IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 19, 2020, accepted December 26, 2020, date of publication January 1, 2021, date of current version January 12, 2021.

Digital Object Identifier 10.1109/ACCESS.2020.3048743

Efficient Threshold Private Set Intersection

EN ZHANG, JIAN CHANG, AND YU LI
College of Computer and Information Engineering, Henan Normal University, Xinxiang 453007, China

Corresponding author: En Zhang (zhangenzdrj@163.com)

This work was supported by the National Natural Science Foundation of China under Grant U1604156, Grant U1804164, and Grant 61901160.

ABSTRACT Threshold private set intersection (TPSI) allows a receiver to obtain the intersection when the cardinality of the intersection is greater or equal to the threshold, which has a wide range of applications such as fingerprint matching, online dating and ridesharing. Existing TPSI protocols are inefficient because almost all of them rely on lots of expensive public-key techniques or require an exponential number of possible combinations among the shares. In this work, we design an efficient TPSI protocol, which achieves computational security in semi-honest model. To improve the efficiency of the TPSI protocol, we design a new TPSI protocol based on garbled Bloom filter (GBF) and threshold secret sharing, which uses a small amount of public-key operations. Moreover, our protocol combines with the Reed-Solomon decoding algorithm to reconstruct the secret which is a feasible method to avoid calculating all possible combinations among the shares. The performance analysis shows that our protocol is more efficient than the previous TPSI protocols. To the best of our knowledge, the optimal TPSI protocol implemented by Zhao and Chow (WPES'18) has an online time of 78 seconds to compute the intersection of two datasets of 100 elements each with threshold $t = 50$. In contrast, our protocol has a total time of 2.988 seconds.

INDEX TERMS Garbled bloom filter, secure multiparty computation, threshold private set intersection, threshold secret sharing.

I. INTRODUCTION

Private set intersection (PSI) is one of the important branches of secure multiparty computation (MPC). An ideal private set intersection protocol guarantees the security of elements other than the intersection while each participant gets the intersection. There are several common PSI application scenarios, such as measure advertisement conversion rates, search potential friendships via social networks, botnet detection, human genomes testing and proximity testing. Google introduced a browser Password Checkup in 2019, giving browser users a tool to check whether their personal log-in passwords are leaked. The principle of the Password Checkup tool is based on PSI.

The development of the Internet has spawned many new industries in recent years, such as ridesharing and online dating. Taking ridesharing as an example, the routes of the two strangers may not be completely the same under the normal circumstance (the possibility of complete consistency is negligible). In order to protect their own privacy, the two parties do not want to share their own route to the other party. How to share the common route shared by the two parties is the top priority in this scene. Therefore, Hallgren *et al.* [1] introduced the threshold PSI to solve this problem. This provides a new way to study PSI. Threshold PSI can also be used in typical scenarios such as fingerprint matching, online dating and ridesharing. In general, these data are private, and this approach provides good privacy protection for the data owner.

The associate editor coordinating the review of this manuscript and approving it for publication was Mohamad Afendee Mohamed.

A. RELATED WORK

Yao [2] introduced the idea of secure computation, which enables parties to securely compute a function that relies on private datasets. As an important branch of MPC, PSI [3]–[7] has been widely concerned by scholars. PSI is a vibrant research field, and lots of PSI protocols are proposed in recent years. These protocols are classified as follows.

Public-Key-Based PSI: The protocol based on public-key encryption is mainly to perform relatively complicated public-key encryption operations on set elements, and then carry out corresponding calculations on ciphertext, such as homomorphic encryption. The PSI protocol based on oblivious polynomials combined with homomorphic encryption was introduced by Freedman *et al.* [3], and they gave specific

protocols in the semi-honest and the malicious setting respectively. Subsequently, Freedman *et al.* [8] used different hash structures to represent set elements to reduce computational complexity, compared their performance, and implemented the semi-honest security protocol in the standard model. Kolesnikov *et al.* [9] constructed an efficient PSI protocol that used oblivious pseudorandom function (OPRF) batch which is an improvement on [10]. A new and improved method, that is, 1-out-of-2 oblivious transfer (OT) extension, was used in this protocol, which is especially efficient when generating large numbers of OPRF instances. Lv *et al.* [11] studied the unbalanced private set intersection cardinality based on commutative encryption in the semi-honest model.

OT-Based PSI: Dong *et al.* [12] constructed a novel data structure called garbled Bloom filter (GBF) and proposed first PSI protocol based on GBF and OT extension which handles set elements up to the size of billions and mainly relies on efficient symmetric encryption operations. However, there are two problems in this protocol: one is the malicious sender may send the wrong shares, the other is the two input datasets are not independent. Rindal and Rosulek [13] further proposed a new structure called random garbled Bloom filter and solved the above problems by using the new structure and cut-and-choose. They gave the two-party PSI protocol in the malicious setting. Zhang *et al.* [6] further proposed and implemented the multi-party PSI protocol, which guarantees the malicious security in the case that there are two non-colluding servers. Their experimental data show that both time and communication cost depend on the number of participants. Pinkas *et al.* [10] described a new PSI protocol based on OT extension. They compared the performance of several different protocols on the same platform and their protocol improves the operational efficiency of previous protocols. Pinkas *et al.* [14] found a better balance between the communication and computational overhead of the two-party PSI protocol. In general, the protocol has the lower monetary cost than other protocols.

Circuit-Based PSI: The circuit-based protocols are general security computing protocols that allow arbitrary functions to be computed. Although the PSI protocols based on the circuit have universality, flexibility and scalability, the design of circuit-based PSI protocols generally requires a large number of gates and circuit depth. So protocols based on circuit technology generally have low computation efficiency. But they have improved greatly in efficiency in recent years.

Huang *et al.* [15] explored three PSI protocols for datasets with different characteristics based on garbled circuit. Pinkas *et al.* [16] applied a new method, which denotes as Phasing, for PSI protocols based on circuit. Their result shows that Phasing can significantly improve the performance and their protocol is faster than the protocol proposed by Huang *et al.* [15]. In the two-party model, the PSI protocol based on symmetric-key primitives proposed by Rindal and Rosulek [17] is secure against malicious adversary and is 12 times faster than their previous protocol [13].

Pinkas *et al.* [18] constructed variants of Cuckoo hashing and proposed new PSI protocols with almost linear comparisons. In addition, their protocol can be extended to multi-party. To our knowledge, PSI protocols based on Cuckoo hashing are hard against malicious adversaries. Because the participant who uses simple hashing may not map his elements to both tables at the same time in the malicious setting. To solve the above problem, Pinkas *et al.* [19] designed a novel structure probe-and-XOR of strings and proposed the malicious PSI protocol by using cuckoo hashing.

Cloud-Server-Based PSI: With the development of cloud technology, the PSI protocol based on cloud server has also become the research interests of most researchers. The advantage of cloud server is that it has good computing power and storage capacity, which effectively improves the efficiency of these protocols, and provides a mature optimization method for the existing PSI protocols.

Kerschbaum proposed two PSI protocols for outsourcing computation. The first one [20] realized anti-collusion outsourcing PSI protocol by using one-way functions, and the second one [21] combining Boneh Goh Nissim homomorphic encryption [22] and Sander Young Yung technology [23] had a great impact on performance. Kamara *et al.* [24] designed the PSI protocols in three different models, and used dummy sets to prevent the malicious server from sending wrong results. At the same time, they were still able to achieve good efficiency under the condition that the data volume of all parties was 100 million. Aydin *et al.* [25] implemented the static semi-honest adversary's multi-party PSI protocol and assumed that the server is not colluding with the participants. The protocol uses polynomials in point-value form to represent the set elements, which is different from the general method of coefficient vector representation. Ali *et al.* [26] first proposed the attribution-based outsourcing PSI scheme. The solution provides fine-grained access control and the data owners do not have to be online after the data is outsourced.

Threshold PSI: In addition to the standard PSI functionality, many works achieve TPSI functionality which enables parties to get the intersection if the cardinality of the intersection is greater or equal to the threshold t. Suppose that $t = 4$, if the cardinality of intersection $|X \bigcap Y| < 4$, then the function outputs $\perp$, as shown in Fig 1(a). If the number of intersection $|X \bigcap Y| \geq 4$, the function outputs the intersection $X \bigcap Y$, as shown in Fig 1(b).

The simplest way to achieve the threshold PSI is to first compute the cardinality of the intersection, then compare it with the threshold and decide whether to output the intersection, such as [1], [3], [18], [27], [28]. Freedman *et al.* [3] described a variant of private matching for set cardinality that the client could learn whether $|X \bigcap Y| \geq t$. Similarly, Pinkas *et al.* [18] also proposed a simple method to determine whether the cardinality of the intersection is greater than the threshold, which is based on the size of the intersection. Zhao and Chow [29] proposed a method called secret transfer with access structure (STAS). The receiver can obtain the secret (the message) when a certain condition is met, which is the

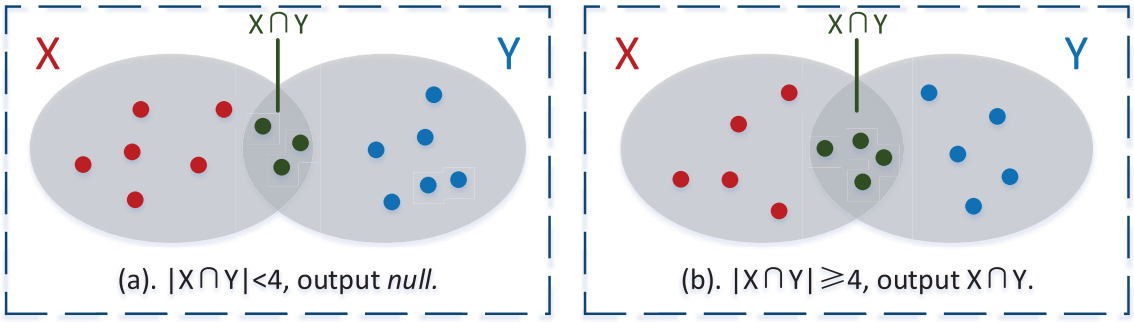

FIGURE 1. Threshold PSI with a threshold $t = 4$.

general case of threshold PSI. In the threshold PSI scenarios, the parties can obtain the intersection when the cardinality of the intersection is greater or equal to the threshold t. Zhao constructed threshold PSI for special scenarios, and implemented two threshold PSI protocols by using STAS. Hallgren *et al.* [1] constructed a threshold-key encapsulation mechanism which implements a threshold PSI protocol and applied the threshold PSI to privacy-preserving ridesharing. Their protocol only works well on small datasets since it needs to compute C_n^t possible combinations to reconstruct the secret. The cost of time will be very high when the datasets are large. Zhao and Chow [27] not only studied the threshold PSI with $|X \bigcap Y| \geqslant t$, but also the threshold PSI with $|X \bigcap Y| \leqslant t$, where X and Y are private sets of sender and receiver respectively. Moreover, Zhao *et al.* also outsourced heavy computations in the protocols to a cloud server to achieve better efficiency. To date there are only two TPSI protocols [1], [27] implemented. Ghosh and Nilges [28] proposed a new tool called oblivious linear function evaluation, and constructed a threshold PSI protocol by using the algebraic method. Ghosh and Simkin [30] studied the communication cost of threshold PSI, and proposed the first protocol with the communication complexity that relies on threshold t. Subsequently, Badrinarayanan *et al.* [31] designed two multi-party TPSI functionalities in which the communication complexity increases with the size of set difference. The first functionality allows parties to learn the intersection if the sets and the intersection differ by at most t. The second one allows parties to learn the intersection if the union and the intersection differ by at most t, which is more efficient than the first protocol.

B. CONTRIBUTIONS

In this work, we propose an efficient TPSI protocol. To the best of our knowledge, to date only Hallgren *et al.* [1] and Zhao and Chow [27] implemented their TPSI protocols. We analyzed the reason why TPSI is difficult to achieve. A series of works [1], [27], [28], [30], [31] propose the TPSI functionality using lots of expensive public-key techniques. In addition, some of them require an exponential number of possible combinations among the shares. Specifically, Hallgren *et al.* [1] designed a threshold-key encapsulation mechanism needs to compute C_n^t possible combinations to reconstruct the secret to implement TPSI protocol. Furthermore, the existing TPSI protocols are only suitable for small datasets. However, our protocol can support larger datasets. Our contributions are as follows:

- We implement a novel TPSI protocol based on GBF and threshold secret sharing. Our protocol requires a small amount of base OTs and avoids heavily public-key operations.
- Moreover, our protocol combines Reed-Solomon decoding algorithm to reconstruct the secret. It does not require calculating all possible combinations among the shares.
- Our protocol supports larger datasets than the existing TPSI protocols. In the existing TPSI protocols, the maximum data size tested is 4092 in [1] which has a total time of 5629 seconds. In contrast, our overall running time is 30.4 seconds. Further, we set the maximum data size is 16384 and the overall time is 519 seconds.

The experiment shows that our protocol is more efficient. Specifically, we set $n = 100$ and $t = 50$, where n is the number of set element and t is the threshold and the time cost is 2.988 s. Our protocol is much faster than [27] which has a total time of 3.6 minutes. In [1], their threshold t is set to $80\%n$ and the running time is 5629 seconds when n is set to 4092. In contrast, our overall running time is 30.4 seconds. See section 5 for more details.

C. ROADMAP

In this paper, the structure is as follows. Chapter II mainly describes some basic knowledge that our protocols involved. Chapter III describes our new threshold PSI protocol. In Chapter IV, we analyse the parameters and the security of the protocol. We test our TPSI protocol and compare the performance of our protocol with the existing protocols in Chapter V. Finally, in Chapter VI, we make a summary for the paper.

II. PRELIMINARIES

A. NOTATION

In this paper, we use λ and σ to represent computational and statistical security parameter, respectively. We write $[n]$ to denote $\{1, \ldots, n\}$ and $[m]$ to denote $\{1, \ldots, m\}$. There are

Ideal functionality for PSI $\mathcal{F}_{\mathrm{PSI}}$:
Parameter: There are two parties sender S and receiver R, who have the private datasets X and Y, respectively. These datasets have n elements and are represented as $|X| = |Y| = n$. Let x_i and y_i represent the i-th element in the set X and Y respectively, where $i \in [n]$. λ is the security parameter and denotes the length of elements x_i (or y_i).

- **Input:** Wait for input X from the sender S and input Y from the receiver R.
- **Computation:** The ideal functionality $\mathcal{F}_{\mathrm{PSI}}$ computes $X \bigcap Y$.
- **Output:** Finally, the ideal functionality $\mathcal{F}_{\mathrm{PSI}}$ sends the result to the receiver R.

FIGURE 2. Ideal functionality for PSI.

two participants sender S and receiver R, who have their own private datasets X and Y, respectively. The elements in X and Y are expressed as $x_i \in \{0,1\}^\lambda$ and $y_i \in \{0,1\}^\lambda$ where $i \in [n]$. The hash function is denoted by $h(x) : \{0,1\}^* \to [m]$. We write $s \in \mathbb{F}$ to denote the random secret where $\mathbb{F}$ is a finite field. Let $s_i, Ind_i \in \{0,1\}^\lambda$ denote the i-th share of s and the corresponding index which satisfies $f(Ind_i) = s_i$ in Shamir' scheme.

B. PSI AND THRESHOLD PSI

PSI allows parties to learn the intersection from their datasets securely, which protects their private elements. We give the ideal function of the two-party PSI with one-side output in Fig 2.

Threshold PSI is a variant of traditional PSI, which guarantees that the receiver can obtain the intersection when the cardinality of the intersection is greater or equal to the threshold t. We give the ideal function of two-party threshold PSI in Fig 3.

C. THRESHOLD SECRET SHARING AND REED-SOLOMON CODE

Secret sharing is a basic cryptographic primitive that splits a secret $s \in \mathbb{F}$ into n shares $s_1, \ldots, s_n$ and distributes them to n parties. The parties take n shares to reconstruct the secret s. Shamir [32] and Blakley [33] proposed threshold secret sharing schemes based on Lagrange interpolation theorem and Gaussian elimination, respectively. They allow t or more correct shares to reconstruct the secret instead of n shares, where t is the threshold and smaller than n. Our protocol calls the sharing phase of Shamir' sharing scheme and denotes it by $\mathcal{F}_{\mathrm{TSS}}$. We give the ideal functionality of Shamir's scheme in Fig 4. See [32] for the specific scheme.

Reed-Solomon codes are a kind of error-correcting codes proposed by Reed and Solomon [34]. Later, a series of Reed-Solomon code algorithms [35]–[37] are proposed. The Reed-Solomon decoding algorithms provide extensions and generalizations of Shamir's scheme. In order to reconstruct the secret, Reed-Solomon decoding algorithm ignores a few incorrect shares and outputs the polynomial. Our protocol combines with the Reed-Solomon decoding algorithm [37] to reconstruct the secret s, which uses fast Fourier transforms and does not reveal the error position or magnitudes.

Ideal functionality for Threshold PSI $\mathcal{F}_{\mathrm{TPSI}}$:
Parameter: There are two parties sender S and receiver R, who have the private datasets X and Y, respectively. These datasets have n elements and are represented as $|X| = |Y| = n$. Let x_i and y_i represent the i-th element in the set X and Y respectively, where $i \in [n]$. λ denotes the length of elements x_i (or y_i) and t is the threshold which is public.

- **Input:** Wait for input X from the sender S and input Y from the receiver R. Then they input the threshold t.
- **Computation:** The ideal functionality $\mathcal{F}_{\mathrm{TPSI}}$ computes $|X \bigcap Y|$ and $X \bigcap Y$.
- **Output:** Finally, the ideal functionality $\mathcal{F}_{\mathrm{TPSI}}$ sends the result to the receiver while $|X \bigcap Y| \geqslant t$. If $|X \bigcap Y| < t$, the receiver can not learn anything.

FIGURE 3. Ideal functionality for Threshold PSI.

Ideal functionality for Shamir's sharing scheme $\mathcal{F}_{\mathrm{TSS}}$:
Parameter: Give integers n, t $(n < t)$ and the finite filed $\mathbb{F}$. Fix n distinct points $ind_1, ..., ind_n$ from $\mathbb{F}$.
Sharing: For input $s \in \mathbb{F}$, choose a random polynomial $f(x)$ of degree t which satisfies $f(0) = s$ and output the set $S = \{f(ind_1) = s_i, ..., f(ind_n) = s_n\}$.
Reconstruction: For the input set $R \subset S$ of size $t+1$, use Lagrangian interpolation to compute the polynomial $f'(x)$ which satisfies $f'(ind_i) = s'_i$, where $s'_i \in R$ and output $f'(0)$.

FIGURE 4. The functionality of $\mathcal{F}_{\mathrm{TSS}}$.

D. OBLIVIOUS TRANSFER

Oblivious transfer [38], [39] is an important cryptographic primitive. There are a sender holding data $m_0, m_1 \in \{0,1\}^*$ and a receiver holding a choice bit $b \in \{0,1\}$. The receiver can safely get m_b from the sender according to the choice bit b by performing the OT functionality. Meanwhile, the sender does not know whether the receiver receives m_0 or m_1. The construction of OT needs a heavy cost in practical application, because the implementation of OT requires expensive public-key operations. Oblivious transfer extension [40], [41] provides a more efficient way to apply in practical applications which uses a small amount of base OTs and avoids heavily public-key operations.

E. (GARBLED) BLOOM FILTER AND PSI WITH GBF

Bloom filter [42] is an efficient structure that can store and query items by using k hash functions easily. In general, the

Parameter: Let BF_C denote the Bloom filter generated by the participant C. The participant C has the private set $X = \{x_1, \ldots, x_n\}$ where $|X| = n$. We write m to denote the length of the Bloom filter. The party randomly selects $h_1, \ldots, h_k : \{0,1\}^* \to [m]$ as the hash functions of Bloom filter .

- **Setup:** The BF_C has m bins, and each bin is set to 0, which is denoted by $BF[i] = 0$ where $i \in [m]$.
- **Insert:** For $i \in [n]$, each element x_i in the set X, there are k hash functions to compute the indexes $h_1(x_i), \ldots, h_k(x_i)$. And the indexes are set to 1.
- **Query:** There is an element y and we want to to query if it is in the BF_C.
  For $i \in [k]$, compute the indexes $h_1(y), \ldots, h_k(y)$ and check the value $BF[i] = 1$ or not. If all the values are 1, then it is determined that y is in the BF_C. That is, $y \in X$. Otherwise, it is determined that $y \notin X$.

FIGURE 5. The data structure of Bloom Filter.

capacity of Bloom filter is expressed as m, and k hash functions are expressed as $h_1, h_2 \ldots, h_k$. Supposed that we have an item x and we want to store it in this data structure, then the item x will be mapped by k hash functions one by one into k bins. Let $h_i(x)$ denote the location where x is indexed by the i-th hash function in Bloom filter. But Bloom filter has a property called false positive. In other words, the item y is not stored in the Bloom filter, but the values in the k bins are all 1, and the index positions of these bins are $h_1(y), \ldots, h_k(y)$. The probability of false positive is determined by the parameters k and m. Generally speaking, we hope that the probability of false positive p is negligible. If n is the number of elements, m and k satisfy $m \geq n \log_2 e \cdot \log_2 1/p$ and $k = (m/n) \cdot \ln 2$ respectively in the optimal case. The formal description of Bloom filter is given in Fig 5.

Garbled Bloom filter (GBF) is an improved data structure of Bloom filter proposed by Dong *et al.* [12] to better represent set elements. The structure takes advantage of the feature of secret sharing, and stores the elements into share values in the GBF data structure with negligible false positive when querying an element. The formal description of GBF is given in Fig 6. Let's review the protocol of DCW [12] and the formal description is given in Fig 7.

F. SECURITY MODEL

We consider the security of the protocol in the static semi-honest adversaries setting. There is an adversary who controls one of the parties and fully follows the protocol specification in the semi-honest model. But the adversary tries to get more additional information from the protocol. Please refer to [43], [44] for a more detailed and formal description.

In the threshold PSI protocol, there are two parties: a sender S with input X and a receiver R with input Y. They execute

Parameter: Let GBF_S denote the garbled Bloom filter generated by the participant S. The participant S has the private set $X = \{x_1, \ldots, x_n\}$ where $|X| = n$. We write m to denote the length of the Bloom filter. The party randomly selects $h_1, \ldots, h_k : \{0,1\}^* \to [m]$ as the garbled Bloom filter hash functions.

- **Setup:** The GBF_S has m bins, and each bin is set to $null$, which is denoted by $GBF[i] = null$ where $i \in [m]$.
- **Insert:** For $i \in [n]$, each element x_i in the set X, there are k hash functions to compute the indexes $h_1(x_i), \ldots, h_k(x_i)$. The element x_i is separated into k shares x_i^j with λ bits which satisfies $x_i^1 \oplus \cdots \oplus x_i^k = x_i$. The share x_i^j is stored into $GBF_S[h_j(x_i)]$ where $j \in [k]$. The remaining bins that do not store strings are put into random strings r_i where $i \in [m]$. For convenience, we use m_i to represent strings stored in GBF_S where $i \in [m]$.
- **Query:** There is an element y and we want to query if it is in the GBF_S.
  For $i \in [k]$, compute the indexes $h_1(y), \ldots, h_k(y)$ and check $m_{h_1(y)} \oplus \cdots \oplus m_{h_k(y)} = y$ or not. If the XOR result of k strings is y, then it is determined that y is in the GBF_S. That is, $y \in X$. Otherwise, it is determined that $y \notin X$.

FIGURE 6. The data structure of Garbled Bloom Filter.

the two-party protocol which computes the function f and outputs $f(X, Y)$.

We define the functionality $f : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$, where $f = X \bigcap Y$ and f is a deterministic functionality (X and Y are the input sets of the sender and receiver respectively). Define π as a two-party protocol that computes the functionality f. The view of the sender and the receiver are denoted by $\mathsf{view}_S^\pi(X, Y, \lambda)$ and $\mathsf{view}_R^\pi(X, Y, \lambda)$ respectively, where λ is the computational security parameter. The output of the sender is $\perp$ and the output of the receiver is $\mathsf{output}_R^\pi(X, Y, \lambda)$ which is *null* or $X \bigcap Y$. That is, if $|X \bigcap Y| \geq t$, $\mathsf{output}_R^\pi(X, Y, \lambda) = X \bigcap Y$. Otherwise, $\mathsf{output}_R^\pi(X, Y, \lambda) = null$.

Definition 1: Let f be a deterministic functionality. The protocol π computes the functionality f against semi-honest adversary securely if there exist probabilistic polynomial-time (PPT) algorithms Sim_S and Sim_R satisfy:

$$Sim_S(1^\lambda, X, \perp) \overset{c}{\equiv} \mathsf{view}_S^\pi(X, Y, \lambda) \quad (1)$$

$$Sim_R(1^\lambda, Y, \mathsf{output}_R^\pi(X, Y, \lambda)) \overset{c}{\equiv} \mathsf{view}_R^\pi(X, Y, \lambda) \quad (2)$$

where $X, Y \in \{0,1\}^*$ and $|X| = |Y|$.

III. THE THRESHOLD PRIVATE SET INTERSECTION PROTOCOL

In the section, a detailed description of the TPSI protocol is given. Our protocol is secure against passive adversary in semi-honest setting and the new TPSI protocol is shown in

Parameter: Let GBF_S denote the garbled Bloom filter generated by the sender S and BF_R denote the Bloom filter generated by the receiver R. The participants S and R have the private sets $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$. We write m to denote the length of the (garbled) Bloom filter. Parties randomly select $h_1, \ldots, h_k : \{0,1\}^* \to [m]$ as the (garbled) Bloom filter hash functions.

- **Setup:** The sender S inserts his elements in the set X into the garbled Bloom filter GBF_S, and the receiver R inserts her elements in the set Y into the Bloom filter BF_R. The bits in BF_R are represented as b_i. The sender S randomly selects m random strings r_i as parameters of the OT stage where $i \in [m]$.
- **OT extension:** The sender S holds $2m$ strings r_i and m_i, and the receiver R holds m bits b_i in BF_C. During the execution of OT extension, if $b_i = 0$, R gets the string r_i, and if $b_i = 1$, R gets the string m_i. After executing the OT protocol, R will get the $GBF_{X\cap Y}$.
- **Query and Output:** The receiver R queries her elements in the set Y whether they are in the $GBF_{X\cap Y}$. For $j \in k$, R computes the indexes $h_1(y_i), \ldots, h_k(y_i)$ and checks $m_{h_1(y_i)}{}' \oplus \cdots \oplus m_{h_k(y_i)}{}' = y_i$ or not. If the XOR result of k strings is y_i, then it is determined that y_i is in the GBF_S. That is, $y_i \in X$. And the element y_i is put into the set $Z = X \cap Y$. Otherwise, it is determined that y_i is not in the GBF_S. That is, $y_i \notin Z$. In the end, the receiver R outputs the intersection Z.

FIGURE 7. The PSI Protocol of DCW.

Fig 8. The main idea of our protocol is that determine the relationship between $|X \cap Y|$ and the threshold t through the secret sharing scheme. The protocol satisfies that enough correct shares to reconstruct the secret s. This means there are enough elements in the intersection and the receiver could compute the polynomial of the secret sharing scheme. Only when the above conditions are met, can the receiver obtain the intersection.

For the sender, a simple approach is to assign a share of the secret s to each element. If the receiver can get enough correct shares, she could output the intersection. But there is a serious problem: If the cardinality of the intersection does not reach the threshold, the receiver can get additional information from the correct shares. That means the elements corresponding to the correct shares are in the intersection.

To solve the above problem, we design a new GBF that can establish the relationship between set elements and the secret shares and use it as threshold detection for the threshold PSI. In addition, the index corresponding to each share is also stored in the GBF, and the receiver can check whether the share she obtained is correct. The construction of the new GBF is similar to GBF, and the setting of parameters is the same as GBF. If $|X \cap Y| \geq t$, the receiver can get enough correct shares, and then reconstruct the secret s. Otherwise, the protocol is terminated and outputs $\perp$, and the receiver cannot get any information from the execution of the protocol.

We suppose that the size of dataset is n. The sender randomly selects secret s from the finite field $\mathbb{F}$ and calls Shamir' scheme to generate n shares. It satisfies t or more correct shares to recover s. Based on GBF and Shamir' scheme, we give Algorithm 1 to generate the new GBF of a set X. Similar to Dong's GBF, the security of the new GBF structure depends on the parameter k. Namely, the false positive probability of the new GBF is at most 2^{-k}. Different from Dong's GBF, the new GBF has two changes:

- Instead of storing x_i, we store $x_i \bigoplus s_i$ in GBF, where x_i is the i-th element and s_i is the i-th share.
- Each random string $m_{h_j(x_i)}$ in the bin will be connected to another random string $I_{h_j(x_i)}$, and satisfy $I_{h_1(x_i)} \oplus \cdots \oplus I_{h_k(x_i)} = ind_i$, where ind_i is the index of the share s_i of the secret s, that is, $f(ind_i) = s_i$. For security, $I_{h_j(x_i)}$ has the same length as $m_{h_j(x_i)}$.

In addition, our protocol combines with the Reed-Solomon decoding algorithm to reconstruct the secret which is a feasible method to avoid calculating all possible combinations among the shares, and we need to ensure that at any value of threshold t, the receiver has the possibility of getting the intersection. In order to we design both parties to add the same d dummy elements. See section IV-A for specific analysis.

IV. PROTOCOL ANALYSIS

A. PARAMETERS

In order to use Reed-Solomon code to achieve **arbitrary** threshold PSI, we design both parties to add the same d dummy elements. The number of dummy elements is related to the threshold t and the cardinality of the set of parties n. The specific calculation process is given below.

We assume that the size of dataset of parties is n and the threshold is t. This means the number of intersection elements is greater or equal to t, the receiver can obtain the intersection. According to the requirements of our protocol, the parameters need to satisfy the following constraints:

- For TPSI protocol, the receiver gets the intersection $X \cap Y$ if $|X \cap Y| \geq t$.
- For Reed-Solomon code, the secret s is reconstructed in the TPSI protocol if $|X \cap Y| \geq t + (n-t)/2$.

If we apply the Reed-Solomon code to secret reconstruction, it requires that the number of wrong shares should be less or equal to $(n-t)/2$, that is, the number of correct shares should be greater or equal to $t + ((n+d)-t)/2$. In order to ensure that the Reed-Solomon code can reconstruct the secret in the requirements of the TPSI parameters, we add $n-t$ dummy elements to the sets of both parties.[1]

[1]Let d denote the number of dummy elements, and compute the equation $t + d = t + ((n+d)-t)/2$ to get $d = n - t$.

Parameter: Let GBF_S denote the garbled Bloom filter generated by the sender S and BF_R denote the Bloom filter generated by the receiver R. The participants S and R have the private sets $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$. We write m to denote the length of the (garbled) Bloom filter. Parties randomly select $h_1, \ldots, h_k : \{0,1\}^* \to [m]$ as the (garbled) Bloom filter hash functions. Let t denote the threshold and $H : \{0,1\}^* \to \{0,1\}^\lambda$ denote the cryptographic hash function.

Setup:

1. The two parties produce $seed_S \in \{0,1\}^\lambda$ and $seed_R \in \{0,1\}^\lambda$ respectively, and send them to each other. They then compute $seed_S \bigoplus seed_R$ and generate $n-t$ dummy elements individually. These dummy elements will be considered as common elements of both parties.
2. The sender S calls the functionality $\mathcal{F}_{\mathrm{TSS}}$ of threshold secret sharing to generate the secret s and $2n-t$ shares $s_1, \cdots, s_{2n-t}$ with $f(ind_i) = s_i$, where $i \in [2n-t]$ and ind_i is the index of the i-th share. Then, the sender S sends $H(s)$ to the receiver R. The dummy elements are represented as $x_{n+1}, \cdots, x_{2n-t}$ and $X' = \{x_1, \cdots, x_n, x_{n+1}, \cdots, x_{2n-t}\}$. The sender S inserts $x_i \in X'$ into the new garbled Bloom filter GBF_S, where $i \in [2n-t]$. The string in the j-th bin is represented as m_j, where $j \in [m]$. S randomly selects m random strings r_j as parameters of the OT extension stage. Similarly, the receiver R sets $Y' = \{y_1, \cdots, y_n, y_{n+1}, \cdots, y_{2n-t}\}$ and inserts $y_i \in Y'$ into the Bloom filter BF_R, where $i \in [2n-t]$. The bits in BF_R are represented as b_j, where $j \in [m]$.

OT extension:

The sender S holds $2m$ strings r_i and m_i, and the receiver R holds m bits b_i in BF_R. During the execution of OT extension, if $b_i = 0$, R gets the string r_i; otherwise, R gets the string m_i. After executing the OT extension protocol, R gets the $GBF_{X' \cap Y'}$.

Computation:

R connects λ-bit 0 to her elements in Y and these elements are represented as y'_i. For $i \in [n]$, R computes $s'_i || ind'_i = y'_i \oplus m'_{h_1(y_i)} \oplus \cdots \oplus m'_{h_k(y_i)}$. The former λ-bit is considered to be the share s_i computed by S, and the latter λ-bit is considered to be the index ind_i corresponding to the share. Then, R carries out the Reed-Solomon decoding algorithm for the former λ-bit strings. If correct shares (which are consist with the shares of S') are enough, R gets the polynomial $f(x)'$ and checks $H(f(0)') \stackrel{?}{=} H(s)$. Otherwise, outputs $\perp$.

Query and Output:

For $i \in [n]$, the receiver R checks whether the string s'_i is a valid share. If $s'_i = f'(ind'_i)$, then it is determined that y_i is in the $GBF_{X \cap Y}$. That is, y_i is the element in the intersection. Then y_i is putted into the set $Z = X \cap Y$. Otherwise, it is determined that y_i is not in the $GBF_{X \cap Y}$. That is, $y_i \notin Z$. In the end, the receiver R outputs the intersection Z.

FIGURE 8. The TPSI Protocol based on DCW.

These dummy elements are chosen by pseudo-random generator with the seed selected by the parties. The length of dummy elements is the same as the private elements, which is 128. Therefore, the probability of the collision between dummy elements and private elements is $(n-t)/2^{128}$.

B. SECURITY PROOF

In the section, we first give the security analysis of the protocol. We conduct the security analysis in two different cases, namely the sender or the receiver is corrupted. Since the protocol is not symmetrical, the proof of the two cases will be different.

Theorem 1: The TPSI protocol is secure in semi-honest setting when the threshold secret sharing scheme is semantically secure.

Proof: We prove the above theorem by analyzing the corrupt sender and corrupt receiver respectively. In two different cases, we simulate the input and output of the corrupt party separately.

Case 1: The sender S is corrupted. In the case, there is a simulator Sim_S to generate the computationally indistinguishable view from the following view:

$$\mathsf{view}_S^{\pi}(X, Y, \lambda) = \{X, seed_S, s, s_i, ind_i, X', m_i, r_i, \perp\}$$

which is the view in the real execution, where $seed_S$ is the random seed, s is the secret in Shamir' scheme, s_i is the i-th share of s, ind_i is the index of s_i, X' includes $x_i \in X$ and dummy elements, m_i and r_i are random strings.

Sim_S performs the following to simulate the view. It first creates an empty view and adds its simulated items to the view according to the order of execution of the protocol. In the phasing of setup, it adds the uniformly selected $\widetilde{seed}_S$, $\widetilde{s}$ to the view. Then, it generates $\widetilde{seed}_R$, $\widetilde{X}$ and $\widetilde{Y}$. The Sim_S gets the dummy elements, $\widetilde{X}'$ and $\widetilde{Y}'$. As before, these values are

Algorithm 1 *BuildNewGBF*$(X, n, m, k, H, \lambda, S, I)$

Input: A set X, $n, m, k, H = \{h_1, \ldots, h_k\}$, two sets S, I
Output: A $(n, m, k, H, \lambda, S, I)$-garbled Bloom filter GBF_X

GBF_X = new m-item array of bit strings;
for $i = 1$ *to* m **do**
  $GBF_X[i] = NULL$;
for $\ell = 1$ *to* n **do**
  foreach $x_\ell \in X, s_\ell \in S, ind_\ell \in I$ **do**
    emptyBin= -1, lastShare $= x_\ell \oplus s_\ell || ind_\ell$;
    for $i = 1$ *to* k **do**
      $j = h_i(x_\ell)$;
      if $GBF_X[j] == NULL$ **then**
        if *emptyBin* $== -1$ **then**
          emptyBin=j;
        else
          $GBF_X[j] \xleftarrow{r} \{0,1\}^{2\lambda}$;
          lastShare=lastShare$\oplus GBF_X[j]$;
      else
        lastShare=lastShare$\oplus GBF_X[j]$;
    GBF_X[emptyBin] = lastShare;
for $i = 1$ *to* m **do**
  if $GBF_X[i] == NULL$ **then**
    $GBF_X[i] \xleftarrow{r} \{0,1\}^{2\lambda}$;

appended to the view. In the phasing of OT extension, Sim_S randomly chooses $\widetilde{m}_i$ and $\widetilde{r}_i$ and appends them to the view. Finally, Sim_S adds $\perp$ to the view and outputs the view.

The input is fixed and the intermediate parameters are randomly selected in both views. We conclude that the simulated view and the real view are computationally indistinguishable.

Case 2: The receiver R is corrupted. In this case, there is a simulator Sim_R to generate a computationally indistinguishable view from the following view:

$$\mathsf{view}_R^{\pi}(X, Y, \lambda) = \{Y, seed_R, Y', b_i, s_i' || ind_i', f(x)', F(X \cap Y)\}$$

which is the view in the real execution, where $seed_R$ is the random seed, Y' includes $y_i \in Y$ and dummy elements, b_i is the choose bit, $s_i' || ind_i'$ is the message received from the sender, $f(x)'$ is the polynomial computed by the receiver and $F(X \cap Y)$ is the output.

Sim_R performs the following to simulate the view. It first generates the input set Y' and creates an empty view and adds the items it simulates to the view according to the order of execution of the protocol. In the phasing of setup, it adds the uniformly selected $seed_R'$ to the view. In the phasing of OT extension, Sim_R chooses b_i to obtain the messages from the sender S and appends it to the view. In the end, Sim_R computes $F(X \cap Y)$ and outputs the view.

Since the interaction between the two parties is only in the setup stage and the OT stage, the security proof of the protocol is also in these two stages. It is easy to see that the simulated view and the real view are computationally indistinguishable.

After the above analysis, it's proved that our protocol is secure in the semi-honest setting, and our proof is completed. □

V. PERFORMANCE AND COMPARISON

We implemented the TPSI protocol in C++, and the experiment was run on two Ubuntu 18.04 virtual machines with an Intel(R) Xeon(R) E5-2630 v4 @2.2GHz CPU, 64GB RAM. Let the computational security parameter $\lambda = 128$, the statistical security parameter $\sigma = 40$ and the number of hash functions $k = 128$.

We firstly analyze the communication complexity of our protocol. In the setup phase, the sender sends $seed_S \in \{0,1\}^\lambda$ and $H(s) \in \{0,1\}^\lambda$ to the receiver and receiver sends $seed_R \in \{0,1\}^\lambda$ to the sender. The communication cost of the parties is based on the length of the Bloom filter. For producing m OTs of 256-bit string, the concrete cost of OT extension stage is $m \cdot 256$ bits plus an additive overhead of a small amount of extended OTs. There is no communication between the two parties in the last two stages. Then we discuss the computational complexity of our protocol. Parties generate their garbled Bloom filter or Bloom filter which needs $(2n - t)k$ hash computations. In addition to the hash computations, S also requires a part of the XOR computations, so the time to generate GBF seems to be longer than the time to generate BF. In the OT extension stage, parties require $2m + 336$ hash computations. R reconstructs the secret s using the Reed-Solomon decoding algorithm and the computational complexity of $O(nlogn)$. Finally, R checks her shares which requires n polynomial computations. The result of our analysis is shown in Table 1.

TABLE 1. The communication and computational complexity of our protocol.

	Setup		OT Extension		Computation	Output
Party	S	R	S	R	R	R
Comm.	2λ	λ	$O(\lambda m)$		0	0
Comp.	$O(nk)$	$O(nk)$	$O(m)$		$O(nlogn)$	$O(n)$

The set size of parties is n and the threshold is t. We set t is $30\%n$, $50\%n$, $80\%n$ and $95\%n$ respectively and test the cost of time and communication when the set size is fixed. The cost of time and communication of our protocol are shown in Table 2 and Table 3. Fig 9 and Fig 10 show the total running time and communication of our protocol running on datasets ranging from 512 to 16384 elements with different threshold.

We can see that the time cost increases with the increase of the threshold with the same set size from the Fig 9. Obviously, the communication is inversely proportional to the threshold in Fig 10. The same data size causes different communications because the threshold changes also change the number of dummy elements. We give the number of dummy elements added in Table 4.

TABLE 2. The time cost of our protocol (On seconds).

Set Size	30%	50%	80%	95%
512	0.56	0.54	0.65	0.64
1024	1.89	2.04	2.32	2.44
2048	6.81	8.22	8.62	8.9
4096	26.71	29.1	33.34	34.53
8192	105.52	113.24	123.08	136.81
16384	403.04	444.06	519.26	542.93

TABLE 3. Communication cost of our protocol (In MB).

Set Size	30%	50%	80%	95%
512	10.23	9.04	7.23	6.33
1024	20.47	18.07	14.46	12.64
2048	40.92	36.11	28.89	25.27
4096	81.82	72.21	57.76	50.55
8192	163.63	144.39	115.51	101.08
16384	327.27	288.77	231.02	202.13

TABLE 4. The number of dummy elements added.

Set Size	30%	50%	80%	95%
512	358	256	102	26
1024	717	512	205	51
2048	1434	1024	410	102
4096	2867	2048	819	205
8192	5734	4096	1638	410
16384	11469	8192	3277	819

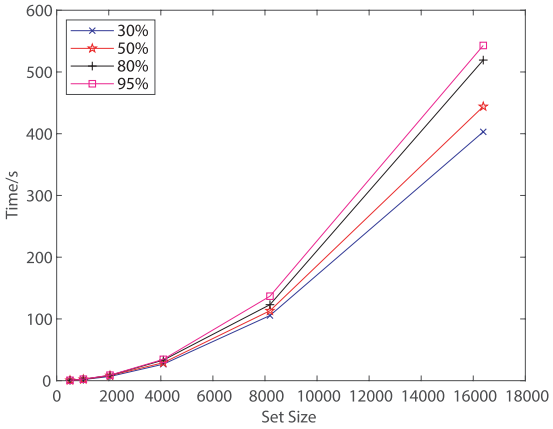

FIGURE 9. The time cost of our protocol.

In order to get the time cost of each stage, we tested the time cost of several stages when the threshold is 80%n and showed our results in Fig 11. As we can see in the figure, the more time cost phases are the query intersection stage and the reconstruction stage, followed by the stage of construction of GBF. In addition, we also give the time cost of the offline and online stages. As we can see from Fig 12, the time cost of the online stage is the major factor.

The asymptotic costs comparison of [1], [27], [28], [30], [31] and our protocol is shown in Table 5 for the set size n, security parameter λ, the number of hash functions k, the threshold t and the length of Bloom filter m. [31] is multi-party threshold PSI, and N is the number of parties.

In the existing TPSI protocol, only [1] and [27] were implemented, so we compared our protocol with [1] and [27].

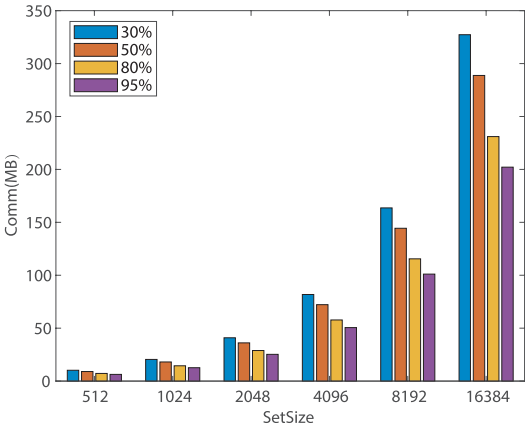

FIGURE 10. The communication of our protocol.

TABLE 5. Asymptotic costs comparison.

Protocol	Comm.	Comp.
[1]	$O(n\lambda)$	$O(n^2)$
[27]	$O(n\lambda log(k+1))$	$\omega(log\lambda)O(n)$
[28]	$O(n\lambda)$	$O(nlog^2n)$
[30]	$\Omega(t)$	$O((n-t)^4)$
[31]	$O(Nt)$	$O((n-t)^4)$
Ours	$O(\lambda m)$	$O(nlogn)$

TABLE 6. Compared time cost with [1] and [15] (On seconds).

Protocol	32	64	128	256	512	1024	2048	4092
[1]	0.02	0.07	0.36	2.08	13.57	96.78	728	5627
[15]	2.48	2.91	3.92	5.59	8.66	15	30.26	62.15
Ours	0.04	0.04	0.08	0.21	0.65	2.32	8.62	30.43

We give a comparison with [1], [15] in Fig 13. It is easy to see that the total time of our protocol is less than [15]. The maximum set size is 4092 and their tests were conducted on a single machine with an Intel i7-4790 @3.6GHz CPU, 16GB RAM. As can be seen from the Fig 13, our protocol is more efficient. We give a comparison between the protocol of [1], [15] and ours in Table 6. The protocol [1] is not efficient since it requires to compute C_n^t possible combinations to reconstruct the secret. By contrast, our protocol avoids calculating all possible combinations among the shares and saves the cost of time.

In [27], their protocol is also based on Bloom filter. In order to ensure the privacy of the cardinality, Zhao *et al.* spent much time in the steps of encrypting Bloom filter, encrypting polynomials and evaluating polynomials. Our protocol uses threshold secret sharing to determine whether to output intersection, which avoids the above expensive operations. Zhao and Chow [27] tested the overhead of set size $n = 100$, threshold $t = 50$, and the length of Bloom filter $m = 4500$. Their experiment was run on a Windows machine with 2 Intel(R) Core(TM) i5-4590 @3.3GHz CPUs and 8GB RAM and their total time cost is 85.278s for P_1 and 139.755s for P_2. In contrast, we tested our experiment on a Windows machine with an Intel(R) Core(TM) i7-8700 @3.2GHz CPU and 8GB RAM and our total time cost is 2.988s. Obviously,

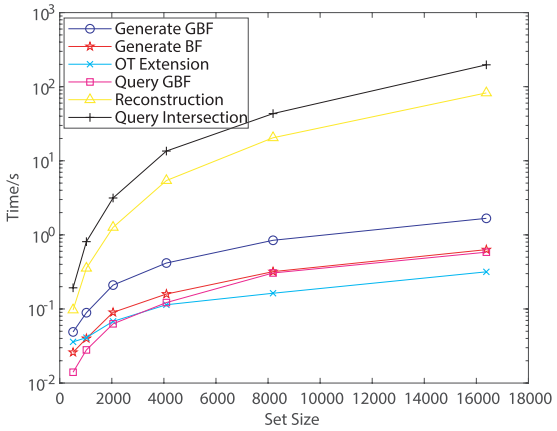

FIGURE 11. The time cost of each stage.

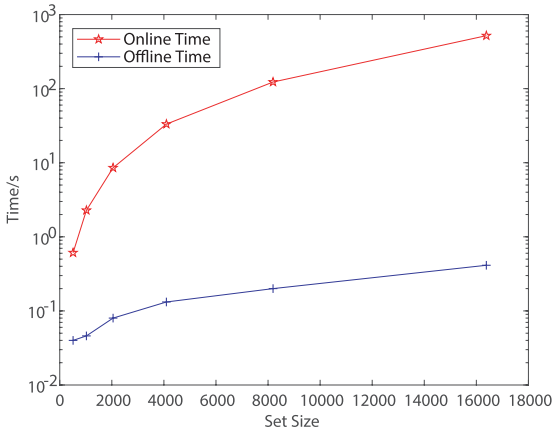

FIGURE 12. Online time and offline time of our protocol.

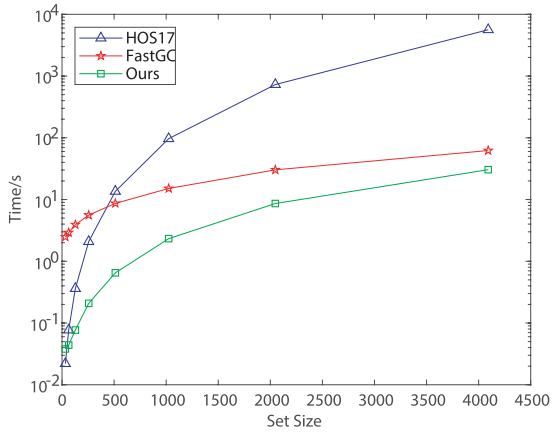

FIGURE 13. Compare with [1] and [15].

our protocol is more efficient and suitable for bigger data.

VI. CONCLUSION

Threshold PSI has a wide range of applications, such as fingerprint matching, online dating and privacy ridesharing. In this work, we design an efficient TPSI protocol with semi-honest security. To improve the efficiency of the TPSI protocol, we design a novel TPSI protocol based on GBF and threshold secret sharing, which uses a small amount of public-key operations. In addition, our protocol combines with Reed-Solomon decoding algorithm to reconstruct the secret, which avoids calculating all possible combinations among the shares. Finally, the performance analysis shows that our protocol is more efficient than the previous TPSI protocols.

REFERENCES

[1] P. Hallgren, C. Orlandi, and A. Sabelfeld, "PrivatePool: Privacy-preserving ridesharing," in *Proc. IEEE 30th Comput. Secur. Found. Symp. (CSF)*. Santa Barbara, CA, USA: IEEE Computer Society, Aug. 2017, pp. 276–291.

[2] A. C.-C. Yao, "Protocols for secure computations (extended abstract)," in *Proc. 23rd Annual Symp. Found. Comput. Sci.*, Nov. 1982, pp. 160–164.

[3] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Advances in Cryptology—EUROCRYPT*, vol. 3027. Berlin, Germany: Springer, 2004.

[4] E. Zhang, F. Li, B. Niu, and Y. Wang, "Server-aided private set intersection based on reputation," *Inf. Sci.*, vol. 387, pp. 180–194, May 2017.

[5] R. Shi, "Efficient quantum protocol for private set intersection cardinality," *IEEE Access*, vol. 6, pp. 73102–73109, 2018.

[6] E. Zhang, F.-H. Liu, Q. Lai, G. Jin, and Y. Li, "Efficient multi-party private set intersection against malicious adversaries," in *Proc. ACM SIGSAC Conf. Cloud Comput. Secur. Workshop (CCSW)*, 2019, pp. 93–104.

[7] O. Ruan, Z. Wang, J. Mi, and M. Zhang, "New approach to set representation and practical private set-intersection protocols," *IEEE Access*, vol. 7, pp. 64897–64906, 2019.

[8] M. J. Freedman, C. Hazay, K. Nissim, and B. Pinkas, "Efficient set intersection with simulation-based security," *J. Cryptol.*, vol. 29, no. 1, pp. 115–155, Jan. 2016.

[9] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient batched oblivious PRF with applications to private set intersection," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 818–829.

[10] B. Pinkas, T. Schneider, and M. Zohner, "Faster private set intersection based on OT extension," in *Proc. 23rd USENIX Secur. Symp.* 2014, pp. 797–812.

[11] S. Lv, J. Ye, S. Yin, X. Cheng, C. Feng, X. Liu, R. Li, Z. Li, Z. Liu, and L. Zhou, "Unbalanced private set intersection cardinality protocol with low communication cost," *Future Gener. Comput. Syst.*, vol. 102, pp. 1054–1061, Jan. 2020.

[12] C. Dong, L. Chen, and Z. Wen, "When private set intersection meets big data: An efficient and scalable protocol," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 789–800.

[13] P. Rindal and M. Rosulek, "Improved private set intersection against malicious adversaries," *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2017, pp. 235–259.

[14] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "Spot-light: Lightweight private set intersection from sparse OT extension," in *Advances in Cryptology—CRYPTO*. Cham, Switzerland: Springer, 2019, pp. 401–431.

[15] Y. Huang, D. Evans, and J. Katz, "Private set intersection: Are garbled circuits better than custom protocols?" in *Proc. 19th Annu. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2012, pp. 1–15.

[16] B. Pinkas, T. Schneider, G. Segev, and M. Zohner, "Phasing: Private set intersection using permutation-based hashing," in *Proc. 24th USENIX Secur. Symp.*, 2015, pp. 515–530.

[17] P. Rindal and M. Rosulek, "Malicious-secure private set intersection via dual execution," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1229–1242.

[18] B. Pinkas, T. Schneider, C. Weinert, and U. Wieder, "Efficient circuit-based PSI via cuckoo hashing," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2018, pp. 125–157.

[19] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "PSI from PaXoS: Fast, malicious private set intersection," in *Advances in Cryptology—EUROCRYPT 2020* (Lecture Notes in Computer Science), vol. 12106, A. Canteaut and Y. Ishai, Eds. Cham, Switzerland: Springer, 2020, pp. 739–767.

[20] F. Kerschbaum, "Collusion-resistant outsourcing of private set intersection," in *Proc. 27th Annu. ACM Symp. Appl. Comput. (SAC)*, 2012, pp. 1451–1456.
[21] F. Kerschbaum, "Outsourced private set intersection using homomorphic encryption," in *Proc. 7th ACM Symp. Inf., Comput. Commun. Secur. (ASIACCS)*, 2012, pp. 85–86.
[22] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-DNF formulas on ciphertexts," in *Proc. Theory Cryptogr. Conf.*, 2005, pp. 325–341.
[23] T. Sander, A. Young, and M. Yung, "Non-interactive cryptocomputing for NC1," in *Proc. 40th Annu. Symp. Found. Comput. Sci.*, 1999, pp. 554–567.
[24] S. Kamara, P. Mohassel, M. Raykova, and S. S. Sadeghian, "Scaling private set intersection to billion-element sets," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, 2014, pp. 195–215.
[25] A. Aydin, T. Sotirios, and D. Changyu, "O-PSI: Delegated private set intersection on outsourced datasets," in *ICT Systems Security and Privacy Protection*. Cham, Switzerland: Springer, 2015, pp. 3–17.
[26] M. Ali, J. Mohajeri, M.-R. Sadeghi, and X. Liu, "Attribute-based fine-grained access control for outscored private set intersection computation," *Inf. Sci.*, vol. 536, pp. 222–243, Oct. 2020.
[27] Y. Zhao and S. S. M. Chow, "Can you find the one for me?" in *Proc. Workshop Privacy Electron. Soc.*, D. Lie, M. Mannan, and A. Johnson, Eds. Toronto, ON, Canada: ACM, 2018, pp. 54–65.
[28] S. Ghosh and T. Nilges, "An algebraic approach to maliciously secure private set intersection," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Cham, Switzerland: Springer, 2019, pp. 154–185.
[29] Y. Zhao and S. S. M. Chow, "Are you the one to share? Secret transfer with access structure," *PoPETs*, vol. 2017, no. 1, pp. 149–169, 2017.
[30] S. Ghosh and M. Simkin, "The communication complexity of threshold private set intersection," in *Advances in Cryptology—CRYPTO 2019*, vol. 11693. Cham, Switzerland: Springer, 2019, pp. 3–29.
[31] S. Badrinarayanan, P. Miao, and P. Rindal, "Multi-party threshold private set intersection with sublinear communication," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 600, May 2020. [Online]. Available: https://eprint.iacr.org/2020/600
[32] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.
[33] G. R. Blakley, "Safeguarding cryptographic keys," in *Proc. AFIPS Nat. Comput. Conf.*, vol. 48, 1979, pp. 313–317.
[34] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, vol. 8, no. 2, pp. 300–304, Jun. 1960.
[35] B. Elwyn and W. Lloyd, "Error correction for algebraic block codes," WO Patent DE3 479 688, Oct. 1989. [Online]. Available: https://www.freepatentsonline.com/DE3479688.html
[36] S. B. Wicker and V. K. Bhargava, "Reed–Solomon codes in frequency-hop communications," in *Reed-Solomon Codes and Their Applications*. Piscataway, NJ, USA: IEEE Press, 1994, pp. 150–174, doi: 10.1109/9780470546345.ch8.
[37] S. Gao, "A new algorithm for decoding Reed–Solomon codes," in *Communications, Information and Network Security* (The Springer International Series in Engineering and Computer Science: Communications and Information Theory), vol. 712, V. K. Bhargava, H. V. Poor, V. Tarokh, and S. Yoon, Eds. Boston, MA, USA: Springer, 2003, pp. 55–68.
[38] J. Kilian, "Founding cryptography on oblivious transfer," in *Proc. 20th Annu. ACM Symp. Theory Comput.*, 1988, pp. 20–31.
[39] O. Goldreich and R. Vainish, "How to solve any protocol problem-an efficiency improvement," in *Proc. Conf. Theory Appl. Cryptograph. Techn.*, 1987, pp. 73–86.
[40] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *Proc. Annu. Int. Cryptol. Conf.*, 2003, pp. 145–161.
[41] M. Keller, E. Orsini, and P. Scholl, "Actively secure OT extension with optimal overhead," in *Proc. Annu. Cryptol. Conf.*, 2015, pp. 724–741.
[42] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.
[43] O. Goldreich, *The Foundations of Cryptography: Basic Applications*, vol. 2. Cambridge, U.K.: Cambridge Univ. Press, 2004. [Online]. Available: http://www.wisdom.weizmann.ac.il/%7Eoded/foc-vol2.html, doi: 10.1017/CBO9780511721656.
[44] Y. Lindell, "How to simulate it—A tutorial on the simulation proof technique," in *Tutorials on the Foundations of Cryptography*. Cham, Switzerland: Springer, 2017, pp. 277–346.

EN ZHANG received the Ph.D. degree from the Beijing University of Technology, in 2013. From 2014 to 2016, he worked as a Postdoctoral Researcher with the Institute of Information Engineering, Chinese Academy of Sciences. He is currently an Associate Professor with Henan Normal University. His research interests include cryptography and information security.

JIAN CHANG was born in Hebei, Henan, China, in 1996. She received the bachelor's degree in computer and information engineering from Anyang Normal University, in 2018. She is currently pursuing the master's degree in computer and information engineering with Henan Normal University. Her research interests include cryptography and information security.

YU LI was born in Anyang, Henan, China, in 1998. She received the bachelor's degree in computer and information engineering from Henan Normal University, in 2020. Her research interests include cryptography and information security.