oplus SR_B \);

Client C: \( c_2[\alpha] = Sum_C \);
Server B: \( b_2[\alpha] = Sum_B \oplus SR_A \oplus SR_B \);

Client C: \( c_3[\alpha] = x \oplus r_\tau \);
Server A: \( a_3[\alpha] = Sum_A \oplus (Sum_B \ominus CT[\tau] \oplus r_\gamma) \);

Client C: \( c_4[\alpha] = x \oplus r_\gamma \);
Server B: \( b_4[\alpha] = Sum_B \ominus (Sum_A \ominus CT[\tau] \oplus r_\gamma) \).

(iii) Updates ra, rb and hint:
(Dummy updates, since no set from \( \{S_i\}_{i \in [m]} \) has been chosen under the rare case.)
Number of sets: \( m = m + 1 \);
Server A samples \( ra_m \leftarrow \{0,1\}^l \), sends Server B: \((Sum_A \oplus ra_m)\);
Server B samples \( rb_m \leftarrow \{0,1\}^l \), sends Client C: \((Sum_A \oplus ra_m) \oplus (Sum_B \oplus rb_m)\);
Client C updates \( hint_m := hint_\beta \oplus (Sum_A \oplus r_0) \oplus (Sum_B \oplus r_1) \).
of the cuckoo hashing, suppose the hashing index of each item \( x \) or each item \( x \) functionally:

Subsets received by \( CD \), Server \( A \) sends \((rca_m \oplus ra_m)\) to \( B \); Server \( B \) sends \((rca_m \oplus ra_m)\) to \( A \)

Server \( A \) updates qmask: \( qmaskA_m = (rca_m \oplus ra_m \oplus rb_m)\);

Server \( B \) updates qmask: \( qmaskB_m = (rb_m \oplus ra_m \oplus rb_m)\)

At the end of every \( k \) queries, do 4 batched PETs:

\[ \text{PET}_{C, A}(c_1, a_1); \text{PET}_{C, B}(c_2, b_2); \text{PET}_{C, A}(c_3, a_3); \text{PET}_{C, B}(c_4, a_4); \]

And add the query item \( x \) to the PSI output vector if one of 4 batched PETs return True.

**Protocol:** PSI (Two servers \( A \) & \( B \), client \( C \))

**[PET]:**

Without loss of generality, we show the procedure of PET between Client \( C \) and Server \( A \), and name the vectors they hold as \( c \) and \( a \) respectively.

- Client \( C \) randomly generates \( k \) masks \( \{r_i \in \{0,1\}^l\}_{i \in [k]} \) and sends them to Server \( A \);
- Server \( A \) computes \( X : \{X_i := a[i] \oplus r_i\}_{i \in [k]} \); Client \( C \) computes \( Y : \{Y_i := c[i] \oplus r_i\}_{i \in [k]} \);
- Client \( C \) sends \( Y \) to Server \( B \);
- Server \( A \) generates a random permutation \( \Pi : [k] \rightarrow [k] \), and sends it to Server \( B \);
- Server \( A \) and \( B \) apply the random permutation \( \Pi : [k] \rightarrow [k] \) on \( X \) and \( Y \) respectively;
- Server \( A \) generates \( k \) sets of \((l + 1)\) random \( \lambda \)-bit strings \( \{s^i_0, s^i_1, ..., s^i_l\}_{i \in [k]} \);
- Server \( A \) computes \( M_i := s_0 \oplus (\bigoplus_j X_{[j]=1} s^i_j) \) and sends \( \{M_i\}_{i \in [k]} \) to Client \( C \);
- Server \( B \) computes \( M'_i := s_0 \oplus (\bigoplus_j Y_{[j]=1} s^i_j) \) and sends \( \{M'_i\}_{i \in [k]} \) to Client \( C \);
- Client \( C \) returns \( True \) if \( \exists i \in [k], M_i = M'_i \); otherwise returns \( False \)

**Functionality \( \mathcal{F}^l_{\text{PSI}} \):**

**Parties:** Two servers \( A, B \) and a client \( C \).

**Inputs:** Two servers \( A \) and \( B \) input the same data set \( DB \subseteq \{0,1\}^l \). The client \( C \) inputs the data set \( CD \subseteq \{0,1\}^l \).

**Functionality:** On receiving \( DB \) from \( A, B \) and \( CD \) from \( C \), send the set intersection \( v = CD \cap DB \) back to \( C \).

Figure 1: Ideal functionality \( \mathcal{F}^l_{\text{PSI}} \) for computing the PSI.

**Correctness** For correctness, we prove that the client succeeds in receiving the intersection of the client database \( CD \) and the server database \( DB \) with all but a negligible probability.

Without loss of generality, here we prove the correctness for Client \( C \)’s each item: \( \forall x \in CD, x \in \mathcal{F}_{\text{PSI}}^l(CD, DB) \) if and only if \( x \in DB \).

For each item \( x \), the potential hashing locations are: \( h_\alpha(x), \alpha \in \{1, 2, \cdots, k\} \). Due to the property of the cuckoo hashing, suppose the hashing index of \( x \) in \( DB \) is \( \psi = h_\alpha(x) \). And we denote the subsets received by \( A \) and \( B \) at [Online Phase] step (i) are \( S_A \) and \( S_B \) respectively.

For each query, it will be processed as either Common case or Rare case. We prove the two cases separately:
Common case:
\[ S_B = S_{\beta \setminus \{\psi\}} \], and B computes the \( \text{Sum}_B = \bigoplus_{i \in S_B} CT[i] = \bigoplus_{i \in S_{\beta \setminus \{\psi\}}} CT[i] \);
Since \( q = \{0\}^{3-1} \cdot \{1\}^{m-3} = q_0 \oplus q_1 \), Server B’s vector:
\[
b_2[\alpha'] = \text{Sum}_B + \text{SR}_A + \text{SR}_B = \bigoplus_{i \in S_{\beta \setminus \{\psi\}}} CT[i] \oplus \bigoplus_{i \in \{m\} s.t. q_0[i]=1} \text{qmask}_A_i \oplus \bigoplus_{i \in \{m\} s.t. q_1[i]=1} \text{qmask}_B_i
\]
Client C’s vector:
\[
c_2[\alpha'] = \text{Sum}_C = \text{hint}_\beta \oplus x \oplus \bigoplus_{i \ s.t. \ q_0[i]=1} \text{rca}_i \oplus \bigoplus_{j \ s.t. \ q_1[j]=1} \text{rcb}_j
\]
So the second from 4 PET batches:
\[ PET_{C,B}(c_2, b_2) \] returns True if and only if \( x = CT[\psi] \), which means \( x \in DB \).

Rare case:
w.l.o.g. we show the proof under \( b' = 0 \), which is exact the protocol shows above.
\( S_A = S_{\text{new} \setminus \{\psi\}} \) and \( S_B = S_{\text{new} \setminus \{\gamma\}} \), and A sends \( (\text{Sum}_A \oplus CT[\gamma] \oplus r_\gamma) \) to B;
Server B’s vector:
\[
b_1[\alpha'] = \text{Sum}_B \oplus (\text{Sum}_A \oplus CT[\gamma] \oplus r_\gamma) = CT[\psi] \oplus CT[\gamma] \oplus CT[\gamma] \oplus r_\gamma = CT[\psi] \oplus r_\gamma
\]
Client C’s vector:
\[
c_1[\alpha'] = x \oplus r_\gamma
\]
So the forth from 4 PET batches:
\[ PET_{C,B}(c_4, b_4) \] returns True if and only if \( x = CT[\psi] \), which means \( x \in DB \)

**Security**

**Theorem 4.1.** The protocol \( \Pi_{\text{PSI}} \) securely computes the ideal functionality \( F^l_{\text{PSI}} \) against a semi-honest adversary that corrupts either the client C or one of the two servers (A, B).

**Corrupted Server A:** Since the whole protocol is lengthy and intricate to some extend, we formalize and expand the view of Server A in detail:

\[
(\text{View}_A(\text{DB}_A, \text{DB}_B, \text{CD}_C)) = \\
(m_1^A = \{\text{rcb}_i\}_{i \in \{m\}}, m_2^A = \{(\text{rca}_i \oplus \text{rb}_i)\}_{i \in \{m\}}, m_3^A = S_{\text{new}}, m_4^A = \gamma, m_5^A = r_\gamma, m_6^A = q_0, m_7^A = (\text{Sum}_B \oplus CT[\gamma] \oplus r_\gamma), m_8^A = \text{SR}_B, m_9^A = \text{rcb}_m, m_{10}^A = (\text{rca}_m \oplus \text{rb}_m))
\]
For security against the semi-honest server A, we construct \( S_A \) as follows.

\( S_A \) runs the protocol to generate its view with the following exceptions:
- for \( m_2^A \), \( S_A \) sends \( m \) uniformly random strings to A: \( m_2^A \xleftarrow{}\{0,1\}^i_{i \in \{m\}} \)
- for \( m_3^A \), \( S_A \) sends uniformly sampled \( S_{\text{new}} \) with size \( S_{\text{new}} = \sqrt{N_e} - 1 \) to A.
• for $m_2^A$ and $m_4^A$, $S_A$ sends uniformly sampled string to $A$: $m_2^A \xleftarrow{\$} \{0,1\}^l$, $m_4^A \xleftarrow{\$} \{0,1\}^l$

Since the idea functionality $F^\psi_{\text{PSI}}$ (shown in Figure 1) is deterministic, we can therefore prove the theorem separately on correctness and privacy.

For any input $DB \subseteq \{0,1\}^l$ and $CD \subseteq \{0,1\}^l$,

$$\left(\text{View}^\Pi_A (DB_A, DB_B, CD_C)\right) \approx \left(S_A \left(1^\lambda, f_A (DB_A, DB_B, CD_C)\right), \right)$$

Corrupted Server $B$: As shown in our protocol’s description, we can observe that the server $B$’s role is very similar to server $A$’s, in spite of some minor differences. The construction of the simulator $S_B$ and the proof of view indistinguishability is also almost the same.

Corrupted Client $C$: For security against the semi-honest server $C$, we construct $S_C$ as follows.

$S_C$ runs the protocol to generate its view with the following exceptions:

• at [Offline Phase] (iii), $S_C$ randomly samples $\{\text{hint}_j\}_{i \in [m]} \xleftarrow{\$} \{0,1\}^l$.

• at [Online Phase] (iii), $S_C$ randomly samples $\text{hint}_{\text{update}} \xleftarrow{\$} \{0,1\}^l$

• at the end of every $k$ queries, given the client $C$’s output, if the current query item $x \notin C$’s output, $S_C$ randomly samples $\{M_j\}_{i \in [k]}$ and $\{M'_j\}_{i \in [k]}$ where all elements are sampled from a uniform distribution over $\{0,1\}^\lambda$; if the current query item $x \notin C$’s output, $S_C$ randomly samples $\{M_j\}_{i \in [k]}$ and $\{M'_j\}_{i \in [k]}$ in the same way except that they share a common element at a same index.

Next we prove that for any input $DB \subseteq \{0,1\}^l$ and $CD \subseteq \{0,1\}^l$,

$$\left(\text{View}^\Pi_C (DB_A, DB_B, CD_C), \text{Out}^\Pi_{AB} (DB_A, DB_B, CD_C)\right) \approx \left(S_C \left(1^\lambda, f_C (DB_A, DB_B, CD_C)\right), f_{AB} (DB_A, DB_B, CD_C)\right)$$

5 Performance and Future Work

The experiments are run on the benchmark machine which has Intel Core i5 6 Core (i5-8500T) 2.10 GHz, and 8 GB RAM, and 500 GB HDD. The online running time of our protocol for the input size $n = 5535, N = 2^{20}$ is 587ms, where $n$ is the cardinality of the smaller set and $N$ is the cardinality of the larger set. As shown in our protocol description and procedures, for the Online Phase, the asymptotic computation complexity is $O(n \cdot \delta \sqrt{N_c})$, where we have stated in the protocol that $\delta$ is the statistical parameter that $\forall i \in \{0,1, \ldots, N_c - 1\}$, $\exists j \in [m], i \in S_j$ with probability $\geq 1 - 2^{-\lambda}$. However, due to the impact of the statistical parameter would have on the concrete performance, the future work would probably be aimed at lowering constant factors of the online time. In addition, although it is considered that most of the heavy work in the offline phase can be done through night or using high-speed Wi-Fi connection, our protocol’s offline time hardly has advantage over the state-of-the-art online-offline based PSI protocols.
References


