Oblivious Transfer Extension-Based Private Set Intersection: A Survey of the Major Results

Huiyang He
School of Cyber Science and Technology
University of Science and Technology
Anhui, China
hhe@mail.ustc.edu.cn

Chenglin Li
School of Cyber Science and Technology
University of Science and Technology
Anhui, China
li1116@mail.ustc.edu.cn

***Abstract*—As one of the most active field of Secure Multiparty Computation (MPC), Private Set Intersection (PSI) had derived many different approaches to securely compute the intersection of sets from different parties. While Oblivious Transfer (OT) extension greatly solved the problem of inefficiency of OT, which is a basic primitive of MPC, it has not been utilized well in PSI until 2013 when Dong etc. suggested the Garbled Bloom Filter (GBF). After that, more practical PSI protocols based on OT extension was constructed some of them had been showed to have the most balanced performance in most scenes. In this survey, we briefly reviewed the PSI, classification of PSI protocols and two major results from the OT Extension-Based class. We also presented and analyzed the results of comparisons between different PSI protocols, conducted by the authors of those two protocols.**

Keywords—MPC, PSI, OT, PSI Protocol

I. INTRODUCTION

Private Set Intersection (PSI) enables two parties and holding their own sets to jointly compute the intersection of their sets using protocols, at the end of which nothing more than the intersection is revealed. Given the motivation of practice and privacy problem in Big Data era, PSI problems has been extensively studied for several decades, and has been utilized in many area, such as human genome research [1], botnet detection [2], privacy-preserving data mining [3], social networks [4] and cheater detection in online games [5]. In the following section, we will give more detailed description of some PSI applications.

PARAMETERS:
Two parties: Sender S and Receiver R.
S has set X.
R has set Y.
FUNCTIONALITY:
S outputs $\perp$.
R outputs $X \cap Y$.

Fig. 1. Basic Functionality of PSI

A. PSI Applications

The basic PSI functionality can be used in applications where two parties want to perform JOIN operations. JOIN is a very basic operation for database tables, and thus is also great needed in many scenes: Data Scientists use JOIN for data analysis; Service providers use JOIN operations to build new features; Merchant use JOIN for intelligent business. PSI allows parties to perform JOIN operations over database tables that must be kept private.

- **Measuring advertisement conversion rate.** With the coming of the digital era, it is common for merchants to advertise on digital media. When the advertiser and the dealer calculate the fee, they must agree on an advertisement conversion rate. Usually, the first step to calculate Advertisement conversion rate is to identify the people who have seen the advertisement, also have completed a transaction with the merchant. This problem can be generalized as that the merchant holds a list of customers who have done a transaction and the advertiser holds a list of users who have seen the advertisement. Then they want to find the intersection of those two lists. This, of course, can be done by one of them sending its whole list to the other. However, in real worlds, those data are valuable property and neither of them would be willing to provide the other with its own data. However, if those two parties use a safe PSI protocol, then the computation is secured, and they can learn nothing more than the intersection. Thus, the privacy problem is solved, and they both protect their legal property through utilizing PSI.
- **Contact discovery** is a generic functionality for social media service nowadays. When a new user signs up for the service, the service provider might want to identify a list of registered users who can be found in the contact of the new user, and provide better service based on this list, such as friend recommendations or news pushing. Neither is it possible to ask the service provider to present its complete database, nor should the user's privacy be undermined. This, again, can also simply be accomplished by applying PSI.
- **Appointment making**. Making appointment can be a very usual activity, such as with the doctor, bank adviser or business partner. It helps people stop from wasting time and make their lives more organized. With the modern lives becoming more complicated, the need of appointment is also raising. When people start to use computer to organize appointments, finding the intersection of two people's availability is not a hard work. However, not everyone is willing to share their time schedule with others, and PSI can complete this computation in a privacy-preserving manner.

Besides those concrete examples, PSI is sometimes also a basic tool when one wants to apply other privacy preserving

mechanics. For example, PSI can be used to do data alignment as a preparation for the privacy preserving machine learning.

B. Different Scenes of PSI

Generally, there are four different scenes when applying PSI protocols.

- **PSI on small sets** might be the best studied and most practical one. Since the size of the sets are not large, ex pensive pre-computation can be used to lower the overhead of communication cost.
- **PSI on large sets** usually happens between big data centers. Data centers usually have enough computation and storage resource and are connected by backbone network, and therefore, balanced PSI protocols might perform best.
- **PSI on unbalanced sets** means that one party has a larger set than the other, just like the case of contact discovery. Usually, the party with the larger set also has more computation resource, so PSI protocols that put heavy computation tasks on only side should be better in this case.
- **Secure computation on intersection** is an extended functionality of PSI. Although intuitively one might think it is easy to compute on the intersection with a PSI protocol, it is non-trivial to securely compute on the intersection, because in this case, more information needs to be protected, i.e., only the result of the computation is revealed, but the element in the intersection is still kept secret.

C. Security Model

As a specific problem of MPC, PSI shares the same security model with MPC. Different the conventional attack model, an attacker in a MPC security model does not directly involves in the protocol but would control one or more participants instead and tries to perform attacks through those corrupted parties.

- **Semi-Honest Security.** A semi-honest adversary is only curious and tries to learn as much as possible. Therefore, parties controlled by a semi-honest attacker would follow the protocol as expected. However, the views of those corrupted parties are known by the attacker, who would then combining all those views to build knowledge about the protocol and the infer more information. This attacker is also known as a passive one.
- **Malicious Security.** A malicious, or an active adversary is one such that might control the corrupted parties to deviate from the designed protocol to violet security. Without doubt, this is a much more powerful attacker, and therefore protocols with malicious security need better and more sophisticated scheme compared with semi-honest security. While they might be less efficient due to the extra protections, they are also more meaningful and practical in real world applications.

Security proofs for MPC problems are different from the ones for traditional security model, and involves a new form of formal proof, called the ideal-real paradigm. It introduces an ideal world, in which all the security guarantees needed are captured by a trusted third party. Then in the ideal world, the participants compute with the help of the third party, while in real world, the parties communicate using the protocol. The main point of using this ideal-real paradigm is to simulate an adversary in the ideal world and prove that this simulated adversary can capture all the views of the adversary in the real world, in other worlds, a real attacker cannot infer more information than what is seen by the attacker in the ideal world. We only give a brief idea of how the ideal-real paradigm works here, and for ones who have interest in the security proves of MPC problem, we highly recommend they refer to "A Pragmatic Introduction to Secure Multi-Party Computation", in which there is a more detailed and systematic description of the security assumptions of MPC problems.

For this survey, all the OT-based protocols we discuss are semi-honest safe. Intuitively, one may ask why we should study those semi-honest security protocol, given that they might not meet the security needs in real world, but more researchers constructed their protocol using the semi-honest model. The first reason is that semi-honest is a good starting point and be a base for building the malicious secure protocol. Moreover, semi-honest has less security requirements, and thus researchers can focus on the efficiency instead of disturbed by the complex security proves.

II. Some Basic Primitives

A. Oblivious Transfer

PARAMETERS:

- *Two parties: Sender S and Receiver R.*
- *S has two secretes, $x_0, x_1 \in \{0,1\}^n$,*
- *R has a selection bit $r \in \{0,1\}$*

FUNCTIONALITY:

- *R outputs x_r*
- *S outputs $\perp$*

Fig. 2. Functionality of 1-out-of-2 OT

Oblivious Transfer (OT) is introduced By Rabin in 1981 [6]. It is a kind of protocol which enables a sender to transfer one of potentially many messages to a receiver but remains oblivious as to what piece of information has been transferred and the receiver cannot know messages other than its choice. Many protocols have been constructed since 1981 and Naor-Pinkas OT proposed in 2001 is one of the most used OT protocol. They proved their protocol is secure under semi-honest setting based on the decisional discrete logarithm problem [7]. Its security is only proven according to a "half-simulation" definition of security, where an ideal world simulator is demonstrated only in the case of a cheating receiver and may fall to selective-failure attacks, where the sender causes a failure that depends upon the receiver's selection.

Later in 1988, Impagliazzo and Rudich showed that a black-box reduction from OT to a one-way function would imply $P \neq NP$ [8]. It is further not known whether such non-black-box reductions exists. This means that OT can only be built based on the public key system, and thus limits the efficiency of OT. Beaver was the first to propose OT extension, i.e., using a small number of OTs and one-way functions to obtain many OTs [8]. However, his construction is not practical, but it proved the possibility of OT extension.

Preliminaries:

- *The protocol operates over a group Z_q of prime order. g is the generator.*
- *The computational Diffie-Hellman Assumption holds.*
- *The protocol uses a function H which is assumed to be a random oracle.*

Initialization:

The sender chooses a random element $C \in Z_q$ and publishes it. (It is only important that the receiver will not know the discrete logarithm of C to the base g)

Protocol:

1) *The receiver picks a random $1 \le k \le q$, sets public key $PK_r = g^k$ and $PK_{1-r} = C/PK_r$, and sends PK_0 to the sender.*
2) *The sender computes $PK_1 = C/PK_0$ and chooses random s $\subset Z_q$. The sender encrypts x_0 by $E_0 = [g^s, H(PK_0^s) \oplus x_0]$, x_1 by $E_1 = [g^s, H(PK_1^s) \oplus x_1]$, and the sends E_0 , E_1 to the receiver.*
3) *The receiver computes $H((g^s)^k) = H(PK_r^s)$, and uses it to decrypt E_r.*

Fig. 3. Naor-Pinkas 1out of 2 OT

Ishai, Kilian, Nissim, Petrank continued the thought of Beaver, and constructed their IKNP OT extension [9] which is truly a treasure that makes most MPC become practical than they have ever been. Let OT_l^m means m instances of OT, each of which transfers l bits. IKNP OT extension realize $\binom{2}{1} OT_l^m \rightarrow \binom{2}{1} OT_m^k$, where $k \ll m$. Their construction is an analog to the combination of public-key system and private-key system. The extension protocol is composed of two main parts: the base OTs and the extended OTs. In the base OTs, the sender S and receiver R switch their roles, i.e., S is the receiver and R is the sender in the base OT. Let t^idenote the *i-th* column of matrix T and t_j denote the *j-th* row of matrix T. R holds the selection bits r and first forms two matrix T and U such that $u^i = r \oplus t^i$. They use k base $\binom{2}{1}$ OT to transfer strings of length k and thus can form a $m \times k$ matrix Q. The key observation here is that:

$$q_j = t_j \oplus [r_j \cdot s] = \begin{cases} t_j, & if\ r_j = 0 \\ t_j \oplus s, & if\ r_j = 1 \end{cases}$$

Let H be a hash function. Then the sender can compute two random strings: $H(j, q_j)$ and $H(j, q_j \oplus s)$. Notice that t_j equals either q_j or $q_j \oplus s$, depending on the receiver's choice bit r_j. Then for each row of the matrix, S can compute and sends $(y_{j,0}, y_{j,1})$ where $y_{j,0} = x_{j,0} \oplus H(j, q_j)$ and $y_{j,1} = x_{j,1} \oplus H(j, q_j \oplus s)$. After receiving $(y_{j,0}, y_{j,1})$, receiver can only learn one of them depends on its choice bit: if it is 0, then receiver can compute $H(j, q_j)$ and get $x_{j,0} = y_{j,0} \oplus H(j, q_j)$. If the bit is 1, then receiver can compute $H(j, q_j \oplus s)$ and get $x_{j,1}$. Because s is sender's secret and receiver has no way to learn it, the receiver cannot decrypt the other secret. The next problem is what value should k be or how small k can be so that we can use as less OTs as possible. Ishai etc. stated that k here is the computational security parameter and is very similar to the concept in the private encryption system and thus is usually taken to be 128 or 256.

It is also mentioned by them that it can be further reduced from $\binom{2}{1} OT_m^k$ to $\binom{2}{1} OT_k^k$, using random OT (ROT). ROT is a very important optimization technic for OT. What is difference in ROT is that rather than transfer one of the secrets to receiver, the sender now holds no messages at the beginning of the protocol, but the receiver still has a choose bit. They use a PRG and an agreed key to generate messages, such that the sender holds two messages, and the receiver can generate exactly one of the messages based on his choose bit. It is worth noticing that ROT is also a very powerful optimization technic for the KK OT Extension and OT-based PSI protocols which will be discussed below, but the way ROT is built into those protocols are not the same and might take some time to understand. The details are omitted for the simplicity of this survey.

PARAMETERS:

- *Two parties: Sender S and Receiver R.*
- *R has a selection bit $r \in \{0,1\}$.*

FUNCTIONALITY:

- *S outputs two random strings x_0, x_1.*
- *R outputs x_r.*

Fig. 4. Random Oblivious Transfer

OT has been subjected to very little implements after [9] until when Kolesnikov and Kumaresan suggested their KK OT extion in 2013 [10]. They creatively showed their fancy way of interpreting the IKNP OT extension from the view of coding theory. Let **C** be an encoding function, then the IKNP OT extension uses what they called repetition code:

$$C_{rep}(r_j) = r \cdot 1^k \stackrel{\text{def}}{=} \begin{cases} 1^k, & if\ r_j = 1 \\ 0^k, & if\ r_j = 0 \end{cases}$$

Then in the 2nd step of IKNP protocol, the receiver forms two matrix such that $t_j \oplus u_j = C_{rep}(r_j)$, and in the 3rd step, it becomes:

$$q_j = t_j \oplus [C_{rep}(r_j) \cdot s]$$

So that receiver can only learn exactly one of q_j or $q_j \oplus [C_{rep}(r_j) \cdot s]$. Form this coding view, Kolesnikov and Kumaresan found that it is not necessary to use repetition code here, and this code actually cannot carry much information and if an error correction code is used here, then the information from the codes can then be used to distinguish different messages. Therefore, they proposed to use Walsh-Hadamard (WH) Code instead of repetition code in the IKNP OT extension.

WH code is a slightly improved version of Hadamard Code and provides a way to recovers some parts of the information. For $\alpha \in \{0,1\}^q$, then the WH Code of α is:

$$WH(\alpha) = (\alpha, x)_{x \in \{0,1\}^q}$$

where the inner product between the two vector is taken modulo 2. It is the 2^q-bit string consisting of inner products of each q-bit string with α, and if we take $q = \log_2 k$, then $WH(\alpha)$ outputs a k-bit long string Let C_{WH}^k denotes k codewords using WH Codes. It is a known fact that the relative distance of C_{WH}^k is ½ when k is power of 2, so that it is safe to WH Codes in the OT extension because it assures that the receiver cannot know $C_{WH}^k(r)$ if it does not know r.

Using WH codes to substitute the repetition codes realize $\binom{n}{1} OT_l^m \rightarrow \binom{n}{1} OT_m^k$, where $k \ll m$ and $n < k$. The receiver now forms two matrices T and U, such that $t_j \oplus u_j = C_{WH}(r_j)$. Then after base OTs, the sender obtains $m \times k$ matrix Q, satisfying:

$$q_j \oplus t_j = C_{WH}(r_j) \odot s$$

Then just like in the KK OT extension, the receiver can only one of the n secret on which it choose with $C_{WH}(r_j)$.To have concrete security as IKNP, the computational security parameter k of KK OT extension needs to be set to $k = 2k_{IKNP}$, but since k is a small constant number such as 128 or 256, this would not bring much performance degrade. Considering that it induces very less extra computation but brings the OT extension from $\binom{2}{1}$ OT to $\binom{n}{1}$ OT, this brings great Improvement and make OT more practical.

PARAMETERS:

- Two parties: Sender S and Receiver R.
- S holds a random seed s for the OPRF
- R has an input r for the OPRF

FUNCTIONALITY:

- S outputs $\perp$
- R outputs $F(r, s)$

Fig. 5.Functionality of OPRF

B. Oblivious Pseudorandom Function

OPRF is another atomic operation of MPC. In OPRF, the sender holds a seed s for the PRF, and the receiver has an input i for the PRF. For the receiver, it can only evaluate $f(s, i)$ and other $f(s, j)$ such that $i \neq j$ looks totally random to it (this kind of OPRF is also referred as Relaxed-OPRF, where the Strong-OPRF requires that the receiver cannot get any information about the send's seed). For the sender, it cannot infer any information about the receiver's input i, but since it has the seed, any value can be evaluated at the server side using the PRF f. In the PSI protocols we are going to discuss, they used one-time OPRF. This kind of OPRF can be only used one time to ensure the security guarantee. It looks like to be very irrational to use one-time OPRF, but OT extension can realize lots of one-time OPRF using a small amount of OT and thus is practical to be used in PSI protocols.

C. Cuckoo Hashing

Cuckoo Hashing was first introduced by Pagh and Rodler in 2001 [11], but here we will give a brief description of a variant of it from [12]. Compared with the normal hashing, the first difference one may notice is that it uses two hash functions $h_1,\ h_2$. To avoids collision in cuckoo hashing, we need to relocating elements when a collision is found. First randomly choose a hash function and insert element e to a bin $B_{h(e)}$. If the bin $B_{h(e)}$ is already occupied, then the original element is evicted. However, we need to find a new place for the evicted elements, so it is then relocated using the same procedure. The reallocation is repeated until no more collision, or the threshold number of reallocations is reached, in which case we consider the last evicted element failed to take a proper place in the bins. If an element failed, then it will be put in the stash. The benefit from using more hash function is the raise of occupancy rate of the hash table and thus a smaller table is needed to hold a large number of elements. Although cuckoo hashing is slower than the simple hashing, it is a space-efficient hashing scheme.

D. Secrete Sharing

Secrete Sharing (SS) aims at splitting a secrete into multiple shares so that those shares can be hold by different parties. No share holder can recover the secrete on their own, and it is only when all share holders agree to reveal the secrete and provide their share, the secrete can then be constructed. There are many different approaches to implement secret sharing, such as using Chinese reminder theorem. There is a variant of SS called threshold Secrete Sharing. It allows a secret dealer to split secret into n shares and deliver them to different holders, and the secrete can only be reconstructed if enough, for example l shareholders exchange their shares with each other. A very common and famous threshold SS is Shamir's SS, which is built on Lagrange interpolation polynomial. Note that when $l = w$, the threshold SS is just the normal SS. This survey only involves the simplest secrete sharing using exclusive-or. To get n shares of a secret s, the dealer needs to generate $n - 1$ different random strings, and those strings compose the first $n - 1$ shares $r_1, \dots, r_{n-1}$. To obtain the last share, it can be computed as:

$$r_n = r_1 \oplus \dots \oplus r_{n-1} \oplus s$$

Then the secrete can then be easily reconstructed by xoring all the shares:

$$s = r_1 \oplus \dots \oplus r_{n-1} \oplus r_n$$

It must be that all shares $r_1, \dots, r_n$ are collected and then the shares can be revealed. If there is a missing or wrong share, then the constructed s' is different from the original secret s.

E. Bloom Filteres

Bloom Filter (BF) is a space-friendly data structure designed for membership testing. It comes with several well-defined hash functions and uses a bit map to record a test and can tell whether an element is in the set. Initially, all bits in a BF are set to 0 and when an element is inserted into the BF, it is hashed using all the hash functions, and all the corresponding bits are set to 1. If one wants to check whether an element is in the set or not, the hashes of the element are calculated and all locations are checked if the bit is 1 or 0. Because the hash functions are deterministic, if any of the location is 0, then the element must not be in the set. If all locations are 1, then the element has a high possibility to be in the set. However, it might return false-positive answers, in the case that an element is not in the set, but all hashed positions of this element is set to 1 by some other elements, then the BF would tell that this element is in the set, but the correct answer should be negative.

III. Classification of PSI Protocols

As an active research field, PSI becomes one of the best studied applications of MPC and is a truly practical primitive of today. Many different approaches have been suggested and a comprehensive classification of them has been first given by Pinkas etc. in [13]. We will give a classification based on their work below, but only the Naïve approach and Oblivious Transfer (OT)-Based protocol will be given detailed description.

A. Naïve PSI

During the early time when privacy-preserving intersection computation is needed, some people with cryptography knowledge might come up with a very naïve protocol, in which the JOIN operation is done between the hashes of the elements from two sets. Then both parties can identify which element are in the intersection using the intersection of hashes. While this approach is easy and efficient, it is vulnerable to brute force attack and thus insecure: if the input domain does not have high entropy, a semi-honest participant can easily identify the whole set hold by the other party through calculating hashes of all the possible inputs and compared to the received hashes. However, when the inputs

to PSI have a high entropy, naïve PSI can be proved to be secure [14]. Although this naïve PSI protocol does not always meet the privacy requirements, it is used by some companies, such as Facebook, considering that it is the fastest PSI protocol and also the easiest to implement.

B. Public-Key-Based PSI

In 1986, Meadows suggested a PSI protocol based on Diffie-Hellmann key exchange scheme [15]. Another PSI protocol based on public key system is Blind RSA protocol proposed in [16]. The advantage of public-key-based PSI is that they usually have lower communication cost compared to other approach, especially circuit-based psi if you have a very large circuit that need to be transferred. Another reason why someone should this class of PSI protocols is that when the target sets unbalanced, i.e., one has a large set while the other has a small set, the expensive public key operations can be carried by only one party, who have more computation resource. Furthermore, some public-key-based PSI can be easily extended to solve Private set intersection cardinality (PSI-CA) problem. For example, in the Diffie-Hellmann based PSI protocol, the receiver can only tell whether the element is in the intersection if the computed masks on the same position equals to the one got from the sender. Thus, a PSI-CA protocol can be built by simply adding a shuffle operation to the masks, then the receiver can only obtain a set of masks and thus get the size of intersection but cannot know exactly which element.

C. Circuit-Based PSI

PSI is usually considered as a specialized problem of MPC, which means that purposed protocol usually outperforms the generic circuit approach. However, many efforts have been done to promote the performance of general MPC circuit to make them more practical and there are also researchers trying to utilize those optimizations in PSI protocols. In [17], Huang etc. constructed several Boolean circuits, which is evaluated using Yao's garbled circuit, for PSI and their Java implementations was showed to outperform the blind-RSA protocol of [16]. Because they use the generic circuit, there is no need for extra security proof. Also, they have better extensibility for the fact that the output from a circuit can be an input to another circuit with no additional conversion. Thus, Circuit-based PSI protocols are also great candidates for PSI-CA problem. For example, one can just design a garbled circuit whose functionality is counting the size of the set and then connect it to the PSI circuit. This would make the output of the PSI circuit become the input to the counting circuit. The intermediate result would not be available to either party, and therefore neither of them can know exactly which elements are in the intersection. Moreover, any reinforce or improving technic on the generic circuit can also be applied to the circuit-based PSI. This is already proved by [17], and more optimization technics in the future might make circuit-based PSI protocols more competive.

D. Oblivious Transfer-Based PSI

Compared with other approaches, OT-based PSI was suggested in a relatively late period but has been proven to have the most balanced performance under different scenes.

In [18], Dong etc. proposed an OT-based PSI protocol by utilizing *Blossom Filter* (BF). They showed a new data structure called *Garbled Blossom Filter* which is a combination of *Secret Sharing* and blossom filter. In GBF, instead of using a bit map to record the set, it records the set by storing shares of an element in the table. They designed an algorithm for the sender to generate and distribute the shares of an element to corresponding bins properly. The SS scheme here is simply exclusive-or secret sharing. For example, if the GBF uses four hash functions, then when an element is inserted into the GBF, the elements is hashed by the four hash functions separately and the four corresponding locations hold one share of the element. If any location is already occupied, then the share is re-used. As a result, the insertion failed if all four locations are already occupied. Therefore, the table need to be big enough so that there are not that many collisions. To check whether an element is in the set, the four shares on the hashed position are first collected and then they are used to construct the element, if the recovered element is the one, we want to check, then it is in the set. Unlike BF, GBF does not return false-positive answer, because the SS composes a second check for the existence of the element.

Algorithm 1: $BuildGBF(S, n, m, k, H, \lambda)$

input : A set $S, n, m, k, \lambda, H = \{h_0, ...h_{k-1}\}$
output: An (m, n, k, H, λ)-garbled Bloom filter GBF_S

GBF_S= new m-element array of bit strings;
for $i = 0$ **to** $m - 1$ **do**
  GBF_S[i]=*NULL*; // NULL is the special symbol that means "no value"
end
for each $x \in S$ **do**
  emptySlot = -1, finalShare= x;
  for *i=0* **to** *k-1* **do**
    $j = h_i(x)$; // get an index by hashing the element
    if $GBF_S[j]$==*NULL* **then**
      if *emptySlot* ==-1 **then**
        emptySlot=j; // reserve this location for finalShare
      else
        $GBF_S[j] \xleftarrow{r} \{0,1\}^\lambda$; // generate a new share
        finalShare=finalShare$\oplus GBF_S[j]$;
      end
    else
      finalShare=finalShare$\oplus GBF_S[j]$; // reuse a share
    end
  end
  $GBF_S[emptySlot]$=finalShare; // store the last share
end
for $i = 0$ **to** $m - 1$ **do**
  if $GBF_S[i]$==*NULL* **then**
    $GBF_S[i] \xleftarrow{r} \{0,1\}^\lambda$;
  end
end

Fig. 6. Algorithm 1 BuildGBF

Algorithm 2: $QueryGBF(GBF_S, x, k, H)$

input : A gabled Bloom filter GBF_S, an element x, k, $H = \{h_0, ...h_{k-1}\}$
output: True if $x \in S$, False otherwise

$recovered = \{0\}^\lambda$;
for $i = 0$ **to** $k - 1$ **do**
  $j = h_i(x)$;
  $recovered = recovered \oplus GBF_S[j]$;
end
if $recovered == x$ **then**
  return True;
else
  return False;
end

Fig. 7. Algorithm 2 QueryGBF

Algorithm 3: $GBFIntersection(GBF_S, BF_C, m)$

input : An (m, n, k, H, λ)-garbled Bloom filter GBF_S, an (m, n, k, H)-Bloom filter BF_C, m
output: An (m, n, k, H, λ)-garbled Bloom filter $GBF_{C\cap S}$

$GBF_{C\cap S}$= new m-element array of bit strings;
for $i = 0$ **to** $m - 1$ **do**
  if $BF_C[i] == 1$ **then**
    $GBF_{C\cap S}[i] = GBF_S[i]$;
  else
    $GBF_{C\cap S}[i] \xleftarrow{r} \{0,1\}^\lambda$;
  end
end

Fig. 8. Algorithm 3 GBFIntersection

Another advantage of GBF is that it can be merged with a BF to produce a new GBF, and this GBF can tell which elements are in both the GBF set and the BF set.

Using GBF, a PSI protocol can be constructed. As figures shown above in Fig.6-8, after the sender forms its GBF and receiver forms its BF. OT was then invoked between the sender with GBF and receiver with BF. The sender uses randomly generated shares and GBF as input and receiver uses GB as choice bits. If one position in BF is 0, then the receiver gets random string. If it is 1, then the receiver gets the share on the corresponding position of GBF. After obtaining the merged GBF, the receiver can use the shares in it to check whether an element is in the intersection: for each element, the receiver can try using the shares in the corresponding bins to recover the element. If the element is successfully recovered, then this element is in the intersection. The receiver cannot infer any other elements which are not in the intersection and this protocol is thus secure. It was shown that their protocol can be very efficient for large data sets.

INPUT OF P_1
  a w-bit element x
INPUT OF P_2
  a w-bit element y

CRYPTOGRAPHIC PRIMITIVE
  An ideal $\binom{2}{1}$ OT_l^w primitive
  1) P_1 *obtains w pairs of uniformly distributed and random l-bit strings* (s_0^i, s_1^i) $1 \le i \le w$.
  2) P_1 *and* P_2 *invoke the* $\binom{2}{1}$ OT_l^w *where* P_1 *is the sender with w pairs of random strings, and* P_2 *is the receiver with* y_i *as the choice bit.*
  3) P_1 *then computes* $m_1 = \oplus_{i=1}^w s_{x[i]}^i$ *and sends to* P_2.
  4) P_2 *receives* m_1 *and compares this value to* $m_2 = \oplus_{i=1}^w s_{y[i]}^i$, *and decides that* $x = y$ *iff* $m_1 = m_2$.

Fig. 9. PEQT Protocol

In 2014, Pinkas etc. constructed a fancy PSI protocol based on OT extension [19]. They start the construction with *private equality test* (PEQT), which aims at telling whether two elements are equal in a privacy-preserving manner. The PEQT is done as follow: suppose elements are of size w, and

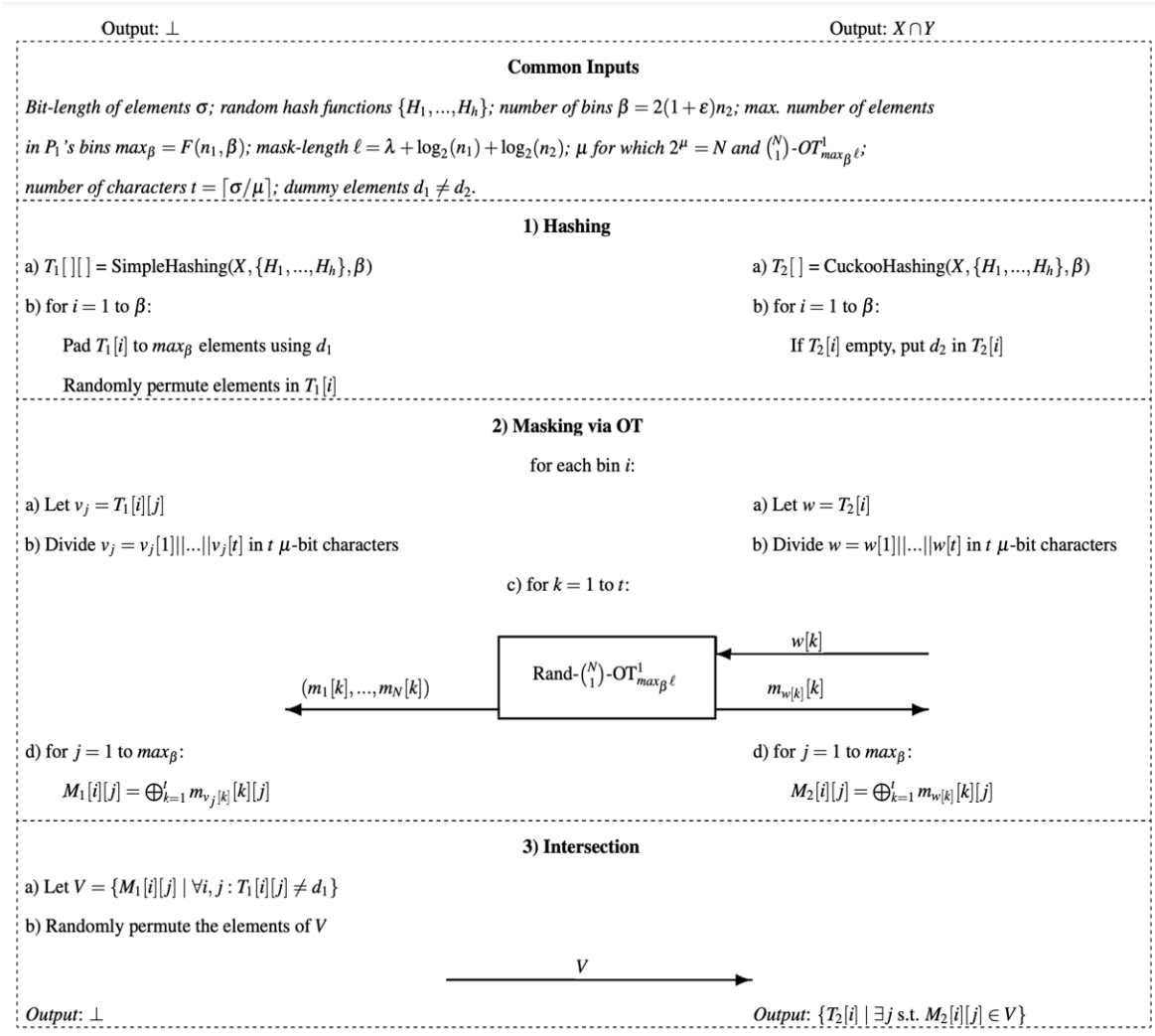

Fig. 10. OT-based PSI protocol of [PSZ 14] without stash

the sender first randomly generate w pairs of strings, each of which is of length l. Receiver and Sender then invoke the $\binom{2}{1}$ OT_l^w where the sender has w pairs of random strings as input, and the receiver use its element as the choice bit string. After the OT is done, the receiver can then compute a mask m_2:

$$m_2 = \oplus_{i=1}^{w} s_{y[i]}^{i}$$

Note that the receiver can only get this mask and at the same time the sender does not know this mask because OT guarantee this. Because the sender has all the string pairs, it can use its own element to compute its corresponding mask:

$$m_1 = \oplus_{i=1}^{w} s_{x[i]}^{i}$$

Then, m_1 is sent to the receiver, who can compare it with m_2. Clearly, if their elements are the same, then $m_1 = m_2$. This protocol needs lots of OTs as base operations and here we see the benefit of OT extension: we can use only g8 or 256 OTs, each of which would transfer $n * w$ bits string, then we can perform the PEQT for n times. In other word, each PEQT will consume several rows in the OT extension matrix. Then, the PEQT protocol was extended to *private set inclusion* protocol. To securely identify whether an element is in a set, the PEQT invoked for each element in the set against the target element. This sounds cumbersome to invoke PEQT one by one, but this can be done paralleled in OT: suppose there are n elements in the sender's set, then the sender can form l pairs of $w \cdot n$ bit strings and use them as input to the OTs. Then at the receiver side, the strings can be split and used to perform PEQT for n times. After this, a prototype of their protocol (PSZ protocol) was shown: one party can simply use the privates set inclusion protocol against the other party's set using every element in his own set, and this would need $n \cdot l$ OTs, each of which transfers a $n \cdot w$ bit string.

In PSZ protocol, the number of OTs needed is proportional to $O(w \cdot n^2)$, where n is the size of the set. To further reduce the OTs needed, they make use of hashing. Given that if two elements are equal, they must be hashed to the same bin using the same hash function, the comparisons need only to be done between those two bins with same index. Pinkas etc. tried three different hashing schemes: simple hashing, balanced hashing and cuckoo hashing. If we use two hash functions, then simple hashing just put element into both hashed positions; balanced hashing would first calculate both hash value and put the element into the bin which contains less element; Cuckoo hashing involves kick and reinsert progress as described in §2 and requires that each bin can only contain one element and an element failed to be inserted is put into the stash. PSI with two-way cuckoo hashing was shown by them have the minimum number of OTs, considering that there is only one element in the bin, then we can use just *private set inclusion* protocol rather than the private set intersection protocol between bins. But in this case, instead of both parties using cuckoo hashing, it must be that the receiver uses the cuckoo hashing, and the sender uses simple hash, so that the sender maps each element to both of two possible bins, which later can be captured by the receiver. If we use cuckoo hashing on both sides, it might happen that the receive maps element e to bin i, but the sender maps the element to bin j and as a result the computed mask are not equal, and then the element is missed in the intersection. Also, to prevent leaking of extra information, both parties must fulfill all the bins and stash, so that the adversary cannot infer any information from the position and hash function. After utilizing cuckoo hashing, the receiver now can have at most one element in each bin, and thus the number of OTs needed is reduced to $O(w \cdot n)$.

Table I. Overall communication for a larger number of hash functions h. Communication is given for a) $n1 = n2 = 2^{20}$ and b) $n2 = 2^8 \ll n1 = 2^{20}$ elements of σ = 32- bit length

h	Util. [%]	#OTs	#Masks	Comm. [MB]	
				$n_1 = n_2$	$n_2 \ll n_1$
2	50.0	$2.00n_2$ t	$\mathbf{2}n_1\ell$	148.0	**17.0**
3	91.8	$1.09n_2$ t	$3n_1\ell$	**99.8**	25.5
4	97.7	$1.02n_2$ t	$4n_1\ell$	105.3	34.0
5	**99.2**	$\mathbf{1.01}n_2$ **t**	$5n_1\ell$	114.6	42.5

Later in [20], serval optimization was applied to the PSZ protocol and thus an even more efficient protocol (PSSZ protocol) was constructed. To begin with, they used permutation-based hashing, which uses a Feistel-like

INPUT OF R
m selection strings $r = (r_1, \ldots, r_m)$, $r^i \in \{0,1\}^*$
PARAMETERS
A $(\kappa, \epsilon) - PRC$ family $\mathbb{C}$ with output length $k = k(\kappa)$.
A κ-Hamming correlation-robust H: $[m] \times \{0,1\}^k \to \{0,1\}^v$.
An ideal OT_m^k primitive.

PROTOCOL
1) *S chooses a random $C \leftarrow \mathbb{C}$ and sends it to R*
2) *S chooses $s \leftarrow \{0,1\}^k$ at random. Let s_i denote the i-th bit of s*
3) *R forms $m \times k$ matrices T, U in the following way:*
   For $j \in [m]$, choose $t_j \leftarrow \{0,1\}^k$ and set $u_j = C(r_j) \oplus t_j$
   Let t^i, u^i denote the i-th column of matrices T, U respectively.
4) *S and R interacts with OT_m^k in the following way:*
   S acts as receiver with input $\{s_i\}_{i\in[k]}$.
   R acts as sender with input $\{t^i, u^i\}_{i\in[k]}$.
   S receives output $\{q^i\}_{i\in[k]}$
   S forms $m \times k$ matrix Q such that the i-th column of Q is the such that i-th column of Q is the vector q^i.
   Let q_j denote the j-th row of Q. Note, $q_j = ((t_j \oplus u_j) \cdot s) \oplus t_j$. Simplifying $q_j = t_j \oplus (C(r_j) \cdot s)$
5) *For $j \in [m]$, S outputs the PRF seed $((C, s), (j, q_j))$.*
6) For $j \in [m]$, R outputs relaxed PRF output (C, j, t_j)

Fig. 11. BaRK-OPRF

PARAMETERS

P_1 *has input* X

P_2 *has input* Y

with $|X| = |Y| = n$

s *is an upper bound on the stash size for Cuckoo hashing*

1) P_2 *specifies random hash functions* $h_1, h_2, h_3: \{0,1\}^* \rightarrow [1.2n]$ *and tells them to* P_1
2) P_2 *assigns his items* Y *into* $1.2n$ *bins using Cuckoo hashing. Let* P_2 *keep track of* $z(y)$ *for each* y *so that if* $z(y) = \perp$ *then* y *is in the stash; otherwise,* y *is in* $h_{z(y)}(y)$*. Arrange the items in the stash in an arbitrary order*

   `P_2 *selects OPRF inputs as follows: for* $i \in [1.2n]$*, if* $bin\#i$ *is empty, then set* r_i *to a dummy value; otherwise, if* y *is in* $bin\#i$ *then set* $r_i = y||z(y)$*. For* $r_i \in [s]$*, if position I in the stash is* y*, then set* $r_i = y$*; otherwise set* r_i *to a dummy value.*
3) *The parties invoke 1.2n+s OPRF instance, with* P_2 *the receiver with input* $(r_1, \dots, r_{1.2n+s})$*.* P_1 *receives* $(k_1, \dots, k_{1.2n+s})$ *and* P_2 *receives* $F(k_i, r_i)$ *for all i.*
4) P_1 *computes:*

$$H_i = \{F(k_{h_i(x),\, x||i}) | x \in X\},\ for\ i \in \{1,2,3\}$$
$$S_j = \{F(k_{1.2n+j\, x||i}) | x \in X\},\ for\ j \in \{1, \dots, s\}$$

   And sends a permutation of each set to P_2
5) P_2 *initializes an empty set* O*, and does the following for* $y \in Y$*:* $If\ z(y) = \perp$ *and* y *is at position* j *in the stash and* $F(k_{1.2n+j,y||i},\ y) \in S_j$ *Then* P_2 *adds* y *to* O*. If* $z(y) \neq \perp$ *and* $F(k_{h_i(y),\, y||i},\ y||z(y)) \in H_{z(y)}$*, then* P_2 *adds* y *to* O*.*
6) P_2 *sends* O *to* P_1 *and both parties output* O

Fig. 12. PSI based on BaRK-OPRF

structure: let $x = x_L|x_R$ be the bit representation of an input item, and then input item is mapped to $x_L \oplus f(x_R)$, and the value stored is x_R. The mapping function ensures that if two elements x_1, x_2 store the same value in the same bin, then it must hold that $x_1 = x_2$. This speed up the process of hashing and make the value stored in the bins shorter. Because the value would be used to generate masks and compared, the short the value, the less OTs are needed. Moreover, they make the sender sends all the computed mask in a randomly permuted order, so the position information is hided and there is no need for dummy items anymore. This saves a lot of both computation and network resource, but the price is that on the receiver side, it has to search the whole mask set to see whether there is a match mask. However, this can be solved by efficient data structure and powerful search algorithm. Nevertheless, three-way cuckoo hash has been proven to achieve better occupancy of the hash table and thus a smaller table and stash is needed on the receiver side to hold the elements. Thus, the number of comparisons is reduced because there are less bins, but each element on the sender side must produce one more mask use the third hash function so that all possible positions and masks can be covered. Those three optimizations all brought huge improvements to the PSZ protocol and make this protocol more practical.

Then in 2017, [21] further boosted this OT extension-based protocol. They continued the thinking of [13] [14] and presented their efficient construction of OPRF based on KK OT extension. Their main technical observation of KK OT extension if that they do not need the Code to be efficiently decodable in OPRF. They suggested that the only requirement is that for all possible codes, the hamming weight of their exclusive OR is at least equal to the computational security parameter. This requires the matrix width of OT extension to be $3k < k' < 4k$, where k is the computational security parameter. They built their Batch-Related-key-OPRF (BaRK-OPRF) from the KK OT extension using AES as the encoding function. Follow the notations from OPRF, then just like in the KK OT extension, the receiver learns $t = q \oplus [C(r) \cdot s]$ for its input r and thus can compute $H(t)$, which is a little more than just $H(t)$ but does not violate the security assumptions, Because the sender holds q and s, it can compute PRF on any value it wants using $H(q \oplus [C(r') \cdot s])$ for any r'. This protocol realizes many instances of this kind of "PRF" but with related keys, and with BaRK-OPRF, each comparison now consumes only one OPRF and thus only one row in the OT extension matrix, make the number of OTs relevant to the length of elements. Therefore, the number of OTs is now irrelevant to the element size. In more details, the PSI protocol now only need $O(n)$ OTs to complete the comparisons. This is a great improvement on PSI on large set with large elements. This, again, make the PSI protocol has a performance closer to the unsecure naïve hash method and therefore more practical.

E. Third Party-Based PSI

Third party-based PSI introduce an additional party to help the secure computation between sender and receiver. This kind of PSI protocols usually applies the naïve hash or OPRF with the help of the third party, and therefore can be only several factor slower than the naïve approach. Hazay and Lindell used a trusted hardware token to evaluate the OPRF [22]

Table II. Run-time in ms for protocols with n = n1 = n2 elements.
(Protocols with (*) are in a different security)

Setting	**LAN**					**WAN**			
Protocol	$n = 2^8$	$n = 2^{12}$	$n = 2^{16}$	$n = 2^{20}$	$n = 2^{24}$	$n = 2^8$	$n = 2^{12}$	$n = 2^{16}$	$n = 2^{20}$
Naive Hashing(*) §3.1	1	4	48	712	13,665	97	111	558	3,538
Server-Aided(*) [KMRS14]	1	5	78	1,250	20,053	198	548	2,024	7,737
DH-based ECC [Mea86]	231	3,238	51,380	818,318	13,065,904	628	10,158	161,850	2,584,212
Bit-length $\sigma = 32$*-bit*									
OT PSI [PSZ14]	184	216	3,681	62,048	929,685	957	1,820	9,556	157,332
OT-Phasing §6	179	202	437	4,260	46,631	912	1,590	3,065	14,567
Bit-length $\sigma = 64$*-bit*									
OT PSI [PSZ14]	201	485	7,302	125,697	—	977	1,873	18,998	315,115
OT-Phasing §6	180	240	865	10,128	137,036	1,010	1,780	5,009	29,387
Bit-length $\sigma = 128$*-bit*									
OT PSI [PSZ14]	201	485	8,478	155,051	—	980	1,879	21,273	392,265
OT-Phasing §6	181	240	915	13,485	204,593	1,010	1,780	5,536	37,422

Table III. Communication in MB for PSI protocols with n = n1 = n2 elements. l = λ +log2 (n1) +log2 (n2). Assuming intersection of size 1/2 · n for TTP-based protocol.
(Protocols with (*) are in a different security model.)

Protocol	$n = 2^8$	$n = 2^{12}$	$n = 2^{16}$	$n = 2^{20}$	$n = 2^{24}$	**Asymptotic [bit]**
Naive Hashing(*) §3.1	0.01	0.03	0.56	10.0	176.0	$n_1\ell$
Server-Aided(*) [KMRS14]	0.01	0.16	2.5	40.0	640.0	$(n_1+n_2+\lvert X\cap Y\rvert)\kappa$
DH-based ECC [Mea86]	0.02	0.28	4.56	74.0	1,200.0	$(n_1+n_2)\varphi+n_1\ell$
Bit-length $\sigma = 32$*-bit*						
OT PSI [PSZ14]	0.09	1.39	22.58	367.20	5,971.20	$0.6n_2\sigma\kappa+6n_1\ell$
OT-Phasing §6	0.06	0.73	8.74	136.8	1,494.4	$2.4n_2\kappa(\lceil\frac{\sigma-\lfloor\log_2(1.2n_2)\rfloor}{8}\rceil)+(3+s)n_1\ell$
Bit-length $\sigma = 64$*-bit*						
OT PSI [PSZ14]	0.14	2.59	41.78	674.4	10,886.4	$0.6n_2\kappa*min(\ell,\sigma)+6n_1\ell$
OT-Phasing §6	0.09	1.34	18.34	290.4	3,952.0	$2.4n_2\kappa(\lceil\frac{min(\ell,\sigma)-\log_2(n_2)}{8}\rceil)+(3+s)n_1\ell$
Bit-length $\sigma = 128$*-bit*						
OT PSI [PSZ14]	0.14	2.59	46.58	828.0	14,572.8	$0.6n_2\ell\kappa+6n_1\ell$
OT-Phasing §6	0.09	1.34	20.74	367.2	5,795.2	$2.4n_2\kappa(\lceil\frac{\ell-\log_2(n_2)}{8}\rceil)+(3+s)n_1\ell$

Table IV. Run-time in ms for PSI protocols with n2 ≪ n1 elements
(Protocols with (*) are in a different security model.)

Setting	**LAN**						**WAN**			
Protocol	$n_2 = 2^8$			$n_2 = 2^{12}$			$n_2 = 2^8$		$n_2 = 2^{12}$	
	$n_1 = 2^{16}$	$n_1 = 2^{20}$	$n_1 = 2^{24}$	$n_1 = 2^{16}$	$n_1 = 2^{20}$	$n_1 = 2^{24}$	$n_1 = 2^{16}$	$n_1 = 2^{20}$	$n_1 = 2^{16}$	$n_1 = 2^{20}$
Naive Hashing(*) §3.1	33	464	7,739	35	466	7,836	560	2,775	562	2,797
Server-Aided(*) [KMRS14]	74	680	8,935	75	696	8,965	629	2,923	731	2,951
DH-based ECC [Mea86]	28,387	421,115	6,848,215	29,810	422,712	6,849,534	112,336	1,743,400	111,642	1,753,595
OT-Phasing §6										
Bit-length $\sigma = 32$	360	906	9,465	369	2,949	12,634	2,139	4,780	3,143	11,399
Bit-length $\sigma = 64$	555	1,506	15,789	581	6,146	22,368	3,349	6,879	3,923	20,345
Bit-length $\sigma = 128$	571	1,942	21,843	649	7,291	31,932	3,352	7,999	4,391	23,209

Table V. Communication in MB for special purpose PSI protocols with n2≪n1 elements. l = λ + log2 (n1) + log2 (n2). Assuming intersection of size 1/2 · n2 for the TTP-based protocol
(Protocols with (*) are in a different security model.)

Protocol	$n_2 = 2^8$			$n_2 = 2^{12}$			**Asymptotic [bit]**
	$n_1 = 2^{16}$	$n_1 = 2^{20}$	$n_1 = 2^{24}$	$n_1 = 2^{16}$	$n_1 = 2^{20}$	$n_1 = 2^{24}$	
Naive Hashing(*) §3.1	0.5	8.5	144.0	0.5	9.0	152.0	$n_1\ell$
Server-Aided(*) [KMRS14]	1.0	16.0	256.0	1.1	16.1	256.1	$(n_1+n_2+\lvert X\cap Y\rvert)\kappa$
DH-based ECC [Mea86]	2.5	40.5	656.0	2.7	41.1	664.1	$(n_1+n_2)\varphi+n_1\ell$
OT-Phasing §6							
Bit-length $\sigma = 32$	1.1	18.1	288.1	2.0	18.9	320.9	$4.8n_2\kappa(\lceil\frac{\sigma-\lfloor\log_2(2.4n_2)\rfloor}{8}\rceil)+2n_1\ell$
Bit-length $\sigma = 64$	1.1	18.1	288.1	3.2	20.1	322.1	$4.8n_2\kappa(\lceil\frac{\sigma-\lfloor\log_2(2.4n_2)\rfloor}{8}\rceil)+2n_1\ell$
Bit-length $\sigma = 128$	1.1	18.2	288.2	3.5	20.4	322.7	$4.8n_2\kappa(\lceil\frac{\sigma-\lfloor\log_2(2.4n_2)\rfloor}{8}\rceil)+2n_1\ell$

IV. Comparisons of PSI protocols

Comparisons of protocols by different authors are usually considered as a very complicated work, given the difference of hardware environment, programming language and optimization technics. Pinkas etc. completed a wide comparison between different PSI protocols in [19] for the first time and they did a more comprehensive comparison in [20]. This section will mainly discuss their results from [20] [21].

A. Environment Settings

In [20], Pinkas etc. used two benchmark settings: a LAN setting and a WAN setting. The LAN setting consists of two desktop PCs (Intel Haswell i7-4770K with 3.5 GHz and 16GB RAM) connected by Gigabit LAN. The WAN setting consists of two Amazon EC2 m3. medium instances (Intel Xeon E5-2670 CPU with 2.6 GHz and 3.75 GB RAM) located in the US east coast (North Virginia) and Europe (Frankfurt) with an average bandwidth of 50 MB/s and average latency (round-trip time) of 96 ms.

In [21], Kolesnikov etc. implemented their protocol on a server with Intel(R) Xeon(R) CPU E5-2699 v3 2.30GHz CPU and 256 GB RAM. They run both clients on the same machine but simulate a LAN and WAN connection using the Linux tc command. In the WAN setting, the average network bandwidth, and the average (round-trip) latency are set to be 50 MB/s and 96 ms, respectively. In the LAN setting, the network has 0.2ms latency. All their experiments use a single thread for each party.

The hardware used by the authors of [21] and [20] do not have comparable resource, and thus our analysis will mostly base on their conclusion and tries to find out the features of different types of PSI protocols.

B. Benchmark Results

1) Naïve PSI.

Naïve PSI using hashing has both the best runtime and best communication cost. It has linear time consumption, but it is insecure and thus not suitable for most cases where the participants need high security guarantees. However, when the security needs are not enforced or the parties have very limited computation or network resource, Naïve PSI is still a good choice.

2) Server-Aided PSI.

Server-Aided PSI is only a small factor slower than the Naïve PSI because it is built on hashing or pseudorandom function which are not very costly operations. With the introducing of the third party, it is somehow more secure than the Naïve PSI, but its communication cost is also higher considering one more participant in the protocol. This should be very practical if a trusted third party exists.

Table VI. Running time in ms for PSI protocols with n elements per party
(Protocols with (*) are in a different security model.)

Setting	Protocol	Bit length ℓ	set size n 2^8	2^{12}	2^{16}	2^{20}	2^{24}
LAN	(insecure) naïve hashing	{32, 64, 128}	1	6	75	759	13,529
	PSSZ	32	306	380	770	**4,438**	**42,221**
		64	306	442	1,236	10,501	137,383
		128	307	443	1,352	13,814	213,597
	bOPRF-PSI	{32, 64, 128}	**197**	**227**	**484**	4,555	71,238
WAN	(insecure) naïve hashing	{32, 64, 128}	97	101	180	1,422	22,990
	PSSZ	32	609	701	1,425	**8,222**	**81,234**
		64	624	742	2,142	18,398	248,919
		128	624	746	2,198	23,546	381,913
	bOPRF-PSI	{32, 64, 128}	**471**	**516**	**1,055**	8,866	121,072

Table VII. Communication in MB for PSI protocols with n elements per party. Communication costs for PSSZ and for our protocol ignore the fixed cost of base OTs for OT extension

Protocol	Bit length ℓ	set size n 2^8	2^{12}	2^{16}	2^{20}	2^{24}	Asymptotic [bit]
naïve hashing	{32, 64, 128}	0.01	0.03	0.59	10.49	184.55	$n(\sigma + 2\log n)$
PSSZ	32	0.07	0.80	9.62	149.74	1,650.88	$2\kappa(1.2n+s)(\lceil\frac{\ell-\lceil\log(1.2n)\rceil}{8}\rceil)+(3+s)nv$
	64	0.10	1.43	19.69	310.80	4,227.86	$2\kappa(1.2n+s)(\lceil\frac{\min(v,\ell)-\log(n)}{8}\rceil)+(3+s)nv$
	128	0.18	2.69	39.82	632.92	9,381.82	$2\kappa(1.2n+s)(\lceil\frac{\ell-\log(n)}{8}\rceil)+(3+s)nv$
bOPRF-PSI	{32, 64, 128}	0.05	0.60	8.91	139.67	2,134.06	$k(1.2n+s)+(3+s)nv$

Table VIII. Communication in MB for PSI protocols with n elements per party. Communication costs for PSSZ and for our protocol ignore the fixed cost of base OTs for OT extension

Protocol	Bit length ℓ	set size n 2^8	2^{12}	2^{16}	2^{20}	2^{24}	Asymptotic [bit]
naïve hashing	{32, 64, 128}	0.01	0.03	0.59	10.49	184.55	$n(\sigma + 2\log n)$
PSSZ	32	0.07	0.80	9.62	149.74	1,650.88	$2\kappa(1.2n+s)(\lceil\frac{\ell-\lceil\log(1.2n)\rceil}{8}\rceil)+(3+s)nv$
	64	0.10	1.43	19.69	310.80	4,227.86	$2\kappa(1.2n+s)(\lceil\frac{\min(v,\ell)-\log(n)}{8}\rceil)+(3+s)nv$
	128	0.18	2.69	39.82	632.92	9,381.82	$2\kappa(1.2n+s)(\lceil\frac{\ell-\log(n)}{8}\rceil)+(3+s)nv$
bOPRF-PSI	{32, 64, 128}	0.05	0.60	8.91	139.67	2,134.06	$k(1.2n+s)+(3+s)nv$

3) Public-Key-Based PSI

The Diffie Hellman-based protocol, which was the first PSI protocol, has the best performance with respect to communication cost. However, its computational complexity is very high due to the expensive public key computations. Therefore, it is suitable for settings with distant parties which have strong computation capabilities but limited network resource, such as big data centers. Also, another advantage of public-key-based PSI protocols is that they are memory-friendly. Pinkas etc. in [19] noted that some PSI protocols like circuit-based PSI and OT-based PSI usually involves huge number of public encryption operations, which would quickly consume all memory and abort the protocol. Although a very naïve solution is to store part of the data on disk or swap out some nonactive data, but for OT-based PSI, all masks are active data and random access is very important for search and match progress, storing data on disk would greatly slow down the protocol. Therefore, public-key-based PSI is also a better choice than circuit-based PSI and OT-based PSI when the participants does not have lots of memory resource. Public-key-based PSI protocol usually can be easily paralleled because public encryption operations can be conducted on serval elements at the same time.

4) Circuit-Based PSI

Generic circuit-based protocols do not perform as good as the newer, OT-based constructions. However, given the extensibility of the generic protocol, they are more flexible and can easily be adapted for computing variants of the set intersection functionality, for example, finding the cardinality of the intersection while keeping the elements in intersection secret. Moreover, any optimization to the generic protocols can be applied to the circuit-based PSI protocol as well, so we might expect to see that current circuit-based PSI become more practical in the future. The result of this experiment also affirms the claim of that circuit-based PSI protocols are faster than the blind-RSA-based PSI. Nevertheless, because of the structure of circuit protocols, the computations are usually massive on one side and light on the other side, so this kind of PSI protocol might best fit in the unbalanced setting.

5) OT-Based PSI

If not considering the Naïve PSI and server-aided PSI, OT-based protocol has the best performance for large data set in LAN setting and is the only protocol that maintains its performance advantage in WAN setting and even has better performance than the public-key-based PSI protocols for a mobile network setting.

In [18], the authors uses GBF which stores strings as shares of the element from the set and thus need more space than the bit map of BF. The whole GBF needs to be stored in the memory so that all data can be access randomly, this requires both the parties to have enough memory to store the GBF. For the works of [21], the first memory demands comes from the cuckoo hash table, but considering that cuckoo hashing is a very compact hashing scheme so that the hash table has very high occupancy rate and that the use of phasing to shorten the messages that need to be stored, the OT-based PSI protocols are actually more space-efficient on storing data structures. The second memory demand is that after receiving the masks from the sender, the receiver must search the whole mask set to decide whether an element is in the intersection or not. This requires that the receiver can random access all the masks, so the mask set might be better put on the memory. It is worth noticing that the authors of [21] proposed an optimization which somehow amortized this memory needs. Besides the element itself, the receiver now also needs to record which hash function put an element into the bin, and

this information, the index of the hash function, is also used as a parameter to generate the masks and on the sender side, the masks are put into three different mask sets, and then each set is randomly permuted and sent to the receiver. The receiver needs to check only one set of masks based on the hash function it used for the element. Moreover, the OT extension can be easily paralleled and be precomputed using the ROT technics. However, both the process of building GBF and cuckoo hash table cannot be paralleled, and thus might become the bottle neck of computation.

In comparison to the original OT-based PSI protocol of PSZ, Pinkas etc. claimed that, using the Feistel-like structure, the permutation technics and the three-way cuckoo hashing, their improved Phasing PSI protocol can reduce the communication for all combinations of elements and bit-lengths by factor 2.5 – 4, and thus is much more efficient than the PSZ protocol of 2014. They also stated that their optimizations have more obvious effect when doing PSI on data with more bits. The experiment showed that their protocol even has a lower communication for $\sigma = 128$ than the PSZ PSI protocol has for $\sigma = 32$.

Nevertheless, Kolesnikov etc. showed that their bOPRF-PSI protocol outperforms PSSZ in almost all the case studies, especially for the long bit length of input and large values of the input size, because their BaRK-OPRF make the number of OTs irrelevant to the size of element. In more details, considering the results in the LAN setting, for the input size of 220, their approach can have 2.3 times and 3 times improvement on the performance comparing with PSSZ for the bit lengths of 64 bits and 128 bits, respectively. Also, it is worth mentioning that it takes about 1 minute to compute the intersection for the sets of size n = 224. Similar observations can be inferred for the WAN setting. However, they also noticed that for smaller bit lengths, the PSSZ protocol can be faster than our PSI protocol. This is the case, for example, when the bit length is 32 bits and n = 220 in LAN setting.

Although bOPRF-PSI protocol is not always a better choice than the PSSZ protocol, it should be relatively straightforward to implement a hybrid program that switch between those two subprotocols based on the data size, given that the two protocols are very similar, differing only in the choice of OPRF subprotocol. Similar results can be observed for the communication cost, their bOPRF-PSI protocol has a 4.4–5.0× improvement on the communication cost with respect to PSSZ protocol for PSI of 128- bit strings and sufficiently large sets. Another example is that, for the input size of 220, their protocol can improve 4.5 times the performance of PSSZ for the bit lengths 128 bits.

REFERENCES

[1] G. Ateniese, E. D. Cristofaro and G. Tsudik, "(If) Size Matters: Size-Hiding Private Set Intersection," 2010.

[2] S. Nagaraja, P. Mittal, C.-Y. Hon, M. Caesa and N. Borisov, "BotGrep: Finding P2P Bots with Structured Graph Analysis," 2010.

[3] C. C. Aggarwal and P. S. Yu, "Privacy-Preserving Data Mining: Models and Algorithms," 2008.

[4] G. Mezzour, A. Perrig, V. Gligor and P. Papadimitratos, "Privacy-Preserving Relationship Path Discovery in Social Networks," 2009.

[5] E. Bursztein, M. Hamburg, J. Lagarenne and D. Boneh, "OpenConflict: Preventing Real Time Map Hacks in Online Games," 2011.

[6] M. Rabin, "How to Exchange Secrets by Oblivious Transfer," 1981.

[7] M. Naor and B. Pinkas, "Efficient Oblivious Transfer Protocols," 2001.

[8] "Correlated pseudorandomness and the complexity of private computations," 2001.

[9] Y. Ishai, J. Kilian, K. Nissim and E. Petrank, "Extending Oblivious Transfers Efficiently," 2003.

[10] V. Kolesnikov and R. Kumaresan, "Improved OT Extension for Transferring Short Secrets," 2013.

[11] R. Pagh and F. Rodler, "Cuckoo Hashing," 2001.

[12] A. Kirsch, M. Mitzenmacher and U. Wieder, "More Robust Hashing: Cuckoo Hashing with a Stash," 2009.

[13] B. Pinkas, T. Schneider and M. Zohner, "Faster private set intersection based on OT extension," 2014.

[14] M. N. D. Cristofaro, A. Dmitrienko, N. Asokan and A.-R. Sadeghi, "Do I know you?: efficient and privacy-preserving common friend-finder protocols and applications," 2013.

[15] C. Meadows, "A More Efficient Cryptographic Matchmaking Protocol for Use in the Absence of a Continuously Available Third Party," 1986.

[16] E. Cristofaro and G. Tsudik, "Practical Private Set Intersection Protocols with Linear Complexity," 2010.

[17] Yan Huang, David Evans and J. Katz, "Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?," 2012.

[18] C. Dong, L. Chen and Z. Wen, "When private set intersection meets big data: an efficient and scalable protocol".

[19] B. Pinkas, T. Schneider and M. Zohner, "Faster Private Set Intersection Based on OT Extension," 2014.

[20] B. Pinkas, T. Schneider, G. Segev and M. Zohner, "Phasing: Private Set Intersection using Permutation-based Hashin," 2015.

[21] V. Kolesnikov, Kumaresan, Ranjit, Rosulek, Mike, Trieu and Ni, "Efficient Batched Oblivious PRF with Applications to Private Set Intersection".

[22] C. Hazay and Y. Lindell, "Constructions of Truly Practical Secure Protocols using Standard Smartcards," 2009.