**International Journal of Advanced Engineering and Technology Research**
ISSN: 3106-8677 (Print), ISSN: 3106-8685 (Online) | Vol. 1, No. 1, 2026

# A Cloud-Assisted Multi-Party Private Set Intersection Protocol

**Yanru Zeng *, Yufeng Wang**

College of Software, Henan Polytechnic University, Jiaozuo, China

* **Corresponding author**

**Abstract:** In cloud environments, addressing the optimization problem between security and computational and communication overhead in multi-party private set intersection protocols. This paper proposes a multi-party Private Set Intersection protocol utilizing cloud server-assisted computation, enhancing the protocols scalability and efficiency while safeguarding data privacy and security. This protocol is designed under the semi-honest security model, Participants process the elements of the set using keyed pseudorandom functions and hash functions, and introduce a pseudorandom generator to randomize participant information. This is combined with an oblivious key-value pair storage structure to complete data encoding, enabling cloud servers to determine intersections solely based on the encoded results, without access to any raw set information from participating parties. Theoretical analysis demonstrates that the proposed protocol correctly implements the computation of the intersection of multiple sets and satisfies privacy protection requirements under the specified security model.

**Keywords:** Privacy set intersection; Cloud-assisted; Inadvertent key-value pair storage; Pseudorandom function; Privacy security.

## 1. Introduction

The continuous advancement of digitalisation has driven the growth of data scale and the expansion of application scenarios, making the need for collaborative data utilisation across entities increasingly prominent. The effective extraction of data value typically relies on data sharing and collaborative analysis, a process frequently constrained by factors such as data privacy, regulatory compliance, and a lack of trust. Against this backdrop, Secure Multi-Party Computation (MPC) has emerged. It enables participants to jointly complete agreed computational tasks without disclosing their respective raw data, thereby providing a viable technical pathway to resolve the conflict between collaborative computation and privacy leakage. This achieves the core objective of 'data usability without visibility'.

Private Set Intersection (PSI)[1], as one of the principal research areas within secure multi-party computation, permits two or more untrusted parties $P = \{P_1, P_2, ..., P_m\}$ to introduce their private sets as input and compute their intersection, without enabling the inference of any other information. Owing to PSIs broad applicability across diverse fields, it has attracted considerable attention from numerous researchers, such as, In medical research, healthcare institutions may share electronic health records of patients common to their existing patient populations without sharing data pertaining to other users not included within this shared cohort[2]; among commercial institutions, identifying common users based on each organization's customer lists without disclosing user data, in order to provide targeted financial services[3]. In these scenarios, privacy computing offers a viable technical pathway for both parties. Consequently, PSI has become an indispensable core technology component within the privacy computing framework. Kavousi et al[4] proposed a privacy-preserving set intersection protocol for multi-party environments using inadvertent pseudorandom functions. By employing chained communication between clients, they achieved a load-balanced MP-PSI protocol. However, as the number of participants increases and the scale of data expands, traditional multi-party computation faces challenges in balancing security against computational and communication overhead. In recent years, driven by advances in cloud-assisted computing technology, the incorporation of cloud servers with substantial computational and storage capabilities into protocols has emerged as a key research direction for enhancing protocol scalability and practicality. Abadi et al.[5] proposed a method combining additive homomorphic encryption with set-point-value polynomial representations for cloud-assisted delegated PSI computation scenarios, achieving secure computation under a semi-honest adversary model. Fan et al.[6] designed secure computation protocols based on fully homomorphic encryption and constructed a PSI protocol supporting multiple users via a single cloud server under a semi-honest model. Jolfaei et al.[7] designed an outsourced private set intersection cardinality protocol for multi-party environments. This protocol utilizes the ElGamal cryptosystem to enable two or more participants to outsource their private input sets to the cloud. The protocol is secure under the semi-honest model, with the sizes of the parties sets being mutually independent. The computational and communication complexity for each party is independent of the other participants. Although multi-party PSI protocols in cloud-assisted environments can leverage technologies such as public-key cryptography to ensure privacy security, they still face certain limitations in optimizing computational and communication overhead.

Building upon this foundation, this paper proposes a cloud-assisted multi-party private set intersection protocol based on cloud-assisted technology and the OKVS scheme, denoted as $\Pi_{PSI}$. The protocol employs random additional data that can acquire key-value pairs as its core encoding mechanism, utilizing pseudorandom functions and hash functions to generate pseudo-random values that conceal genuine element information. By encoding the collection of random key-value

pairs into an OKVS structure, it achieves secure and efficient data transmission and computation. The protocol then designed an asymmetric interaction process, thereby avoiding direct interaction between the participating parties. Each party completes data preparation locally. Communication between participants involves only the transmission of a pseudorandom number, processed by a pseudorandom number generator, from the central node to each other participant. This prevents potential privacy leaks during interactions. Finally, the cloud server receives this processed data. The protocol will determine that the intersection calculation task should be outsourced to the cloud server. By executing the specified computational tasks, the cloud server completes the intersection determination. In the absence of any additional data, the protocol ensures that all participating parties can fairly obtain the result.

# 2. Relevant technology

## 2.1. Pseudorandom function

Pseudorandom function(PRF)[8] is a deterministic function, Let $F{:}K \times A \to B$ denote a family of functions, $K$ be the key space, $A$ and $B$ be the input and output spaces respectively. There exists a function $F_k{:}\{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ ,that is $b \leftarrow F_k(k,a)$ ,where $k$ denotes the key, and $a,b$ denotes the input and the output. PRFs exhibit determinism, meaning that they produce the same output given the same key and input, as well as unpredictability and computational indistinguishability.

## 2.2. Hash Function

A Hash Function[9] takes input data as parameters and, through a series of complex algorithms, converts it into a binary string of a specific length, which can be represented as $H{:}\{0,1\}^* \to \{0,1\}^{\kappa}$ , $\kappa$ denotes the output length. Hash functions are one-way, meaning it is impossible to deduce the original input data from the hash value. This property enables hash functions to be utilised in cryptography. Moreover, hash functions are deterministic, meaning that for any given input, the same input always produces the same output. Furthermore, hash functions exhibit computational collision resistance, such that it is computationally infeasible to find two distinct inputs that produce the same hash value.

## 2.3. Pseudorandom Generatorv

Pseudorandom Generator (PRG)[10]is one of the fundamental components of modern cryptography and an efficient algorithm. A pseudorandom generator is a deterministic polynomial-time algorithm that takes a string as input and outputs a string. Transforming its outputs statistical randomness into computational security, a secure pseudorandom generator must satisfy the condition that, when faced with an attacker performing computations within probabilistic polynomial time, the generators output is computationally indistinguishable from a genuinely random string of equal length. Pseudorandom generators serve to provide the requisite randomness support for diverse cryptographic protocols. The random sequences they generate exhibit robust unpredictability and pseudorandom characteristics, playing a pivotal role in preventing information leakage and resisting potential attacks. They thereby furnish crucial safeguards for the overall security of cryptographic algorithms. The fundamental functions of a pseudorandom generator are as follows.

Input a random seed $seed \in X$ , where $G{:}\ X \to Y$ denotes a pseudorandom generator capable of producing a long random number $G(seed) \in Y$ . For any adversary $A$ computable in polynomial time, it cannot readily distinguish between a random value $x'$ of equal length and the output $G(seed)$ of the pseudorandom generator, i.e.,

$$|\Pr[A(G(seed)) = 1] - \Pr[A(x') = 1]| \le negl(\cdot) \quad (1)$$

where $negl(\cdot)$ is a function which the probability of success is negligible.

## 2.4. Oblivious key-value store

Oblivious key-value store (OKVS)[11]is a compact and secure data structure designed for efficiently representing and maintaining correct mappings between keys and their corresponding values. It features low communication overhead, a compact structure, and the ability to conceal keys. Primarily composed of encoding and decoding algorithms, it serves as a data structure for storing collection elements, described as follows:

Parameters: key set $K = (k_1, k_2, ..., k_n)$ , value set $V = (v_1, v_2, ..., v_n)$

Encoding algorithm $Encode(K,V)$ ; Input a set of key-value pairs with mapping relationships $\{(k_1, v_1), \ldots, (k_n, v_n)\} \in K \times V$ , output the data structure $X \leftarrow Encode(K,V)$ .

Decoding algorithm $Decode(X,k)$ ; Given the data structure $X$ and a query key $k_i$ , output the corresponding value $v_i$ .if $k_i \in K$ , then $v_i = Decode(X,k_i) \in V$.

Many efficient OKVS schemes possess linear homomorphism. Suppose $X_1$ and $X_2$ are the data structures encoded from two sets of key-value pairs. Then the decoding result satisfies $Decode(X_1 \oplus X_2, k) = Decode(X_1, k) \oplus Decode(X_2, k)$ ,where $X_1 \oplus X_2$ . This property allows the protocol to more easily extend from two parties to multiple parties. Multiple participants can independently or collaboratively perform linear operations on the same OKVS structure, ultimately completing complex multi-party secure computations. Moreover, by shifting substantial computations to the preprocessing stage or employing efficient linear operations to replace complex cryptographic operations, the number of communication rounds and computational latency during the online phase of the protocol can be significantly reduced.

# 3. Protocol Design

## 3.1. System Model

This chapter constructs a multi-party private set intersection protocol model assisted by a cloud server. The protocol involves m participants and a single cloud server. Each participant holds a private set, and the protocol aims to compute the intersection of all m sets without revealing any participant's set contents. The cloud server possesses substantial computational and storage capabilities and assists in executing the computationally intensive tasks of the

protocol, while it cannot access any participant's raw data. Regarding the security model, the protocol is designed and analyzed under the semi-honest model, where all entities strictly follow the protocol but may attempt to infer additional information.

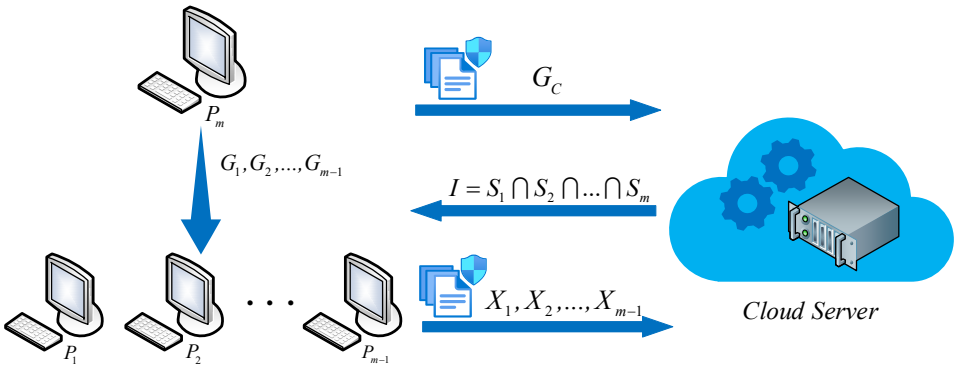


**Figure 1.** Protocol System Model Diagram

## 3.2. Design Approach

The protocol design transforms the set intersection problem into performing specific operations and result verification on the blinded data structures processed between participants $P_i, i = 1,2,...,m-1$ and participant $P_m$. Communication among participants is limited to the central node $P_m$ transmitting the data $G_i$, which has been processed by a pseudorandom generator, to the ordinary participants $P_i, i = 1,2,...,m-1$.The cloud server can only access the data structures $X_i, i = 1,2,...,m-1$ and $G_C$ generated by the OKVS encoding scheme, whose security ensures that the server cannot infer any information about the key-value pairs contained in the set $K_i, i = 1,2,...,m$ that forms these structures.

First, the protocol has the central participant $P_m$ processes the elements in its set $S_m$ using a pseudorandom function $F_k : \{0,1\}^* \to \{0,1\}^\ell$ and a collision-resistant hash function $H : \{0,1\}^* \to \{0,1\}^\kappa$, generating pseudorandom values $F_k(s_j^{(m)})$ and $v_j^{(m)} = H(s_j^{(m)})$. The values corresponding to $s_1^{(m)}, s_2^{(m)}, ..., s_n^{(m)}$ are then formed into the key-value pair set $K_m = \{(v_1^{(m)}, F_k(s_1^{(m)})), (v_2^{(m)}, F_k(s_2^{(m)})), ..., (v_n^{(m)}, F_k(s_n^{(m)}))\}$, which is used for OKVS encoding computation, resulting in $X_m \leftarrow Encode\{K_m\}$.

Next, $P_m$ generates $m-1$ random values $r_i \leftarrow \{0,1\}^\kappa$ for $P_1, P_2, ..., P_{m-1}$ and computes $PRG(r_i) = G_i, i = 1,2,...,m-1$, as well as $G_C = X_m \oplus G_1 \oplus G_2 \oplus ... \oplus G_{n-1}$. It sends the computation result $G_C$ to the cloud server $C$, and sends $G_1, G_2, ..., G_{m-1}$ to $P_1, P_2, ..., P_{m-1}$.Upon receiving $G_i$, each $P_i, i = 1,2,...,m-1$ computes $y_j^i = Decode(G_i, v_j^{(i)})$ with the randomized values $v_j^{(i)} = H(s_j^{(i)})$ of all elements in its set $S_i$. It then forms the key-value pair set $K_i = \{(v_1^{(i)}, y_1^i), (v_2^{(i)}, y_2^i), ..., (v_n^{(i)}, y_n^i)\}$ from the computation results corresponding to $s_1^{(i)}, s_2^{(i)}, ..., s_n^{(i)}$, performs OKVS encoding computation to obtain $X_i \leftarrow Encode\{K_i\}$, where, $s_j^{(i)} \in S_i$, $j = 1,2,...,n$, and sends $X_i$ to the cloud server C.

Finally, after receiving $G_C$ and $X_1, X_2, ..., X_{m-1}$, the cloud server C computes for all elements $s_j^{(N)}, j = 1,2,...,N$ in the set $S_N$:

$$E(s_j^{(N)}) = Decode(X_1, H(s_j^{(N)})) \oplus Decode(X_2, H(s_j^{(N)})) \oplus ... \oplus Decode(X_{m-1}, H(s_j^{(N)})) \oplus Decode(G_C, H(s_j^{(N)})) \oplus F_k(s_j^{(N)})$$

if $s_j^{(N)}$ is an intersection element, then $s_j^{(N)}$ exists within the set $S_i$ and $H(s_j^{(N)}) = H(s_j^{(i)}) = v_j^{(i)}$. Since the $K_i$ used to generate $X_i$ via encoding contains $(v_j^{(i)}, y_j^i)$, when computing $Decode(X_i, H(s_j^{(N)}))$, the result $y_j^i = Decode(G_i, v_j^{(i)})$ —i.e., $Decode(G_i, H(s_j^{(i)}))$ is obtained. If $H(s_j^{(1)}) = H(s_j^{(2)}) = ... = H(s_j^{(m-1)})$ and all exist within $S_1, S_2, ..., S_{m-1}$, then according to the homomorphic properties of OKVS, the following holds:

$$\begin{aligned} &Decode(G_1, H(s_j^{(N)})) \oplus Decode(G_2, H(s_j^{(N)})) \oplus ... \oplus Decode(G_{m-1}, H(s_j^{(N)})) \oplus Decode(G_C, H(s_j^{(N)})) \\ &= Decode(G_1 \oplus G_2, ..., G_{m-1} \oplus G_C, H(s_j^{(N)})) \end{aligned} \tag{2}$$

Since $G_C = X_m \oplus G_1 \oplus G_2 \oplus ... \oplus G_{n-1}$, after these operations, it is only necessary to determine the result of $Decode(X_m, H(s_j^{(N)}))$ .If $H(s_j^{(N)}) = H(s_j^{(m)}) = v_j^{(m)}$, because the $K_m$ used to encode $X_m$ contains the entry $(v_j^{(m)}, F_k(s_j^{(m)}))$, the result of $Decode(X_m, H(s_j^{(N)}))$ will be $F_k(s_j^{(m)})$. Clearly, $H(s_j^{(N)}) = H(s_j^{(i)}), i = 1,2,...,m$, indicating that $s_j^{(N)}$ is an intersection element. The cloud server aggregates all elements satisfying this condition into the intersection $I$ and distributes it to all participating parties $P_i, i = 1,2,...,m$.

## 3.3. Specific Scheme

Parameters: $m$ participants $P_1, P_2, ..., P_i, ..., P_{m-1}, P_m$, each holding a private set $S_i, i = 1,2,...,m$ of size n ; a keyed

pseudorandom function $F_k:\{0,1\}^* \to \{0,1\}^\ell$ ; a collision-resistant hash function $H:\{0,1\}^* \to \{0,1\}^\kappa$ ; a pseudorandom generator $PRG(\cdot)$ ; and a security parameter $\kappa$ .

Input: $P_i, i=1,2,...,m$ with private sets $S_i=\{s_1^{(i)}, s_2^{(i)},...,s_n^{(i)}\}$ .

Output: $I=S_1 \bigcap S_2 \bigcap ... \bigcap S_m$ .

(1). Data Processing Phase

① $P_m$ computes the pseudorandom value $F_k(s_j^{(m)})$ and $v_j^{(m)}=H(s_j^{(m)})$ for all elements $s_1^{(m)}, s_2^{(m)},...,s_n^{(m)}$ in set $S_m$ . . It then forms key-value pairs $(v_j^{(m)}, F_k(s_j^{(m)}))$ and aggregates all such pairs from $S_m$ into a key-value set $K_m=\{(v_1^{(m)}, F_k(s_1^{(m)})),(v_2^{(m)}, F_k(s_2^{(m)})),..., (v_n^{(m)}, F_k(s_n^{(m)}))\}$ , which is then encoded using OKVS to obtain $X_m \leftarrow Encode\{K_m\}$ .

② $P_m$ generates $m-1$ random values $r_i \leftarrow \{0,1\}^\kappa, i=1,2,...,m-1$ , computes $PRG(r_i)=G_i$ , and sends $G_i$ to $P_i (i=1,2,...,m-1)$ . It then computes $G_C=X_m \oplus G_1 \oplus G_2 \oplus ... \oplus G_{n-1}$ and sends $G_C$ and sends $C$ .

③Upon receiving $G_i$ , each $P_i, i=1,2,...,m-1$ computes the hash value $v_j^{(i)}=H(s_j^{(i)})$ for all elements $s_1^{(i)}, s_2^{(i)},...,s_n^{(i)}$ in set $S_i$ . Using $v_j^{(i)}$ , it computes $y_j^i=Decode(G_i, v_j^{(i)})$ , forms key-value pairs $(v_j^{(i)}, y_j^i)$ , and aggregates all such pairs from $S_i$ into a key-value set $K_i=\{(v_1^{(i)}, y_1^i),(v_2^{(i)}, y_2^i),...,(v_n^{(i)}, y_n^i)\}$ . This set is then encoded using OKVS to obtain $X_i \leftarrow Encode\{K_i\}$ , where $s_j^{(i)} \in S_i$ , $j=1,2,...,n$ . Finally, $X_i$ is sent to the cloud server C.

(2). Intersection Phase

After receiving $G_C$ and $X_i, i=1,2,...,m-1$ , the cloud server C computes for each element $s_j^{(N)}$, $j=1,2,...,N$ in the set $S_N$:

$$E(s_j^{(N)})=Decode(X_1, H(s_j^{(N)})) \oplus Decode(X_2, H(s_j^{(N)})) \oplus ... \oplus Decode(X_{m-1}, H(s_j^{(N)})) \oplus$$

$Decode(G_C, H(s_j^{(N)})) \oplus F_k(s_j^{(N)})$ . If the result of the computation is 0, then the element $s_j^{(N)}$ is a part of the intersection. The cloud server aggregates all elements satisfying this condition into the intersection $I$ and distributes it to all participating parties $P_i, i=1,2,...,m$ .

# 4. Protocol Analysis

## 4.1. Correctness Analysis

Theorem 1: The protocol can be securely implemented in the presence of a semi-honest adversary.

Proof: Correctness. According to the protocol flow, the central participant $P_m$ computes $F_k(s_j^{(m)})$ and $v_j^{(m)}$ for each element in the set $S_m=\{s_1^{(m)}, s_2^{(m)},...,s_n^{(m)}\}$ . It then forms the corresponding key-value pair $(v_j^{(m)}, F_k(s_j^{(m)}))$ for the same element, collects the $n$ key-value pairs into a key-value set $K_m$ ,and performs OKVS encoding on this set to obtain the structure $X_m$ . Subsequently, $P_m$ generates random values $r_1, r_2,..., r_{m-1}$ for the other $m-1$ participants, uses a pseudorandom generator to compute $G_i=PRG(r_i), i=1,2,...,m-1$ , and distributes each $G_i$ to the corresponding participant $P_i, i=1,2,...,m-1$ . Finally, $P_m$ computes $G_C=X_m \oplus G_1 \oplus G_2 \oplus ... \oplus G_{m-1}$ and sends it to the cloud server. $P_i, i=1,2,...,m-1$ calculates $v_j^{(i)}=H(s_j^{(i)})$ , then computes $y_j^i=Decode(G_i, v_j^{(i)})$ , and forms a key-value pair $(v_j^{(i)}, y_j^i)$ from the result of each element. $P_i$ then collects the key-value pairs formed from its set elements into a key-value set $K_i$ , performs OKVS encoding to obtain $X_i \leftarrow Encode\{K_i\}$ , and sends $X_i$ to the cloud server.

Based on the security and determinism properties of OKVS, when the cloud server uses the same data $H(s_j^{(N)}), s_j^{(N)} \in S_N$ to decode all received OKVS structures $X_1, X_2,..., X_{m-1}$ and $G_C$ , if this data was an element in the key set during encoding, the corresponding value paired with that key in the key-value pair can be decoded. Furthermore, if the element $s_j^{(N)} \in I$ , i.e., $s_j^{(N)} \in S_i, i=1,2,...,m$ , Then, according to the protocol, the computational procedures required in the cloud server's processing phase are shown in the following formula:

$$\begin{aligned}
E(s_j^{(N)}) &= Decode(X_1, H(s_j^{(N)})) \oplus Decode(X_2, H(s_j^{(N)})) \oplus ... \oplus Decode(X_{m-1}, H(s_j^{(N)})) \oplus Decode(G_C, H(s_j^{(N)})) \\
&= Decode(G_1, H(s_j^{(N)})) \oplus Decode(G_2, H(s_j^{(N)})) \oplus ... \oplus Decode(G_{m-1}, H(s_j^{(N)})) \oplus Decode(G_C, H(s_j^{(N)})) \\
&= Decode(G_1 \oplus G_2,..., G_{m-1} \oplus G_C, H(s_j^{(N)})) \\
&= Decode(G_1 \oplus G_2,..., G_{m-1} \oplus X_m \oplus G_1 \oplus G_2 \oplus ... \oplus G_{m-1}, H(s_j^{(N)})) \\
&= Decode(X_m, H(s_j^{(N)})) \\
&= F_k(s_j^{(N)})
\end{aligned}$$

If $s_j^{(N)} \notin I$ , meaning that the element is not in the set of a certain participant, then when that participant encodes $X_i \leftarrow Encode\{K_i\}$ , we have $(v_j^{(i)}, y_j^i) \notin K_i$ , where $v_j^{(i)}=H(s_j^{(N)})$ . Therefore, $Decode(X_i, H(s_j^{(N)}))$ cannot

compute $Decode(G_i, v_j^{(i)})$ , and in subsequent operations, it is impossible to decode $F_k(s_j^{(N)})$ .

From the above, it can be concluded that the protocol can be correctly implemented.

## 4.2. Security Analysis

For the convenience of the proof, this paper divides the participating entities of the protocol into: the collusion set $P_C \in \{P_1, P_2, ..., P_m\}$ $(1 < |P_C| < m-1)$ , the set of other participants $P_H = \{P_1, P_2, ..., P_m\} - P_C$ , and the cloud serve $C$ . Under the semi-honest participant model, protocol participants will follow the established protocol flow. The following proves that the simulated view obtained according to the protocol process in the ideal model is indistinguishable from the real protocol view.

(1) The simulator generates the view $Sim_{PSI}^{P_C}$ for a collusion set $P_C \in \{P_1, P_2, ..., P_m\}, (1 < |P_C| < m-1)$ that does not contain $P_m$ :

Data Processing Phase: The simulator executes the protocol. Since $P_m$ is not part of $P_C \in \{P_1, P_2, ..., P_m\}$, $Sim_{PSI}^{P_C}$ is not updated at this stage. Simulating $P_i, i = 1, 2, ..., m-1$ After receiving $G_i$ , they compute $Decode(G_i, v_j^{(i)})$ using the random value $v_j^{(i)} = H(s_j^{(i)})$ for each element in their own set $S_i$ . The result $y_j^i = Decode(G_i, v_j^{(i)})$ obtained from the calculation for each element obtained from the calculation for each element $v_j^{(i)}$ to form a key-value pair $(v_j^{(i)}, y_j^i)$ . All key-value pairs for the elements $(v_1^{(i)}, y_1^i), (v_2^{(i)}, y_2^i), ..., (v_n^{(i)}, y_n^i)$ are collected into a set $K_i$ , which is then OKVS encoding to produce the result $X_i \leftarrow Encode\{K_i\}$ , $s_j^{(i)} \in S_i$ ; $j = 1, 2, ..., n$ . Consequently, $Sim_{PSI}^{P_C}(S_i, G_i, v_j^{(i)}, y_j^i, X_i)$ is updated.

Intersection Phase: $P_i$ does not participate, therefore $Sim_{TE-PSI}^{P_C}$ is not updated.

The security of the pseudorandom generator guarantees that $G_i$ is computationally indistinguishable; $v_j^{(i)}$ is a random value generated by participants using a hash function, thus $v_j^{(i)}$ and the key-value pair set are computationally indistinguishable to other participants and the cloud server; $X_i$ is the result of OKVS encoding the key-value pair set, and the security of $v_j^{(i)}$ and $Decode(G_i, v_j^{(i)})$ is guaranteed by the OKVS protocol; $X_i$ is computationally indistinguishable. Furthermore, since $P_m \notin P_C$ , at most $m-1$ participants in $P_C$ possess a given element, making it impossible to determine whether another participants set contains that element. Therefore, $Sim_{PSI}^{P_C}$ and $View_{PSI}^{P_C}$ are computationally indistinguishable.

(2) The simulator generates $Sim_{PSI}^{P_m}$ for $P_C = \{P_m\}$ ;

Data Processing Phase: The simulator simulates $P_m$ according to the protocol, computing for each element $s_j^{(m)} \in S_m, j = 1, 2, ..., n$ the pseudorandom value $F_k(s_j^{(m)})$ and the random value $v_j^{(m)}$ , forming the key-value pair $(v_j^{(m)}, F_k(s_j^{(m)}))$ . All such key-value pairs are collected into the set $K_m = \{(v_1^{(m)}, F_k(s_1^{(m)})), (v_2^{(m)}, F_k(s_2^{(m)})), ..., (v_n^{(m)}, F_k(s_n^{(m)}))\}$ , which is then OKVS encoding to obtain $X_m \leftarrow Encode\{K_m\}$ . Subsequently, random values are generated for the other $m-1$ participants, and $G_i = PRG(r_i), i = 1, 2, ..., m-1$ is computed, which are distributed to $P_1, P_2, ..., P_{m-1}$ . Finally, the simulator emulates $P_m$ in computing $G_C = X_m \oplus G_1 \oplus G_2 \oplus ... \oplus G_{m-1}$ and sends it to the cloud server. Update $Sim_{PSI}^{P_m}(S_m, F_k(s_j^{(m)}), v_j^{(m)}, X_m, G_i, G_C)$ 。

Intersection Phase: $P_m$ does not participate in the reconstruction, so $Sim_{PSI}^{P_m}$ is not updated.

Since the simulator, while emulating $P_m$ , does not participate in the computations of other users, $Sim_{PSI}^{P_m}$ and $View_{PSI}^{P_m}$ are indistinguishable.

(3)The simulator generates $Sim_{PSI}^{P_C}$ for the colluding set $P_C$ $(1 < |P_C| < m-1)$ containing $P_m$ .

Data Processing Phase: The simulator emulates $P_m$ by computing for each element in the set $S_m$ the pseudorandom value $F_k(s_j^{(m)})$ and the random value $v_j^{(m)}$, forming the key-value pair $(v_j^{(m)}, F_k(s_j^{(m)})), s_j^{(m)} \in S_m, j = 1, 2, ..., n$ . All key-value pairs are aggregated into the set $K_m = \{(v_1^{(m)}, F_k(s_1^{(m)})), (v_2^{(m)}, F_k(s_2^{(m)})), ..., (v_n^{(m)}, F_k(s_n^{(m)}))\}$ , which is then OKVS-encoded to obtain $X_m \leftarrow Encode\{K_m\}$ . Then, for the other $m-1$ participants, the simulator computes $G_i = PRG(r_i)$ and $G_l = PRG(r_l)$ , distributing them to the colluding participants $P_i$ and the other honest participants $P_l$ , respectively, where $P_i \in P_C$ , $P_l \in P_H$ . Next, the simulator emulates $P_m$ in computing $G_C = X_m \oplus G_1 \oplus G_2 \oplus ... \oplus G_{m-1}$ and sends it to the cloud server. The simulator also emulates each colluding participant $P_i$ : upon receiving $G_i$ , they compute $Decode(G_i, v_j^{(i)})$ using the random value $v_j^{(i)} = H(s_j^{(i)}), s_j^{(i)} \in S_i$ for each element in their own set $S_i$ . The result $y_j^i = Decode(G_i, v_j^{(i)})$ for each element $s_j^{(i)} \in S_i$ is paired with the corresponding $v_j^{(i)}$ to form the key-value pair $(v_j^{(i)}, y_j^i)$ . All such key-value pairs $(v_1^{(i)}, y_1^i), (v_2^{(i)}, y_2^i), ..., (v_n^{(i)}, y_n^i)$ are aggregated into the set $K_i$ , which is then OKVS-encoded to yield $X_i \leftarrow Encode\{K_i\}$ , $s_j^{(i)} \in S_i$ ; $j = 1, 2, ..., n$ . Update $Sim_{PSI}^{P_C}(S_i, S_m, F_k(s_j^{(m)}), v_j^{(m)}, X_m, G_i, G_l, G_C, v_j^{(i)}, X_i)$ .

Intersection Phase: $P_m$ and $P_i$ do not participate in the reconstruction, so $Sim_{PSI}^{P_C}$ is not updated.

Based on the simulation results from (1) and (2), it is proven that: $F_k(s_j^{(m)}), v_j^{(m)}, X_m, G_i, G_l, G_C, v_j^{(i)}, X_i$ are computationally indistinguishable. Although the simulator possesses the participants $G_i$ values, the colluding set satisfies $|P_C| < m-1$, preventing the simulator from accurately computing the intersection elements. Therefore, $Sim_{PSI}^{P_C}$ and $View_{PSI}^{P_C}$ are indistinguishable.

(4) The simulator generates $Sim_{PSI}^{C}$ for the cloud server $C$:

Data Processing Phase: The cloud server $C$ does not participate, so $Sim_{PSI}^{C}$ is not updated.

Intersection Phase: The simulator emulates the protocol execution. It receives $G_C$ and $X_i, i = 1,2,...,m-1$, and computes

$$E(s_N) = Decode(X_1, H(s_N)) \oplus Decode(X_2, H(s_N)) \oplus ... \oplus Decode(X_{m-1}, H(s_N)) \oplus Decode(G_C, H(s_N))$$

. If $E(s_N) = F_k(s_N)$, the element is added to the intersection set $I_m$. Then, $Sim_{TE-PSI}^{C}(G_C, X_i, E(s_N), I_m)$ is updated.

Since the cloud server $C$ receives $G_C$ and $X_i$, and based on the security properties of OKVS and the pseudorandom generator, $G_C$ and $X_i$ are indistinguishable. Furthermore, the computation process relies solely on OKVS decoding and XOR operations, ensuring that the cloud server $C$ gains no information about the private data. Therefore, $Sim_{PSI}^{C}$ and $View_{PSI}^{C}$ are indistinguishable.

(5)The simulator generates the simulation for the cloud server $C$ colluding with m-1 participants, denoted as $Sim_{PSI}^{C \cup P_C}$:

Case 1: $P_m \notin P_C$

Data Processing Phase: The simulator emulates participants $P_i, i = 1,2,...,m-1$ according to the protocol. For each element in their own set $S_i$, using the random value $v_j^{(i)} = H(s_j^{(i)})$, they perform the calculation $Decode(G_i, v_j^{(i)})$. The result $y_j^i = Decode(G_i, v_j^{(i)})$ obtained for each element $s_j^{(i)} \in S_i$ is then paired with the corresponding $v_j^{(i)}$ to form a key-value pair $(v_j^{(i)}, y_j^i)$. All key-value pairs for the elements $(v_1^{(i)}, y_1^i), (v_2^{(i)}, y_2^i), ..., (v_n^{(i)}, y_n^i)$ are collected into a set $K_i$, which is then OKVS encoding to produce the result $X_i \leftarrow Encode\{K_i\}$, $s_j^{(i)} \in S_i$; $j = 1,2,...,n$. Consequently, $Sim_{PSI}^{C \cup P_C}(S_i, G_i, v_j^{(i)}, y_j^i, X_i)$ is updated.

Intersection Phase: The simulator emulates the protocol execution. It receives $G_C$ and $X_i, i = 1,2,...,m-1$, and computes

$$E(s_N) = Decode(X_1, H(s_N)) \oplus Decode(X_2, H(s_N)) \oplus ... \oplus Decode(X_{m-1}, H(s_N)) \oplus Decode(G_C, H(s_N))$$

If $E(s_N) = F_k(s_N)$, the element is stored in the set $I_m$.

Update $Sim_{PSI}^{C \cup P_C}(S_i, G_i, v_j^{(i)}, y_j^i, X_i, G_C, E(s_N), I_m)$.

Proof based on the simulation results from (1) and (4): $G_i, v_j^{(i)}, y_j^i, X_i, G_C, E(s_N)$ are computationally indistinguishable. Furthermore, the computation process is based solely on OKVS decoding and XOR operations, ensuring that the cloud server $C$ gains no information. Therefore, $Sim_{PSI}^{C \cup P_C}$ and $View_{PSI}^{C \cup P_C}$ are indistinguishable.

Case 2: $P_m \in P_C$

Data Processing Phase: The simulator emulates participant $P_m$ by computing the pseudorandom value $F_k(s_j^{(m)})$ and the random value $v_j^{(m)}$ for each element in the set $S_m$, forming key-value pairs $(v_j^{(m)}, F_k(s_j^{(m)})), s_j^{(m)} \in S_m, j = 1,2,...,n$. All key-value pairs are aggregated into the set $K_m = \{(v_1^{(m)}, F_k(s_1^{(m)})), (v_2^{(m)}, F_k(s_2^{(m)})), ..., (v_n^{(m)}, F_k(s_n^{(m)}))\}$, which is then OKVS encoding to obtain $X_m \leftarrow Encode\{K_m\}$. Subsequently, for the other $m-1$ participants, the simulator computes $G_i = PRG(r_i)$ and $G_l = PRG(r_l)$, distributing them to the colluding participant $P_i$ and other participants $P_l$, respectively, where $P_i \in P_C$, $P_l \in P_H$.

Then, the simulator emulates $P_m$ in calculating $G_C = X_m \oplus G_1 \oplus G_2 \oplus ... \oplus G_{m-1}$ and sends it to the cloud server. The simulator also emulates each colluding participant $P_i$: upon receiving $G_i$, they use the random value $v_j^{(i)} = H(s_j^{(i)}), s_j^{(i)} \in S_i$ for each element in their own set $S_i$ to compute $Decode(G_i, v_j^{(i)})$. The result $y_j^i = Decode(G_i, v_j^{(i)})$ for each element $s_j^{(i)} \in S_i$ is paired with the corresponding $v_j^{(i)}$ to form a key-value pair $(v_j^{(i)}, y_j^i)$. All key-value pairs $(v_1^{(i)}, y_1^i), (v_2^{(i)}, y_2^i), ..., (v_n^{(i)}, y_n^i)$ are aggregated into a set $K_i$, which is then OKVS encoding to produce the result $X_i \leftarrow Encode\{K_i\}$, $s_j^{(i)} \in S_i$; $j = 1,2,...,n$.

Update $Sim_{PSI}^{C \cup P_C}(S_i, S_m, F_k(s_j^{(m)}), v_j^{(m)}, X_m, G_i, G_l, G_C, v_j^{(i)}, X_i)$.

Intersection Phase: The simulator emulates the protocol execution, receiving $G_C$ and $X_i, i = 1,2,...,m-1$, and calculates

$$E(s_N) = Decode(X_1, H(s_N)) \oplus Decode(X_2, H(s_N)) \oplus ... \oplus Decode(X_{m-1}, H(s_N)) \oplus Decode(G_C, H(s_N))$$

. If $E(s_N) = F_k(s_N)$, the element is stored in the set $I_m$.

Update $Sim_{PSI}^{C \cup P_C}(S_i, S_m, F_k(s_j^{(m)}), v_j^{(m)}, X_m, G_i, G_l, G_C, v_j^{(i)}, X_i, E(s_N), I_m)$

.

Based on the simulation results from (3) and (4): $F_k(s_j^{(m)}), v_j^{(m)}, X_m, G_i, G_l, G_C, v_j^{(i)}, X_i, E(s_N)$ are computationally indistinguishable. Moreover, the cloud servers computation process upon receiving $G_C$ and $X_i$ involves only OKVS decoding and XOR operations. Based on the security properties of OKVS, the cloud server gains no information. Although the simulator possesses the participants $G_i$ values, the colluding set satisfies $|P_C| < m-1$, preventing the simulator from accurately computing the intersection elements. Therefore, $Sim_{PSI}^{C \cup P_C}$ and $View_{PSI}^{C \cup P_C}$ are indistinguishable.

From the above, it can be concluded that the protocol can be securely implemented in the presence of a semi-honest adversary.

# 5. Performance Analysis

## 5.1. Theoretical Performance Analysis

To better evaluate the performance of the protocol, this section analyzes its efficiency from three perspectives: the number of communication rounds, communication complexity, and computational complexity.

Communication Rounds: During the data processing phase, $P_m$ locally computes over its elements and performs OKVS encoding to obtain $X_n$. It then uses a pseudorandom generator to generate random values for the other $m-1$ participants, i.e., $G_i = PRG(r_i)$, and computes $G_C = X_n \oplus G_1 \oplus G_2 \oplus ... \oplus G_{n-1}$, $P_m$ sends $G_C$ to the cloud server $C$ and sends $G_i$ to $P_i, i = 1,2,...,m-1$, which constitutes one round of parallel communication among these entities. $P_i, i = 1,2,...,m-1$ locally computes over its own set elements according to the protocol and performs OKVS encoding, then sends the encoded structure $X_i$ to the cloud server, resulting in one round of communication. Finally, the cloud server computes the intersection according to the protocol and returns the result to the participants in one additional round of communication. In total, the protocol involves three communication rounds: one round between $P_m$, $P_i, i = 1,2,...,m-1$ and $C$; one round between $P_i, i = 1,2,...,m-1$ and $C$; and one round between the cloud server $C$ and $P_m$.

Communication Complexity: $P_m$ locally computes over its elements and performs OKVS encoding to obtain $X_n$, incurring a communication cost of $O(n)$. It then generates $G_i = PRG(r_i)$ for the other $m-1$ participants and computes $G_C = X_n \oplus G_1 \oplus G_2 \oplus ... \oplus G_{n-1}$. Participant $P_m$ sends $G_C$ to the cloud server $C$, and sends each $G_i$ to $P_i, i = 1,2,...,m-1$, transmitting a total of $n$ data items. Hence, the total communication cost of $P_m$ is $O(n)$. The communication cost between each $P_i, i = 1,2,...,m-1$ and the cloud server $C$ is $O(1)$. Each $P_i, i = 1,2,...,m-1$ locally computes over its set elements and performs OKVS encoding, with communication complexity $O(n)$. It then sends the encoded structure $X_i$ to the cloud server $C$, resulting in communication complexity $O(n)$ per participant. The total communication complexity of the cloud server $C$ is $O((m-1)n)$. Finally, the cloud server returns the intersection result. Since the size of the intersection is at most $n$, the communication complexity of $C$ in this step is $O(n)$. In summary, the total communication complexity of $P_m$ is $O(2n)$; the total communication complexity of each $P_i, i = 1,2,...,m-1$ is $O(2n+1)$; and the total communication complexity of the cloud server $C$ is $O(mn+1)$.

Computational Complexity: $P_m$ applies the pseudorandom function $F_k: \{0,1\}^* \to \{0,1\}^\ell$ and the hash function $H: \{0,1\}^* \to \{0,1\}^\kappa$, each ordinary participant locally computes the pseudorandom values $F_k(s_j^{(m)})$ and $v_j^{(m)} = H(s_j^{(m)})$, $s_j^{(m)} \in S_m, j = 1,2,...,n$ for $n$ elements in its set, resulting in a computational cost of $O(2n)$; Subsequently, the $n$ pairs of corresponding values $v_j^{(m)}$ and $F_k(s_j^{(m)})$ are constructed into key–value pairs, and the resulting set of $n$ key–value pairs is encoded using OKVS, leading to a total computational cost of $O(n)$. Next, the central participant $P_m$ generates $m-1$ random values and employs a pseudorandom generator $G_i = PRG(r_i)$ to produce $G_C = X_n \oplus G_1 \oplus G_2 \oplus ... \oplus G_{m-1}$, followed by computing the corresponding values, leading to a total computational cost of $O(2m-2)$。Then, $P_i, i = 1,2,...,m-1$ computes the hash values $v_j^{(i)}$ for the $n$ elements in $S_i$,and computes $Decode(G_i, v_j^{(i)})$ for $n$ times, with a computational cost of $O(2n)$, After that, the set consisting of $n$ roups of key–value pairs is encoded via OKVS, which also requires $O(n)$ computational cost. Finally, the cloud server performs $m$ decoding operations and $2m-2$ XOR operations for each element, resulting in a total computational cost of $O(2mN)$. In summary, the total computational cost of the central participant $P_m$ is $O(3n+2m-2)$; the total computational cost of the ordinary participant $P_i (i = 1,2,...,m-1)$ is $O(3n)$; and the total computational cost of the cloud server is $O((3m-2)N)$.

Due to differences in functional objectives and design priorities $O((m-1)n^2)$ $O((m-1)n^2)$ among various protocols, discrepancies in their system architectures are inevitable, which makes rigorous and fair comparisons across protocols challenging. Therefore, this section focuses on multi-party PSI schemes under a cloud-assisted architecture. The theoretical efficiency comparisons of the ordinary participants, the central participant, and the cloud server are summarized in Table X. In the Table 1, let $m$ denote the number of participants, $n$ denote the set size of each participant, $k$ denote the number of hash functions, $\lambda$ denote the security parameter of the Paillier public-key cryptosystem, and $\varepsilon$ denote the Boolean value in the protocol.

**Table 1.** Comparison of communication efficiency among ordinary participants, central participants, and cloud servers

| Protocol | Communication Complexity | | | Computational Complexity | | |
|---|---|---|---|---|---|---|
| | Central Party | Ordinary Party | Cloud | Central Party | Ordinary Party | Cloud |
| **[5]** | $O((m+1)n\lambda+n)$ | $O(n\lambda+n)$ | $O(2mn\lambda+n)$ | $O((2m+1)n^2+mn+3n)$ | $O(n^2+3n)$ | $O((n+2)mn)$ |
| **[6]** | $O((\varepsilon+1)n)$ | $O(\varepsilon n)$ | $O((m+1)\varepsilon n)$ | $O(2mn+3n)$ | $O(3n+1)$ | $O((m-1)n^2)$ |
| **[7]** | $O(m+n+1)$ | $O(2n+1)$ | $O(mn+n)$ | $O((k+1)n+m)$ | $O(2kn+n)$ | $O(mn+n)$ |
| **Ours** | $O(2n)$ | $O(2n+1)$ | $O(mn+1)$ | $O(3n+2m-2)$ | $O(3n)$ | $O((3m-2)N)$ |

## 5.2. Experimental Analysis

This section analyzes the performance of the proposed protocol through simulation experiments. The experimental environment is as follows: Ubuntu 22.04 LTS, a 4-core 1.00 GHz Intel(R) Core(TM) i5-1035G1 CPU, 3 GB of RAM, and implementation in C++. To evaluate the impact of the number of elements in each set on protocol efficiency, the set size is configured as $n=\{10^3,10^4,10^5,10^6\}$, and the number of participants is set to $m=10$. For each parameter combination, the experiment is independently executed 10 times, and the average result is reported to mitigate random fluctuations. To provide a clearer comparison of efficiency, the total running time of the proposed protocol $\Pi_{PSI}$ is compared with that of related protocols, as shown in the Figure 2.

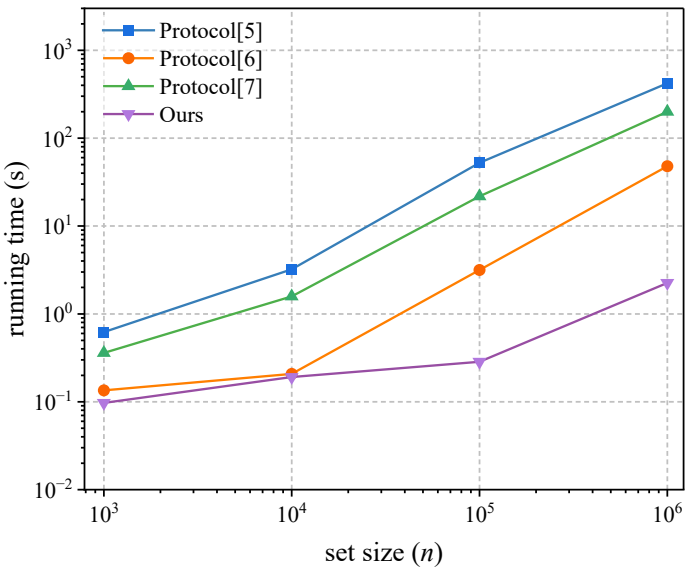


**Figure 2.** The impact of Dataset Size on Protocol Running Time

From the experimental results illustrated in the figure, the proposed protocol $\Pi_{PSI}$ demonstrates consistently stable and satisfactory performance across different set sizes. Its running time increases with the growth of the data scale, but the overall trend remains gradual. For smaller data sets, the protocol $\Pi_{PSI}$ incurs low overhead and minimal baseline costs, while for larger data sets, the running time remains relatively low. As the set size increases, the time cost of all schemes generally grows linearly; however, under identical set sizes, the proposed protocol consistently exhibits lower running time compared to other schemes. Overall, the protocol achieves a certain degree of optimization in time efficiency, mitigating the impact of expanding set sizes on computational overhead.

# 6. Summary

This paper addresses the need to balance privacy protection and computational efficiency in multi-party collaborative data analysis by proposing a cloud-assisted multi-party private set intersection protocol. Under the semi-honest security model, the protocol introduces a cloud server to assist computation and, through participant division of labor and protocol workflow design, reduces local computation and communication overhead. In the protocol design, each participant employs keyed pseudorandom functions and hash functions to randomize set elements, and encodes them using an OKVS structure, so that the cloud server can perform only judgment operations based on the encoded results. The cloud server performs unified computations over the structures transmitted by multiple participants, and an element is included in the final intersection only if it meets the protocol-defined judgment conditions, ensuring correctness and consistency. Under the semi-honest adversary model, the protocol effectively prevents participants from inferring any additional information through protocol interactions.

Although the proposed protocol balances privacy and

efficiency under the semi-honest model, certain limitations remain. Future work may focus on optimizing data processing, reducing encoding redundancy and communication costs, and improving practical applicability. Moreover, the protocol's functionality can be extended to suit specific application scenarios.

# References


[1] Morales Daniel, Isaac Agudo, and Javier Lopez. "Private set intersection: A systematic literature review", Computer Science Review, Vol. 49, pp. 100567, 2023. https://doi.org/10.1016/j.cosrev.2023.100567

[2] Lai Chengzhe, Hanyue Zhang, Rongxing Lu, and Dong Zheng. "Privacy-preserving medical data sharing scheme based on two-party cloud-assisted PSI", IEEE Internet of Things Journal, Vol. 11, No. 9, pp. 15855-15868, 2024. https://doi.org/10.1109/jiot.2024.3350029

[3] Chen Yuchi, and Kuan-Chun Huang. "JEDI: joint and effective privacy preserving outsourced set intersection and data integration protocols", IEEE Transactions on Information Forensics and Security, Vol. 18, pp. 4504-4514, 2023. https://doi.org/10.1109/tifs.2023.3295941

[4] Kavousi Alireza, Javad Mohajeri, and Mahmoud Salmasizadeh. "Efficient scalable multi-party private set intersection using oblivious PRF", In: International Workshop on Security and Trust Management, pp. 81-99, Cham: Springer International Publishing, 2021. https://doi.org/10.1007/978-3-030-91859-0_5

[5] Abadi Aydin, Sotirios Terzis, Roberto Metere, and Changyu Dong. "Efficient delegated private set intersection on outsourced private datasets." IEEE Transactions on Dependable and Secure C omputing, Vol. 16, No. 4, pp. 608-624, 2017. https://doi.org/10.1109/tdsc.2017.2708710

[6] Fan Cunqun, Peiheng Jia, Manyun Lin, Lan Wei, Peng Guo, Xiangang Zhao, and Ximeng Liu. "Cloud-assisted private set intersection via multi-key fully homomorphic encryption." Mathematics, Vol. 11, No. 8, pp. 1784, 2023. https://doi.org/10.3390/math11081784

[7] Jolfaei Amirhossein Adavoudi, Hamid Mala, and Maryam Zarezadeh. "EO-PSI-CA: Efficient outsourced private set intersection cardinality." Journal of Information Security and Applications, Vol. 65, pp. 102996, 2022. https://doi.org/10.1016/j.jisa.2021.102996

[8] Banerjee Abhishek, Chris Peikert, and Alon Rosen. "Pseudorandom functions and lattices." In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 719-737. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. https://doi.org/10.1007/978-3-642-29011-4_42

[9] Damgård Ivan Bjerre. "A design principle for hash functions." In Conference on the Theory and Application of Cryptology, pp. 416-427. New York, NY: Springer New York, 1989. https://doi.org/10.1007/0-387-34805-0_39

[10] Håstad Johan, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. "A pseudorandom generator from any one-way function." SIAM Journal on Computing, Vol. 28, No. 4, pp. 1364-1396, 1999. https://doi.org/10.1137/s0097539793244708

[11] Pinkas Benny, Mike Rosulek, Ni Trieu, and Avishay Yanai. "PSI from PaXoS: Fast, malicious private set intersection." In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 739-767. Cham: Springer International Publishing, 2020. https://doi.org/10.1007/978-3-030-45724-2_25.