Mathematical Biosciences and Engineering

http://www.aimspress.com/journal/MBE

MBE, 19(2): 1861–1876.
DOI: 10.3934/mbe.2022087
Received: 08 October 2021
Accepted: 14 December 2021
Published: 20 December 2021

Research article

SE-PSI: Fog/Cloud server-aided enhanced secure and effective private set intersection on scalable datasets with Bloom Filter

Shuo Qiu, Zheng Zhang*, Yanan Liu, Hao Yan and Yuan Cheng

School of Software Engineering, Jinling Institute of Technology, Nanjing 211169, China

* **Correspondence:** Email: zhangzheng@jit.edu.cn.

Abstract: Private Set Intersection (PSI), which is a hot topic in recent years, has been extensively utilized in credit evaluation, medical system and so on. However, with the development of big data era, the existing traditional PSI cannot meet the application requirements in terms of performance and scalability. In this work, we proposed two secure and effective PSI (SE-PSI) protocols on scalable datasets by leveraging deterministic encryption and Bloom Filter. Specially, our first protocol focuses on high efficiency and is secure under a semi-honest server, while the second protocol achieves security on an economic-driven malicious server and hides the set/intersection size to the server. With experimental evaluation, our two protocols need only around 15 and 24 seconds respectively over one million-element datasets. Moreover, as a novelty, a *multi-round* mechanism is proposed for the two protocols to improve the efficiency. The implementation demonstrates that our *two-round* mechanism can enhance efficiency by almost twice than two basic protocols.

Keywords: private set intersection; Bloom Filter; multi-round; scalability

1. Introduction

Private set intersection (PSI) is a cryptographic protocol that allows two parties to jointly calculate set intersection over their datasets without leaking any additional information. Formally, a PSI protocol is described as follows: Two clients, Alice and Bob, have different (or same) size of the datasets, where Alice has a set $X = \{x_1, x_2, \cdots, x_m\}$ and Bob has a set $Y = \{y_1, y_2, \cdots, y_n\}$. At the conclusion of the protocol, if the protocol is symmetric, it outputs $(X \cap Y, X \cap Y) \leftarrow \text{PSI}(X, Y)$ to Alice and Bob simultaneously; If the protocol is asymmetric, it outputs $(X \cap Y, \Lambda) \leftarrow \text{PSI}(X, Y)$ to one client, where Λ denotes the empty string.

PSI has been applied in many scenarios. For instance, the department of homeland security (DHS), who maintains a terror watch list (or 'do-not-fly' list), wants to check whether its terrorist suspects are in the list of passengers from a flight. Obviously, DHS is not allowed to obtain the private information

of the innocent passengers in the list. As another classic example, two real estate companies would like to identity homeowners who have signed double contracts with both companies for profit purpose, and both two are not willing to reveal any extra information of other homeowners. Furthermore, PSI also has been utilized extensively in a wide range of emerging cloud computing paradigm setting such as internet-based Personal Healthy Records (PHRs) systems [1], mobile social networks [2], human genomic research [3], and online advertising [4].

With the rapid development of big data, traditional PSI protocols (e.g., [5–7]) are never well applicable for large-scale dataset. How to design an efficient and scalable privacy-preserving PSI solution is an open problem. Dong et al. [6] combined Garbled Bloom filter to propose a two-party PSI protocol for million-element datasets, and hereafter several works focused on enhancing the efficiency of PSI. Especially, Pinkas et al. [8] improved the efficiency with Oblivious Transfer extension technique, and then proposed another new circuit-based protocols for computing variants of the intersection with an almost linear number of comparisons based on new variants of Cuckoo hashing in two dimensions. And recently, Chase et al. [9] presented an efficient PSI protocol with lightweight Oblivious Pseudo-Random Functions in a two-party setting model. However, such two-party model introduces multiple interactions and cannot be scalable to billion-scale datasets.

Kamara et al. [10] introduced an efficient PSI protocol for billion-element datasets using Pseudo-random Permutation under a *server-aided* setting by parallel mode and hardware acceleration. Later, Zhang et al. [11] proposed a two-server-aided PSI protocol with multiple keys with scarifying a little efficiency compared with [10]. There are also amount of works focusing on server-aided privacy preserving outsourced computations (e.g. [12–15]), and especially [16–19] realize outsourced private set intersection with combing functional encryption algorithm. Regrettably, these solutions require either multiple interactions or very complex operations over the outsourced ciphertexts. Moreover, in the hidden size model, the clients have to undertake most of set intersection calculation. All of these obstacles make them unscalable to a very large datasets (such as billion set size).

Our Contributions. Motivated by previous PSI works on massive datasets, we exploit the Bloom Filter technique and lightweight deterministic encryption algorithm to design an efficient fog/cloud server-aided PSI protocol in this paper. Our Protocol can output an approximate intersection due to the false positive in Bloom Filter, while this approximate result is guaranteed to be almost the same with the actual intersection through setting the suitable parameter of Bloom Filter. The main contributions can be summarized as below:

- We present the first basic protocol with extremely high efficiency and it is secure against a semi-honest server. Our first protocol is also regarded as a basic building block for the second protocol.
- For more security, we propose the second protocol with secure against a lazy or cheating malicious server. Specifically, by adding dummy sets, our second protocol not only can preserve the set/intersection size from being leaked to the server, and also can check whether the server honestly returns the intersection result with a quite high probability.
- Experimental evaluation shows that our two basic protocols need only around 15 and 24 seconds respectively over one million-element datasets. Furthermore, we propose a novel *multi-round* mechanism to improve the efficiency and accuracy. Through the implementation, a *two-round* mechanism can enhance efficiency by almost twice than two basic protocols in average with maintaining certain accuracy.

2. Related work

Private Set Intersection (PSI) proposed by Freedman et al. [20] is a representative cryptographic computation protocol between two parties. Subsequent works focus on the study of PSI protocols with more efficiency and security. We summarize them with two modes as follows: *two-party* PSI mode and *server-aided* PSI mode.

Two-party PSI Mode. Freedman et al. [20] proposed two secure PSI protocols with Oblivious Polynomial Evaluation, and lots of subsequent works focused on improving the computation efficiency or obtaining more security (e.g., [21, 22]). There exist several solutions using Pseudo-Random Functions (PRF) (e.g., [2, 9, 23]), which are instantiated effectively since with symmetric-key technique. What's more, several other works combining different techniques focused on realizing high efficiency and security over a large-scale dataset (e.g., Bloom Filter [6],cut-and-choose [24], Circuits-based techniques [25, 26], and Oblivious Transfer (OT) [8, 27–29]). Unfortunately, these *two-party* setting solutions are seemingly infeasible with low capability clients since multiple interactions would incur an extreme computation cost. So a large body of works under *server-aided* setting for PSI were introduced (e.g., [7, 10, 11, 16–19, 30, 31]).

Server-aided PSI Mode. In the *Server-aided* mode, two parties with weak computation capability in the protocols can outsource their storage and intersection computation to an aided server, and the server returns the intersection results without learning any additional private information of each parties. A branch of works (e.g., [7, 31, 32]) used homomorphic encryption to outsource the encrypted datasets and delegate the intersection operations to the server. Woefully, the lower efficiency of homomorphic encryption prevents these schemes from performing on a large scale dataset (e.g., billion set size). In order to get a higher efficiency, [10, 16, 18] leveraged Pseudo-Random Functions with no public-key operations. Moreover, an effective data structure, Bloom Filter, is also utilized to do set operations in variety of work [16, 19, 33, 34]. Unfortunately, as illustrated in Section 1, multiple interactions with complex operations make these solutions infeasible for a very large-scale set. [30] proposed a fine-grained access control private set intersection (PSI) scheme with non-interaction, while due to the inefficiency of the attribute-based encryption, this scheme is unable to scale to the PSI computation over billion size sets. In terms of efficiency, [10] is a quite feasible solution, but it cannot minimize privacy disclosure because the set/intersection size will be leaked to the server. In this paper, we focus on designing an efficient and secure protocol via leveraging Bloom Filter and lightweight symmetric deterministic encryption, which can also preserve the set/intersection size from leaking to the aided server.

3. Definitions

3.1. System definition

Three entities are consisted in our server-aided PSI system model: Alice, Bob, and an aided server (the fog/cloud server), where two clients Alice and Bob can engage in a computation of set intersection by the assistance of an un-fully trusted server without leaking any extra private information. Specifically, each client, Alice and Bob, holds a secret set respectively, denoted as S_A and S_B. Before conducting the set intersection, two clients encrypt their original sets respectively and outsource them

to the aided server. Then, the server is delegated to compute the intersection over two encrypted sets of S_A and S_B without gaining any more private information, and finally outputs the private set intersection of $S_A \cap S_B$ to Alice and Bob. For that, we aim to build a more efficient solution with leaking minimal privacy to the server and saving more computation cost of two clients, and we formalize our security model as follows.

3.2. Security definition

Similar to the related works (e.g., [6, 10]), we also consider two different models of adversaries, including the *semi-honest* and *malicious* adversaries, and only one of the parties can be corrupted with an adversary at the same time. Specifically, a semi-honest adversary can execute the protocol with the specified steps honestly, but it curiously records all intermediate computations to derive other parties' private data; a malicious client would provide a false input to infer extra privacy of the other client, and a malicious server would arbitrarily deviate from the specified executions in the protocol or temper with the results of computation, such as deleting or modifying elements of the computed intersection or even randomly outputting a false result without any calculations.

In addition, we suppose that no collusion is occurred in any two parties in our protocols under the above two types of adversary model and this non-collusion setting is referred in most previous protocols (e.g., [10, 35]).

Ideal vs. Real. We introduce the general *ideal vs. real* model like [36] to illustrate the privacy of our protocols in this paper, which is a common security model of secure multiple computation.

Generally speaking, we assume that there is a fully Trusted Third Party (TTP) and a probabilistic polynomial-time independent simulator $\mathcal{S}$ in the ideal world, they evaluate two parties' set intersection through simulating π. $\mathcal{S}$ outputs $\mathbf{Ideal}_{\mathcal{S}}^{\pi}$ as its view. While in a real world, the valid participants and $\mathcal{A}$ execute π without TTP. In the end of the execution, $\mathcal{A}$ outputs $\mathbf{Real}_{\mathcal{A}}^{\pi}$ as its view. The protocol π is secure only if the views for any adversary in the real and ideal world are computationally indistinguishable.

Assuming there is no collusion occurred in our executions, for the security parameter λ, all feasible inputs $\bar{x}$, randomness $\bar{z}$, and each $i \in \{1, 2, 3\}$

$$\left\{\mathbf{Ideal}_{\mathcal{S},\bar{z}}^{\pi,(i)}(\lambda;\bar{x})\right\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \left\{\mathbf{Real}_{\mathcal{A},\bar{z}}^{\pi,(i)}(\lambda;\bar{x})\right\}_{\lambda\in\mathbb{N}},$$

where " $\overset{c}{\approx}$ " indicates computational indistinguishability [36].

Table 1. Notations.

Notation	Description
λ, sk	λ represents security parameter and sk is the secret key for encryption
n, m	n is number of elements in the set and m is length of Bloom Filter
k	k is the number of hash functions in Bloom Filter
p	p is false positive probability generated from Bloom Filter
π	π is a pseudo-random permutation
DE	DE is lightweight symmetric deterministic encryption algorithm
BF_S	BF_S represents the Bloom Filter of set S
S_A, S_B	S_A and S_B represent the datasets of two clients *Alice* and *Bob* respectively

4. Preliminaries

To achieve a better efficiency, we adopt an effective data structure in this paper, namely Bloom Filter. In addition, we use a lightweight symmetric deterministic encryption to encrypt the original datasets, and to preserve the position privacy, a pseudo-random permutation is also introduced in our protocols. Moreover, for better readability, we first describes some main mathematical notations referred to in this paper in Table 1.

4.1. Bloom Filter

Bloom Filter, a simple space-efficient randomized data structure, supports fast membership testing in a set. A Bloom Filter of set S is an array with certain length m, denoted as BF_S, which is generated as in Figure 1.

- Init(H, m, k): It initializes
$$BF_S[0] = 0, \cdots, BF_S[m-1] = 0,$$
and chooses k hashes independently
$$h_0, h_1, \cdots, h_{k-1} : \{0,1\}^* \to \{0, 1, \cdots, m-1\}.$$
- CreatBF(S): For inserting a value $s \in S$ into BF_S, it computes k hash values for each h_i $(0 \le i < k)$ and then sets
$$BF_S[h_0(s)] = 1, \cdots, BF_S[h_{k-1}(s)] = 1.$$
If $BF_S[j] = 1$ before, no operation is done. The above process is repeated until all elements in S are added into BF_S.
- Query(s, BF_S): To test if the element $s \in S$, s is hashed with k hash functions h_i $(0 \le i < k)$ respectively, and if
$$BF_S[h_0(s)] = 0 \vee \cdots \vee BF_S[h_{k-1}(s)] = 0,$$
s is not in S. Else $s \in S$.

Figure 1. Generation of Bloom Filter for a set.

Note that, it sometimes occurs false positive in Bloom Filter, meaning that it is possible that s is not in S, while all $BF_S[h_i(s)] = 1$ $(0 \le i < k)$. The probability of false positive p can be computed as:

$$p = (1 - (1 - \frac{1}{m})^{kn})^k \approx (1 - e^{-kn/m})^k, \tag{4.1}$$

According to [6], given a particular false positive probability p, Bloom Filter length is at least $m \ge -n \log_2 e \cdot \log_2 p$. It is obviously that the probability of false positive and Bloom Filter length are negative correlation. Therefore, to maintain a better accuracy and efficiency, we set $m = -n \log_2 e \cdot \log_2 p$ in this paper.

4.2. Deterministic encryption

In a symmetric deterministic encryption (DE) cryptosystem [37], a same original plaintext would be encrypted to a same ciphertext [37], and thus it can be used to do equality checking or membership

testing over encrypted data [10]. We generally state $DE = \{KeyGen, Enc, Dec\}$ as follows,

- $sk \leftarrow KeyGen(1^\lambda)$: for security parameter λ, return secret key $sk \in \mathcal{K}$.
- $c \leftarrow Enc(sk, d)$: for secret key sk and plaintext $d \in \mathcal{M}$, return ciphertext c.
- $d \leftarrow Dec(sk, c)$: for secret key sk and ciphertext $c \in \mathcal{C}$, return plaintext d.

Here, *KeyGen* is a probabilistic algorithm while *Enc* and *Dec* are deterministic for equality checking in our protocols.

4.3. Pseudo-random permutation

Suppose that F is an efficient pseudo-random permutation, then for all probably distinguishers D and a random β-bit permutation functions f_β, there exists a negligible $\varepsilon(\cdot)$ such that:

$$| Pr[D^{F_\alpha(\cdot), F_\alpha^{-1}(\cdot)}(1^\beta) = 1] - Pr[D^{f_\beta(\cdot), f_\beta^{-1}(\cdot)}(1^\beta) = 1] | \leq \varepsilon(\beta),$$

where $\alpha \xleftarrow{R} \{0,1\}^\beta$.

Especially, to preserve the element's positions in the Bloom Filter, we introduce $\pi : [|S|] \rightarrow [|S|]$ as a pseudo-random permutation in this paper. That is to say, $\pi(S)$ permutes the real positions in Bloom Filter, which is denoted as

$$\pi(S) = \{s_{\pi(i)} | s_i \in S\}.$$

5. Our protocols

Here, two protocols are proposed to compute the private set intersection with the assistance of an aided server. More specifically, each client firstly generates a Bloom Filter BF_j ($j \in \{A, B\}$) of the corresponding S_j, and then encrypts BF_j with DE and sends it to the server. Finally, the aided server performs equality test on the encrypted Bloom Filters. Since there exist only two different values (i.e, 0 or 1) in the Bloom Filters, to keep the computation correct, we encrypt $BF[i]$ with different cases,

$$C[i] = \begin{cases} \mathsf{Enc}(sk, i||BF[i]), & if\ BF[i] = 1, \\ \mathsf{Enc}(sk, r_i||BF[i]), & otherwise, \end{cases} \tag{5.1}$$

where r_i ($i \in \{0, 1, \cdots, m-1\}$) is randomly chosen from message domain $\mathcal{M}$, and "||" denotes concatenation. Then we can obtain the ciphertext of Bloom Filter as $C = \{C[1], C[2], \cdots, C[m-1]\}$ based on Eq (5.1).

5.1. Protocol I under semi-honest model

In Protocol I, the server is honest-but-curious about inferring the private information of other parties with a record of the intermediate computations. More details are described in Figure 2.

Then we analyze the *correctness* and *security* of our Protocol I. The correctness is obvious, for $\forall i,\ i \in \{0, 1, \cdots, k-1\},\ C_A[h_i(a)] = C_B[h_i(b)],\ a \in S_A,\ b \in S_B$, since the encryption algorithm DE is deterministic, we have $BF_A[h_i(a)] = BF_B[h_i(b)]$, and then $h_i(a) = h_i(b)$, that is $a = b$.

In case of security, our Protocol I is secure under a semi-honest server or a malicious client without distinguishing the views of ideal and real world. Then we formalize it as follows:

Initialization: Given $\mathsf{DE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, and `Alice` and `Bob` have their inputs sets S_A and S_B respectively.

1. `Alice` (`Bob`) initializes a blank bloom filter with $\mathsf{Init}(H, m, k)$ as shown in Fig 1 and runs

$$BF_A \leftarrow \mathsf{CreatBF}(S_A), \qquad BF_B \leftarrow \mathsf{CreatBF}(S_B),$$

   to build m-bits BF_A for S_A and BF_B for S_B respectively. Here the hash functions $H : h_0, h_1, \cdots, h_{k-1}$ and the parameters m, k for creating bloom filters are public.
2. `Alice` (`Bob`) encrypts BF_A and BF_B as

$$C_A \leftarrow \mathsf{DE.Enc}(sk, BF_A), C_B \leftarrow \mathsf{DE.Enc}(sk, BF_B),$$

   based on Eq (5.1). Then, each client permutes C_A and C_B with $T_A = \pi(C_A)$ and $T_B = \pi(C_B)$, and finally sends T_A and T_B to `Server`.
3. Then `Server` conducts, for $\forall i \in \{0, 1, \cdots, m-1\}$,

$$T_A[i] \stackrel{?}{=} T_B[i],$$

   then outputs the indexes $I = \{i | T_A[i] = T_B[i]\}$ to `Alice` and `Bob` respectively.
4. `Alice` (`Bob`) calculates the intersection for Bloom Filter

$$BF_{S_A \cap S_B} = \{BF_{S_A \cap S_B}[i] = 1 | i \in \pi^{-1}(I)\}.$$

   `Alice` and `Bob` both get $S_A \cap S_B$ by running $\mathsf{Query}(S_j, BF_{S_A \cap S_B})$ for $j \in \{A, B\}$.

Figure 2. Details for protocol I under semi-honest model.

Theorem 1. *Since the pseudo-randomness for permutation function π and DE, our Protocol I is secure if: 1) with a semi-honest server and two honest clients, no extra information is revealed to the server; 2) with a honest server and one of malicious clients, no extra private information is leaked to the other client.*

Proof. Recall that, in order to prove security against an adversary $\mathcal{A}$ corrupting the semi-honest server and attempting to get more private information of the input sets, we need to construct a simulator S in the ideal-world to simulate the server's behavior in the real-world executions with the honest Alice and Bob. Our Protocol I is secure if the joint distribution of the view generated by $\mathcal{A}$ in the real-world is distinguishable from the view of S in the ideal-world executions.

It is clear to see that the only information that the simulator S obtains in the ideal-world executions is the encrypted Bloom filters $C_A = \mathsf{DE.Enc}(sk, BF_A)$ and $C_B = \mathsf{DE.Enc}(sk, BF_B)$. Then, S computes $C_A \cap C_B$ and finally returns the computed result to the adversary $\mathcal{A}$. With the pseudo-randomness of DE encryption, $\mathcal{A}$ cannot discriminate these ciphertexts from the real and ideal world. The only difference between the view of S and the adversary $\mathcal{A}$ is the length of the returned intersection result. However, this length is randomly distributed in $[0, m]$ (m is bloom filter length.), which guarantees that $\mathcal{A}$ cannot distinguish it from the real world and the ideal world. Additionally, due to the private permutation function π, $\mathcal{A}$ is unable to guess the real bit positions of the intersection in the Bloom filter. Therefore, there is no way to obtain any elements in the intersection via brute-force attack based

on the intersection Bloom filter $C_A \cap C_B$.

Note that, from the aspect of the clients, the only information obtained by both two is the last intersection which is exactly what they should know. Therefore, each client obtains no extra private information of the other one in the above protocol.

5.2. Protocol II under malicious model

Our Protocol I shown in above is proven to be secure under a semi-honest model, where the server always runs the protocol and outputs the result honestly. Unfortunately, such assumption can not satisfy the actual requirements since the server sometimes could perform cheating in practice to make more profits or save computational expands with economically-driven nature, such as returning a random results with no computation or an incorrect intersection after computation to both clients without being detected.

Moreover, the server somewhat may conjecture the intersection size via the intersection Bloom Filter in our Protocol I, which is private information in some situations. To solve this problem, we present our second protocol with adding a checking mechanism over our Protocol I. Concretely, given the original sets S_A and S_B, Alice negotiates S_0 to S_2 with Bob privately, and they get $D_A = S_A \cup S_0 \cup S_1$ and $D_B = S_B \cup S_0 \cup S_2$ respectively. Then, Alice, Bob and the server compute the dummied intersection of D_A and D_B by running Protocol I, and finally Alice and Bob gain the original $S_A \cap S_B$ via deducting S_0 from dummied intersection returned by the server. Here, with a short shared random seed, we can leverage a pseudorandom generator to effectively generate three disjoint dummy sets mentioned above, which can not only reduce time overhead compared to directly agreeing on these dummy sets and also maintain the randomness of them. Our Protocol II is illustrated as shown in Figure 3.

Initialization: Given $\mathsf{DE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, and `Alice` and `Bob` have their inputs sets S_A and S_B respectively, and with out of generality we assume that $|S_A| = |S_B| = n$ and $S_A, S_B \subseteq \mathbb{D} \subseteq \mathbb{E}$, and $\mathbb{E}$ is data space. The secret key sk used for encryption is generated via $\mathsf{DE.KeyGen}(1^\lambda)$ and secretly shared with `Alice` and `Bob`. π is beforehand shared between two clients as a secret random permutation.

1. `Alice` randomly selects $S_0, S_1, S_2 \subseteq \mathbb{D}' \subseteq \mathbb{E}$ as dummy sets, and secretly shares them with `Bob`, where $|S_0| = |S_1| = |S_2| = t$ (For simplicity, suppose three sets are with same length t.) and $\mathbb{D}' \cap \mathbb{D} = \varnothing$.
2. `Alice` (`Bob`) respectively produces the dummied sets as

$$D_A = S_A \cup S_0 \cup S_1, \qquad D_B = S_B \cup S_0 \cup S_2.$$

3. `Alice`, `Bob` and `Server` firstly run Protocol I to get $DI = D_A \cap D_B$.
4. `Alice` (`Bob`) verifies: if

$$(S_0 \subseteq DI) \wedge (S_i \cap DI = \varnothing), i \in 1, 2,$$

the intersection is $S_A \cap S_B = DI - S_0$; Else, aborts.

Figure 3. Details of Protocol II under malicious model.

The correctness of Protocol II can be easily verified as the Protocol I. Next, we define its security as follows:

Theorem 2. *Our Protocol II is secure under the following cases: 1) with a malicious server deceiving the honest users during the protocol, which can be detected with a very high probability; 2) with a malicious client, no extra private information can be inferred to this malicious client.*

Proof. Firstly, similar to the proof of Theorem 1, suppose that there is a malicious adversary $\mathcal{A}$ corrupting the server to execute the protocol and try to obtain as much as private information of both two clients' inputs. The view of $\mathcal{A}$ in the execution includes the ciphertexts of the bloom filters for two dummied sets D_A and D_B, and the approximate size of $D_A \cap D_B$. As the pseudo-randomness of DE, $\mathcal{A}$ cannot get any extra privacy belonging to the input sets. With the computed intersection result of two dummied encrypted Bloom filters, the adversary would not infer any private information of the real intersection $S_A \cap S_B$. Specifically, $\mathcal{A}$ can approximate the size of $D_A \cap D_B$ from its view in the executions. However, the server is unable to obtain the size of $S_A \cap S_B$ unless it can correctly guess the size of dummy set S_0. Unfortunately, the size of S_0 is randomly chosen from a large domain, and thus the adversary has only a negligible probability to correctly guess it in polynomial time. Therefore, our Protocol II can successfully preserve the intersection size from the malicious server.

Another advantage of Protocol II is that it can prevent a cheating server from removing or adding element into the computed intersection or a lazy server from returning a random result without computation. Specifically, by adding the dummy sets, the intersection result $DI = D_A \cap D_B$ is impossible to be empty due to the existence of S_0 and also impossible to be an entire set of one of the sets D_A or D_B since neither S_1 nor S_2 should be contained in the intersection, so the server is not able to arbitrarily return an empty intersection or a whole set from D_A and D_B without detection. Furthermore, it also can guarantee that the server cannot remove or add some elements into the intersection, because it needs to make sure that the intersection set includes all the elements in S_0 while all the elements in S_1 or S_2 must be not in the intersection. Therefore, we can easily know that the probability of a lazy server randomly returning a result without detection is $(\frac{1}{n+2t})^t \cdot (1 - \frac{1}{n+2t})^{2t}$($n$ is the size of original sets and t is size of dummy sets), which would almost trend to zero with a large n and a suitable t. What's more, for a cheating server, according to [10], the security can be significantly increased against a malicious server by choosing suitable parameters.

For the case of one client being corrupted with a malicious adversary $\mathcal{A}$ (Without loss of generality, suppose Alice is corrupted with $\mathcal{A}$, which is able to provide arbitrary inputs.), $\mathcal{A}$ is unable to learn any extra private information of the other client Bob. Specifically, the most powerful malicious behavior of $\mathcal{A}$ with arbitrary input is that it sets all bits of the Bloom filter to be 1, thus the result returned from the server would contain all the elements provided by Bob, including the dummy set S_2. However, even if Alice recovers all the elements based on the intersection Bloom filter, he/she is still unable to decide which elements should be contained in Bob's private set since S_2 is also included in the recovery set, unless via two times executions of the protocol with the same S_2. While Bob can detect such malicious behavior with checking whether S_2 is contained in the intersection once receiving the intersection Bloom filter, and stops carrying out the private set intersection with Alice if so. Therefore, $\mathcal{A}$ cannot gain extra private information of Bob with the above malicious case.

Note that, the intersection size is not leaked to the server since the size of dummy set S_0 is secret only known to both two clients. That is to say, our Protocol II achieves the intersection size-hiding from the server.

5.3. Enhanced Efficiency with Multi-Round Executions

Since there exits a false positive probability in the Bloom Filter used in our protocols, two clients may recover a different intersection by running $Query(BF_{S_A \cap S_B})$ at the end of two protocols. Based on Eq (4.1), we observe that the accuracy of the intersection is higher with the increasing of Bloom Filter length m, while it would lead to lower efficiency with more computation and storage overhead. To leverage a better efficiency and accuracy, we present a multi-round mechanism to achieve a higher efficiency with a desired accuracy, and our two basic protocols in this paper can be regarded as one-round protocols. Intuitively, we assume that p_i is the false positive probability and m_i is the Bloom Filter length of i-th round, then

$$\begin{cases} m = m_1 + m_2 + \cdots + m_\tau, \\ p = p_1 \cdot p_2 \cdot \cdots \cdot p_\tau, \end{cases} \tag{5.2}$$

where τ denotes total rounds in the multi-round algorithm, and $m_i = -n \log_2 e \cdot \log_2 p_i$ ($i \in \{1, \cdots, \tau\}$). m is the total length of all the Bloom Filters used in the multi-round algorithm and p is the final achieved false positive probability. As shown in Algorithm 1, we introduce that $Optimal(p_i)$ is the most optimal false positive of i-th round, thoroughly guaranteeing a shortest Bloom Filter length m as illustrated in Eq (5.2).

The security of Algorithm 1 follows the security of two basic protocols in this paper since the main step of Algorithm 1 is step 5. With the following experimental implementations, the results show that the multi-round executions significantly enhance the performance of our two basic protocols.

Algorithm 1 Multi-Round-PSI (A, B)

Require: `Alice` and `Bob` input private sets A and B respectively, and for simplify, suppose$|A| = |B| = n$. p is the desired false positive probability in the protocol. Here we initialize $p' = 1$ and $j = 0$.

Ensure: The output of the set intersection: $(A \cap B, A \cap B)$.

1: $j++$;
2: $p_j = Optimal(p_j)$;
3: $m_j = -n \log_2 e \cdot \log_2 p_j$;
4: $p' = p' \cdot p_j$;
5: **call** Protocol I or II stated above and obtain $A_j = \{a \mid a \in A \wedge a \in A \cap B\}$, $B_j = \{b \mid b \in B \wedge b \in A \cap B\}$;
6: **if** $(p' <= p)$ **then**
7: **return** (A_j, B_j);
8: **else**
9: Multi-Round-PSI (A_j, B_j);
10: **end if**

6. Performance evaluation

6.1. Efficiency of Protocol I and II

To evaluate the performance of our protocols, we set the experimental environments as: two clients run on Linux operating system, 12GB RAM and 8 vCPUs; and the server runs on Linux operating

system, 8 vCPUs and 14GB RAM. We use Java language to implement our protocols, and leverage Crypto++ library and AES-ECB mode with 128-bit security to realize our cryptographic operations with Deterministic Encryption. Furthermore, to get more efficiency, we adopt MD5 * to construct Bloom Filters.

Table 2. Average execution for our two basic protocols, where set size is from 10 thousand to 100 million in pipeline and parallel modes. ms is milliseconds and s is seconds. We perform 10 times testing for average value for set sizes in the table.

Set Size	Pipeline Time		Parallel Time	
	Protocol I	Protocol II	Protocol I	Protocol II
10 K	765(ms)	1.19(s)	175(ms)	268(ms)
100 K	7.9(s)	13.1(s)	1.61(s)	2.49(s)
1 M	76(s)	119(s)	13(s)	22.3(s)
10 M	789(s)	1,167(s)	129(s)	218(s)
100 M	7,984(s)	11,108(s)	1,326(s)	2,101(s)

In the following part, we evaluate the performance of our protocols in the pipeline mode and parallel mode as described in [6]. Specifically, all computation on each side is executed in a single thread under the pipeline mode, while in the parallel mode, multiple threads are used to perform the protocols simultaneously on multiple CPUs. The experimental results given below show that the performance can be critically enhanced under the parallel mode with multi-core machines since all dominating computations in both two protocols, including constructing and encrypting the Bloom Filters, and performing equality checking over the encrypted Bloom filter, can be executed simultaneously in parallel.

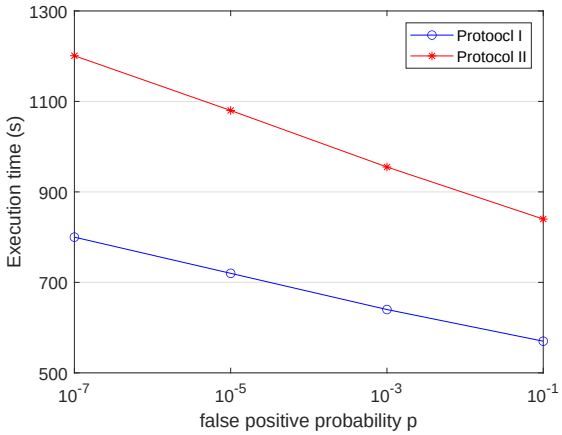

Figure 4. The performance for our protocols with the false positive probability p ranging from 10^{-7} to 10^{-1}, where the size of data set $n = 1.0 \times 10^7$.

Table 2 presents the average execution time of our two basic protocols for the set size varied from 10 thousand to 100 million. To guarantee the correctness and efficiency of our Protocol II, we set the size of the dummy sets to be $n/2$ and a feasible false positive probability to be $1/n$, where n is the

*MD5 guarantees a sound accuracy in most situations, and it is enough in our works and can be replaced to SHA-1 if needed.

size of original set. Experimental results show that our setting of false positive probability $1/n$ may ensure a good accuracy, and the efficiency of both two protocols can be boosted via performing on the parallel mode. For instance, when considering one million set size, our two protocols cost 79 seconds and 123 seconds in the pipeline mode, while in parallel mode, both two can be respectively reduced to 15 seconds and 24.3 seconds in average.

To verify the relation of false positive probability p and performance for our protocols, we set dataset size $n = 1.0 \times 10^7$ and p varying from 10^{-7} to 10^{-1}. As shown in Figure 4, it is clear to see that the execution time linearly decreases with the increase of the false positive probability which indicates a lower accuracy. This is consistent with our theoretical analysis in Subsection 4.1.

6.2. Enhanced efficiency of multi-round algorithm

For simplify, we just verify a two-round mechanism over our two basic protocols since all multi-round (more than two rounds) protocols can be easily realized with a two-round block. Suppose that m_1 and m_2 are the Bloom filters length in this two-round algorithm respectively, and p_1 and p_2 are the corresponding false positive probability. Based on Eq (5.2), we can get

$$\begin{cases} m = m_1 + m_2, \\ p = p_1 \cdot p_2, \end{cases} \tag{6.1}$$

where $m_i = -n \log_2 e \cdot \log_2 p_i, (i \in \{1, 2\})$. Then, with a desirable p in Eq (6.1), the optimal p_1 can be easily computed and can also get the minimum m:

$$m = -n \log_2 e(\log_2 p_1 + (\beta + p_1) \log_2 \frac{p}{p_1}), \tag{6.2}$$

where $\beta = \frac{|S_A \cap S_B|}{n}$, and $p < p_1 < 1$.

Table 3. The optimal value of p_1 for the minimal m in our two-round algorithm, where the size of dataset $n = 1.0 \times 10^7$, $p = 1.0 \times 10^{-7}$, and $\beta = \frac{|S_A \cap S_B|}{n}$ varying from 0.1 to 0.9 with step 0.2.

Intersection ratio β	0.1	0.3	0.5	0.7	0.9
Optimal false positive probability p_1	0.062	0.049	0.036	0.022	0.008

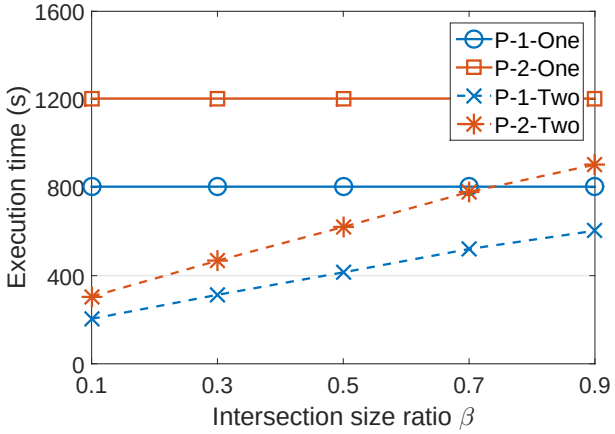

Figure 5. Performance for two basic protocols and two-round protocols over 10 million set size under pipeline mode, where the size of the data set $n = 1.0 \times 10^7$, the false positive probability $p = 1.0 \times 10^{-7}$, and the intersection size ratio β varies from 0.1 to 0.9 with step 0.2. *P*-1-*One* and *P*-2-*One* represent our basic Protocol I in Figure 2 and Protocol II in Figure 3 respectively, which are also regarded as one-round protocols. The basic Protocol I and Protocol II with two-round setting is represented as *P*-1-*Two* and *P*-2-*Two* respectively, which are named as two-round protocols.

Table 3 lists the optimal false positive probability p_1 in the first round computed from Eq (6.2) with β ranging from 0.1 to 0.9 with step 0.2, where $n = 1.0 \times 10^7$ and $p = 1.0 \times 10^{-7}$. Through our implementation, the experimental results are almost consistent with the theoretical results in Table 3. Next, we verify the efficiency of our two-round algorithm with these optimal parameters under the pipeline mode.

We can observe that the two-round protocols execute almost twice more efficient than one-round setting in average in Figure 5. Specifically, given a 10 million set size, our Protocol I under one-round protocol requires at least 13 minutes, while it only needs less than 7 minutes in the two-round setting. Note that, it is clear to see that the execution time in our two-round protocol linearly increases with β, which is the ratio of intersection size to the original set size; the output intersection of the first round in our two-round protocol is the input of the second round execution, and thus the efficiency of the second round is linear to β. However, two-round setting still runs more efficient than one-round setting even though in the worst case $\beta = 0.9$ (Actually, β would not be very high in most applications.), as shown in Figure 5. Furthermore, we can also deeply optimize the performance for two-round setting with a parallel mode, and make our protocols more scalable to large datasets.

Additionally, we compare the efficiency of our enhanced protocols with the server-aided PSI protocols in [10]. According to [10], we can observe that it costs around 7s for the basic protocol SHPSI and 82 s for the enhanced protocol SizePSI over 10 million set size under a parallel mode with 100-threads. For the efficient of our protocols, as stated above in Figure 5, our Protocol I and Protocol II run around 400 and 600 s in average respectively under pipeline mode with one thread. It is obviously that our Protocol II has significant efficiency improvements over SizePSI in [10] after the same acceleration of parallel mode with 100-threads.

7. Conclusions

In this paper, we present two enhanced secure and efficient PSI protocols with assistance of an aided server (a fog/cloud server) by leveraging Bloom Filter technique and lightweight symmetric deterministic encryption. Especially, our Protocol II is able to hide the size of the set/intersection from leaking to the server via adding dummy sets, and also can improve security against a malicious server with modifying the computed intersection or randomly returning a false result. Our novel *multi-round* mechanism can significantly enhance efficiency of the basic protocols with maintaining a desirable accuracy, and it is quite feasible to the practical application.

Acknowledgments

This work was supported by Program for Scientific Research Foundation for Talented Scholars of Jinling Institute of Technology (JIT-B-201639, JIT-B-201726), Natural Science Research Projects of Universities (19KJB520033), Philosophy and Social Science Foundation of the Jiangsu Higher Education Institutions of China (2021SJA0448), Natural Science Foundation of Jiangsu Province (BK20210928), Higher Education Research Project of Nanjing Institute of Technology (2021ZC13), and National Natural Science Foundation of China (61902163).

Conflict of interest

The authors declare there is no conflict of interest.

References

1. Q. Tang, Public key encryption supporting plaintext equality test and user-specified authorization, *Secur. Commun. Networks*, **5** (2012), 1351–1362. doi: 10.1002/sec.418.
2. D. Kales, C. Rechberger, T. Schneider, M. Senker, C. Weinert, Mobile private contact discovery at scale, in *28th USENIX Security Symposium (USENIX Security 19)* , (2019), 1447–1464.
3. P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, G. Tsudik, Countering gattaca: Efficient and secure testing of fully-sequenced human genomes , in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, (2011), 697–702. doi: 10.1145/2046707.2046785.
4. M. Ion, B. Kreuter, E. Nergiz, S. Patel, S. Saxena, K. Seth, et al., Private intersection-sum protocol with applications to attributing aggregate ad conversions, *IACR Cryptol. ePrint Arch.*, (2017), 738.
5. E. D. Cristofaro, G. Tsudik, Practical private set intersection protocols with linear complexity, *Lect. Notes Comput. Sci.*, **6052** (2010), 143–159. doi: 10.1007/978-3-642-14577-3_13.
6. C. Dong, L. Chen, Z. Wen, When private set intersection meets big data: An efficient and scalable protocol, in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, (2013), 789–800. doi: 10.1145/2508859.2516701.
7. S. Qiu, J. Liu, Y. Shi, Identity-based symmetric private set intersection, *Wuhan Univ. J. Nat. Sci.*, **19** (2014), 426–432. doi: 10.1007/s11859-014-1035-z.

8. B. Pinkas, M. Rosulek, N. Trieu, A. Yanai, Spot-light: Lightweight private set intersection from sparse ot extension, *Annu. Int. Cryptology Conf.*, (2019), 401–431. doi: 10.1007/978-3-030-26954-8_13.
9. M. Chase, P. Miao, Private set intersection in the internet setting from lightweight oblivious prf, *Lect. Notes Comput. Sci.*, Springer, **12172** (2020), 34–63. doi: 10.1007/978-3-030-56877-1_2.
10. S. Kamara, P. Mohassel, M. Raykova, S. Sadeghian, Scaling private set intersection to billion-element sets, *Financ. Cryptography Data Secur.*, **8437** (2014), 195–215. doi: 10.1007/978-3-662-45472-5_13.
11. E. Zhang, F. Li, B. Niu, Y. Wang, Server-aided private set intersection based on reputation, *Inf. Sci.*, **387** (2017), 180–194. doi: 10.1016/j.ins.2016.09.056.
12. H. Liu, H. Zhang, L. Guo, J. Yu, J. Lin, Privacy-preserving cloud-aided broad learning system, *Comput. Secur.*, **112** (2022), 102503. doi: 10.1016/j.cose.2021.102503.
13. T. Wang, J. Zhou, X. Chen, G. Wang, A. Liu, Y. Liu, A three-layer privacy preserving cloud storage scheme based on computational intelligence in fog computing, in *IEEE Transactions on Emerging Topics in Computational Intelligence*, **2** (2018), 3–12. doi: 10.1109/TETCI.2017.2764109.
14. H. Zhang, J. Yu, C. Tian, G. Xu, J. Lin, Practical and secure outsourcing algorithms for solving quadratic congruences in internet of things, *IEEE Int. Things J.*, **7** (2020), 2968–2981. doi: 10.1109/JIOT.2020.2964015.
15. H. Zhang, J. Yu, M. S. Obaidat, P. Vijayakumar, R. Hao, Secure edge-aided computations for social internet-of-things systems, *IEEE Trans. Comput. Soc. Syst.*, (2020), 1–12. doi: 10.1109/TCSS.2020.3030904.
16. A. Abadi, S. Terzis, C. Dong, Feather: lightweight multi-party updatable delegated private set intersection, *IACR Cryptol. ePrint Arch*, 2021.
17. A. Abadi, S. Terzis, R. Metere, C. Dong, Efficient delegated private set intersection on outsourced private datasets, *IEEE Trans. Dependable Secure Comput.*, **16** (2017), 608–624. doi: 10.1109/TDSC.2017.2708710.
18. A. Kavousi, J. Mohajeri, M. Salmasizadeh, Improved secure efficient delegated private set intersection, in *2020 28th Iranian Conference on Electrical Engineering*, (2020), 1–6. doi: 10.1109/ICEE50131.2020.9260663.
19. L. Tajan, D. Westhoff, F. Armknecht, Private set relations with bloom filters for outsourced sla validation, *IACR Cryptol. ePrint Arch.*, (2019), 993.
20. M. J. Freedman, K. Nissim, B. Pinkas, Efficient private matching and set intersection, *Lect. Notes Comput. Sci.*, **3027** (2004), 1–19. doi: 10.1007/978-3-540-24676-3_1.
21. S. Ghosh, M. Simkin, The communication complexity of threshold private set intersection, *Lect. Notes Comput. Sci.*, **11693** (2019), 3–29. doi: 10.1007/978-3-030-26951-7_1.
22. C. Hazay, Oblivious polynomial evaluation and secure set-intersection from algebraic prfs, *J. Cryptol.*, **31** (2018), 537–586. doi: 10.1007/s00145-017-9263-y.
23. A. Kavousi, J. Mohajeri, M. Salmasizadeh, Efficient scalable multi-party private set intersection using oblivious prf., *Secur. Trust Manage.*, (2021), 81–99. doi: 10.1007/978-3-030-91859-0_5.

24. P. Rindal, M. Rosulek, Improved private set intersection against malicious adversaries, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, (2017), 235–259. doi: 10.1007/978-3-319-56620-7_9.
25. B. Pinkas, T. Schneider, O. Tkachenko, A. Yanai, Efficient circuit-based psi with linear communication, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, (2019), 122–153. doi: 10.1007/978-3-030-17659-4_5.
26. B. Pinkas, T. Schneider, C. Weinert, U. Wieder, Efficient circuit-based psi via cuckoo hashing, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, (2018), 125–157. doi: 10.1007/978-3-319-78372-7_5.
27. M. Ciampi, C. Orlandi, Combining private set-intersection with secure two-party computation, *Lect. Notes Comput. Sci.*, Springer, **11035** (2018), 464–482. doi: 10.1007/978-3-319-98113-0_25.
28. G. Garimella, B. Pinkas, M. Rosulek, N. Trieu, A. Yanai, Oblivious key-value stores and amplification for private set intersection, *Lect. Notes Comput. Sci.*, **12826** (2021), 395–425. doi: 10.1007/978-3-030-84245-1_14.
29. B. Pinkas, T. Schneider, M. Zohner, Scalable private set intersection based on ot extension, *ACM Trans. Priv. Secur.*, **21** (2018), 1–35. doi: 10.1145/3154794.
30. M. Ali, J. Mohajeri, M. R. Sadeghi, X. Liu, Attribute-based fine-grained access control for outscored private set intersection computation, *Inf. Sci.*, **536** (2020), 222–243. doi: 10.1016/j.ins.2020.05.041.
31. C. Dong, L. Chen, J. Camenisch, G. Russello, Fair private set intersection with a semi-trusted arbiter, *Lect. Notes Comput. Sci.*, **7964** (2013), 128–144. doi: 10.1007/978-3-642-39256-6_9.
32. H. Chen, K. Laine, P. Rindal, Fast private set intersection from homomorphic encryption, in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, (2017), 1243–1255. doi: 10.1145/3133956.3134061.
33. S. K. Debnath, R. Dutta, Secure and efficient private set intersection cardinality using bloom filter, *Lect. Notes Comput. Sci.*, Springer, **9290** (2015), 209–226. doi: 10.1007/978-3-319-23318-5_12.
34. D. Stritzl, *Privacy-Preserving Matching Using Bloom Filters: An Analysis And An Encrypted Variant*, Master's thesis, University of Twente, 2019.
35. H. Carter, B. Mood, P. Traynor, K. Butler, Outsourcing secure two-party computation as a black box, 2016. doi: 10.1002/sec.1486.
36. O. Goldreich, *Foundations of cryptography: volume 2, basic applications*, Cambridge University Press, 2004.
37. J. Katz, Y. Lindell, *Introduction to Modern Cryptography*, CRC Press, 2007.

AIMS Press