Hindawi
Security and Communication Networks
Volume 2021, Article ID 3159269, 11 pages
https://doi.org/10.1155/2021/3159269

Research Article

Outsourced Mutual Private Set Intersection Protocol for Edge-Assisted IoT

Jing Zhang,[1] **Rongxia Qin**,[1] **Ruijie Mu**,[1] **Xiaojun Wang**,[2] **and Yongli Tang**[1]

[1]*College of Computer Science and Technology, Henan Polytechnic University, Jiaozuo 454003, China*
[2]*Network Innovations Centre (NIC), Dublin City University, Dublin 9, Ireland*

Correspondence should be addressed to Yongli Tang; yltang@hpu.edu.cn

Received 30 August 2021; Revised 17 October 2021; Accepted 9 December 2021; Published 31 December 2021

Academic Editor: Shifeng Sun

The development of edge computing and Internet of Things technology has brought convenience to our lives, but the sensitive and private data collected are also more vulnerable to attack. Aiming at the data privacy security problem of edge-assisted Internet of Things, an outsourced mutual Private Set Intersection protocol is proposed. The protocol uses the ElGamal threshold encryption algorithm to rerandomize the encrypted elements to ensure all the set elements are calculated in the form of ciphertext. After that, the protocol maps the set elements to the corresponding hash bin under the execution of two hash functions and calculates the intersection in a bin-to-bin manner, reducing the number of comparisons of the set elements. In addition, the introduction of edge servers reduces the computational burden of participating users and achieves the fairness of the protocol. Finally, the IND-CPA security of the protocol is proved, and the performance of the protocol is compared with other relevant schemes. The evaluation results show that this protocol is superior to other related protocols in terms of lower computational overhead.

1. Introduction

The vigorous development of fifth-generation technology (5G), Internet of Things (IoT), edge computing, and other technologies has spawned new medical and life modes such as smart medical treatment [1], smart home [2], and smart bus [3]. Intelligent IoT devices have been widely used in daily life and have brought great changes to people's life. With the assistance of proxy devices and edge servers [4], these data can be outsourced to edge storage for subsequent analysis and use. However, although data outsourcing based on edge computing reduces the storage and computing overhead on the user side, it also exposes the user's sensitive data to the risk of leakage [5]. How to protect the privacy of data stored on edge servers and share data with designated data consumers (such as service and product providers, medical professionals, and educators) has become the focus of research.

Private Set Intersection (PSI), as an efficient encryption technology that allows secret sharing of data information, can ensure the security of the data stored on the edge server when making full use of the data for intersection calculation. So it has become an important research object to solve the problem of edge-assisted Internet of Things data privacy sharing. PSI protocol [6] refers to the intersection of two participants calculating their private sets. In the edge computing environment, two clients encrypt their respective data sets and outsource them to the edge server. Then the edge server effectively performs the intersection operation but cannot know any information in the set. Then one or two clients can obtain the intersection result, while their respective sets remain private. Usually, only one client obtains the intersection result is the one-way PSI protocol [7], which can be applied in situations such as contact tracing of the novel coronavirus COVID-19 [8]. At this time, the client unilaterally obtains the intersection result to determine whether it belongs to the contact. However, one-way PSI protocol cannot guarantee that both clients obtain the intersection result at the same time. Obviously, one-way PSI cannot satisfy both clients in the situation such as profile matching [9], in which both clients want to obtain the intersection results to realize the medical information sharing

between patients. In this case, mutual PSI (mPSI) protocol [10] is the better choice. That is our focus.

Debnath et al. [11] proposed a secure mutual oblivious pseudorandom function (mOPRF) under the malicious model based on composite order group to maintain the fairness of the mPSI protocol and use homomorphic encryption algorithm to protect data privacy. However, since the protocol is constructed based on composite order, the efficiency is lower than that based on prime order. Based on [11], Debnath et al. [12] proposed another mPSI protocol using prime order groups. The protocol also uses homomorphic encryption algorithm to ensure the security of data and uses semihonest offline arbiter to achieve fairness between two participants. However, the complexity of the protocol is still high. The mPSI protocol in [13] is also constructed based on prime order. It uses the multiplicative homomorphic encryption ElGamal and the distributed ElGamal cryptosystem to ensure the security of data and the offline semihonest arbiter to achieve fairness. But in order to ensure the security under the malicious model, the verifiable Cramer-Shoup cryptographic system used makes the protocol more complicated. Overall, although these mPSI protocols achieve data privacy and fairness, they have high computational complexity and low efficiency.

In this paper, we propose an outsourced mPSI (O-mPSI) protocol in the aid of the edge server. The protocol not only protects the data privacy of parties and achieves the fairness of the results obtained by both parties, but also improves the efficiency. The main contributions are as follows:

(1) The O-mPSI protocol improves the method of preprocessing set elements in [14] by increasing the number of elements stored in each hash bin instead of using the stash. It only needs to compare the elements in the two hash tables in a bin-to-bin manner to calculate the intersection, which further reduces the number of comparisons of set elements and decreases the computational cost of the protocol. As a result, the computation and communication overhead of this protocol is lower than the existing mPSI protocol and realizes the efficiency of the O-mPSI protocol.

(2) The protocol adopts ElGamal threshold encryption algorithm to encrypt the elements in the hash table to ensure that the set elements of both parties are compared in the form of ciphertext, so that the users can correctly and safely calculate the intersection and realize the data privacy of the O-mPSI protocol.

(3) The protocol introduces a semihonest edge server as the third party, and the work of set element comparison is transferred to the server, which further reduces the computational burden of users. At the same time, it enables two users to process collection elements in parallel. This ensures that, after the implementation of the agreement, both parties can know the intersection results at the same time and realize the fairness of O-mPSI protocol.

2. Related Work

Since Agrawal et al. [15] proposed the concept of PSI, a series of work has been done to construct the PSI protocol. Ordinarily, they are divided into one-way PSI and mutual PSI.

2.1. One-Way PSI. At present, there are many types of research on the one-way PSI protocol, and the design methods are mainly based on public key encryption, garbled circuit (GC), oblivious transfer (OT), and cloud computing. Based on the design method of public key encryption system, literature [16] uses Fully Homomorphic Encryption (FHE) to construct the PSI protocol. The protocol uses bloom filters (BF) to process data, so that the complexity of the protocol has nothing to do with the size of the client set. Huang and Evans designed a PSI protocol [17] through GC that can resist semihonest adversaries, which proved to be suitable for the intersection calculation of sets of different sizes. Pinkas et al. realized the PSI protocol [18] with linear communication complexity based on GC and oblivious programmable pseudorandom functions (OPPRF). This protocol is superior to previous circuit-based PSI protocols in terms of efficiency. Literature [14] constructed oblivious pseudorandom function (OPRF) based on OT extension and then proposed a PSI protocol combined with OPRF and hash algorithm, and the security is proved under the semihonest opponent model. Kavousi et al. proposed a PSI protocol in [19], which takes the OPRF and the garbled BF as its main components, avoiding costly operations of computation and having high scalability. The protocol in [20] allows users to store their private data sets on cloud server and also entrust computing of the intersection to the server and use homomorphic encryption (HE) and oblivious polynomial evaluation (OPE) to process the data. It greatly reduces the workload of users and improves the computational efficiency of the protocol. The PSI protocol designed by Abadi et al. [21] also uses cloud storage of private data sets. It is mainly constructed by hash table and OPE without any public key encryption operation. However, all parties need to establish a secure channel in advance; otherwise it is easy for an attacker to eavesdrop on the communication between honest parties. Literature [22] proposed an improved PSI scheme based on [21]. There is no need for any secure channel in the scheme, which is superior to the scheme in [21] in terms of confidentiality and complexity.

2.2. Mutual PSI. The research on mutual PSI protocol began in 2005; Kissner and Song et al. proposed the first mPSI protocol [23], which is based on the mathematical properties of OPE and uses HE to calculate PSI on ciphertexts. The fairness of the protocol depends on the fairness of the threshold encryption scheme used by the protocol. Camenisch and Zaverucha proposed another mPSI authentication set protocol [24] based on Camenisch-Lysyanskaya signature (CLS) and OPE. The disadvantage of this protocol is that its computational overhead is quadratic and the computational complexity is relatively high. Fairness of the protocol in [24] is realized by a fair exchange scheme. However, if the

input is not authenticated, it usually does not work. Kim et al. coupled the prime representation (PR) technology with threshold additive HE and realized an mPSI protocol [25] with linear complexity for the first time in the semihonest security model and through the nature of threshold decryption to achieve the fairness of the protocol. Dong and Chen et al. proposed a fair mPSI protocol [26] with a semihonest arbitrator based on HE and OPE. The arbitrator in the protocol can handle conflicts and cannot know the user's private input and output. The mPSI protocol in [11] is constructed by HE and mutual oblivious pseudorandom function (mOPRF). The fairness of the mOPRF ensures the fairness of the protocol. And in the standard model, it has been proven to resist the attack of malicious adversaries. In [12], an mPSI protocol with linear communication and computational overhead is constructed by using prime order group. The protocol uses a distributed ElGamal encryption algorithm and an offline semitrusted third party to achieve the fairness of the protocol and the security under the malicious adversary model. The mPSI protocol in [13] uses multiplicative HE and a distributed ElGamal cryptosystem to protect data privacy and uses an offline semihonest arbiter to achieve fairness. Under the decisional Diffie-Herman hypothesis, it is proved that the protocol is safe under the malicious adversary model.

We show the differences between these protocols in Table 1.

3. Preliminaries

3.1. Decisional Diffie-Hellman Assumption

Definition 1. Let G be a cyclic group of prime order p, g is the generator of G, and $a, b \leftarrow_R \mathbb{Z}_p$. For the following two four-tuple distributions: $R = (g, g^a, g^b, g^c) \in G^4$, $D = (g, g^a, g^b, g^{ab}) \in G^4$, and for any probability polynomial time (PPT) adversary A when distinguishing between R and D, its advantage $\mathrm{Adv}_{\mathscr{A}}^{\mathrm{DDH}}(\lambda) = |\Pr[\mathscr{A}(R) = 1] - \Pr[\mathscr{A}(D) = 1]|$ is negligible with security parameter λ, where R is a random four-tuple and D is a Diffie-Hellman four-tuple.

3.2. Security Model

3.2.1. Adversary Model. We consider the environment where a PPT adversary A exists; the definition and model follow [27]. In this setting, the adversary A can eavesdrop on the messages transmitted through the communication channel and allow A to corrupt participant to obtain private key. The capabilities of the adversary A are shown in Table 2.

3.2.2. Formal Security Model. We mainly describe the formal security model of the O-mPSI protocol in this section. It is based on the model in [28] and we made some modifications according to the specific requirements of the O-mPSI protocol.

Initialization. In a two-party set intersection scheme O-mPSI, three entities are involved, that is, two participants $P_1, P_2 \in$ **User** and an edge server $S \in$ **Server**. Each of them may have several instances called oracle, which involve different, concurrent executions of O-mPSI. We denote user instances as $P_1^i, P_2^i (i \in \mathbb{Z})$, server instances as S^i, and any kind of instance as $U \in$ **User** $\cup$ **Server**. In addition, P_1 holds key pair (pk_1, sk_1), P_2 has key pair (pk_2, sk_2), and pk is the system public key.

Queries. The adversary A only interacts with the participants of the protocol through oracle queries, which simulates the capabilities of adversary A in a real attack. And all possible oracle queries are shown below:

(1) Execute (P_1^i, P_2^i, S^i): The adversary's eavesdropping attack is simulated in this oracle query (see C1 in Table 2), and the messages exchanged during the actual implementation of the O-mPSI protocol are returned.

(2) Send (U, m): A sends a message to instance U in this query; then the response generated by U processing m based on O-mPSI protocol is returned.

(3) Corrupt (U, a): The corruption ability of the adversary is simulated in this query (see C2 in Table 2). A can only corrupt one of the two users, not both.

If $a = 1$, it returns the private key sk_1 and data set X of P_1.

If $a = 2$, it returns the private key sk_2 and data set Y of P_2.

(4) Collude (S): The collusion ability of adversary A is simulated in this query (i.e., C3 in Table 2). Any information stored on the server is returned.

(5) Reveal (U): The system private key shared by the instances P_1^i and P_2^i is returned to the adversary in this query. But instances P_1^i, P_2^i and their partner S^i must not have been queried by corrupt-query and test-query; otherwise it returns $\perp$.

(6) Challenge (U, m_0, m_1): In this query, A selects two messages m_0 and m_1 of equal length and sends them to instance U. The instance U selects $b \leftarrow_R \{0, 1\}$ randomly, then encrypts m_b, and returns the ciphertext c. Among them, P_1^i, P_2^i, and S^i in U have not been inquired by corrupt-query and reveal-query; otherwise it returns $\perp$. And this query is called only once during execution.

(7) Test (U): After querying the oracle, if the guessed bit $b' = b$, return 1; otherwise return 0. This query can only point to the instance in the challenge-query and is called only once during execution.

IND-CPA security. During the execution of the O-mPSI protocol, A can require polynomial degree to execute, send, corrupt, collude, and reveal queries. A can also send a challenge-query and a test-query to an instance that has not been queried. At the end of the game execution, for the bit b in the challenge-query, A outputs a guess bit b' in the test-query. If $b' = b$, it means that A wins and it is recorded as Succ. The advantage that the adversary A can destroy the IND-CPA security of O-mPSI protocol is defined as

Table 1: Difference between the mentioned protocols.

Type	Reference	Structure	Adversary model	Fairness
Public key	[12]	HE	Malicious	Yes
	[13]	HE	Malicious	Yes
	[16]	FHE + BF	Semihonest	No
	[23]	HE + OPE	Malicious	Yes
	[24]	HE + OPE + CLS	Malicious	Yes
	[25]	HE + PR	Semihonest	Yes
	[26]	HE + OPE	Malicious	Yes
Garbled circuit	[17]	GC	Semihonest	No
	[18]	GC + OPPRF	Semihonest	No
Oblivious transfer	[11]	mOPRF + HE	Malicious	Yes
	[14]	hash + OPRF	Semihonest	No
	[19]	OPRF + garbled BF	Semihonest	No
Cloud computing	[20]	HE + OPE	Semihonest	No
	[21]	Hash + OPE	Semihonest	No
	[22]	Hash + OPE	Semihonest	No

Table 2: Capabilities of the adversary.

Types	Descriptions
C1	A can learn the messages transmitted in the communication channel.
C2	A can corrupt one of the two participants to obtain its secret data and private key.
C3	A can collude with dishonest edge server S to obtain the information stored in the server.

$$\mathrm{Adv}_{O-m\mathrm{PSI}}^{\mathrm{IND-CPA}}(\mathscr{A}) = \Pr[\mathrm{Succ}(\mathscr{A})] - \frac{1}{2} = \Pr[b' = b] - \frac{1}{2}. \quad (1)$$

Definition 2. For any PPT adversary A, if there is a negligible function $\varepsilon(\cdot)$ such that $\mathrm{Adv}_{O-m\mathrm{PSI}}^{\mathrm{IND-CPA}}(\mathscr{A})\Pr[\mathrm{Succ}(\mathscr{A})] - 1/2 \le \varepsilon(\lambda)$, then the O-mPSI protocol is IND-CPA secure.

3.3. Hash Algorithm. Cuckoo hashing. Cuckoo hashing [29] is a method to solve hash conflict, which is widely used in PSI protocols. In this hashing technique, two hash functions h_1, h_2: $\{0,1\}^* \longrightarrow \{1,\ldots,m\}$ are used to map n elements into m bins, where each bin has at most b elements. When storing an element x in the cuckoo hash table, calculate the two bin positions $h_1(x)$ and $h_2(x)$ corresponding to x. If both bins are not full, insert x into either of them; if one of the two bins is full and the other is not full, then insert x into the bin that is not full; if both bins are full, then randomly select a position to kick out one of the elements y and then insert x into it; the kicked y is reinserted into the cuckoo table using the same algorithm. This process is called relocation, and the relocation process is executed recursively until all elements are stored in the cuckoo hash table.

Simple hashing. This hashing technique is similar to cuckoo hashing. Two hash functions h_1, h_2: $\{0,1\}^* \longrightarrow \{1,\ldots,m\}$ are used to map n elements into m bins, and each bin also has at most b elements. However, in simple hash mapping, when the element x is mapped to its corresponding two positions $h_1(x)$ and $h_2(x)$, the elements are stored in both $h_1(x)$ and $h_2(x)$.

3.4. ElGamal Threshold Encryption Algorithm. The ElGamal threshold encryption algorithm [30] is implemented according to the additive homomorphism of the ElGamal encryption algorithm, which is composed of *KeyGen, Encrypt, Decrypt,* and *Rerandomize* four algorithms. The specific algorithm is as follows:

KeyGen. Given a cyclic group G of order p and its generator g, the security parameter is λ. Parties P_1 and P_2 randomly select $sk_i \leftarrow Z_p (i = 1, 2)$ and then compute $pk_i = g^{sk_i} \bmod p$, where sk_i is their respective private key, and pk_i is their respective public key. Then the public key of the threshold encryption scheme is $pk = pk_1 \cdot pk_2$.

Encrypt. For a message m, map m to $H(m)$ using the hash function $H(\cdot)$, where $H(\cdot)$ is defined as $\{0,1\}^* \longrightarrow G$. Then select a random number r_1, and compute the ciphertext of the message m as $ct = (g^m \cdot pk^{r_1} \bmod p)$.

Decrypt. For a ciphertext ct, P_1 half-decrypts it as $ct' = (g^m \cdot pk^{r_1}/(g^{r_1})^{sk_1} \bmod p)$; then P_2 fully decrypts ct' as $g^m = ct'/(g^{r_1})^{sk_2} \bmod p$.

Rerandomize. For the given ciphertext ct, select a random number r_2 to rerandomize it to $ct^* = (g^m \cdot pk^{r_1} \cdot pk^{r_2}) = (g^m \cdot pk^{r_1+r_2})$.

4. Outsourced Mutual PSI Protocol

4.1. Overview. The system model of O-mPSI protocol is shown in Figure 1. There are three entities involved, two participating users P_1, P_2 and a semihonest edge server S. Among them, P_1 has private data set $X = \{x_1, \ldots x_{n_1}\}$ and user P_2 has private data set $Y = \{y_1, \ldots, y_{n_2}\}$. They hope to calculate the intersection $X \cap Y$ of X and Y through the server S without revealing their own set information.

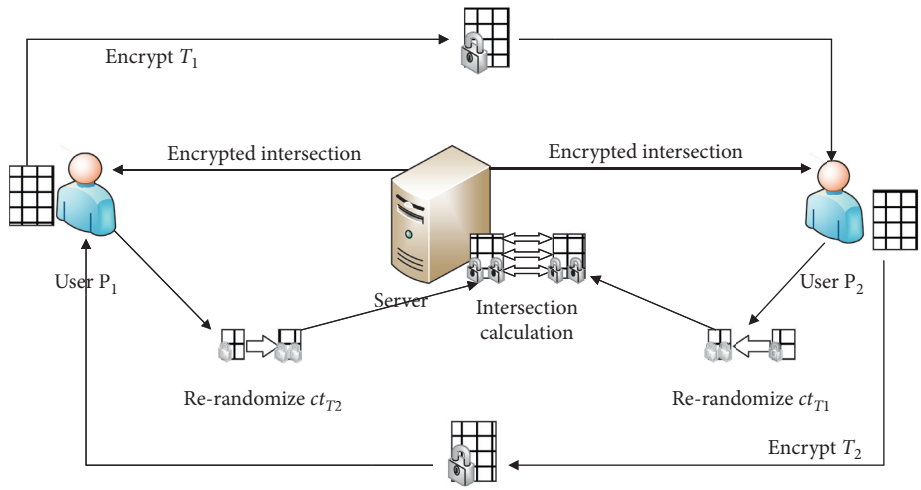

Figure 1: System model of O-mPSI protocol.

The design idea of our protocol is based on [14], where P_1 and P_2 select three hash functions $\{h_1, h_2, h_3\}$: $\{0,1\}^* \longrightarrow \{1,\ldots,m\}$ to generate three hash bin positions corresponding to the set elements in the hash table. P_1 uses the cuckoo hashing to select one of the three bins to map each element in the set X to hash table T_1, each bin stores one element, and the remaining elements are stored in the stash ST. P_2 uses simple hashing to map each element in set Y to the corresponding three bins in hash table T_2, and each bin stores b elements. Then, the elements in the hash bin of T_1 are compared with the elements in the corresponding hash bin of T_2. And each element in the stash ST is compared to all elements in set Y. In this way, the comparison times of elements are reduced from $O(n^2)$ to $O(n)$, and the intersection $X \cap Y$ is all the equal elements obtained by comparison.

But we diverge from the protocol of [14] in the method. In our protocol, parties P_1 and P_2 agree on two hash functions h_1, h_2: $\{0,1\}^* \longrightarrow \{1,\ldots,m\}$ to generate the two hash bin positions corresponding to the set elements in the hash table. In this way, the elements in the set Y of P_2 only need to be stored twice, which saves storage space and reduces the subsequent computational burden. Moreover, our protocol sets the size of each bin of hash table T_1 to b. This avoids that the elements cannot be stored in the hash table T_1 due to too many relocations when hash conflicts occur, so there is no need to increase the stash to store the elements. Therefore, it is only necessary to compare the elements in the bin of T_1 with the elements in the corresponding bin of T_2 to obtain the intersection. This further decreases the number of comparisons and reduces the complexity of the protocol.

However, there is still a problem of how to protect the privacy of user set information while calculating the intersection correctly. We use the ElGamal threshold encryption algorithm to solve this problem, as shown in Section 4.2.1. Participant $P_a\,(a = 1,2)$ uses the public key pk to encrypt the elements in the hash table T_a and then sends it to the other party to rerandomize. Combined with the hash algorithm, we can see that, for $\forall x_i \in X$ in the hash bin of T_1, if the same element $y_j \in Y$ in T_2 is stored in the corresponding hash bin, there must be $E^*(x_i) = E^*(y_j)$ in the two hash bins with the same bin number. Therefore, the use of ElGamal threshold encryption algorithm enables the comparison operation of elements to be performed in the form of ciphertext and ensures that the encrypted results of equal elements on both sides are the same. It creates conditions for the smooth implementation of intersection calculation.

In addition, in order to decrease the computational burden of both parties, the element comparison work is handed over to the edge server. Specifically, $P_a\,(a = 1,2)$ sends the hash table encrypted by the ElGamal encryption algorithm to the server. Then the server compares the elements in the two hash tables in a bin-to-bin manner in the form of ciphertext. That is, it compares the elements in the hash bin of T_1 with the elements in the corresponding hash bin of T_2. In this way, all the equal ciphertext elements in the m bins are the intersection $X \cap Y$ in the ciphertext form. Then P_1 and P_2 cooperate to decrypt the intersection in the form of ciphertext in parallel, and the plaintext intersection $X \cap Y$ can be obtained at the same time. Among them, the comparison work of m bins can be performed in parallel, which reduces the time overhead of O-mPSI protocol.

4.2. O-mPSI Protocol. In this section, we introduce the proposed O-mPSI protocol. For easier understanding, we divide the protocol into two parts: the data encryption part and the set intersection computation part, which are introduced in the following two subsections, respectively.

4.2.1. ElGamal Threshold Encryption Protocol. In this section, we describe how ElGamal threshold encryption algorithm works in the protocol, and the details are shown in Algorithm 1.

Remark 1. In the key generation phase, parties P_1 and P_2 jointly generate the public key pk so that both parties can use the same public key for encryption in the subsequent encryption phase.

Inputs:
P_1: The cuckoo hash table T_1
P_2: The hash table T_2
Output: the encrypted cuckoo hash table $ct^*_{T_1}$ and $ct^*_{T_2}$.
Key generation phase
P_1 and P_2 calculate as below:
1. Determine a hash function H: $\{0,1\}^* \longrightarrow G$.
2. Run the *KeyGen* algorithm in ElGamal threshold encryption to generate the key pairs (pk_i, sk_i), $i = 1, 2$.
3. Publish the public keys $pk_i\,(i = 1, 2)$ and keep the private key $sk_i\,(i = 1, 2)$.
4. Each party computes $pk = pk_1 \cdot pk_2$.
Encryption phase
1. P_1 encrypts the cuckoo hash table T_1 by bins, for all $k \in [m]$, P_1 computes as follows:
(1) choose a random number r_1^k, for all items $x_i^k\,(i = 1, \ldots, b)$ in the k$^{\text{th}}$ cuckoo hash table T_1^k, using ElGamal threshold encryption algorithm to encrypt them: $ct_{T_1^k} = E(x_i^k) = (g^{r_1^k}, g^{H(x_i^k)} \cdot pk^{r_1^k})$
(2) sends $ct_{T_1} = (ct_{T_1^1}, \ldots, ct_{T_1^m})$ to P_2 in shuffled order.
2. P_2 encrypts the hash table T_2 by bins, for all $k \in [m]$, P_2 computes as follows:
(1) choose a random number r_2^k, for all items $y_i^k\,(i = 1, \ldots, b)$ in the k$^{\text{th}}$ hash table T_2^k, using ElGamal threshold encryption algorithm to encrypt them: $ct_{T_2^k} = E(y_i^k) = (g^{r_2^k}, g^{H(y_i^k)} \cdot pk^{r_2^k})$
(2) sends $ct_{T_2} = (ct_{T_2^1}, \ldots, ct_{T_2^m})$ to P_1 in shuffled order.
3. P_1 re-randomizes the received ct_{T_2} by bins. For all $k \in [m]$ and $\forall i \in [b]$, P_1 computes as follows: $ct^*_{T_2^k} = E^*(y_i^k) = (g^{r_1^k + r_2^k}, g^{H(y_i^k)} \cdot pk^{r_1^k + r_2^k})$.
Then $ct^*_{T_2} = (ct^*_{T_2^1}, \ldots, ct^*_{T_2^m})$.
4. P_2 re-randomizes the received ct_{T_1} by bins. For all $k \in [m]$ and $\forall i \in [b]$, P_2 computes as follows: $ct^*_{T_1^k} = E^*(x_i^k) = (g^{r_1^k + r_2^k}, g^{H(x_i^k)} \cdot pk^{r_1^k + r_2^k})$
Then $ct^*_{T_1} = (ct^*_{T_1^1}, \ldots, ct^*_{T_1^m})$.

Algorithm 1: ElGamal threshold encryption protocol.

Remark 2. In the encryption phase, after encrypting the hash table, each party $P_i\,(i = 1, 2)$ needs to send ct_{T_i} to the other party for rerandomization. Because each party privately chose a random number for encryption, two equal elements are encrypted differently by the two parties. In addition, to calculate the set intersection correctly, adding the rerandomization operation performed by the other party to make the encryption results of equal elements the same is necessary.

4.2.2. Outsourced Two-Party Set Intersection Protocol. In this section, we will explain the intersection calculation part of the O-mPSI protocol. Detailed description can be seen in Algorithm 2.

Remark 3. In the data storage phase, P_1 hashes all its elements into one of its two corresponding bins, and each element is stored in only one bin. P_2 hashes all its elements into both of its two corresponding bins, and each element is stored in both bins. In this case, suppose that there is $x_i = y_j$ ($x_i \in X$ and $y_j \in Y$); the two bin positions $h_1(x_i)$ and $h_2(x_i)$ corresponding to x are the same as the two bin positions $h_1(y_j)$ and $h_2(y_j)$ corresponding to y. Then no matter which bin($h_1(x_i)$ or $h_2(x_i)$) x is stored in, the corresponding element y can be found in the two bins $h_1(y_j)$ and $h_2(y_j)$. Therefore, it can be known that storing all elements in set Y in both bins can avoid missing intersection elements when computing the set intersection.

Remark 4. In the data encryption phase, parties P_1 and P_2 use the ElGamal threshold encryption algorithm in Section 4.2.1 to encrypt the elements in their hash table and at the same time add a permutation sequence, so that both parties cannot know the specific location of the element. After hash tables T_1 and T_2 undergo the same permutation process, their bin numbers still correspond, so the intersection can be calculated correctly by the bins.

Remark 5. In the intersection calculation phase, if the server continues to calculate on the encrypted intersection according to the addition homomorphism of ElGamal homomorphism encryption algorithm, the encrypted intersection sum can be obtained. Then continuing step 2, the plaintext intersection sum will be obtained by both P_1 and P_2.

5. Security Analysis

The security proof of the O-mPSI protocol based on decisional Diffie-Hellman (DDH) assumption is shown in this section.

Theorem 1. *Let G be a represented group of order p. Let A be a PPT adversary against the IND-CPA security within a time limit t, and A can send q_{send} send-queries, q_{exe} execute-queries, and q_h random oracle queries at most. Then we can get*

Inputs:
P_1: Set $X = \{x_1, \ldots, x_{n_1}\}$
P_2: Set $Y = \{y_1, \ldots, y_{n_2}\}$
Output: The intersection set I, where $I = X \cap Y$.
Data storage phase:
1. P_1 and P_2 determine the number m of bins, the size b of bins and the two hash functions h_1, h_2: $\{0,1\}^* \longrightarrow \{1, \ldots, m\}$.
2. For $x_i \in X, i \in [n_1]$, P_1 computes the two bin positions $h_1(x_i)$ and $h_2(x_i)$ corresponding to x_i, and inserts x_i into one of them. P_1 fills the bin with less than b elements with dummy elements, then generates the cuckoo hash table T_1.
3. For $y_j \in Y, j \in [n_2]$, P_2 computes the bin positions $h_1(y_j)$ and $h_2(y_j)$ corresponding to y_j, and inserts y_j into both of them. P_2 fills the bin with less than b elements with dummy elements, then generates the hash table T_2.
Data encryption phase:
1. P_1 calls the ElGamal threshold encryption protocol to compute the encrypted hash table $ct^*_{T_2}$, and then sends $ct^*_{T_2}$ to the server in shuffled order.
2. P_2 calls the ElGamal threshold encryption protocol to compute the encrypted cuckoo hash table $ct^*_{T_1}$, and then sends $ct^*_{T_1}$ to the server in shuffled order.
Intersection calculation phase:
1. After the $ct^*_{T_1}$ and $ct^*_{T_2}$ are received, the server proceeds as follows:
(1) computes the encrypted set intersection by bins: for $k \in [m]$, computes $ct_{I^k} = ct^*_{T_1^k} \cap ct^*_{T_2^k}$.
(2) then the final encrypted intersection $ct_I = ct_{I^1} \cup ct_{I^2} \cup \cdots \cup ct_{I^m}$.
(3) publishing $ct_{I=X\cap Y}$..
2. After receiving ct_I, the users P_1, P_2 and the server proceed as follows:
(1) each party $P_i (i = 1, 2)$ half-decrypts ct_I to ct_{I_i}' using its private key, and then sends ct_{I_i}' to the server.
(2) then the server sends ct_{I_2}' to P_1 and sends ct_{I_1}' to P_2.
Each party fully decrypts the received other party half-decrypted intersection, and gets the plaintext intersection set $I = \{z_1, \ldots, z_w\}$.

Algorithm 2: Outsourced two-party set intersection protocol.

$$\mathrm{Adv}^{\mathrm{IND-CPA}}_{O-m\mathrm{PSI}}(\mathscr{A}) \le q_h \mathrm{Adv}^{\mathrm{DDH}}_{\mathscr{A}}\left(\left(t + \frac{1/3 \cdot q_{\mathrm{send}} + q_{\mathrm{exe}} + 1}{p^2}\right) + \frac{q_h^2 + \left(1/3 \cdot q_{\mathrm{send}} + q_{\mathrm{exe}}\right)^2}{2p}\right). \tag{2}$$

Proof. Let A be the adversary against the IND-CPA security of O-mPSI protocol. Then construct PPT adversaries to attack the DDH assumption through A. If A can break the IND-CPA security, then at least one PPT adversary has successfully broken the DDH assumption. We utilize hybrid games to prove Theorem 1. The game starts from the real attack and ends when the adversary does not have any advantage. An event Succ_n is defined for each game $G_n (0 \le n \le 4)$, which indicates that A guesses the bit b in the test-query correctly.

(1) Game G_0: The game G_0 corresponds to a real attack in the random oracle model. According to Definition 2, we can get

$$\mathrm{Adv}^{\mathrm{IND-CPA}}_{O-m\mathrm{PSI}}(\mathscr{A}) = \Pr[\mathrm{Succ}_0] - \frac{1}{2}. \tag{3}$$

(2) Game G_1: We simulate the random oracle H: $\{0,1\}^* \longrightarrow G$ (and there is also a random oracle H' that will appear in G_3) in G_1 by maintaining hash list L (and L') as usual. Besides, Send, Execute, Corrupt, Collude, Reveal, Challenge, and Test oracles will be simulated as in the real attack (see Table 3). It is easy to know that G_1 is completely indistinguishable from the real attack, so that

$$\left|\Pr[\mathrm{Succ}_1] - \Pr[\mathrm{Succ}_0]\right| = 0. \tag{4}$$

(3) Game G_2: Like G_1, we simulate all oracles in G_2, except for games where some collisions occur: $H(x)$ and $H(y)$. Because x or y is simulated, they are randomly and uniformly selected. Therefore, from the birthday paradox, we know that

$$\left|\Pr[\mathrm{Succ}_2] - \Pr[\mathrm{Succ}_1]\right| \le \frac{q_h^2 + \left(1/3 \cdot q_{\mathrm{send}} + q_{\mathrm{exe}}\right)^2}{2p}. \tag{5}$$

(4) Game G_3: In G_3, we use the private oracles H' instead of the oracle H to calculate $s(ct_{T_1})$ and $s(ct_{T_2})$, which are completely independent of ct_{T_1} and ct_{T_2}. The games G_3 and G_2 are indistinguishable unless the event $\mathrm{Ask}H_3$ occurs: A queries the hash function H for ct_{T_1} or ct_{T_2}. In addition, no matter what bit b in challenge-query is, the answer is random. Therefore, it can be see that

$$\begin{aligned} \left|\Pr[\mathrm{Succ}_3] - \Pr[\mathrm{Succ}_2]\right| &\le \Pr[\mathrm{Ask}H_3] \\ \Pr[\mathrm{Succ}_3] &= \frac{1}{2}. \end{aligned} \tag{6}$$

(5) Game G_4: In G_4 , we simulate the executions through the random self-reducibility of the Diffie-Hellman problem, and given one DDH instance $(A = g^x, B = g^y)$, we randomly select $\alpha, \beta \in \mathbb{Z}_p$ and

Table 3: Simulation of the queries in O-mPSI protocol.

Number	Queries
(1)	Hash $H(m)$ (or $H'(m)$): If there is a record (m, r) existing in list L, return r. Otherwise, select a random number $r \in G$ to return, and add the record (m, r) to the list L (or list L').
(2)	Send (P^i, a, start): For each of the m hash bins, if $a = 1$, choose a random number $r_1^k (k \in [m])$, compute $E(x^k) = (g^{H(x^k)} \cdot pk^{r_1^k})$, and then return $ct_{T_1} = (ct_{T_1^1}, \ldots, ct_{T_1^m})$; if $a = 2$, choose a random number r_2^k, compute $E(y^k) = (g^{H(y^k)} \cdot pk^{r_2^k})$, and then return $ct_{T_2} = (ct_{T_2^1}, \ldots, ct_{T_2^m})$.
(3)	Send (P^i, a, ct_{T_a}): For $k \in [m]$, if $a = 1$, compute $E^*(y^k) = (g^{H(y^k)} \cdot pk^{r_1^k + r_2^k})$, and then return $ct^*_{T_2} = (ct^*_{T_2^1}, \ldots, ct^*_{T_2^m})$; if $a = 2$, compute $E^*(x^k) = (g^{H(x^k)} \cdot pk^{r_1^k + r_2^k})$, and then return $ct^*_{T_1} = (ct^*_{T_1^1}, \ldots, ct^*_{T_1^m})$.
(4)	Send $(S^i, (ct^*_{T_1}, ct^*_{T_2}))$: For each of the m hash bins, compute $ct_{I^k} = ct^*_{T_1^k} \cap ct^*_{T_2^k}$ and $ct_I = ct_{I^1} \cup ct_{I^2} \cup \cdots \cup ct_{I^m}$. Then return ct_I.
(5)	Execute (P_1^i, P_2^i, S^i): According to the successive simulations of the send-queries, return $ct_{T_1} \leftarrow$ send-query (P_1^i, start), $ct_{T_2} \leftarrow$ send-query (P_2^i, start), $ct^*_{T_2} \leftarrow$ send-query (P_1^i, ct_{T_2}), $ct^*_{T_1} \leftarrow$ send-query (P_2^i, ct_{T_1}), and $ct_I \leftarrow$ send-query $(S^i, (ct^*_{T_1}, ct^*_{T_2}))$.
(6)	Corrupt (U^i, a): Return sk_1 and X if $a = 1$; return sk_2 and Y if $a = 2$.
(7)	Collude (S^i): Return $ct^*_{T_1}$, $ct^*_{T_2}$, and ct_I of S^i.
(8)	Reveal (U^i): Compute $sk = sk_1 + sk_2$ and return the private key sk of P_1^i and P_2^i.
(9)	Challenge (U^i, m_0, m_1): The instance U^i randomly selects $b \leftarrow_R \{0, 1\}$, then computes $E(m_b) = (g^r, g^{H(m_b)} \cdot pk^r)$, and returns the $E(m_b)$.
(10)	Test-query (U^i): According to the $E(m_b)$ obtained from challenge-query, A outputs a guess bit b'. If $b' = b$, return 1; otherwise return 0.

let $E(x) = A^\alpha$, $E(y) = B^\beta$, so we can get a quadruple $\text{DDH}(g, A, B, \text{DDH}(E(x), E(y)))$. Then we have

$$\text{DDH}(E(x), E(y)) = \text{DDH}\left(A^\alpha, B^\beta\right) = \text{DDH}(A, B)^{\alpha\beta}. \tag{7}$$

Moreover, $\text{Ask}H_4$ means that the adversary A had queried the random oracle H on $E(x)$ or $E(y)$. Then we get

$$\begin{aligned} &\Pr\left[\text{Ask}H_3\right] = \Pr\left[\text{Ask}H_4\right] \\ &\Pr\left[\text{Ask}H_4\right] \le q_h \text{Adv}_{\mathcal{A}}^{\text{DDH}}\left(t + \frac{1/3 \cdot q_{\text{send}} + q_{\text{exe}} + 1}{p^2}\right). \end{aligned} \tag{8}$$

According to the above equations, we can conclude that

$$\text{Adv}_{O-m\text{PSI}}^{\text{IND-CPA}}(\mathcal{A}) \le q_h \text{Adv}_{\mathcal{A}}^{\text{DDH}}\left(\left(t + \frac{1/3 \cdot q_{\text{send}} + q_{\text{exe}} + 1}{p^2}\right) + \frac{q_h^2 + (1/3 \cdot q_{\text{send}} + q_{\text{exe}})^2}{2p}\right). \tag{9}$$

The simulation queries involved in the protocol are as follows. □

6. Performance Evaluation

6.1. Theoretical Evaluation. In this section, regarding the complexity of calculation and communication, we compare the O-mPSI protocol with the protocols in [11, 12, 20]. We choose these three protocols because both parties of the protocols in [11, 12] can know the intersection, and the protocol in [20] supports outsourcing. These protocols are very similar to our protocol. The comparison results are shown in Table 4.

Computation complexity. Our O-mPSI protocol uses a cuckoo hash table to store data, and the ElGamal threshold encryption algorithm to encrypt data. We use the number of modular exponential operations to evaluate the computational overhead of O-mPSI protocol. Party P_1 and party P_2 each performs $2m + mb$ exponential operations when encrypting the hash tables T_1 and T_2 and performs w exponential operations in the decryption phase. The server only performs the intersection calculation which does not involve any exponential operations, where m represents how many bins are in the hash table, b is the size of the bin, and w is the intersection-cardinality. Therefore, the protocol O-mPSI has performed $2(2m + mb + w)$ modular exponential operations in total. We define $n = n_1 \gg n_2$, $b = 4$, $mb = 1.5n$, where $n_i (i = 1, 2)$ is the cardinality of the private set of party P_i. The possible values of intersection w are in the range $[0, n]$, where we take its maximum value n. Therefore, the computational complexity is $6.5n$ in the O-mPSI protocol.

Table 4: Comparison of the properties of O-mPSI protocol and related protocols.

Protocol	Comp. Cost		Comm. Cost	Fairness
	User	Server		
Debnath's mPSI1 [11]	$48n$	—	$103n$	Yes
Debnath's mPSI2 [12]	$23n + 7$	—	$12n + 4$	Yes
Abadi's PSI [20]	$10n + 15$	$10n + 15$	$16n + 24$	No
Our O-mPSI	$6.5n$	0	$6n$	Yes

The protocol in [11] is based on the two-way oblivious pseudorandom function mOPRF, without the participation of a third-party server, and its computation complexity is $48n$. The protocol in [12] also does not involve the third-party server; its computation complexity is $23n + 7$. In the verifiable delegated PSI protocol of [20], the user performs $10n + 15$ exponential operations and the server performs $10n + 15$ exponential operations, so the overall computation complexity of [20] is $20n + 30$.

From the analysis above, it is clear that our O-mPSI protocol has much lower computational complexity than the other three protocols.

Communication complexity. Our O-mPSI protocol uses the number of transmitted ciphertexts to express the complexity of communication. Party P_1 sends mb encrypted elements $E(x_i^k)$ and mb rerandomized elements $E^*(y_i^k)$, for $1 \le i \le b$ and $1 \le k \le m$, to P_2. Party P_2 sends mb encrypted elements $E(y_i^k)$ and mb rerandomized elements $E^*(x_i^k)$, for $1 \le i \le b$ and $1 \le k \le m$, to P_1. Thus, the O-mPSI protocol generates a total of $4mb$ ciphertexts transmissions. The communication complexity of the protocol O-mPSI is $6n$, due to $mb = 1.5n$.

In [11], the protocol generates $103n$ ciphertexts interactions, the protocol in [12] generates $12n + 4$ ciphertexts interactions, and the protocol in [20] generates $16n + 24$ ciphertexts interactions.

To sum up, we can see that our O-mPSI protocol is superior to the other three protocols in terms of the communication complexity.

6.2. Experimental Evaluation. To verify the theoretical analysis results in Section 6.1, the computational costs of our O-mPSI protocol and the protocols in [11, 12, 20] are compared through experiments. The experimental platform is Windows 10, AMD Ryzen 5 4600H with Radeon Graphics 3.00 GHz, 16 GB RAM, and the compilation environment is MyEclipse 2017. In the experiment, we set $b = 4$ and $mb = 1.5n$, where m represents how many bins are in the hash table, b is the size of the bin, and n is the set cardinality.

Since our O-mPSI protocol and the protocols in [11, 12, 20] all use homomorphic encryption algorithm to encrypt data, we first compare the time it takes for the four protocols to execute with different modulus lengths. In this experiment, we set $n = 1000$; for modulus of 128 bit, 256 bit, 512 bit, and 1024 bit, the different homomorphic encryption times of the 4 protocols are shown in Figure 2.

It can be seen from Figure 2, with the increase of modulus length, the time of homomorphic encryption performed by the four protocols also increases. Among them, the time cost

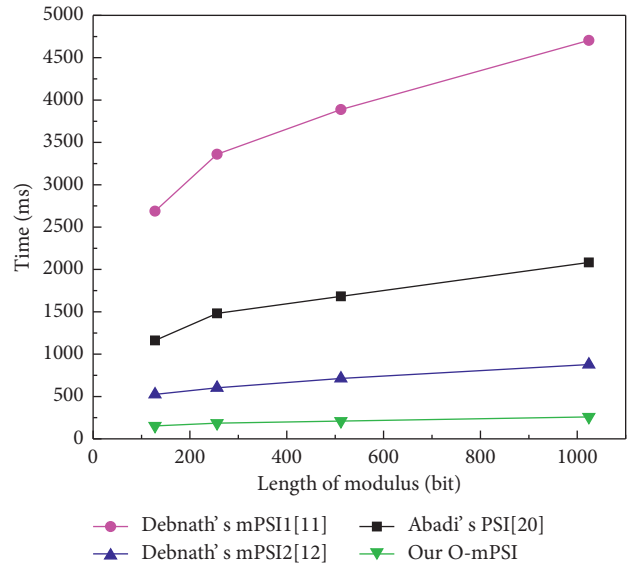

Figure 2: Time to perform homomorphic encryption of different modulus length.

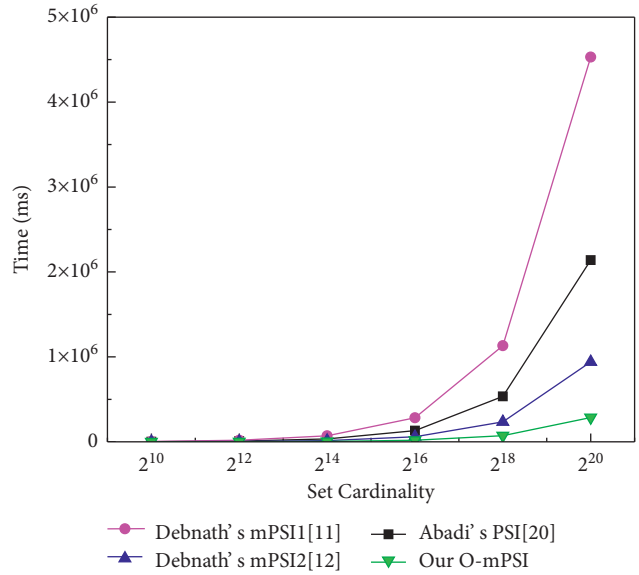

Figure 3: Running time at different set cardinality.

of the O-mPSI protocol is lower than that of Debnath's mPSI1, mPSI2 protocol, and Abadi's PSI protocol, and the growth trend is the slowest. The time cost in this experiment depends on how many modular exponential operations are used in the scheme. And according to Table 3, the modular

exponential operation is least used in the calculation of the O-mPSI protocol. Therefore, the increase of modulus length has the least impact on O-mPSI protocol.

We further compare the time it takes for the four protocols to execute at different set cardinalities. In this experiment, we fixed the modulus length to 1024 bit; for the cardinality n of $2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, 2^{20}$, the running time of the four protocols can be seen in Figure 3.

It can be seen from Figure 3 that the execution time of the four protocols increases with the increasing size of the data set. Among them, the execution time overhead of the O-mPSI increases most slowly compared to that of Debnath's mPSI1, mPSI2 protocol, and Abadi's PSI protocol. This is because the O-mPSI protocol in this paper uses hash algorithm to process the set elements in advance. Therefore, as the size of the data set continues to increase, the time cost curve of O-mPSI protocol has grown slowly, while the time cost curve of the other three protocols has increased significantly.

7. Conclusions

To address the problem of data privacy and security sharing in edge-assisted IoT, we proposed a fair outsourced mPSI protocol with a lower computational cost. The proposed O-mPSI scheme uses the existing hash bin-to-bin method to calculate the intersection and improves it to further reduce the number of element comparisons. The calculation of intersection is outsourced to the edge server, which reduces the computing burden on both sides. The scheme adopts ElGamal threshold encryption algorithm to ensure data security. Only when both parties cooperate can all ciphertexts be completely decrypted. Therefore, this scheme can effectively resist the collusive attack between the server and any one of the two parties. And it proves that this scheme is IND-CPA secure under the DDH assumption. Through theoretical analysis and experimental evaluation, it is shown that the computational cost of our proposed O-mPSI protocol proposed in this paper is lower than that of other mutual PSI protocols.

The combination of edge computing and IoT improves the intelligence of IoT devices and introduces intelligent devices into all aspects of life. The massive use of smart IoT devices has led to a sharp increase in the amount of data generated by users. Privacy protection and secure sharing of big data in the IoT have become the focus of current research. Therefore, research on more secure and efficient PSI technology applied to edge-assisted IoT is our future research direction.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Fund of China (no. 61802117), the PhD Foundation of Henan Polytechnic University (no. B2021-41), the Support Plan of Scientific and Technological Innovation Team in Universities of Henan Province, China (no. 20IRTSTHN013), the youth backbone teacher support program of Henan Polytechnic University (2018XQG-10), and the Research Foundation of Young Core Instructor in Henan Province, China (2018GGJS058).

References

[1] C. Zhou, J. Hu, and N. Chen, "Remote care assistance in emergency department based on smart medical," *Journal of Healthcare Engineering*, vol. 2021, Article ID 9971960, 10 pages, 2021.

[2] R. L. Fritz, M. Wilson, G. Dermody, M. E. Schmitter, and D. J. Cook, "Automated smart home assessment to support pain management: multiple methods analysis," *Journal of Medical Internet Research*, vol. 22, no. 11, Article ID e23943, 2020.

[3] R. S. Krishnan, S. Manikandan, J. R. F. Raj, and Y. H. Robinson, "Android Application Based Smart Bus Transportation System for Pandemic Situations," in *Proceedings of the 2021 Third International Conference On Intelligent Communication Technologies And Virtual Mobile Networks (ICICV)*, pp. 938–942, Tirunelveli, India, February 2021.

[4] K. Sabri, A. Toufik, and M. Mohamed, "Edge-based Safety Intersection Assistance Architecture for Connected Vehicles," in *Proceedings of the 2021 International Wireless Communications and Mobile Computing (IWCMC)*, pp. 272–277, Harbin City, China, july 2021.

[5] D. Fadolalkarim, E. Bertino, and A. Sallam, "An Anomaly Detection System for the Protection of Relational Database Systems against Data Leakage by Application Programs," in *Proceedings of the 2020 IEEE 36th International Conference On Data Engineering (ICDE)*, pp. 265–276, Dallas, TX, USA, April 2020.

[6] S. K. Debnath, K. Sakurai, K. Dey, and N. Kundu, "Secure Outsourced Private Set Intersection with Linear Complexity," in *Proceedings of the 2021 IEEE Conference On Dependable and Secure Computing (DSC)*, pp. 1–8, Aizuwakamatsu, Fukushima, Japan, February 2021.

[7] Y. Shi and S. Qiu, "Delegated key-policy attribute-based set intersection over outsourced encrypted data sets for CloudIoT," *Security and Communication Networks*, vol. 2021, no. 4, 11 pages, Article ID 5595243, 2021.

[8] S. Dittmer, Y. Ishai, S. Lu et al., "Function Secret Sharing for PSI-CA: With Applications to Private Contact Tracing," 2020, https://arxiv.org/abs/2012.13053.

[9] Y. Qian, J. Shen, P. Vijayakumar, and P. K. Sharma, "Profile matching for IoMT: a verifiable private set intersection scheme," *IEEE Journal of Biomedical and Health Informatics*, vol. 25, no. 10, pp. 3794–3803, 2021.

[10] S. K. Debnath and R. Dutta, "Provably secure fair mutual private set intersection cardinality utilizing bloom filter,"vol. 10143, pp. 505–525, in *Proceedings of the International Conference on Information Security and Cryptology*, vol. 10143, Springer, Beijing, China, November 2016.

[11] S. K. Debnath and R. Dutta, "Towards fair mutual private set intersection with linear complexity," *Security and Communication Networks*, vol. 9, no. 11, pp. 1589–1612, 2016.

[12] S. K. Debnath and R. Dutta, "New realizations of efficient and secure private set intersection protocols preserving fairness,"vol. 10157, pp. 254–284, in *Proceedings of the International Conference on Information Security and Cryptology*, vol. 10157, Springer, Beijing, China, November 2016.

[13] S. K. Debnath and R. Dutta, "Fair mP. S. I. and mPSI-C. A.: Efficient constructions in prime order groups with security in the standard model against malicious adversary," *IACR Cryptol. ePrint Arch*, vol. 216, 2016.

[14] B. Pinkas, T. Schneider, and M. Zohner, "Scalable private set intersection based on OT extension," *ACM Transactions on Privacy and Security*, vol. 21, no. 2, pp. 1–35, 2018.

[15] R. Agrawal, A. Evfimievski, and R. Srikant, "Information sharing across private data-bases," in *Proceedings of the 2003 ACM SIGMOD International Conference On Management Of Data*, pp. 86–97, California, CA, USA, June 2003.

[16] Y. Cai, C. Tang, and Q. Xu, "Two-party privacy-preserving set intersection with FHE," *Entropy*, vol. 22, no. 12, pp. 1339–1353, 2020.

[17] Y. Huang, D. Evans, and J. Katz, "Private Set Intersection: Are Garbled Circuits Better than Custom Protocols," in *Proceedings of the nineteenth Network and Distributed Security Symposium (NDSS 2012)*, San Diego, CA, USA, February 2012.

[18] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai, "Efficient circuit-based PSI with linear communication," in *Proceedings of the Advances in Cryptology - EUROCRYPT 2019*, vol. 11478, pp. 122–153, Springer, Darmstadt, Germany, May 2019.

[19] A. Kavousi, J. Mohajeri, and M. Salmasizadeh, "Efficient scalable multi-party private set intersection using oblivious PRF," in *Proceedings of the International Workshop on Security and Trust Management*, vol. 13075, pp. 81–99, Springer, Darmstadt, Germany, October 2021.

[20] A. Abadi, S. Terzis, and C. Dong, "Verifiable delegated private set intersection on outsourced private datasets,"vol. 9603, pp. 149–168, in *Proceedings of the International Conference on Financial Cryptography & Data Security*, vol. 9603, pp. 149–168, Springer, Christ Church, Barbados, February 2016.

[21] A. Abadi, S. Terzis, R. Metere, and C. Dong, "Efficient delegated private set intersection on outsourced private datasets," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 4, pp. 608–624, 2019.

[22] A. Kavousi, J. Mohajeri, and M. Salmasizadeh, "Improved Secure Efficient Delegated Private Set Intersection," in *Proceedings of the 2020 twentyeigth Iranian Conference on Electrical Engineering (ICEE)*, pp. 1–6, Tabriz, Iran, May 2020.

[23] L. Kissner and D. Song, "Privacy-preserving Set Operations," in *Proceedings of the Annual International Cryptology Conference*, pp. 241–257, Springer, Santa Barbara, California, CA, USA, August 2005.

[24] J. Camenisch and G. M. Zaverucha, "Private intersection of certified sets,"vol. 5628, pp. 108–127, in *Proceedings of the Financial Cryptography and Data Security*, vol. 5628, Springer, Accra Beach, Barbados, February 2009.

[25] M. Kim, H. T. Lee, and J. H. Cheon, "Mutual Private Set Intersection with Linear Complexity,"vol. 7115, pp. 219–231, in *Proceedings of the International Workshop on Information Security Applications*, vol. 7115, pp. 219–231, Springer, Jeju Island, Republic of Korea, August 2011.

[26] C. Dong, L. Chen, J. Camenisch, and G. Russello, "Fair private set intersection with a semi-trusted arbiter,"vol. 7964, pp. 128–144, in *Proceedings of the Computer Science in Data and Applications Security and Privacy XXVII*, vol. 7964, Springer, Newark, NJ, USA, July 2013.

[27] C. Wang, D. Wang, Y. Tu, G. Xu, and H. Wang, "Understanding node capture attacks in user authentication schemes for wireless sensor networks," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[28] D. Wang and P. Wang, "Two birds with one stone: two-factor Authentication with security beyond conventional bound," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 4, pp. 708–722, 2018.

[29] R. Pagh and F. F. Rodler, "Cuckoo hashing," *Journal of Algorithms*, vol. 51, no. 2, pp. 122–144, 2004.

[30] P. Miao, S. Patel, M. Raykova, K. Seth, and M. Yung, "Two-sided malicious security for private intersection-sum with cardinality,"vol. 12172, pp. 3–33, in *Proceedings of the Advances in Cryptology - CRYPTO 2020*, vol. 12172, Springer, Santa Barbara, CA, USA, August 2020.