Efficient Cryptographic Solutions for Unbalanced Private Set Intersection in Mobile Communication

Wuzheng Tan , Shenglong Du * , Jian Weng

Posted Date: 26 April 2024

doi: 10.20944/preprints202404.1701.v1

Keywords: Cryptographic; Private Set Intersection; Cuckoo Filter; Cloud Assistance; Private Contact Discovery)

Article

Efficient Cryptographic Solutions for Unbalanced Private Set Intersection in Mobile Communication

Wuzheng Tan [1,2], Shenglong Du [1,2,*] and Jian Weng [1,2]

[1] College of Cyber Security, Jinan University, Guangzhou 510632, China; tanwuzheng@jnu.edu.cn (W.T.); cryptjweng@gmail.com (J.W.)
[2] Guangdong Key Laboratory of Data Security and Privacy Preserving
* Correspondence: jndsl@stu2021.jnu.edu.cn

Abstract: Private Set Intersection (PSI) is a cryptographic method in secure multi-party computation that allows entities to identify common elements in their datasets without revealing their private data. Traditional approaches assume similar-sized datasets and equal computational power, overlooking practical imbalances. In real-world applications, dataset sizes and computational capacities often vary, particularly in the Internet of Things and mobile scenarios where device limitations restrict computational types. Traditional PSI protocols are inefficient here, as computational and communication complexities correlate with the size of larger datasets. Thus, adapting PSI protocols to these imbalances is crucial. This paper explores unbalanced PSI scenarios where one party (the receiver) has a relatively small dataset and limited computational power, while the other party (the sender) has a large amount of data and strong computational capabilities. It introduces three innovative solutions for unbalanced PSI: unbalanced PSI protocol based on Cuckoo filters, unbalanced PSI protocol based on single cloud assistance, and unbalanced PSI protocol based on dual cloud assistance, each addressing the shortcomings of the previous solution. These protocols primarily optimize database computational load, storage capacity, protocol execution time, and security. Depending on performance and security needs, different protocols can be employed for applications such as private contact discovery.

Keywords: cryptographic; private set intersection; cuckoo filter; cloud assistance; private contact discovery

1. Introduction

In today's digital age, data privacy and security have become critically important issues worldwide. With technological advancements and explosive growth in data volumes, individuals and institutions face unprecedented challenges in protecting their privacy. Privacy computing technologies have emerged in response to these challenges, enabling the secure computation and analysis of data without exposing the details of personal information. This is crucial for driving data-driven innovation and services while safeguarding personal privacy and data protection.

Private Set Intersection (PSI) technology is a key technique in the field of privacy computing. It allows two or more parties to identify the common elements in their data sets without revealing any other non-shared data. This technology is highly useful in multiple application scenarios, such as cross-institutional data cooperation, fraud detection, and private contact discovery, without compromising user privacy. It has been applied in various fields, including genetic testing of fully sequenced human genomes [1], private contact discovery [2], and botnet detection [3].

Despite the robust privacy protection features of PSI, there are several limitations and challenges. One of the main issues is the low efficiency in scenarios where data set sizes are imbalanced. Traditional PSI schemes typically assume that participants have similarly sized data sets and computational capabilities, which is not always feasible in practice. Particularly in scenarios where one party is a server with a large amount of data and the other is a mobile device with limited computational power, this imbalance is pronounced. Unbalanced PSI was proposed to address this issue by optimizing algorithms and protocol designs to enhance efficiency and feasibility between participants with different computational powers and data set sizes.

 doi:10.20944/preprints202404.1701.v1

To overcome the drawbacks of traditional Private Set Intersection protocols in scenarios with significant disparities in dataset sizes of the participants, this paper proposes a more suitable unbalanced PSI protocol based on Cuckoo filters. However, in the unbalanced PSI protocol based on Cuckoo filters, the client-side involves complex cryptographic operations and requires managing and processing a substantial amount of filter data. This poses a significant computational and storage burden on resource-constrained client devices, such as smartphones or other portable devices, limiting the practicality of these technologies and potentially affecting device performance and user experience.

In light of this, the second approach in this paper considers introducing cloud computing as a solution, leveraging its robust computational and storage capabilities to alleviate the load on client devices. By outsourcing part of the computation tasks to cloud servers, the client's workload can be significantly reduced, thereby speeding up the entire set intersection process. However, outsourcing data processing tasks to the cloud environment introduces a new problem: the risk of collusion attacks. When the cloud server colludes with one of the participants, it could threaten the privacy security of the entire scheme.

To address this challenge, the paper proposes a third, more secure design aimed at thwarting potential collusion attacks while maintaining the efficiency and practicality of unbalanced Private Set Intersection. This design incorporates multiple security technologies and strategies, effectively reducing the computational and storage pressure on client devices while ensuring data privacy in the cloud environment, even in the face of collusion threats. This is crucial for advancing the development and application of unbalanced Private Set Intersection technologies, providing a secure, efficient, and practical framework for future privacy-preserving computations.

Moreover, the unbalanced Private Set Intersection protocols proposed in this paper are particularly suited for private contact discovery applications. This scenario requires identifying and verifying common contacts between individuals or organizations under the premise of maintaining individual privacy, which is extremely important for social networking services, emergency response coordination, and business cooperation. Using the protocols proposed in this paper, users can securely and quickly identify common contacts without disclosing their complete contact lists. This not only enhances user privacy but also facilitates complex social network analyses and emergency contact networks, while avoiding security risks associated with data breaches or improper handling.

Overall, this paper's research primarily addresses the shortcomings of traditional PSI protocols in unbalanced scenarios by proposing three unbalanced private set intersection protocols for different contexts. There is a progressive relationship between each protocol, with each new proposal improving upon the deficiencies of the previous one. Specifically, this paper introduces an unbalanced PSI protocol based on Cuckoo filters that resolves the traditional PSI protocol's issues with unbalanced scenarios, a single-cloud assisted unbalanced PSI protocol that transfers most computational and storage tasks from the client to the cloud, and a double-cloud assisted unbalanced PSI protocol that can resist collusion attacks. In practical applications, different schemes can be selected based on varying performance and security needs. The main work and innovations of this study are as follows:

1. To address the performance shortcomings of traditional PSI protocols in the face of significant differences in dataset sizes between participants, this paper introduces the first protocol, which is an unbalanced PSI protocol based on Cuckoo filters. This protocol successfully constructs the first unbalanced private intersection protocol by integrating exchange encryption technologies with Cuckoo filter functionalities for private information retrieval, followed by experimental analysis.
2. To alleviate the computational and storage burden on clients in the first protocol, the paper further proposes an unbalanced PSI protocol based on single cloud assistance and conducts experimental analysis. This strategy effectively migrates computational and storage tasks to cloud services, significantly optimizing resource utilization efficiency.
3. To safeguard against data leakage risks inherent in the unbalanced PSI protocol based on single cloud assistance which cannot resist collusion attacks, the paper further designs an unbalanced PSI protocol based on dual cloud assistance. By employing homomorphic encryption and

 doi:10.20944/preprints202404.1701.v1

other security technologies, this scheme resolves potential data leakage risks in the single-cloud protocol while effectively preventing potential collusion attacks.
4. Building on the unbalanced PSI protocol based on dual cloud assistance, the research introduces the concept of a PSI network and formulates corresponding data update strategies, significantly enhancing the practicality of the protocol.

2. Related Works

2.1. Design framework of private set intersection protocol

2.1.1. Design Framework Based on Public Key Encryption

The basic idea behind early private set intersection protocols is to encrypt data elements and then perform comparison operations on the encrypted data. The most widely used technique in this method is homomorphic encryption: the sender encrypts their dataset and sends it to the receiver. The receiver processes these ciphertexts using the properties of homomorphic encryption and returns the results to the sender. The sender then decrypts these results using their own private key to obtain the intersection of the datasets. This public-key-based method generally relies on three main security assumptions [4]:

1. Based on Diffie-Hellman (DH) theory: Meadows [5] used the DH key exchange mechanism, which is based on the discrete logarithm problem, to implement a PSI protocol. In contrast, Huberman [6] and his team explored the use of elliptic curve cryptography in PSI, noting its significant advantages in security and efficiency compared to traditional discrete logarithm-based PSI methods.
2. Based on the RSA assumption: DeCristofaro and others [7] developed a semi-honest PSI protocol using RSA blind signature technology based on the integer factorization problem. Another study [8] showed that PSI schemes based on discrete logarithm cryptography demonstrated higher efficiency compared to those based on integer factorization cryptography.
3. Based on homomorphic encryption: Freedman and his team [9] innovatively represented elements as roots of polynomials and encrypted the coefficients of these polynomials using Paillier homomorphic encryption technology, combined with zero-knowledge proofs, to implement a two-party PSI protocol resistant to malicious attacks. In 2016, Freedman et al. [10] further improved computational efficiency through the ElGamal encryption mechanism and reduced the protocol's computational complexity using Cuckoo Hash technology [4]. Abadi et al. [11] introduced a set representation method based on point-value pairs of d-degree polynomials, implemented through the Paillier encryption scheme, reducing the multiplication complexity from $O(d^2)$ to $O(d)$ [4]. Kissner and other researchers [12] adopted different polynomial representation methods, significantly reducing computational costs to be linearly proportional to the number of participants. Jarecki and others [13] used additive homomorphic encryption and zero-knowledge proofs to implement pseudorandom functions (PRF). Hazay and others [14] developed an additive homomorphic encryption scheme that supports threshold decryption for implementing multi-party semi-honest PSI protocols. Dou Jiawei and others [15] combined Paillier encryption to propose a PSI protocol based on the formula for calculating the area of triangles and rational number encoding.

Public key encryption-based Private Set Intersection (PSI) schemes typically feature fewer communication rounds and are suitable for environments with strong computational capabilities. However, in practice, communication bandwidth and time complexity often pose significant constraints [4].

2.1.2. Design Framework Based on Garbled Circuits

Garbled circuit technology can transform any function into a Boolean circuit, thereby securely computing the function. Early methods based on universal circuits, like the DPSZ scheme [16],

 doi:10.20944/preprints202404.1701.v1

demonstrated how to use arithmetic circuits to solve the set intersection problem: the circuit builder encrypts the circuit gates using a symmetric key, then creates a garbled circuit and sends it to the circuit evaluator. The evaluator decrypts specific paths in the garbled circuit to obtain the intersection results, while being unable to access other paths in the circuit. As the circuit depth increases, its construction complexity also increases. Additionally, PSI protocols based on this circuit design can also perform various symmetric function operations, such as calculating the threshold intersection, the number of intersection elements, and their sum. For adversaries under semi-honest conditions, there are two types of garbled circuits: the Yao [17] protocol and the GMW [18] protocol. Pinkas et al. [19–21] and Chandran et al.based on hash storage structures and GMW circuits, implemented a more efficient OPRF circuit PSI scheme through private membership tests, reducing the number of comparisons and the depth of circuit equivalence comparisons. Meanwhile, Huang [22] and others created a semi-honest secure disordered circuit PSI scheme through the combination of Yao circuits, performing equivalence tests and specific sorting on adjacent elements. Despite these advantages, these methods still require additional key calculations and communication processes, such as key exchanges between participants.

2.1.3. Design Framework Based on Oblivious Transfer

Oblivious Transfer (OT) [23] is a cryptographic protocol that allows a sender to transmit information to a receiver without revealing any private information. In the OT protocol, the sender has two options, but the receiver can only obtain information about one of them without access to the other, and the sender does not know which option the receiver has chosen. OT is widely used in many secure areas due to its cryptographic robustness and privacy features. Its applications include secure protocol negotiation, secure online auction systems, and secure voting systems. In 2013, Dong [24] et al. proposed a new data structure—the garbled Bloom filter (GBF)—and based on the GBF and OT extension, they introduced a PSI protocol. This protocol utilized efficient symmetric encryption operations and could handle billions of elements. However, this protocol faced two issues: one is that the malicious sender might send incorrect shared information, and the other is that the input datasets are not independent. To address these problems, in 2016, Rindal and Rosulek [25] proposed a new randomized garbled Bloom filter using the "cut-and-choose" technique. They successfully implemented a two-party malicious model PSI protocol. Subsequently, Zhang [26] et al. based on this scheme, further proposed and implemented a multi-party PSI protocol, ensuring malicious security in the presence of two non-colluding servers, with computational and communication costs depending on the number of participants. Pinkas et al. [27] based on the OOS17-OT [28] protocol, built a maliciously secure PSI protocol. Rindal [29] et al. based on the semi-honest secure Schoppmann et al. [30] protocol and the maliciously secure Weng et al. [31] protocol, respectively proposed maliciously secure and semi-honest secure PSI protocols. Overall, PSI protocols based on oblivious transfer typically feature lower computational and communication overhead.

2.2. Private contact discovery

As shown in Figure 1, mobile privacy contact discovery refers to when you install a communication application such as WhatsApp on your phone, the first thing this app does is check your contact list to see which of your contacts are also using their service. To achieve this functionality, the application could simply tell the service provider about the users in your contact list. The service provider can then inform you which of these users are also using their service. A typical example is WeChat, which recommends friend accounts based on the mobile phone contact list. In this process, the WeChat server cross-matches the mobile numbers in the user's contact list with its own WeChat account database, thus identifying all users in the contact list who have registered WeChat accounts, and provides friend account recommendations based on this information.

 doi:10.20944/preprints202404.1701.v1

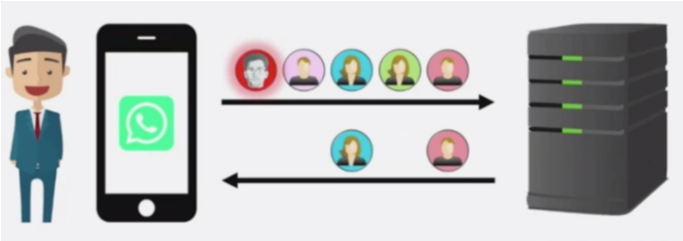

Figure 1. private contact discovery

This naturally involves privacy issues, as through our contacts, many aspects of our lives can be inferred, such as spending level and home address, and more seriously, it can lead to the leakage of our entire social graph. Furthermore, if we communicate with a highly influential person who holds many secrets, attackers could also potentially access these secrets through you.

The insecure solution to the privacy contact discovery problem allows users to send their contact set to the service provider, who then performs the intersection on behalf of the users. While this protects the privacy of the service provider, it exposes the users' private contacts to the service provider. In much research, current social media privacy contact discovery recommendation methods mainly include: i) not offering privacy protection promises, ii) seeking privacy protection through ambiguity (for example, using multiple signals for recommendations without explaining the reasons [32]), and iii) using temporary thresholds to block the simplest attacks (typically, these thresholds are applied to the number of mutual friends between two users to decide whether to recommend one to the other). The only known attempt to rigorously address this issue in practice was the recent study by Signal [33], but it addressed a more limited problem than the one discussed in this paper—privately finding out whether contacts in a phonebook are Signal users. Theoretical research suggests adopting structured graph perturbation [34] and randomized recommendation methods [35,36] to achieve strict privacy guarantees of differential privacy [37]. Although differential privacy has begun to see practical application in other data mining applications [38], theoretical methods for privacy contact discovery have not yet been deployed, possibly because the trade-offs between privacy and utility they require are too harsh (i.e., they often recommend people who users are almost unlikely to know, significantly negatively impacting user experience quality). William Brendel [39] and others proposed a method using an auxiliary graph for deanonymization. Instead of modifying existing privacy contact discovery algorithms to protect privacy, they modify the graph used by the algorithm to create a candidate graph more resistant to practical brute force attacks, aiming to improve privacy with auxiliary graph information.

Some applications use a naive hashing protocol, where the client sends only the hash results of phone numbers to the service provider. Unfortunately, this technique is almost considered insecure. Because the entropy of phone numbers is low, naive hashing methods are susceptible to dictionary attacks. Generally speaking, PSI (Private Set Intersection) protocols are provably secure cryptographic protocols. They allow two parties to compute the intersection of their input sets without revealing any information other than the intersection. However, in mobile contact discovery scenarios, a common problem with most PSI protocols is that the online phase of the protocol, the computation of the intersection, has communication complexity linearly related to the size of the two input sets. In mobile contact discovery scenarios, the server-side database may contain millions or even billions of entries, while it is generally assumed that the client has about 1000 contacts. This makes the communication complexity of the protocol extremely high and impractical to apply.

3. Related theories and technologies

3.1. Multi-party secure computation security model

The mathematical concept of Multi-Party Computation (MPC) involves several participants (such as $P_1, P_2, \ldots, P_n$) each holding private input data (x_i). These participants collaboratively execute

 doi:10.20944/preprints202404.1701.v1

a computation of the function $f(x_1, x_2, \ldots, x_n)$ with the goal of ensuring that each participant can only access their own computational results, while being unable to ascertain the inputs and results of others. There are generally two security models employed in secure multi-party computation protocols [40] [41]:

1. **Semi-honest model:** In this model, participants adhere to the protocol's execution rules but may attempt to gather other participants' inputs, outputs, and any accessible information during the execution of the protocol. This model assumes that the participants do not deviate from the established procedural rules but will use all available information to deduce the private data of others.
2. **Malicious adversary model:** Unlike the semi-honest model, the malicious adversary model accounts for the possibility that attackers may manipulate a subset of the participants to perform illicit actions, such as submitting incorrect input data or maliciously altering data to steal the private information of honest participants. Malicious adversaries might also disrupt the protocol by intentionally terminating its execution or by refusing to participate, thus preventing the protocol's completion.

3.2. Cuckoo filter

Determining whether a particular element belongs to a given set is a common problem in computer science, with widespread applications in bioinformatics, machine learning, computer networks, the Internet of Things, and database systems [42]. Filter data structures such as Bloom filters and Cuckoo filters can approximately determine if an element is part of a specified set and have been extensively applied in network routing [43], information retrieval [**?**], file merging [44], spam detection [45], and distributed systems [46].

Filter data structures are used to approximately ascertain if an element belongs to a specific set. In essence, for a given set S and a query element x, the filter can approximately inform the query whether "x is in S". "Approximately" here implies that if x is actually not in S, the filter has a small error probability p of wrongly indicating that "x is in S"; however, if x is indeed in S, the filter will always correctly return that "x is in S". Filter data structures sacrifice some query accuracy to enhance space and time efficiency. Unlike data structures that require storing complete information of each element for precise queries, filters approximate the presence of an element solely through partial information such as hash values or "fingerprints". Based on this principle, existing filter data structures are mainly categorized into two types: one type uses bit arrays as in Bloom filters; the other type, exemplified by Cuckoo filters, is based on element "fingerprints".

The Cuckoo filter [47] is an advanced retrieval structure made up of multiple buckets, each capable of containing several bits. Compared to Bloom filters, Cuckoo filters offer the significant advantage of supporting deletion of elements and having higher space efficiency. With equal storage space, Cuckoo filters can achieve more accurate search results and shorter search times. When querying an element, the time complexity for Cuckoo filters is $O(1)$, meaning constant time complexity. This indicates that the execution time for query operations does not increase with the number of elements in the filter, an important performance feature of the Cuckoo filter design. In this paper, Cuckoo filters are used to store data on database servers.

3.3. Paillier homomorphic encryption

Homomorphic encryption is an encryption technology that allows computations to be performed on encrypted data and to obtain encrypted results, which, when decrypted, are consistent with the results obtained by performing the same computations directly on the original data. This means that homomorphic encryption enables data to be processed and analyzed without revealing any content. It is an important technology for protecting online privacy, allowing cloud computing services to perform complex data processing tasks on users' encrypted data without accessing the actual data.

 doi:10.20944/preprints202404.1701.v1

Paillier homomorphic encryption is a public-key cryptosystem that specifically supports homomorphic addition operations on encrypted data. The applications of the Paillier encryption scheme are extensive, and it can be used to protect the privacy and security of data. For example, in distributed computing, the Paillier encryption scheme can be used to encrypt data and transmit it to various nodes for processing, ensuring the security and privacy of the data. Furthermore, the Paillier encryption scheme can also be used to implement homomorphic secret sharing, private set intersection, and other application scenarios. Overall, the Paillier encryption scheme is an efficient homomorphic encryption scheme with a wide range of application prospects. This paper uses the Paillier cryptosystem in its final scheme. The homomorphic properties utilized in this paper are as follows. The final scheme is based on these two features:

1. **Additive Homomorphism**: If $c_1 = \text{Enc}(m_1)$ and $c_2 = \text{Enc}(m_2)$, then $\text{Dec}(c_1 \cdot c_2 \mod n^2) = m_1 + m_2$. This allows for performing addition operations on ciphertexts without needing to decrypt them first.
2. **Scalar Multiplication Homomorphism**: If $c = \text{Enc}(m)$, then $\text{Dec}(c^k \mod n^2) = k \cdot m$. This means that it is possible to perform multiplication operations between a ciphertext and a plaintext scalar without decryption.

4. Unbalanced PSI Protocol Based on Cuckoo Filters

4.1. PSI Protocol Constructed Based on DH Key Exchange Mechanism

Before proposing the first unbalanced PSI protocol of this paper, we introduces Meadows' PSI protocol constructed based on the DH key exchange mechanism [5]. As shown in Figure 2,the specific process is as follows:

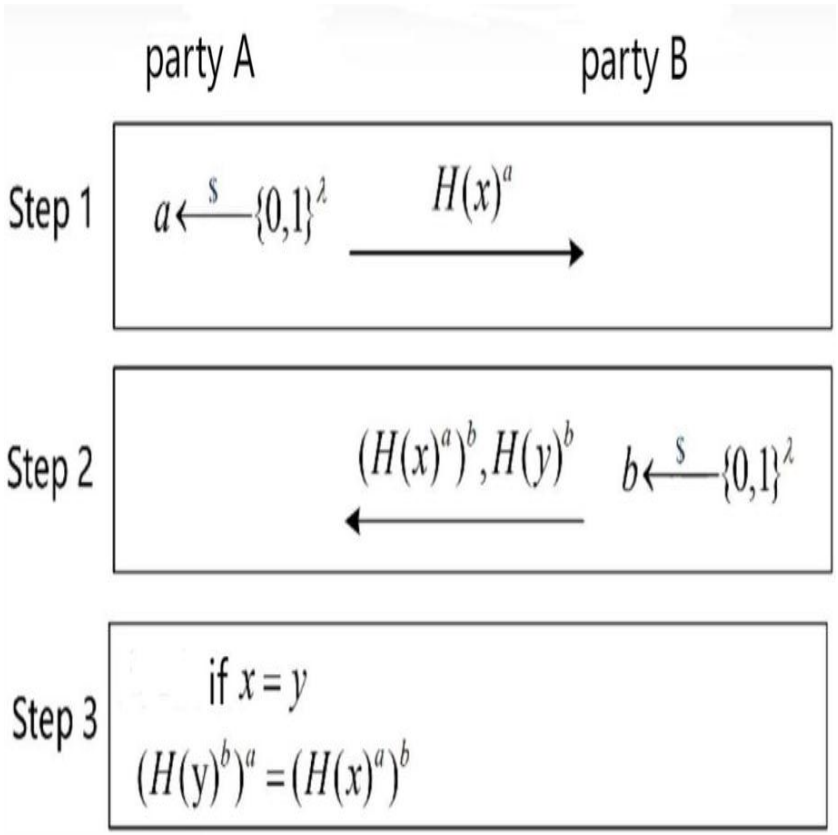

Figure 2. PSI protocol flow chart based on DH Key Exchange Mechanism

 doi:10.20944/preprints202404.1701.v1

4.1.1. Preprocessing Stage:

1. Hash Processing: Each participant applies the same hash function to each element in their dataset to form a hash-processed dataset, ready for subsequent encryption and computation processes.

4.1.2. Exchange and Computation Stage:

1. Exponentiation: Participant A takes each element from its dataset, use a to perform exponentiation operations (where a is A's private key) and forms a new set.
2. Data Exchange: Participant A sends the above-computed set to Participant B.
3. Auxiliary Dataset Construction: Upon receiving the dataset from A, Participant B use b to perform exponentiation operations (where b is B's private key) to build an auxiliary dataset and sends the result set back to A.
4. Exponentiation: At the same time, Participant B also use b to perform exponentiation operations of each element in its own dataset , which is also sent to A.

4.1.3. Intersection Identification Stage:

1. Exponentiation and Comparison: After receiving two datasets from B, participant use a to perform exponentiation operations of the elements in the latter dataset received that has been powered by B. Then, A compares this result with another dataset received from B.
2. Intersection Determination: If an element after being powered a times matches an element in the auxiliary dataset sent by B, then that element belongs to the intersection of datasets A and B.

This protocol effectively protects the participants' data privacy through two exponential operations and hash processing, preventing data leakage during the exchange process.

4.1.4. Experimental Analysis

For this protocol, experiments were conducted and the runtime was recorded for various combinations of dataset sizes, as shown in Table 1, where the dataset cardinality refers to the number of elements in the dataset.

Table 1. Runtime of the PSI Protocol Based on DH Key Exchange Mechanism

Cardinality of Dataset from Participant One	Cardinality of Dataset from Participant Two	Protocol Runtime (seconds)
2^{10}	2^{15}	1.7442
2^{10}	2^{17}	6.8024
2^{10}	2^{20}	55.2655
2^{10}	2^{25}	1849.2111
2^{15}	2^{15}	4.9248
2^{15}	2^{17}	10.0466
2^{15}	2^{20}	58.6051
2^{15}	2^{25}	1852.7098
2^{17}	2^{17}	20.0932
2^{17}	2^{20}	68.9472
2^{17}	2^{25}	1863.5443
2^{20}	2^{20}	165.4733
2^{20}	2^{25}	1964.6669

Through the experimental data, an important phenomenon can be observed. Table 1 shows the estimated runtime of the above protocol under different data volume levels. For example: Initially, when the number of elements in Participant One's dataset is 2^{10} and Participant Two's dataset is 2^{15}, the runtime of the protocol is 1.7442 seconds. In this case, there is a noticeable imbalance between the

 doi:10.20944/preprints202404.1701.v1

smaller side (Participant One) and the larger side (Participant Two). Now, if we expand the number of elements in Participant One's dataset (originally the side with fewer elements) to 2^{15}, while keeping Participant Two's dataset size constant at 2^{15}, the runtime increases to 4.9248 seconds, approximately three times the original. This indicates that although the runtime increases when the datasets are balanced, the increase is limited. However, if we keep Participant One's dataset size at 2^{10} and increase Participant Two's dataset size to 2^{20} (the same scale of change), the runtime dramatically increases to 55.2655 seconds, about 31 times the initial condition. This phenomenon shows that in unbalanced dataset conditions, increasing the number of elements in the larger dataset significantly affects the efficiency of the protocol.

These results reveal the importance of dataset balance in maintaining efficiency during the implementation of this protocol. Unbalanced datasets not only lead to extended runtimes but can also cause low resource utilization and delays in processing. However, in practical applications, when two parties want to perform private set intersection, their sets are often unequal and with a significant gap. Therefore, the current situation requires the design of a new protocol to eliminate the impact of dataset size imbalance on protocol efficiency.

4.2. Unbalanced PSI Protocol Based on Cuckoo Filters

Although Meadows' PSI protocol [5] constructed based on the DH key exchange mechanism provides an effective way to compute the intersection of two datasets, especially under the premise of protecting participants' data privacy, this paper observes that its efficiency is significantly impacted when dataset sizes are extremely unbalanced. In particular, as shown in Table 1, the runtime increases significantly as the size of the larger dataset increases, reflecting the performance limitations of Meadows' PSI protocol when dealing with unbalanced datasets.

In order to overcome these limitations, the first protocol proposed in this paper adopts a different technical strategy, which effectively reduces the computational burden under unbalanced conditions by introducing Cuckoo filters. This not only optimizes the data processing process, but also improves the overall operational efficiency. In the new protocol, the increase in run time is not as dramatic as that in the Meadows [**?**] PSI protocol based on the DH key exchange, even with unbalanced data set sizes, this allows for more efficient and balanced data processing. This improvement is particularly important for data sets of different sizes frequently encountered in practical applications.

Therefore, based on the above introduction, as shown in Figure 3, this paper first proposes a one-way private set intersection (PSI) protocol based on the discrete logarithm problem difficulty and the correctness (high false positive rate) of Cuckoo filters. This protocol is divided into two phases, the specific details of which will be introduced later, and in the subsequent sections, the specific implementation details and optimization strategies of the protocol will be thoroughly explained.

4.2.1. Definition of main participants and related symbols

1. database server:Represents the database server that holds all user data.
2. client:Represents the mobile client who wants to perform private contact discovery services.
3. X and Y represent the dataset of the database server and the client, respectively.
4. α represents the private key of the database server in the Diffie-Hellman encryption algorithm.
5. β_j represents the random number generated by the client for the Diffie-Hellman encryption algorithm.
6. H represents the hash function negotiated by the client and database server for use.
7. CF represents Cuckoo Filter,$CF.insert$ represents the operation of adding an element to the Cuckoo filter,$CF.check$ represents the operation of checking whether a specific element exists in the filter.
8. X_i represents the i-th element of set X. Similarly, Y_i, C_i, etc., also represent similar meanings.
9. $C = \{c_1, c_2, \ldots, c_n\}$ represents the set containing n_2 ciphertexts sent by the client to the database server.
10. C' represents the set containing n_2 ciphertexts sent by the database server to the client.

 doi:10.20944/preprints202404.1701.v1

11. $item_j$ represents the result obtained through a series of exchange and decryption operations, used to retrieve the filter to obtain the intersection.

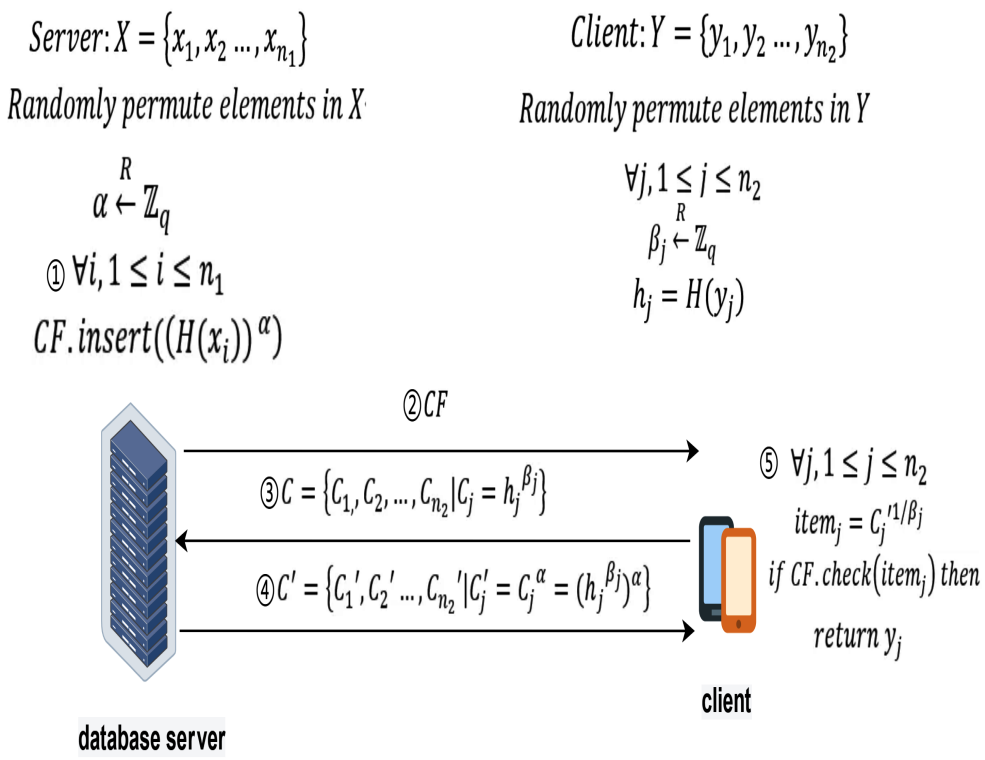

Figure 3. Unbalanced PSI Protocol Based On Cuckoo Filters

4.2.2. Protocol Process

The protocol is divided into two phases: the preprocessing phase and the intersection phase, with specific details as follows.

- Preprocessing:In the preprocessing phase, the client and server need to perform a series of preparatory work to ensure the security and efficiency of subsequent interactions. The specific steps are as follows:

    1. Security parameter negotiation: The client and database server agree on the large prime number q used in the DH encryption algorithm and the hash function H used.
    2. Database Server Generates Private Key: The database server generates its own private key α, used for the Diffie-Hellman (DH) encryption algorithm.
    3. Data Scrambling: The client and database server scramble their own datasets X and Y for randomization, enhancing data privacy and security.
    4. Client Data Preprocessing: The client calculates $h_j = H(y_j)$ and generates n_2 random numbers β_j, used for the Diffie-Hellman (DH) encryption algorithm.
    5. Creation of Cuckoo Filter: The database server generate a Cuckoo filter CF by using the operation $CF.insert((H(x_i))^{\alpha})$, and sends the filter CF to the client for private set intersection queries with privacy protection.

- Intersection: In the intersection phase, the client and database server perform a series of carefully designed encryption and decryption operations to blind the client's elements securely and compute the intersection of the two sets. The specific operations are as follows:

    1. Element Blinding and Interactive Encryption Operations: The client and the database server interact through a series of asymmetric encryption and decryption operations to blind the client's elements. Specifically, the client calculates $C_j = h_j^{\beta_j}$ and sends C to the database

 doi:10.20944/preprints202404.1701.v1

server. The database server uses its private key α to compute $C'_j = C_j^{\alpha}$ and sends C' back to the client.

2. Intersection Computation: After receiving C', the client checks whether they belong to the filter CF through the check operation $CF.check$, thereby calculating the intersection of the sets. Specifically, after receiving $H(X)^{r\alpha}$ sent by the database server, the client computes $item_j = C_j'^{1/\beta_j}$ and uses the result to query the filter CF to obtain the intersection element y_j.

4.2.3. Correctness Analysis

If $x_i = y_j$, then $H(x_i) = H(y_j)$, then $item_j = C_j'^{1/\beta_j} = C_j^{\alpha * 1/\beta_j} = \left(h_j^{\beta_j}\right)^{1/\beta_j * \alpha} = h_j^{\alpha} = H(x_i)^{\alpha} = H(y_j)^{\alpha}$.

Thus, through this scheme, the client can accurately obtain the intersection elements of both parties.

4.2.4. Security Analysis

This section will analyze the security of the protocol in detail, mainly its ability to protect the privacy of both parties.

Firstly, considering that the protocol utilizes the Diffie-Hellman (DH) key exchange mechanism to blind the client's elements, this process's security is based on the difficulty of solving the One-More-Gap-Diffie-Hellman (OMGDH) problem. Since the DH mechanism ensures that even in public communication channels, unauthorized parties cannot decipher the exchanged secret information, the client's data is protected during transmission to the server. The server uses a private key to process the received data and returns the results to the client, this process likewise ensures the security and privacy of the data server.

Secondly, the protocol's use of Cuckoo filters, while efficiently supporting insertion and query operations, its false positive characteristics mean that even if some non-intersecting elements are mistakenly identified as belonging to the intersection, it does not reveal the exact set membership information. This feature provides additional privacy protection to some extent, as even in the event of a false positive error, attackers cannot determine whether a specific element truly exists in the other party's set.

Furthermore, through the interactive computations between the client and the server, the protocol ensures that only elements common to both parties can be accurately identified. The client checks the data returned by the server against its own dataset to ultimately determine the intersection.

In summary, based on the blinding process using the Diffie-Hellman mechanism and the use of Cuckoo filters, this protocol can accurately calculate the intersection of two sets while protecting the participants' privacy.

4.2.5. Experimental Analysis

For this protocol, experiments were conducted, and the runtime was recorded for various combinations of data volumes, as shown in Table 2. The table also compares the runtime of Meadows' PSI protocol constructed based on the DH key exchange mechanism [5]. Since preprocessing can be completed offline, the runtime of the unbalanced PSI protocol based on Cuckoo filters refers to the total time of the outsourcing process and the intersection process. The original protocol refers to the PSI protocol constructed based on the DH key exchange mechanism, and the new protocol refers to the unbalanced PSI protocol based on Cuckoo filters.

 doi:10.20944/preprints202404.1701.v1

Table 2. Comparison of PSI Protocol Runtime Based on DH Key Exchange Mechanism and Cuckoo Filters

Cardinality of Dataset from Participant One	Cardinality of Dataset from Participant Two	Original Protocol Runtime (seconds)	New Protocol Runtime (seconds)
2^{10}	2^{15}	1.7442	0.1539
2^{10}	2^{17}	6.8024	0.1569
2^{10}	2^{20}	55.2655	0.1616
2^{10}	2^{25}	1849.2111	0.1693
2^{15}	2^{15}	4.9247	4.9239
2^{15}	2^{17}	10.0465	5.0232
2^{15}	2^{20}	58.6050	5.1709
2^{15}	2^{25}	1852.7097	5.4172
2^{17}	2^{17}	20.0931	20.0930
2^{17}	2^{20}	68.9471	20.6841
2^{17}	2^{25}	1863.5442	21.6690
2^{20}	2^{20}	165.4732	165.4731
2^{20}	2^{25}	1964.6668	173.3531

Through the experimental data, this paper can observe several key phenomena. First, when the cardinality of the smaller dataset (number of elements) remains constant while the number of elements in the larger dataset increases rapidly, it is observed that the runtime of the protocol does not change much, remaining consistent. This indicates that although the size of the large dataset increases dramatically, the efficiency of the protocol is not significantly affected, thereby proving that the design of this protocol can effectively mitigate the negative impact of dataset size imbalance on protocol efficiency. Especially, the overall runtime of the protocol is more related to the cardinality of the smaller dataset and has very low relevance to the cardinality of the larger dataset.

At the same time, this experiment also found that under balanced dataset conditions, the runtime of the PSI protocol based on the DH key exchange mechanism and the unbalanced PSI protocol based on Cuckoo filters does not differ significantly. This indicates that when the sizes of the sets are similar, both protocols can exhibit comparable performance, providing an efficient data intersection solution.

4.3. Summary of This Chapter

In this chapter, two types of private set intersection (PSI) protocols are discussed in detail: the PSI protocol constructed based on the DH key exchange mechanism [5] and the unbalanced PSI protocol based on Cuckoo filters. By comparing the runtime of these two protocols under different data volume conditions, this chapter has obtained a series of important observations and conclusions.

First, the PSI protocol based on DH key exchange shows a performance decline when dealing with imbalanced datasets. Especially when the size of the smaller dataset remains constant while the size of the larger dataset increases significantly, the runtime of this protocol increases dramatically, showing efficiency issues when processing large datasets.

In contrast, the unbalanced PSI protocol based on Cuckoo filters exhibits more stable performance under various conditions of dataset size imbalance. Even when the size of the large dataset increases significantly, the change in runtime for this protocol is not substantial, proving its superiority in dealing with unbalanced datasets. Additionally, when the datasets are nearly balanced, the performance difference between the two protocols is not significant, indicating that both protocols can work effectively under balanced dataset conditions.

Therefore, facing settings with dataset imbalance, the unbalanced PSI protocol based on Cuckoo filters appears more efficient. It not only maintains a relatively stable runtime in situations where the dataset sizes are extremely unbalanced but also exhibits performance comparable to the PSI protocol based on DH key exchange even when the dataset sizes are similar. This robustness makes the unbalanced PSI protocol based on Cuckoo filters a more ideal choice when facing the common problem of dataset imbalance in practical applications.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 26 April 2024 doi:10.20944/preprints202404.1701.v1

5. Unbalanced PSI Protocol Based on Single Cloud Assistance

The previous chapter has proven that the unbalanced PSI protocol based on Cuckoo filters is more suitable for practical scenarios, especially under unbalanced conditions, this protocol effectively resolves the performance limitations of the PSI protocol constructed using the DH key exchange mechanism in handling unbalanced datasets. However, there is still room for improvement in this protocol. It is observed that in this protocol, clients need to store filters and perform complex cryptographic operations, which can be a significant burden for mobile devices with limited computing power and storage space. A series of encryption operations and storing filters received from the other party becomes a heavy load. To address this issue, it is considered to transfer most of the client's computational and storage tasks to cloud servers. By delegating tasks to cloud servers, clients can significantly reduce computational and storage pressure, especially for mobile devices with limited capabilities.

This section will introduce cloud computing technology, which allows clients with limited computing power and storage space to outsource their private data and request cloud platforms to perform related computations. Currently, whether for individual users or large enterprises, entrusting data storage and computation tasks to cloud services has become a common practice. Based on the introduction above, as shown in Figure 4, this chapter proposes a second unbalanced private set intersection scheme based on the unbalanced PSI protocol using Cuckoo filters.

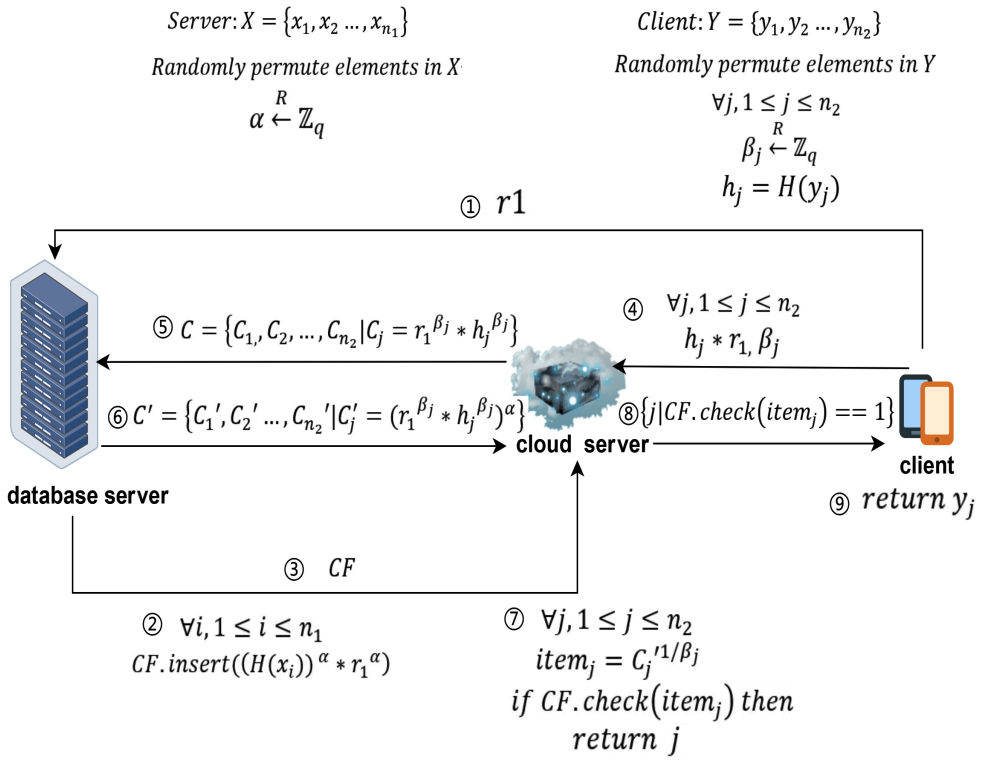

Figure 4. Unbalanced PSI Protocol Based On Single Cloud Assistance

5.1. Definition of main participants and related symbols

1. database server:Represents the database server that holds all user data.
2. client:Represents a mobile client that wants to discover private contacts.
3. cloud server: Represents an auxiliary server that assists the client in performing intersection operations, undertaking most of the computational and storage pressures.
4. X and Y respectively represent the dataset of the database server and the client dataset.
5. α represents the private key of the database server in the Diffie-Hellman encryption algorithm.
6. r_1 represents the random number generated by the client, used to blind the data.

 doi:10.20944/preprints202404.1701.v1

7. β_j represents the random number generated by the client for the Diffie-Hellman encryption algorithm.
8. H represents the hash function negotiated for use by the client and database server.
9. CF represents the Cuckoo Filter, $CF.insert$ represents the operation to add an element to the Cuckoo filter, $CF.check$ represents the operation to check if a specified element exists in the filter.
10. X_i represents the i-th element of the set X. Similarly, Y_i, C_i, etc., also represent similar meanings.
11. $C = \{c_1, c_2, \ldots, c_n\}$ represents the set of n_2 ciphertexts sent by the client to the database server.
12. C' represents the set of n_2 ciphertexts sent by the database server to the client.
13. $item_j$ represents the result obtained through a series of exchange and decryption operations, used to retrieve the filter to obtain the intersection.

5.2. Protocol Process

5.2.1. Preprocessing

1. Security parameter negotiation: Each role discusses the necessary security parameters—all parties share the large prime q used in the DH cryptographic algorithm. The client and database server negotiate to generate r_1 and the hash function H.
2. Database server generates a private key: The database server generates its own private key α, for use in the Diffie-Hellman encryption algorithm.
3. Data scrambling: The client and database server each scramble their own datasets X and Y.
4. Client data preprocessing: The client calculates $h_j = H(y_j)$, generates n_2 random numbers β_j, and calculates $h_j * r_1$.

5.2.2. Outsourcing

1. Database server sends data to the cloud server: The database server uses its private key α to perform the operation $CF.insert((H(x_i))^\alpha * r_1^\alpha)$, creates a Cuckoo filter CF, and sends it to the cloud server.
2. Client sends data to the cloud server: The client sends the random numbers β_j and $h_j * r_1$ to the cloud server. After receiving the data sent by the client, the cloud server calculates $C_j = r_1^{\beta_j} * h_j^{\beta_j}$. At this point, the cloud server has saved the client's blinded data.

5.2.3. Intersection

1. Cloud server sends data: The client cloud server sends the blinded data C_j to the database server.
2. Database server processes data: Upon receiving C_j, the database server uses its private key α to calculate $C'_j = C_j^\alpha$, and sends the result back to the cloud server.
3. Cloud server processes data: After receiving C'_j from the database server, the cloud server calculates $item_j = C_j^{1/\beta_j}$ and uses the result to search CF. If $item_j$ exists in CF, it returns the index j of $item_j$ and sends j to the client.
4. Obtaining the intersection: The client obtains the intersection element y_j through the received index j.

5.3. Correctness Analysis

If $x_i = y_j$, then $H(x_i) = H(y_j)$, so $item_j = C_j'^{1/\beta_j} = C_j^{\alpha * 1/\beta_j} = \left(r_1{}^{\beta_j} * h_j^{\beta_j}\right)^{1/\beta_j * \alpha} = r_1{}^\alpha * h_j^\alpha = r_1{}^\alpha * H(x_i)^\alpha = r_1{}^\alpha * H(y_j)^\alpha$.

Thus, through this scheme, the client can accurately obtain the intersection elements of both parties.

5.4. Security Analysis

In the design of this protocol, the primary security objective is to ensure that, even in a partially trusted cloud environment, neither the client's data nor the database server's data can be accessed

 doi:10.20944/preprints202404.1701.v1

or inferred by unauthorized entities. Specifically, since other participating parties are unaware of the database server's private key α, they cannot deduce the data held by the database server. Similarly, since other parties do not know the client's private random number r_1, they cannot deduce the client's data.

However, this scheme has inherent security risks, primarily because it does not withstand collusion attacks. If the data server and the cloud server collude, they can jointly deduce the client's data. This is possible because the cloud server possesses the blinded data $h_j \times r_1$, and if the data server leaks the private key r_1 to the cloud server, then both the cloud server and the database server could deduce the client's original data h_j. Collusion attacks are a security threat where two or more distinct entities (for example, users, systems, or service providers) secretly cooperate to undermine or circumvent security mechanisms and privacy measures. In cloud computing environments, cloud service providers and cloud users may collude to steal or infer other users' sensitive data stored on the cloud.

5.5. Experimental Analysis

5.5.1. Data Storage Volume

In the research of this paper, the experimental analysis of the unbalanced PSI protocol based on single cloud assistance revealed a key issue: when the client needs to receive a Cuckoo filter from the database server, this poses a significant challenge for mobile clients with limited storage capacity. This challenge is magnified when facing large datasets.

To understand this issue deeply, a series of experiments were conducted to measure the volume of Cuckoo filters needed by the client under different data sizes.The input data size for the experiments was provided by the database server, reflecting the various data volumes that might be encountered in actual application scenarios.As shown in Table 3, the paper meticulously recorded the specific sizes of Cuckoo filters under different input data volumes, revealing the intrinsic relationship between data volume and filter size.Through experiments, it was discovered that as the data volume in the database server increased, the storage burden on the client under the original protocol also increased accordingly, with the size of the Cuckoo filter directly impacted by the input data volume. In applications highly dependent on contact discovery services and containing extensive user information, this storage pressure is particularly evident. For example, for applications containing hundreds of millions of data entries, the volume of the Cuckoo filter could reach an overwhelming 14.850GB, a significant challenge for most mobile devices.

However, the proposed protocol based on cloud-assistance significantly alleviates this pressure. In the protocol, the Cuckoo filter is stored on the cloud server, thereby avoiding direct data transmission to the client. This approach not only effectively reduces the client's storage burden but also maintains the system's efficient operation, especially when handling large-scale data. By making such system design adjustments, the paper not only ensures the protection of data privacy but also significantly improves the feasibility and practicality of contact discovery services in actual applications.

In summary, the experimental results of this chapter emphasize the importance of optimizing storage strategies when dealing with large data scenarios. By outsourcing some storage tasks to the cloud server, the solution proposed in this chapter offers a viable approach for mobile clients needing to perform private contact discovery services, addressing the growing demands for data storage.

Therefore, based on the above analysis and experimental results, it is clear that when the data volume in the database server is excessively large, in other words, when the number of users reaches a certain level, the feasibility of a simple unassisted unbalanced PSI protocol based on Cuckoo filters significantly decreases, especially in applications like private contact discovery. This is because the unbalanced PSI protocol based on Cuckoo filters requires clients to directly receive and process massive Cuckoo filters, which poses a significant challenge for clients with limited storage resources,

 doi:10.20944/preprints202404.1701.v1

particularly mobile devices. Client devices often do not have enough storage space to accommodate these large-volume filter data, let alone process these data to complete PSI operations.

Table 3. Size of Cuckoo Filters at Different Data Volumes

Data Set Count	Size of Cuckoo Filter (MB)
2^{15}	0.535
2^{17}	2.363
2^{20}	21.678
2^{22}	93.645
2^{23}	194.436
2^{24}	403.201
2^{27}	3571.206
2^{28}	7372.835
2^{29}	15206.421

In this context, the introduction of a cloud server scheme shows its unique advantages. By transferring the storage of the filter to the cloud server, the burden on the client is greatly reduced. By this means, even in situations with a massive number of users and large data volumes, the scheme can still maintain efficient operations and ensure the smooth completion of PSI operations.

In summary, through experimental and theoretical analysis, this section concludes that in scenarios with large-scale users and massive data volumes, the introduction of a cloud server scheme is more feasible and efficient than the unbalanced PSI protocol based on Cuckoo filters.

5.5.2. Protocol Running Time

For this protocol, as shown in Table 4, the paper conducted experiments and recorded the running time of the protocol under various data volume combinations. Because preprocessing can be completed offline, the running time of the protocol refers to the total time of the outsourcing process and the intersection process. Table 4 also compares the running times of the unbalanced PSI protocol based on Cuckoo filters and the unbalanced PSI protocol based on single cloud assistance. Here, Protocol 1 refers to the unbalanced PSI protocol based on Cuckoo filters, and Protocol 2 refers to unbalanced PSI protocol based on single cloud assistance.

Table 4. Running Times of Protocol 1 and Protocol 2 Under Different Data Volume Combinations

Client Dataset Size	Database Server Dataset Size	Protocol 1 Running Time (seconds)	Protocol 2 Running Time (seconds)
2^{10}	2^{15}	0.1539	0.1543
2^{10}	2^{17}	0.1569	0.1573
2^{10}	2^{20}	0.1616	0.1611
2^{10}	2^{25}	0.1693	0.1683
2^{15}	2^{15}	4.9239	3.8223
2^{15}	2^{17}	5.0232	3.9145
2^{15}	2^{20}	5.1709	4.0267
2^{15}	2^{25}	5.4172	4.2233
2^{17}	2^{17}	20.0930	15.6768
2^{17}	2^{20}	20.6841	16.1281
2^{17}	2^{25}	21.6690	16.8939
2^{20}	2^{20}	165.4731	129.0516
2^{20}	2^{25}	173.3531	135.2534

From the experimental analysis, the following conclusions can be drawn: In cases of smaller data volumes, the performance differences between the two protocols are not significant. However, as the data volume increases, the running time differences between different protocols gradually become

 doi:10.20944/preprints202404.1701.v1

apparent. This is because, at certain specific levels, the proportion of communication time is relatively high when the data volume is small, significantly impacting the results. For larger data volumes, where computation time dominates, Protocol 2, by placing computational tasks on the more powerful cloud server, gradually widens the running time difference from Protocol 1. Overall, the use of cloud resources in the unbalanced PSI protocol based on single cloud assistance significantly reduces running times, especially when dealing with large-scale datasets.

5.6. Summary of This Chapter

This chapter introduces an unbalanced PSI protocol based on single cloud assistance, which utilizes cloud computing to reduce the computing and storage pressure of the client compared with previous protocols. Additionally, in the absence of collusion between the data server and the cloud server, the protocol effectively protects data from unauthorized access, ensuring the confidentiality of the data and the privacy of the client, making it highly suitable for scenarios where the cloud server is fully trusted.

However, it cannot be denied that although this scheme significantly reduces the computational and storage burden on the client, its security against collusion attacks is insufficient. When the possibility of collusion between the cloud server and data server cannot be completely ruled out, the protocol faces security risks and will require further security enhancement measures. Therefore, the next chapter will introduce a more secure solution to address the security deficiencies of the current scheme, ensuring the security and privacy of client data and database server data in environments where not all parties are fully trustworthy. In other words, the new scheme can resist collusion attacks. server.

6. Unbalanced PSI Protocol Based on Dual cloud Assistance

The previous single-server solution, which efficiently delegated computationally intensive encryption operations such as exponentiation and storage-intensive Cuckoo filters to the cloud server, has indeed alleviated the computational and storage burdens on the client to a certain extent. This is particularly advantageous for clients with limited computing and storage capabilities, allowing them to operate beyond their hardware constraints. However, security analysis reveals that the unbalanced PSI protocol based on single cloud assistance inherent security risks, specifically when collusion between the cloud and data servers is possible, thus compromising its adequacy in protecting client data privacy.

As shown in Figure 5, to preserve the advantages of the previous scheme—namely reducing computational and storage pressures on the client—while addressing these security issues, this chapter proposes a new solution. This design aims to enhance the security during data processing, especially against potential collusion attacks.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 26 April 2024 doi:10.20944/preprints202404.1701.v1

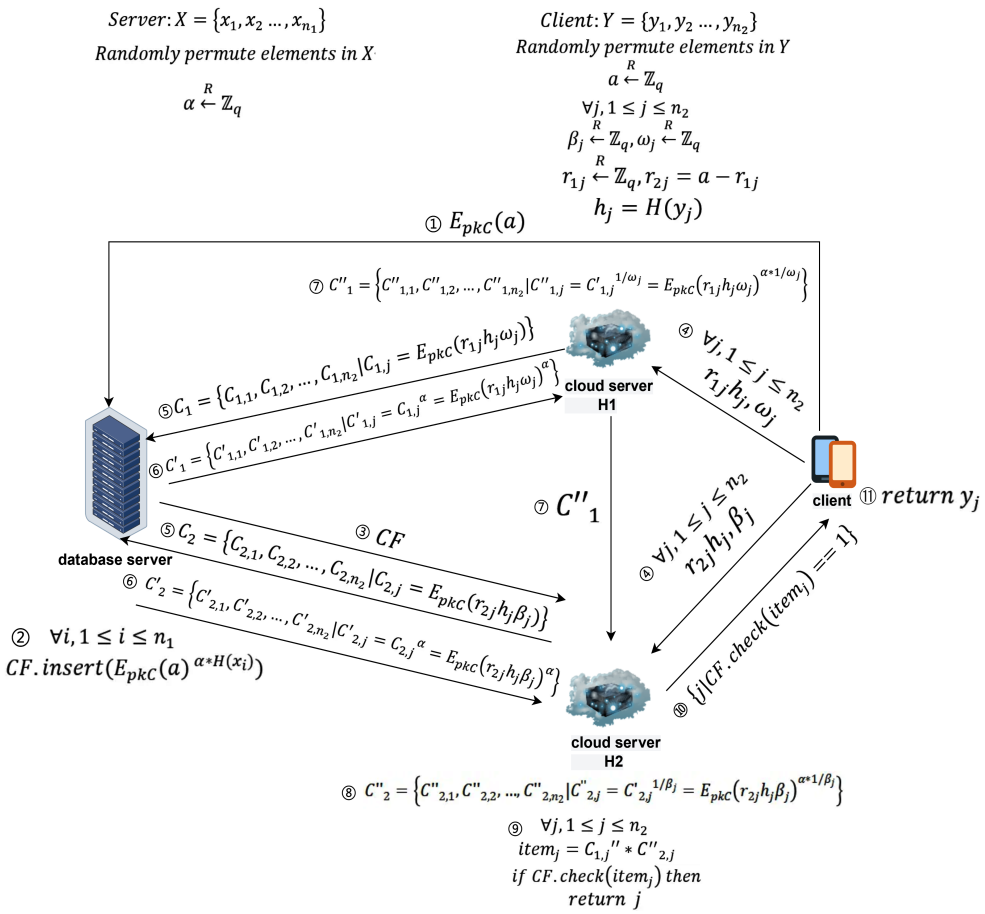

Figure 5. nbalanced PSI Protocol Based on Dual cloud Assistance

6.1. Definition of main participants and related symbols

1. database server: Represents the database server that holds all user data.
2. client: Represents the mobile client that wishes to perform private contact discovery services.
3. cloud server H_1: Acts as an auxiliary server for the client, handling the majority of computation and storage pressures.
4. cloud server H_2: Another auxiliary server handling substantial computational and storage demands.
5. X and Y: Represent the dataset of the database server and the client, respectively.
6. α: Represents the private key of the database server used in the Diffie-Hellman encryption algorithm.
7. H: The hash function agreed upon by the client and the database server for use.
8. CF: Represents the Cuckoo Filter, where $CF.insert$ denotes the operation to add elements, and $CF.check$ checks for the presence of specific elements.
9. ω_j: Random exponentials generated by the client for cloud server H_1, β_j for cloud server H_2.
10. a: A secret value held by the client.h
11. $r_{1,j}$: Random numbers used by the client for sending obfuscated data to cloud server H_1, and $r_{2,j}$ for H_2 where $r_{1,j} + r_{2,j} = a$.
12. C_1: The ciphertext collection sent from cloud server H_1 to the database server, and C_2 from H_2; $C_{1,j}$ and $C_{2,j}$ are specific elements within these collections.
13. C'_1 and C'_2: Processed ciphertext collections returned to H_1 and H_2 from the database server; $C'_{1,j}$ and $C'_{2,j}$ are specific elements within these collections.
14. C''_1 and C''_2: Final processed ciphertext collections at H_1 and H_2 after receiving data from the database server; $C''_{1,j}$ and $C''_{2,j}$ are specific elements within these collections.
15. $item_j$: Represents the result of multiplying $C''_{1,j}$ and $C''_{2,j}$ used to query the filter.
16. j: Represents the index used by the client to obtain the intersection.

 doi:10.20944/preprints202404.1701.v1

6.2. Protocol Process

6.2.1. Preprocessing

1. Discuss security parameters: Each party discusses the necessary security parameters—the large prime q used in DH encryption and the client's public key pk_c required for the Paillier encryption system. The client and the database server negotiate the creation of hash function H.
2. Client sends $E_{pk_c}(a)$: The client generates its private secret number a and sends $E_{pk_c}(a)$ to the database server.
3. Database server generates private key: The database server creates its private key α, used for the DH encryption algorithm.
4. Data scrambling: The client and the database server each shuffle their respective datasets.
5. Client calculates hashes and generates random numbers: The client computes $h_j = H(y_j)$ and generates n_2 random numbers β_j, ω_j, r_{1j}, r_{2j}, and computes $r_{1j}h_j$, $r_{2j}h_j$ where $r_{1j} + r_{2j} = a$.

6.2.2. Outsourcing

1. Client sends data to cloud servers: The client sends $r_{1j}h_j$, ω_j to cloud server H_1, and $r_{2j}h_j$, β_j to cloud server H_2. H_1 computes $C_{1,j} = E_{pk_c}(r_{1j}h_j\omega_j)$, and H_2 computes $C_{2,j} = E_{pk_c}(r_{2j}h_j\beta_j)$. At this point, H_1 and H_2 hold the client's obfuscated data.
2. Database server sends data to cloud servers: Using $E_{pk_c}(a)$, the database server performs the filter insertion operation CF.insert$\left(E_{pk_c}(a)^{\alpha * H(x_i)}\right)$ to generate a Cuckoo filter and sends it to cloud server H_2. H_2 stores the filter sent by the database server.

6.2.3. Intersection

1. H_1 and H_2 send data: H_1 and H_2 each send their respective collections C_1 and C_2 to the database server.
2. Database server processes data: Upon receiving the data, the database server uses its private key α to compute $C'_{1,j} = C^{\alpha}_{1,j} = E_{pk_c}(r_{1j}h_j\omega_j)^{\alpha}$ and sends the results back to H_1. It also processes $C'_{2,j} = C^{\alpha}_{2,j} = E_{pk_c}(r_{2j}h_j\beta_j)^{\alpha}$ and sends the results back to H_2.
3. H_1 processes data: After receiving data from the database server, H_1 uses the random number ω_j to calculate $C''_{1,j} = C'^{1/\omega_j}_{1,j} = E_{pk_c}(r_{1j}h_j\omega_j)^{\alpha * 1/\omega_j}$ and sends the results to H_2.
4. H_2 processes data: Upon receiving data from H_1 and the database server, H_2 calculates $C''_{2,j} = C'^{1/\beta_j}_{2,j} = E_{pk_c}(r_{2j}h_j\beta_j)^{\alpha * 1/\beta_j}$. H_2 checks if $item_j = C''_{1,j} * C''_{2,j}$ exists in CF. If it does, it returns the index j of $item_j$ and sends it to the client.
5. Obtaining the intersection: The client receives the index j and retrieves the intersecting element y_j.

6.3. Correctness Analysis

If $x_i = y_j$, then $H(x_i) = H(y_j)$, which implies that $C''_{1,j} * C''_{2,j} = C'^{1/\omega_j}_{1,j} * C'^{1/\beta_j}_{2,j} = E_{pk_c}(r_{1j}h_j\omega_j)^{\alpha * 1/\omega_j} * E_{pk_c}(r_{2j}h_j\beta_j)^{\alpha * 1/\beta_j} = E_{pk_c}(r_{1j}h_j\alpha) * E_{pk_c}(r_{2j}h_j\alpha) = E_{pk_c}[\alpha h_j(r_{1j} + r_{2j})] = E_{pk_c}(\alpha h_j a) = E_{pk_c}(a)^{\{\alpha * H(x_i)\}}$.

Thus, the client can accurately obtain the intersection elements from both parties.

6.4. Security Analysis

In considering security against collusion attacks, it is generally assumed that there is an adversary who possesses the perspective and information of all participating parties except for the protected entity. This means the adversary can access, control, or receive information and resources from all participants except for the protected party. In this scenario, the adversary attempts to compromise the system's security or privacy by aggregating these insights, such as revealing sensitive data of the protected party. If, in this context, the adversary still cannot learn or infer the protected party's data, then it is proven that the data and privacy of the protected party are sufficiently secured against collusion attacks.

 doi:10.20944/preprints202404.1701.v1

This section defines a game where the security objective is to maintain confidentiality of the data within the set under semi-honest and collusion conditions. The game for securing the client's data set is as follows:

1. The client runs the preprocessing algorithm, sharing the cryptographic hash function H and the large prime q used in the protocol with the adversary.
2. The client simulates the outsourcing algorithm and sends their (encrypted) input to the adversary.
3. The client and the adversary simulate the intersection algorithm and discard any output.
4. The adversary is asked to output a guess $\hat{y}$ of the client's input y.

The game is analogized to a deterministic one-way function, such as a public key encryption scheme. Let S be the simulated messages of the client during the game. Let a one-way function adversary A' be given the information (public key) pk and function (ciphertext) c (encrypted y). The advantage of the adversary Adv_A is defined as the difference between the successful guesses of A and A'. If this advantage is negligible in the security parameter λ, then the outsourced private set intersection is considered secure. That is, let $Adv_A = \Pr[A(S) = y] - \Pr[A'(pk, c) = y]$. If $Adv_A < \frac{1}{\text{poly}(\lambda)}$, then the protocol is said to be secure.

Specifically, after the steps mentioned above, H_1, H_2, and the database server have a complete view of the process. However, under the two-server architecture, as illustrated in Figure 5.1:

1. In step four of Figure 3, since r_{1j} and r_{2j} are unknown to the adversary, h_j cannot be derived. The adversary can only attempt exhaustive guessing, thus making Adv_A negligible.
2. In subsequent steps, as A does not know the client's private key for the Paillier encryption system, it is impractical to decrypt the ciphertexts, making it even more challenging to derive h_j. For instance, $item_j = C''_{1,j} * C''_{2,j} = {C'_{1,j}}^{1/\omega_j} * {C'_{2,j}}^{1/\beta_j} = E_{pk_c}(r_{1j}h_j\omega_j)^{\alpha * 1/\omega_j} * E_{pk_c}(r_{2j}h_j\beta_j)^{\alpha * 1/\beta_j}$, and since the private key used in Paillier's system by the client is unknown, decrypting this compound is complex and hence h_j remains secure.

From the analysis above, it is evident that the advantage of Adv_A is negligible. Therefore, if both cloud servers collude with the database server, they cannot deduce the client's original data.

Next, consider the security of the data in the database server's set. Obviously, apart from the database server itself, none of the parties know the database server's private key α, hence even if both cloud servers colluded with the client, they cannot derive the original data from CF.

It is particularly noted that due to the prevalence of attacks on hash functions, further security enhancements are recommended by protecting the hashed data as the raw data.

In conclusion, the dual-server scheme successfully resists collusion attacks under semi-honest conditions. By thoroughly integrating considerations for security and privacy into the protocol design, both the client's and the database server's data are assured of robust protection. This solution not only provides an effective mechanism for private set intersection but also demonstrates resilience against potential collusion threats.

6.5. Experimental Analysis

6.5.1. Data Computation Volume

When evaluating any protocol's performance, the computational load borne by the client is undoubtedly a critical factor. As all three protocols have been introduced, this section focuses particularly on the client's computational volume, to accurately gauge and compare the efficiency of the three distinct protocols in operation. Specifically, this section will conduct a detailed analysis and comparison of the primary computational tasks that the client must execute across these protocols, to fully assess each protocol's demand on the client's computational resources. This analysis will primarily focus on the types of operations involved, aiming to clarify which protocol demonstrates relative advantages in reducing the client's computational burden, thus providing a solid basis for

 doi:10.20944/preprints202404.1701.v1

selecting the most appropriate protocol. Below is an analysis of the main types of operations involved in each protocol, focusing primarily on the outsourcing and intersection processes as the preprocessing can be completed offline.

1. **unbalanced PSI protocol based on Cuckoo filters:** Two rounds of modular exponentiation operations and filter retrieval.
2. **unbalanced PSI protocol based on single cloud assistance**: A single round of multiplication operations and outputting y_j based on index j.
3. **unbalanced PSI protocol based on dual cloud assistance**: Two rounds of multiplication operations and outputting y_j based on index j.

An analysis of the single-instance time consumption for these four operations offers a practical insight into the computational volume differences:

1. **Modular Exponentiation Operation**: Representing computation-intensive operations, modular exponentiation becomes particularly time-consuming when dealing with large numbers. On standard hardware setups, a single instance of modular exponentiation might take from a few milliseconds to several tens of milliseconds, depending mainly on the size of the numbers involved and the efficiency of the algorithm.
2. **Multiplication Operation**: Compared to modular exponentiation, multiplication operations execute much faster on modern computing systems, even when involving large numbers, thanks to optimized algorithms that can keep times in the microsecond range. Therefore, whether it's a single round of multiplication in the single-cloud protocol or two rounds in the dual-cloud protocol, the processing times are relatively short, typically ranging from a few microseconds to a few hundred microseconds.
3. **Cuckoo Filter Retrieval**: Although relatively quick, the retrieval operation for a Cuckoo filter involves memory access, which may make it slightly slower than simple arithmetic operations. This type of operation typically takes from a few microseconds to several tens of microseconds, depending on the size of the filter and the efficiency of the implementation.
4. **Outputting y_j Based on Index j**: This operation involves retrieving an element from an array or list based on a specific index and is generally very fast, with processing times possibly ranging from a few nanoseconds to a few microseconds, primarily limited by memory access speeds.

After a detailed analysis and comparison, this section has conducted a thorough exploration of the key computational tasks executed by the client across the three different protocols. These tasks include modular exponentiation, multiplication operations, Cuckoo filter retrieval, and data retrieval based on an index. By assessing these types of computations and their specific time consumptions, the following conclusions can be drawn:

1. **Unbalanced PSI Protocol Based On Cuckoo Filtersl**: Primarily relies on two rounds of modular exponentiation, which are computation-intensive, especially when dealing with large numbers, making it the most time-consuming of all the operations reviewed.
2. **Unbalanced PSI Protocol Based On Single Cloud Assistance** : By executing a single round of multiplication and an index-based data retrieval process, it significantly alleviates the computational burden on the client. Multiplication operations, even for large numbers, can be completed within the microsecond range (from a few to several hundred microseconds), and index-based data retrieval takes an extremely short time, usually just a few nanoseconds to a few microseconds.
3. **Unbalanced PSI Protocol Based On Dual cloud Assistance**: Includes two rounds of multiplication operations and an index-based data retrieval process, also aiming to distribute the computational pressure on the client. Although it involves two rounds of multiplication, due to the inherent efficiency of the operation, the total processing time remains within an acceptable range.

 doi:10.20944/preprints202404.1701.v1

Through the meticulous assessment of each protocol's computational types and their time consumptions, it is evident that both the unbalanced PSI protocol based on single cloud assistance and unbalanced PSI protocol based on dual cloud assistance exhibit excellent performance in reducing the client's computational burden, particularly in the efficient execution of multiplication operations and data retrieval. In contrast, the unbalanced PSI protocol based on Cuckoo filters, while potentially offering stronger security provisions, shows some deficiencies in efficiency and timeliness. Therefore, when choosing an appropriate protocol, a balance should be struck based on actual performance requirements and security needs.

6.5.2. Protocol Running Time

After introducing all three protocols, this section primarily discusses the running times of the protocols. The running time of a protocol is an important benchmark for evaluation in this paper because it directly reflects the protocol's efficiency in practical operations. The factors affecting the running time of the protocol include computational time and communication time. As shown in Table 5, experiments were conducted to record the running times of the protocols under various data volume combinations.Table 5 also places the running times of the unbalanced PSI protocol based on Cuckoo filters, unbalanced PSI protocol based on single cloud assistance, and unbalanced PSI protocol based on dual cloud assistance side by side for comparative analysis. Here, Protocol I refers to the unbalanced PSI protocol based on Cuckoo filters, Protocol II refers to the unbalanced PSI protocol based on single cloud assistance, and Protocol III refers to the unbalanced PSI protocol based on dual cloud assistance.

Table 5. Running times of the three protocols under different data volume combinations

Data Volume	Protocol I Running Time (s)	Protocol II Running Time (s)	Protocol III Running Time (s)
$2^{10}\|2^{15}$	0.1539	0.1543	0.1551
$2^{10}\|2^{17}$	0.1569	0.1573	0.1586
$2^{10}\|2^{20}$	0.1616	0.1611	0.1635
$2^{10}\|2^{25}$	0.1693	0.1683	0.1701
$2^{15}\|2^{15}$	4.9239	3.8223	4.3707
$2^{15}\|2^{17}$	5.0232	3.9145	4.4709
$2^{15}\|2^{20}$	5.1709	4.0267	4.6001
$2^{15}\|2^{25}$	5.4172	4.2233	4.8206
$2^{17}\|2^{17}$	20.0930	15.6768	17.8801
$2^{17}\|2^{20}$	20.6841	16.1281	18.4004
$2^{17}\|2^{25}$	21.6690	16.8939	19.2802
$2^{20}\|2^{20}$	165.4731	129.0516	147.2608
$2^{20}\|2^{25}$	173.3531	135.2534	154.3003

It is noteworthy that the preprocessing stages of all three protocols can be completed offline, meaning they do not directly contribute to online operation delays. Therefore, the recorded running times in this paper refer to the total time of all processes excluding preprocessing. Specifically, in Protocol I, this primarily refers to the total duration of the intersection process; in Protocols II and III, it refers to the total duration of both the outsourcing and intersection processes.

Through experimental analysis, the paper draws the following conclusions: At smaller data volumes, the performance differences between the three protocols are not significant. However, as the data volume increases, the differences in running times between the protocols become apparent. Generally, the unbalanced PSI protocol based on Cuckoo filters tends to have the longest running time, while the unbalanced PSI protocol based on single cloud assistance has the shortest running time, and the performance of the unbalanced PSI protocol based on dual cloud assistance is in the middle. This phenomenon can be explained by the complexity of data handling and the differences in communication overhead among the protocols. The unbalanced PSI protocol based on Cuckoo

 doi:10.20944/preprints202404.1701.v1

filters, due to its direct and unoptimized calculations, is less efficient when handling large volumes of data. Nevertheless, at very small data volumes, where the proportion of communication time is relatively high, the impact of data transmission costs on total running time becomes significant. In such cases, the unbalanced PSI protocol based on Cuckoo filters does not necessarily appear inefficient because other protocols might be even less efficient in data transmission. Especially in environments with poor network conditions or limited data transfer rates, the lower communication demands of the unbalanced PSI protocol based on Cuckoo filters might, in some cases, lead to better performance.

Moreover, the running times of the unbalanced PSI protocol based on single cloud assistance and the unbalanced PSI protocol based on dual cloud assistance are significantly reduced through distributed computing and the use of cloud resources, especially when dealing with large-scale datasets. In summary, choosing the appropriate protocol requires a comprehensive consideration of factors such as data volume, computational resources, and network environment. In practical applications, understanding the performance characteristics and suitable scenarios of each protocol is crucial for optimizing data processing workflows and enhancing efficiency.

6.6. Summary of This Chapter

The protocol leverages the computational and storage resources of two cloud servers, significantly reducing the burden on the client by lowering its computational and storage requirements and enhancing the system's efficiency and availability. Through distributed computing and security measures such as homomorphic encryption, it ensures the safety and privacy of data during transmission and processing, adequately protecting sensitive information of both the client and the database server. This solution not only improves the operational efficiency of devices with limited resources but also effectively prevents collusion attacks. Consequently, the unbalanced PSI protocol based on dual cloud assistance excels in private set intersection operations, particularly in applications like private contact discovery, demonstrating both high efficiency and security.

6.7. Extensions

To enhance the practicality of the scheme, this section will explore two key aspects from an engineering practice perspective: the design of the PSI network and the design of the data update mechanism.

First, the design of the PSI network focuses on building an efficient, secure, and scalable network architecture to support large-scale private set intersection computations.

Second, the design of the data update mechanism involves how to update the data sets stored on the cloud servers without interrupting the service. This is particularly crucial for PSI computation scenarios that require frequent data updates.

6.7.1. PSI Network

As previously described, the client delegates PSI computations to two cloud servers. In practice, a vast network of cloud servers can be built to support this delegation. The basic system description is as follows.

- Access and Authentication of Cloud Servers: Any server can apply to become a cloud server, also known as a server assistant. These servers must undergo a series of certification processes (including hardware performance verification, security vulnerability scanning, and compliance checks) to ensure they meet security and performance standards. Servers that pass the certification but later violate regulations will be blacklisted and removed. The system maintains platform security and trust through mechanisms such as regular security scans and real-time monitoring, with any violations leading to immediate removal and further investigation of the server.
- Mechanism for Selecting Server Assistants: When needing to perform PSI, clients choose two cloud servers based on their performance (such as processing power, storage capacity, and network bandwidth), stability, security capabilities, and compliance with regulations, among

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 26 April 2024 doi:10.20944/preprints202404.1701.v1

other hard and soft factors. Cloud servers with high availability promises are preferred to minimize the risk of failures.

- Execution Mechanism for PSI Operations: The PSI network supports client flexibility and system scalability; clients can execute PSI on different database servers by merely changing $E_{pk}(a)$, without needing to redesign the entire system. This design enhances client flexibility and the system's efficiency, reliability, and security.

6.7.2. Summary of the PSI Network

This system design not only achieves the delegation of PSI computations but also introduces multiple cloud servers into the network, thereby enhancing the system's flexibility and stability. Additionally, by implementing authentication and maintaining a blacklist for cloud servers, the system can better guarantee the credibility of the cloud servers, enhancing overall security. This flexible yet secure system design provides clients with more options and makes PSI operations more adaptable to various practical requirements.

In summary, under the existing framework, clients can delegate computing and storage tasks to different cloud servers and perform PSI operations on various database servers by using different random numbers and different $E_{pk}(a)$.This method allows clients to more flexibly use multiple resource nodes and optimize task distribution, thereby further enhancing the overall performance and security of the privacy protection scheme.

6.7.3. Data Updates

To further enhance the practicality of the scheme, this paper also designs a data update mode compatible with the scheme, making the overall scheme more practical and reliable.

- **Data Updates on the Database Server Side:**

As shown in Figure 6, the update details of the database server are as follows:

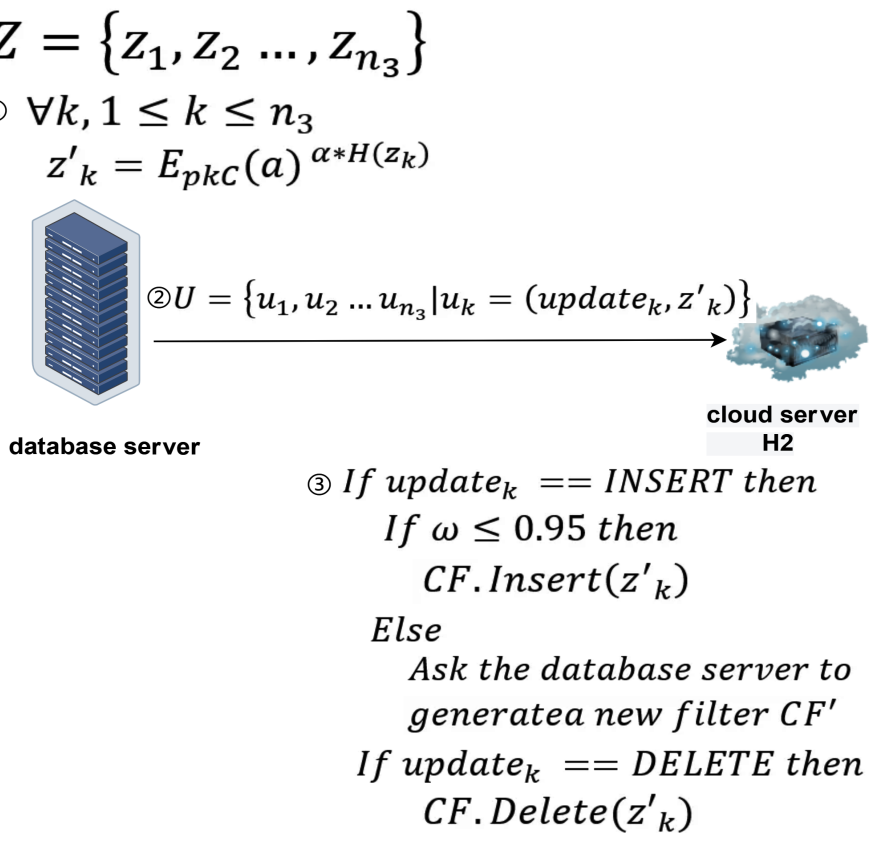

Figure 6. Data Updates on the Database Server Side

 doi:10.20944/preprints202404.1701.v1

Definition of main participants and related symbols:

1. database server: Represents the database server that wants to encrypt and upload updated data to cloud server H_2.
2. cloud server H_2): Represents the cloud-assisted server H_2 that assists the database server in completing update operations.
3. Z represents the set of data to be updated, z_k represents the k-th element of Z.
4. ω represents the load factor of the filter.
5. z'_k represents the data z_k after encryption processing.
6. update$_k$ represents the operation index, used to determine whether the update operation is an insertion or deletion.
7. U represents the set of data sent by the database server to the cloud-assisted server H_2, u_k represents the k-th element of U.

Update process:

1. The database server has a set of elements Z it wants to insert or delete. These elements are blinded before being sent to cloud server H_2. Specifically, $z'_k = E_{pk_c}(a)^{\alpha * H(z_k)}$.
2. In addition to sending the blinded elements, the database server also sends an identifier variable update$_k$ to inform the client whether the operation is an insertion or a deletion.
3. During an insertion operation, H_2 first checks whether the current filter's load factor ω exceeds 0.95.
4. If the load factor ω is greater than 0.95, then H_2 must request the database server to generate a new filter using all elements to maintain high spatial and lookup efficiency of the filter.
5. If the load factor ω is less than or equal to 0.95, then H_2 can directly insert the element into the current filter CF.
6. In a deletion operation, H_2 removes the specified element from the filter CF, a process that does not require generating a new filter.

This section introduces the data update process for the database server under the unbalanced PSI protocol based on single cloud assistance. This series of update operations allows the database server to flexibly handle the insertion and deletion of elements based on the current state of the filter, ensuring the system's efficiency and accuracy..

- **Data Updates on the Client Side:**: As shown in Figure 7, the update details of the database server are as follows:

Definition of main participants and related symbols:

1. client: Represents the client who wants to perform data updates.
2. cloud server H_1: Represents the cloud-assisted server H_1 that assists the database server in completing update operations.
3. cloud server H_2 Represents the cloud-assisted server H_2 that assists the database server in completing update operations.
4. Z represents the set of data to be updated, z_k represents the k-th element of Z.
5. z'_k represents the data after being processed by the hash function H.
6. k represents the data index, used to determine the type of update, either insertion or deletion, and to retrieve the updated data based on the index.
7. When adding data, date$_k$ represents the data processed through the dual-cloud scheme and sent to the two cloud-assisted servers. When deleting, date$_k$ is null.
8. V represents the set of data sent by the database server to the cloud-assisted server H_1, v_k represents the k-th element of V.
9. V' represents the set of data sent by the database server to the cloud-assisted server H_2, v'_k represents the k-th element of V'.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 26 April 2024 doi:10.20944/preprints202404.1701.v1

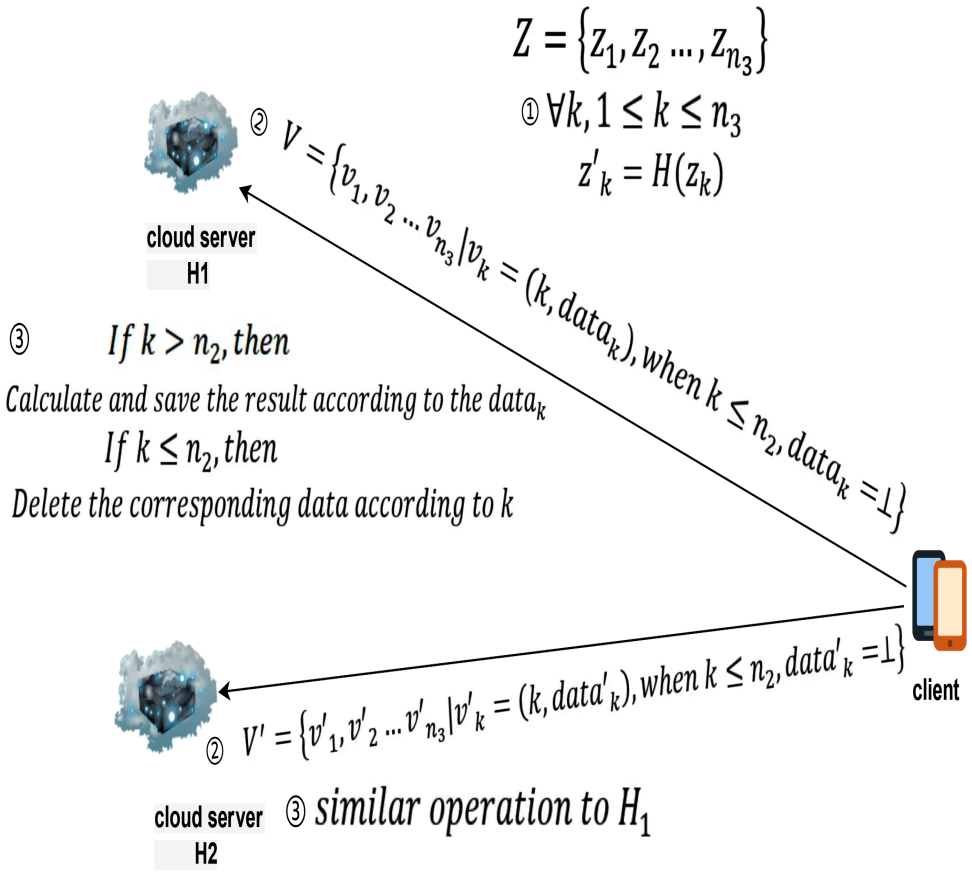

Figure 7. Data Updates on the Client Side

Update process:

1. The client has a set of elements Z it wants to insert or delete. In both cases, the client blinds each element and sends them to H_1 and H_2 respectively.
2. The client sends a data index K to inform the cloud servers about the type of update, whether it is an insertion or a deletion. If the index is less than n_2, it indicates a deletion operation. In this case, data_k is null, and H_1 and H_2 delete the corresponding data based on the index.
3. If the index is greater than n_2, it indicates an addition operation, and the corresponding calculation results and index are saved.
4. After completing a batch of deletion and addition operations, the relative order of the indices also needs to be adjusted. The update process is illustrated in Figure 5.

This section introduces a client data update process based on dual cloud assistance, designed to enhance the database's dynamic management capabilities while ensuring data privacy and efficiency. This update protocol supports both data insertion and deletion operations, and through the cooperation of cloud-assisted servers H_1 and H_2, it optimizes the speed and security of client data updates.

- **Summary of Data Updates:** This section has explored two key data update processes based on the unbalanced PSI protocol based on dual cloud assistance the database server update process and the client update process. Both processes are designed to efficiently handle data insertions and deletions while ensuring data security , and to use cloud server resources to optimize overall operation efficiency.

 doi:10.20944/preprints202404.1701.v1

7. Conclusions and Future Work

7.1. Work Summary

Privacy computing is a technology framework aimed at protecting individual privacy during the process of data use and sharing. It ensures that data can still be effectively utilized without disclosing specific content through various algorithms and protocols. Among many applications of privacy computing, Privacy Set Intersection (PSI) is a common requirement, which allows two or more parties to compute the intersection of their data sets without revealing their exclusive data to each other.

Traditional PSI protocols are primarily designed for cases where data sets are relatively balanced in size, which often does not apply in real scenarios. In many practical situations, the size disparity between participants' data sets is significant, necessitating the use of unbalanced PSI protocols. These protocols are specifically designed to handle such disparities, optimizing computational efficiency and privacy protection to cater to a wider range of practical needs. By adopting unbalanced PSI protocols, not only is the processing efficiency improved, but more precise control over data protection is also offered, thus finding broader application in various data-sensitive industries. This paper proposes three protocols: the unbalanced PSI protocol based on Cuckoo filters, the unbalanced PSI protocol based on single cloud assistance, and the unbalanced PSI protocol based on dual cloud assistance. Here, the unbalanced PSI protocol based on Cuckoo filters addresses performance issues of traditional PSI protocols in handling unbalanced data sets. On this basis, the unbalanced PSI protocol based on single cloud assistance transfers most of the computational and storage burdens from the client to the cloud, enhancing practicality. Faced with the possibility of collusion attacks, the unbalanced PSI protocol based on dual cloud assistance employs security mechanisms such as homomorphic encryption to effectively resist these attacks. The main contributions of this paper are summarized as follows:

1. Addressing the shortcomings of traditional private set intersection protocols when dealing with significant data size disparities among participants, this paper proposes the first protocol, namely the unbalanced PSI protocol based on Cuckoo filters. This protocol successfully constructs a novel private set intersection approach through encrypted exchanges and using Cuckoo filters for private information retrieval.
2. Given the complexities of cryptographic operations and storage demands in the unbalanced PSI protocol based on Cuckoo filters, this paper introduces a unbalanced PSI protocol based on single cloud assistance. This protocol successfully offloads most of the client's computational and storage burdens onto the cloud.
3. In response to potential collusion between the cloud and database servers in the unbalanced PSI protocol based on single cloud assistance, this paper proposes a unbalanced PSI protocol based on dual cloud assistance with security mechanisms like homomorphic encryption, which effectively prevents collusion attacks while offloading computational and storage burdens.
4. Concerning practical issues in the unbalanced PSI protocol based on dual cloud assistance, this paper also introduces a conceptually meaningful PSI network and a data update mode tailored for the unbalanced PSI protocol based on dual cloud assistance.

7.2. Protocol Summary

As shown in Table 6, this section provides a comprehensive summary and recommendations for the three protocols discussed in this paper. The unbalanced PSI protocol based on Cuckoo filters offers high security but involves significant computational and storage demands, making it suitable for clients with strong computational and storage resources. The unbalanced PSI protocol based on single cloud assistance, while being the fastest and offloading computational burdens to the cloud, poses security risks as it cannot withstand collusion attacks, making it appropriate for scenarios where the cloud is fully trusted. The unbalanced PSI protocol based on dual cloud assistance offers an ideal balance of runtime, security, and efficiency, making it the most versatile and practical option.

 doi:10.20944/preprints202404.1701.v1

Table 6. Overall Performance Summary of the Three Protocols

ine **Protocol**	**Security**	**Client Storage & Computational Burden**	**Runtime**
ine Unbalanced PSI Protocol based on Cuckoo Filters	High Security (No collusion attacks)	Requires storing Cuckoo filters and intensive computation	Longest
ine Unbalanced PSI Protocol based on Single Cloud Assistance	Security Risks (Cannot resist collusion attacks)	Shifted to cloud server	Fastest
ine Unbalanced PSI Protocol based on Dual Cloud Assistance ine	High Security (Can resist collusion attacks)	Shifted to cloud server	Moderate

7.3. Future Outlook

Although the protocols proposed in this document are applicable in most scenarios, there are still several aspects that could be optimized for future development:

1. All protocols are designed for two-party unbalanced private set intersections. Extending these protocols to multi-party scenarios is an important future direction, given the practical needs for multi-party computations.
2. The protocols are developed under a semi-honest security model. Extending their robustness to malicious models, where adversaries may actively attempt to undermine the protocols, represents a crucial area for further research.
3. The current protocols focus solely on set intersection. In practical applications, there may be requirements to perform further computations on the intersection results. Developing functionalities to support such computations post-intersection is another significant direction for future work.

References

1. Bald, P.; Baronio, R.; Cristofaro, E.; Gasti, P.; Tsudik, G. Efficient and secure testing of fully-sequenced human genomes. *Biological Sciences Initiative* **2000**, *470*, 7–10.
2. Chen, H.; Laine, K.; Rindal, P. Fast private set intersection from homomorphic encryption. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 1243–1255.
3. Nagaraja, S.; Mittal, P.; Hong, C.Y.; Caesar, M.; Borisov, N. {BotGrep}: Finding {P2P} Bots with Structured Graph Analysis. 19th USENIX Security Symposium (USENIX Security 10), 2010.
4. Li, W.; Liu, J.; Zhang, L.; Wang, Q.; He, C. A Survey on Set Intersection Computation for Privacy Protection. *Journal of Computer Research and Development* **2022**, *59*, 1782–1799.
5. Meadows, C. A More Efficient Cryptographic Matchmaking Protocol for Use in the Absence of a Continuously Available Third Party. Proc. of the 7th IEEE Symposium on Security and Privacy; IEEE Computer Society: Los Alamitos, CA, 1986; pp. 134–134.
6. Huberman, B.; Franklin, M.; Hogg, T. Enhancing Privacy and Trust in Electronic Communities. Proc. of the 1st ACM Conference on Electronic Commerce; ACM: New York, 1999; pp. 78–86.
7. DeCristofaro, E.; Tsudik, G. Experimenting with Fast Private Set Intersection. Proc. of Int. Conf. on Trust and Trustworthy Computing; Springer: Berlin, 2012; pp. 55–73.
8. Pinkas, B.; Schneider, T.; Zohner, M. Faster Private Set Intersection Based on OT Extension. Proc. of the 23rd USENIX Security Symposium; USENIX Association: Berkeley, CA, 2014; pp. 797–812.
9. Freedman, M.; Nissim, K.; Pinkas, B. Efficient Private Matching and Set Intersection. Proc. of the 23rd Int. Conf. on the Theory and Applications of Cryptographic Techniques; Springer: Berlin, 2004. Accessed: 2020-10-16.

 doi:10.20944/preprints202404.1701.v1

10. Freedman, M.J.; Hazay, C.; Nissim, K.; et al.. Efficient Set Intersection with Simulation-Based Security. *Journal of Cryptology* **2016**, *29*, 115–155.
11. Abadi, A.; Terzis, S.; Dong, C. O-PSI: Delegated Private Set Intersection on Outsourced Datasets. Proc of the 27th IFIP International Information Security and Privacy Conference; Springer: Berlin, 2015; pp. 3–17.
12. Kissner, L.; Song, D. Privacy-Preserving Set Operations. Proc of the 25th Annual International Cryptology Conference; Springer: Berlin, 2005; pp. 241–257.
13. Jarecki, S.; Liu, X. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. LNCS 5444: Proc of the 6th Theory of Cryptography Conference; Springer: Berlin, 2009; pp. 577–594.
14. Hazay, C.; Venkitasubramaniam, M. Scalable Multi-party Private Set-Intersection. Proc of the 20th IACR International Workshop on Public Key Cryptography; Springer: Berlin, 2017; pp. 175–203.
15. Dou, J.; Liu, X.; Wang, W.; et al.. Efficient and Secure Calculation of Two-Party Sets in the Field of Rational Numbers. *Chinese Journal of Computers* **2020**, *43*, 1397–1413.
16. Damgård, I.; Pastro, V.; Smart, N.; others. Multiparty Computation from Somewhat Homomorphic Encryption. Proceedings of the 32nd Annual Cryptology Conference; Name, E., Ed.; Springer: Berlin, 2012; Lecture Notes in Computer Science, pp. 643–662.
17. Yao, A.C. Protocols for Secure Computations. Proc of the 23rd Annual Symposium on Foundations of Computer Science (SFCS 1982); IEEE: Piscataway, NJ, 1982; pp. 160–164.
18. Micali, S.; Goldreich, O.; Wigderson, A. How to Play Any Mental Game. Proc of the 19th ACM Symposium on Theory of Computing; ACM: New York, 1987; pp. 218–229.
19. Pinkas, B.; Schneider, T.; Segev, G.; et al.. Phasing: Privateset intersectionusing permutation-basedhashing. Proceedings of the 24th USENIX Security Symposium; USENIX Association, , 2015; pp. 515–530.
20. Pinkas, B.; Schneider, T.; Weinert, C.; et al.. Efficient circuit-based PSI via cuckoo hashing. Proceedings of the 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques; Springer, , 2018; pp. 125–157.
21. Pinkas, B.; Schneider, T.; Tkachenko, O.; et al.. Efficient circuit-based PSI with linear communication. Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques; Springer, , 2019; pp. 122–153.
22. Huang, Y.; Evans, D.; Katz, J. Private Set Intersection: Are Garbled Circuits Better Than Custom Protocols? Proc of the 19th Network and Distributed System Security Symposium; ISOC: Reston, VA, 2012. Accessed: 2020-10-21.
23. Naor, M.; Pinkas, B. Efficient oblivious transfer protocols. SODA, 2001, Vol. 1, pp. 448–457.
24. Dong, C.; Chen, L.; Wen, Z. When private set intersection meets big data: an efficient and scalable protocol. Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, 2013, pp. 789–800.
25. Rindal, P.; Rosulek, M. Improved private set intersection against malicious adversaries. Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer International Publishing, 2017, pp. 235–259.
26. Zhang, E.; Liu, F.H.; Lai, Q.; others. Efficient multi-party private set intersection against malicious adversaries. Proceedings of the 2019 ACM SIGSAC conference on cloud computing security workshop, 2019, pp. 93–104.
27. Pinkas, B.; Rosulek, M.; Trieu, N.; others. PSIfrom PaXoS: Fast, malicious private set intersection. Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 2020, pp. 739–767.
28. Orrù, M.; Orsini, E.; Scholl, P. Actively secure 1-out-of-n OT extension with application to private set intersection. Proceedings of Cryptographers' Track at the RSA Conference. Springer, 2017, pp. 381–396.
29. Rindal, P.; Schoppmann, P. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. IACR Cryptology ePrint Archive, 2021. https://eprint.iacr.org/2021/266.
30. Schoppmann, P.; Gascón, A.; Reichert, L.; others. Distributed vector-OLE: Improved constructions and implementation. Proceedings of the 26th ACM SIGSAC Conference on Computer and Communications Security. ACM, 2019, pp. 1055–1072.

 doi:10.20944/preprints202404.1701.v1

31. Weng, C.; Yang, K.; Katz, J.; others. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for Boolean and arithmetic circuits. Cryptology ePrint Archive, 2020. https://eprint.iacr.org/2020/925.
32. Hill, K. Facebook Figured Out My Family Secrets, And It Won't Tell Me How. *Gizmodo* **2017**. Published on August 25, 2017.
33. Marlinspike, M. Private Contact Discovery for Signal, 2017. Accessed on September 26, 2017.
34. Mittal, P.; Papamanthou, C.; Song, D. Preserving Link Privacy in Social Network Based Systems. NDSS, 2013.
35. Abebe, R.; Nakos, V. Private Link Prediction in Social Networks. Technical report, Harvard University, 2014.
36. Karwa, V.; Raskhodnikova, S.; Smith, A.; Yaroslavtsev, G. Private Analysis of Graph Structure. *PVLDB* **2011**, *4*.
37. Dwork, C. A Firm Foundation for Private Data Analysis. *Communications of the ACM* **2011**.
38. Erlingsson, Ú.; Pihur, V.; Korolova, A. RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response. Proc. of the ACM Conference on Computer and Communications Security (CCS), 2014.
39. Brendel, W.; Han, F.; Marujo, L.; Jie, L.; Korolova, A. Practical privacy-preserving friend recommendations on social networks. Companion Proceedings of the The Web Conference 2018, 2018, pp. 111–112.
40. Su, G.; Xu, M. A Survey on Secure Multi-party Computation Technology and Applications. *Information Communication Technologies and Policy* **2019**, pp. 19–22.
41. Li, A. Research on Multi-party Statistical Computations Based on Functional Encryption. PhD thesis, Wuhan University of Technology, Wuhan, 2017.
42. Wang, H.; Dai, H.; Chen, S.; Chen, Z.; Chen, G. A Survey of Filter Data Structures. *Computer Science* **2024**, *51*, 35–40.
43. Yu, M.; Fabrikant, A.; Rexford, J. BUFFALO: Bloom filter forwarding architecture for large organizations. Proceedings of International Conference on Emerging Networking Experiments and Technologies, 2009, pp. 313–324.
44. Li, P.; Luo, B.; Zhu, W.; et al.. Cluster-based distributed dynamic cuckoo filter system for Redis. *International Journal of Parallel, Emergent and Distributed Systems* **2020**, *35*, 340–353.
45. Wang, F.; Chen, H.; Liao, L.; et al.. The power of better choice: Reducing relocations in cuckoo filter. Proceedings of International Conference on Distributed Computing Systems, 2019, pp. 358–367.
46. Gur, L.; Lis, D.; Dai, H.; et al.. Adaptive online cache capacity optimization via lightweight working set size estimation at scale. Proceedings of USENIX Annual Technical Conference, 2023, pp. 467–484.
47. Reviriego, P.; Martínez, J.; Larrabeiti, D.; et al.. Cuckoo Filters and Bloom Filters: Comparison and Application to Packet Classification. *IEEE Transactions on Network and Service Management* **2020**, *17*, 2690–2701.