symmetry

MDPI

Article

Private Set Intersection Based on Lightweight Oblivious Key-Value Storage Structure

Zhengtao Jiang *,†, Xiaoxuan Guo *,†, Ting Yu, Hanyu Zhou, Jiaqi Wen and Zhengyang Wu

School of Computer and Cyber Science, Communication University of China, Beijing 100024, China; 202120083900010@cuc.edu.cn (T.Y.); zhy0923@cuc.edu.cn (H.Z.); shuyan@cuc.edu.cn (J.W.); wyyw@cuc.edu.cn (Z.W.)

* Correspondence: z.t.jiang@163.com (Z.J.); xiaoxuanguo@cuc.edu.cn (X.G.); Tel.: +86-1314-680-8969 (Z.J.); +86-1768-554-8136 (X.G.)

† These authors contributed equally to this work.

Abstract: At this stage, the application of Private Set Intersection (PSI) protocols is essential for smart homes. Oblivious Key-Value Stores (OKVS) can be used to design efficient PSI protocols. Constructing OKVS with a cuckoo hashing graph is a common approach. It increases the number of hash functions while reducing the possibility of collisions into rings. However, the existing OKVS construction scheme requires a high time overhead, and such an OKVS applied to PSI protocols would also have a high communication overhead. In this paper, we propose a method called 3-Hash Garbled Cuckoo Graph (3H-GCG) for constructing cuckoo hash graphs. Specifically, this method handles hash collisions between different keys more efficiently than existing methods, and it can also be used to construct an OKVS structure with less storage space. Based on the 3H-GCG, we design a PSI protocol using the Vector Oblivious Linear Evaluation (VOLE) and OKVS paradigm, which achieves semi-honest security and malicious security. Extensive experiments demonstrate the effectiveness of our method. When the set size is 2^{18}–2^{20}, our PSI protocol is less computationally intensive than other existing protocols. The experiments also show an increase in the ratio of raw to constructed data of about 7.5%. With the semi-honest security setting, our protocol achieves the fastest runtime with the set size of 2^{18}. With malicious security settings, our protocol has about 10% improvement in communication compared with other existing protocols.

Keywords: private set intersection; oblivious key-value stores; cuckoo hashing graph

Citation: Jiang, Z.; Guo, X.; Yu, T.; Zhou, H.; Wen, J.; Wu, Z. Private Set Intersection Based on Lightweight Oblivious Key-Value Storage Structure. *Symmetry* **2023**, *15*, 2083. https://doi.org/10.3390/sym15112083

Academic Editors: Sergei D. Odintsov, Christos Volos, Konglin Zhu and Pengcheng Wang

Received: 6 October 2023
Revised: 19 October 2023
Accepted: 16 November 2023
Published: 18 November 2023

1. Introduction

With the advent of the Big Data era, user data are generated, collected and consumed in different locations [1]. More potential value can be obtained by integrating and analyzing data scattered in various places. However, the integration of data may bring about leakage of private information and compromise the privacy of users [2]. The application of Private Set Intersection (PSI) protocols has effectively improved this situation [3–9]. Specifically, the adoption of Oblivious Key-Value Stores (OKVS) has gained significant attraction in constructing PSI protocols. This paper aims to tackle the existing challenges related to the high computational complexity and storage demands of OKVS construction. We introduce a novel construction method for OKVS, utilizing a cuckoo hash map. This approach effectively addresses the computational and communication complexities linked to its integration into PSI protocol applications. The method has the potential to enhance privacy and efficiency, particularly in scenarios like federated learning [10]. This method addresses the high computational complexity and high communication overhead of its construction process in PSI protocol applications. Moreover, we are committed to reducing the protocol's computational and communication complexity.

Participants obtain the intersection of sets owned by each other by executing the PSI protocol. And they do not disclose the non-intersection elements they own. Generic

PSI protocol implementations are based on symmetric encryption, public key encryption, Oblivious Transfer (OT), and full homomorphic encryption. In particular, symmetric functions have a wide range of applications in PSI-sum and threshold PSI [11,12]. The PSI protocol includes two main components: the data packaging algorithm and the data transmission algorithm. Algorithms for data transmission include: OT [13], Orrú, Orsini and Scholl (OOS) [14], VOLE [15,16], etc. In the PSI protocol designed in this paper, we have used the VOLE method to implement data transmission. Specifically, VOLE hides the data obtained from encoding through an equational relationship. Data packaging algorithms can be collectively referred to as OKVS [17,18], and our work focuses on improving an OKVS that is less dependent on auxiliary locations.

Existing OKVSs can be categorized into three types based on their structure, i.e., polynomial encoding (PE)-based OKVSs, random matrix-based OKVSs, and cuckoo hashing-based OKVSs. PE is the most basic OKVS structure, which works by interpolating a polynomial through n unstructured points, or computing a polynomial over n points [13,19,20]. However, PE has a high computational overhead for both encoding and decoding and is not efficient in solving encoding and decoding problems with large amounts of data.

The second category, which our work belongs to, is based on the cuckoo hash graph. This class of methods first originated with the work of Pinkas et al. [17], who first proposed Probe-and-XOR of Strings (PaXoS). PaXoS is able to encode a mapping from keys to values to hide the keys. Then, the mapping structure is dispersed into a cuckoo hash map, which outperforms PE in terms of computational overhead. Subsequently, researchers have designed different algorithms based on PaXoS [15,16]. However, PaXoS is only a special binary-type data structure, and this data structure is not universal [17].

To address this problem, Pinkas et al. [18] reconstructed a generic OKVS structure 3H-GCT, using PaXoS as a starting point. Unlike PaXoS, which uses two hash functions to construct the cuckoo hash graph, 3H-GCT increases the number of hash functions for constructing the cuckoo hash graph to three, which greatly improves the efficiency of constructing the cuckoo hash graph. However, the storage requirements of the data structures constructed by 3H-GCT are large, and the computational overhead of the Gaussian elimination method applied to them is also large. When applied to design PSI protocols, 3H-GCT causes a rise in traffic volume, resulting in a large communication overhead.

Aiming at these problems, we investigate the process of encoding cuckoo hash graph for 3H-GCT. In response to the high computational overhead of the Gaussian elimination method, we propose a locally randomized valuation method. To address the problem of high storage requirements, we limit the number of positions to which keys can be mapped to secondary positions, thereby reducing the storage space required. Specifically, we reduce the computational overhead by replacing the Gaussian elimination method with a method that first randomizes the node values, and then determines the null node values from the fixed node values.To address the problem of high storage overhead, we set a location limit during the mapping process of storing data into the cuckoo hashmap. And not every element in our method is stored using auxiliary storage locations. This reduces the dependence on auxiliary locations when constructing the cuckoo hash map to some extent, which ultimately reduces the storage space. To sum up, major contributions of this paper are as follows:

- We propose a new OKVS method, 3H-GCG, whose decoding algorithm consists of two arithmetic formulas, which is less dependent on auxiliary locations, and can be easily adapted to the PSI protocol.
- We leverage the newly proposed 3H-GCG to design a two-party PSI protocol. Our approach incorporates the OKVS and VOLE paradigm [15,16], providing resilience against malicious adversaries. We offer security proofs for our two-party PSI protocol under semi-honest security settings and malicious security settings, respectively.
- We implemented the proposed 3H-GCG and two-party PSI protocols and compared them with existing PSI protocols. When the set size is 2^{18}–2^{20}, our PSI protocol is less computationally intensive than other existing protocols. The experiments show that

for our OKVS, 3H-GCG, the ratio of raw data to constructed data is improved by about 7.5% compared to other existing OKVSs. Under the semi-honest security setting, our PSI protocol achieves the fastest runtime with a set size of 2^{18}. Under the malicious security setting, our PSI protocol has about 10% improvement in communication compared to existing protocols.

The remainder of this paper is organized as follows: in Section 2, we present the symbolic representation of this paper, the security model followed in this paper, and the preparatory knowledge and the related work of the PSI study. In Section 3, we introduce our own OKVS, 3H-GCG, and perform a parametric analysis. In Section 4, we propose the OPRF protocol based on 3H-GCG and further construct the PSI protocol, which can defend against malicious adversaries. In Section 5, we give the correctness analysis and security proof of the protocol. In Section 6, we present the details of our PSI implementation and the performance comparison with existing PSI protocols. In Section 7, we conclude the work of this paper.

2. Preliminaries and Backgrand

This section provides background information. In Section 2.1, we describe the symbolic representations used in this paper. Section 2.2 covers key definitions and properties of OKVS. In Section 2.3, we outline the principles of OKVS implementation using cuckoo hashing. Section 2.4 introduces standard model definitions for semi-honest and malicious security. Lastly, Section 2.5 discusses related work in PSI research.

2.1. Notation

We use X to denote the set of sender and Y to denote the set of receiver. $[x, y]$ denotes the set $\{x, x+1, \ldots, y\}$. $[a]$ denotes the set $[1, a]$. p denotes the row vector $(p_1, p_2, \ldots, p_n)$. $< \cdot, \cdot >$ denotes the inner product operation on vectors; i.e., $< a, b >$ denotes the inner product of a and b. The assignment is noted as :=, and we use = to denote the statement that the values are equal.

2.2. OKVS Structure Definition and Property

Definition 1. *A key-value store (KVS) is characterized by a set of keys (K), a set of values (V), and a set of hash functions (H). It comprises two algorithms:*

- *$Encode_H$: The input is a set of key-value pairs (k_i, v_i) and the output is an object S. In rare cases, an error indicator $\perp$ may be outputted instead.*
- *$Decode_H$: The input is an object S and a key k, and the output is a value v. A KVS is considered correct if for all subsets A of $K \times V$ with distinct keys, the following holds: if (k, v) is in A and $S \leftarrow Encode_H(A) \neq \perp$, then $Decode_H(S, k) = v$.*

The decision of whether $Encode_H$ outputs $\perp$ is determined by the functions H and the keys k_i, and is not influenced by the values v_i. If the data are encoded as a polynomial, $Encode_H$ always succeeds. It is possible to invoke $Decode_H(S, k)$ on any key k. The goal is to make it impossible to determine whether k was used to generate S or not. This is further explained in the following definition [18].

Definition 2. *A KVS is considered as an oblivious KVS (OKVS) if for all distinct sets of keys $\{k_1^0, k_2^0, \ldots, k_n^0\}$ and $\{k_1^1, k_2^1, \ldots, k_n^1\}$, if $Encode_H$ does not output $\perp$ for either set of keys, then the output of $Encode_H\{k_1^0, k_2^0, \ldots, k_n^0\}$ is computationally indistinguishable from that of $Encode_H$ $\{k_1^1, k_2^1, \ldots, k_n^1\}$.*

In other words, if an OKVS encodes random values, it is infeasible to distinguish between an OKVS encoding of the keys of K^0 from an OKVS encoding of the keys of K^1 for any two sets of keys K^0 and K^1. In fact, if the values encoded in the OKVS are random, then the two distributions are perfectly indistinguishable.

Security of OKVS: An OKVS is composed of an encode and a decode algorithm. Encode takes as input a set of key-value pairs (k_i, v_i) and returns a data structure S. Decode takes as

input a data structure like S and a key value k, and outputs a result. Decode can be called on any key, but if it is called on one of the keys used to generate S_{k_i}, then the result is the corresponding v_i. The most fundamental property of an OKVS is that when v_i is random, S hides k_i, reducing the probability that the value of the original data k will be leaked and increasing its security.

2.3. OKVS Construction Based on Cuckoo Hash Graph

This section outlines the construction of the cuckoo hash map proposed in Section 3.2 of document [17]. The implementation details of the original encoding and decoding are described below:

$Encode_H((x_1, y_1), \ldots, (x_n, y_n))$: Given n items (x_i, y_i) where $x_i \in \{0,1\}^*$ and $y_i \in \{0,1\}^l$, let M be an $n \times m$ matrix where the ith row is $v(x_i)$. A data structure (matrix) $D = (d_1, \ldots, d_m)^T \in \{0,1\}^{m \times l}$ can be solved for such that $M \times D = (y_1, \ldots, y_n)^T$. In other words, the following linear system of equations (over the field of order 2^l) is satisfied:

$$\begin{bmatrix} v(x_1) \\ v(x_2) \\ \vdots \\ v(x_n) \end{bmatrix} \times \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_m \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{1}$$

When the $v(x_i)$s are linearly independent, a solution to this system of equations must exist.

$Decode_H(D, x)$: Given a data structure $D \in (\{0,1\}^l)^m$ and a key $x \in \{0,1\}^*$, the corresponding value can be retrieved as follows:

$$y = < v(x_i, D) > = \bigoplus_{j:v(x)_j=1} d_j \tag{2}$$

In other words, solving for a key x in D is equivalent to computing an x-or of a specific position in D. The choice of position is determined by $v(x)$ and depends only on x, not on the data structure D.

For the construction of the data structure D, the paper [17] focuses on the analysis of instantiation with cuckoo hash graphs. The vertices of a cuckoo hash graph are denoted $1, ..., m$, corresponding to the positions of the elements in the data structure D. The edges of the cuckoo hash graph are undirected edges, corresponding to the x_i values to be inserted. The correspondence between the x_i values and the edges in the cuckoo hash graph is $x_i \to \{h_1(x_i), h_2(x_i)\}$. It follows that the cuckoo hash graph may contain self-loops and undirected loops.

2.4. Security Model

In the security proof of the proposed method, we follow the standard security definitions for secure two-party computation. The whole idea of the proof follows the ideal-realistic paradigm. The ideal state means that the participants of the PSI protocol operation truthfully provide the data that should be provided. Strictly following the requirements of the protocol leads to a true intersection result. Specifically, the security certificate is divided into semi-honest security proof and malicious security proof, and the specific requirements are as follows:

Semi-honest security model: For the protocol Π, if there exist probabilistic polynomial-time adversaries S_1 and S_2 such that for all inputs X and Y, the following equation is satisfied:

$$(view_1^{\Pi}(X, Y), out^{\Pi}(X, Y)) \overset{c}{\approx} (S_1(1^n, X, n^2), f(X, Y)) \tag{3}$$

$$view_2^{\Pi}(X, Y) \overset{c}{\approx} S_2(1^n, X, n^2) \tag{4}$$

Then, it means that protocol Π is secure under the semi-honest model. Where $view_1^{\Pi}(X,Y)$ denotes the view of P_1 in protocol Π, $view_2^{\Pi}(X,Y)$ denotes the view of P_2 in protocol Π, $out^{\Pi}(X,Y)$ denotes the output of P_2 in the protocol Π, and $f(X,Y)$ denotes the intersection calculation result of P_2 in the ideal state.

Malicious security model: For protocol Π, for malicious participants P_1 and P_2 under the realistic model that can arbitrarily deviate from the protocol, there exist probabilistic polynomial-time adversaries S_1 and S_2 under the ideal model such that for all inputs X and Y, the following equation is satisfied:

$$Real_1^{\Pi}(X,Y) \stackrel{c}{\approx} (Ideal_{S_1}^{F}(X,Y)) \tag{5}$$

$$Real_2^{\Pi}(X,Y) \stackrel{c}{\approx} (Ideal_{S_2}^{F}(X,Y)) \tag{6}$$

Then, it means that protocol Π is safe under the malicious model. $Real_1^{\Pi}(X,Y)$ denotes the perspective of P_1 when P_1 is a malicious participant under the realistic model, $Ideal_{S_1}^{F}(X,Y)$ denotes the perspective of P_1 when running protocol Π under the ideal model. $Real_2^{\Pi}(X,Y)$ denotes the perspective of P_2 when P_2 is a malicious participant under the realistic model, and $Ideal_{S_2}^{F}(X,Y)$ denotes the perspective of P_2 when running protocol Π under the ideal model.

2.5. Related Work

The idea of polynomial encoding (PE) associated with secure multi-party computation and PSI can be traced back to the work of Manulis et al. [19]. They proposed the concept of index-hiding message encoding (IHME) to solve the privacy-preserving group discovery problem with linear computational and communication complexity. Then, a perfect security construction for IHME using PE is given. Kolesnikov et al. [20] implemented the OPRF protocol using PE, but PE requires a time complexity of $O(n^2)$, which is expensive for large n. Subsequently, Pinkas et al. [21] proposed a polynomial-based PPRF scheme based on the construction of [20] to apportion the cost of batching multiple OPRF calls together. Kolesnikov et al. [22] constructed the reverse private membership test (RPMT) protocol using polynomials and implemented the private set union (PSU) computation protocol using RPMT. Pinkas et al. [13] designed polynomial slicing and streaming using PE and used it to achieve a low communication PSI. Based on the above analysis, existing PE technology suffers from high computational complexity in encoding and decoding.

To solve this problem, the researchers proposed the concept of OKVS. The leading OKVS implementations are currently based on random matrices or cuckoo hashing. Then, the development of OKVS can be traced back to Pinkas et al. [17], who proposed a fast malicious secure two-party PSI protocol using the proposed PaXoS. However, due to the random nature of the results produced by this scheme, there is a high risk of compromising the data information of the participants. Therefore, Rindal et al. [15] designed a variant of PaXoS, XoPaXoS, which effectively solved the data leakage problem. PaXoS has been much more computationally efficient compared to PE, but is still not general enough.

Based on the above gap, Garimella et al. [18] introduced the concept of OKVS using PaXoS as a starting point. According to the concept description, PE belongs to the most basic construct of OKVS, while PaXoS belongs to a specific, binary type of OKVS. A new OKVS structure, 3H-GCT, was designed along with the introduction of the OKVS concept [18]. 3H-GCT expands the number of hash functions of the cuckoo hash map from two to three on the basis of PaXoS, achieving an expansion from the particular to the general. However, the Gaussian elimination method applied by 3H-GCT in the encoding process still places a high demand on computational performance, and in this paper we endeavour to demonstrate protocol solutions with low communication and computation volumes.

3. Proposed OKVS

This section primarily focuses on explaining the encoding and decoding mechanisms of our OKVS. In Section 3.1, we present the codec implementation details, and in Section 3.2, we discuss the parameter selection scheme along with the results for the encoding and decoding mechanisms.

3.1. OKVS Construction Based on Cuckoo Hash Graph

This section begins with the introduction of our proposed OKVS algorithm, which we have named 3H-GCG. The main description of this algorithm is shown in Figure 1, which implements the storage of n key-value pairs into a cuckoo hash graph.

Parameters:

• The algorithm is parameterized with the functions $H = \{h_1, h_2, h_3\}$; each has a range $[m]$. N_i denotes the node of cuckoo hash graph, $i \in [m]$.

• In addition, the algorithm uses the functions $li(\cdot)$ and $ri(\cdot)$, where $li(x)$ outputs a bit-vector of length m with zero at all entries except of entries $h_1(x)$,$h_2(x)$ and $h_3(x)$. The function $ri(x)$ outputs a random bit-vector of length $\kappa + 0.35log(n)$ with zero at all entries except for two random entries.

Algorithm:

$Encode_H((k_1, v_1), \ldots, (k_n, v_n))$:

1. Initialize empty vectors $\boldsymbol{l} \in (\mathbb{F}^l)^m$ and $\boldsymbol{r} \in (\mathbb{F}^l)^{\kappa+0.35log(n)}$.
2. Initialize queue Q.
3. For all key-value pairs (k, v) such that $h_1(k) = h_i(k_a)$, $h_2(k) = h_j(k_b)$, $h_3(k) = h_p(k_c)$($i, j, p \in [3]$ and $a, b, c \in [n]$, but $k_a, k_b, k_c \neq k$), enqueue (k, v) into Q.
4. Store the $k_i \in Q$ into $S := (\boldsymbol{l}, \boldsymbol{r})$.
5. Store the $k_i \notin Q$ into $S := (\boldsymbol{l}, \boldsymbol{r})$.
6. Set any empty position in $\boldsymbol{l}$ (and $\boldsymbol{r}$) with a random value from $\mathbb{F}^m$ (and $\mathbb{F}^r$), output $S = (\boldsymbol{l}, \boldsymbol{r})$.

$Decode_H(\{S, x\})$:

$Decode_1(\{S, x\})$:

1. Compute $li(k_i)$, fetching parameters $S = (\boldsymbol{l}, \boldsymbol{r})$.
2. Return $< li(k_i), \boldsymbol{l} >$.

$Decode_2(\{S, x\})$:

1. Compute $li(k_i)$ and $ri(k_i)$, fetching parameters $S = (\boldsymbol{l}, \boldsymbol{r})$.
2. Return $< (li(k_i), ri(k_i)), (\boldsymbol{l}, \boldsymbol{r}) >$.

Figure 1. The 3-Hash Garbled Cuckoo Graph constructing approach, fitting n key-value pairs (k_i, v_i) to a data structure S.

In this algorithm, the number of hash functions of the cuckoo hash graph is set to 3. The three hash functions are h_1, h_2, h_3, and their mapping ranges are $[m]$. In addition, the node of the cuckoo hash graph is set to N_i and the range of i is also $[m]$, in keeping with the mapping range of the hash function. Further, two mapping functions $li(\cdot)$ and $ri(\cdot)$ are set. In particular, we place some restrictions on the mapping values output by these two functions to make them better serve our OKVS structure. Specifically, the output of the mapping function $li(\cdot)$ is an $m - long$ bit string and all but three positions are zeros, where the three positions of $li(x)$ are determined by $h_1(x), h_2(x)$, and $h_3(x)$, respectively. The mapping function $ri(\cdot)$, on the other hand, outputs an $\kappa + 0.35log(n)$ long string of bits, of which only two random positions are 1 and all other positions are zeros.

The encryption algorithm of 3H-GCG initializes two empty vectors $\boldsymbol{l}$ and $\boldsymbol{r}$, and an empty queue Q, as detailed in steps 1 and 2 of the encoding algorithm in Figure 1. Step 3 aims at classification. Since the mapping range $[m]$ of the hash function is relatively small, it is inevitable that the key values in the set of key-value pairs collide with each other when hashing the values of different hash functions between elements. With the help of queue Q, we can divide the key-value pairs that need to be encoded into two categories. The key-value pairs where all three hashes collide are placed in queue Q, and the rest are placed outside of queue Q. Step 4 is to secretly store the key-value pairs from the queue Q into $\boldsymbol{l}$ and $\boldsymbol{r}$. For k_i, $i \in [m]$ in Q, if $\exists j \in [3]$, $N_{h_j(k_i)}$ is empty, then keep one empty position, fill the other positions with random values if they are empty, and finally decide the value of the retained empty position according to the position that is not empty, such that $< li(k_i), \boldsymbol{l} >$

$= v_i$. Otherwise, compute $ri(k_i)$ and use the random position determined by $r(k_i)$ to assist in storing (k_i, v_i) such that $< (li(k_i), ri(k_i)), (\boldsymbol{l}, \boldsymbol{r}) > = v_i$. Step 5 is to secretly store the key-value pairs outside the queue Q into l and r. For k_i not in Q, if $\exists j \in [3]$, $N_{h_j(k_i)}$ is not empty, then store the value of the empty node based on the nodes already set, such that $< li(k_i), \boldsymbol{l} > = v_i$. Otherwise, take two empty positions and store them randomly, and store the other position according to the position already set, such that $< (li(k_i), ri(k_i)), (\boldsymbol{l}, \boldsymbol{r}) > = v_i$. Note in particular that during the execution of steps 4 and 5, the initialized null vectors $\boldsymbol{l}$ and $\boldsymbol{r}$ are continually updated as key-value pairs are deposited. Until all the key-value pairs are stored, the encoding process is completed by filling in the random values in step 6 to obtain $S := (\boldsymbol{l}, \boldsymbol{r})$.

Unlike existing algorithms, the decoding algorithm of 3H-GCG consists of two algorithms $Decode_1$ and $Decode_2$. The user needs to compute both $Decode_1$ and $Decode_2$ algorithms if they do not know any conditions when decoding. Of the two results obtained, at most one can be matched to the expected result.

Our OKVS construction is constructed with the idea of the cuckoo hash graph and the hypergraph construction. But the concept of cuckoo hashing mentions that if the data cannot be stored anymore, the other deposited elements will be cycled out until a free spot is found to store all the elements. This approach has little efficiency difference in decoding from our proposed 3H-GCG. However, having to constantly determine and restore elements during the encoding process increases computational overhead. Therefore, the position of the elements stored will not change again. If a storage conflict is encountered, the conflict is resolved directly by using the auxiliary position. This makes a considerable improvement to coding efficiency.

In addition, this encoding and decoding approach needs to be used in special scenarios. That is, at least one of the participants needs to have an expectation of the decoded result. If one of the two numbers solved does not follow the number one expects, it is proved that the number of the query is not stored in this OKVS. If one of the two numbers solved can be matched to the expected number before decoding, it is proved that the number one wants to confirm exists at this OKVS check. Based on the above description of the applicable scenarios, we find that this OKVS encoding method can be applied to the PSI protocol. The specific way in which the PSI protocol is constructed is described in detail in Section 4.

3.2. Parameter Analysis

In this work, the parameters to be analyzed are the output length l_1 of H_1, the output length l_2 of H_2, and the number of cuckoo hash graph vertices m and the number of auxiliary positions r. Of these, l_1 and l_2 are introduced when designing the PSI protocol in the next section, and we will first discuss their parameter selection here.

Choices of l_1, l_2: Since H_1 and H_2 control the collision probability of the PSI protocol, l_1 and l_2 can be set to $l_1 = l_2 = \lambda + log(n_1 n_2)$ in a semi-honest security setting, where n_1 and n_2 denote the number of elements of the sender and receiver sets, respectively. Under the malicious security setting, l_1 and l_2 can be set to $l_1 = l_2 = \lambda + log(Q_1 Q_2)$, with Q_1 and Q_2 denoting the number of queries that the sender and receiver can make to the H_1 and H_2 random prediction machines, respectively.

Choices of m, r: In the 3H-GCG mechanism, $S = (\boldsymbol{l}, \boldsymbol{r})$, where $|\boldsymbol{l}| = m$, $|\boldsymbol{r}| = r$. To ensure the success of encoding, $m = 1.2n$ and $r = \kappa + 0.35log(n)$ are taken. In this scheme, the encoding fails when the auxiliary position is less than $\kappa + 0.35log(n)$ at the time of encoding. Refer to [18] for specific calculations.

Based on the aforementioned analysis, we establish a parameter selection scheme tailored to varying numbers of set elements. The security parameters are set to $\lambda = 40$ and $\kappa = 128$. Specifically, we focus on the balanced PSI protocol, where $n_1 = n_2 = n$. The details of this parameter selection scheme are presented in Table 1.

Table 1. Parameter Selection.

n	m	r	l_1&l_2 (Semi-Honest)	l_1&l_2 (Malicious)
2^{16}	$1.2n$	134	72	120
2^{18}	$1.2n$	135	76	122
2^{20}	$1.2n$	135	80	124
2^{22}	$1.2n$	136	84	126
2^{24}	$1.2n$	137	88	128

4. Private Set Intersection

In this section, we outline the PSI protocol scheme we have developed. Section 4.1 introduces the VOLE-based OPRF protocol, while Section 4.2 demonstrates how the OPRF protocol is used to design the VOLE-based PSI protocol.

4.1. OPRF-Based Vector-OLE

We begin by introducing a VOLE function that can be converted at random, which can be adapted to the participant's indication of malicious behaviour, and the whole scheme is resistant to malicious adversaries; the process is illustrated in Figure 2.

Parameters:
- Two parties, a sender and a receiver.
- A finite field $\mathbb{F}$.
- The size of the output vectors: m.

Functionality:
- No input from either sender or receiver.
- In the event that the receiver exhibits malicious behavior, wait for the transmission of C and $A \in \mathbb{F}^m$. Subsequently, sample $\Delta \leftarrow \mathbb{F}$ and calculate $B = C - \Delta A$.
- In the event that the sender exhibits malicious behavior, wait for the transmission of $B \in \mathbb{F}^m$ and $\Delta \in \mathbb{F}$. Subsequently, sample $A \leftarrow \mathbb{F}^m$ and calculate $C := B + \Delta A$.
- If there is no malicious behaviour on either side, sample A and $B \leftarrow \mathbb{F}^m$, as well as $\Delta \leftarrow \mathbb{F}$, then calculate $C := B + \Delta A$.
- The functionality transmits Δ and B to the sender, while sending C, calculated as $\Delta A + B$, and transmits A to the receiver.

Figure 2. A random reversed VOLE functionality realization.

The whole VOLE scheme is adapted to both participants, with a restricted finite field of $\mathbb{F}$ and all vectors of dimension set to m. No input is required from either participant for the entire operation. If no malicious adversary is detected, vectors A and B and the scalar Δ are randomly selected by the VOLE processor, then it can be calculated that $C := B + \Delta A$. If a malicious adversary is present during the implementation of Figure 2, the implementation needs to be discussed on a case-by-case basis:

If receiver is a malicious adversary, receiver needs to generate its own A and C for transmission to the VOLE processor; the processor takes a random value Δ and computes $B := C - \Delta A$, returning B and Δ to sender.

If sender is a malicious adversary, sender needs to generate the scalar Δ and B by himself and transmit both to the VOLE processor. The process then randomizes A and then calculates $C := B + \Delta A$ and transmits A and C to the receiver.

Based on the above random reversed VOLE function that can resist a malicious adversary, an OPRF protocol can be constructed to resist a malicious adversary. The specific OPRF execution process is shown in Figure 3.

Parameters:
- There exists two parties, a sender with set $X \subset \mathbb{F}$ of size n, and a receiver with set $Y \subset \mathbb{F}$ of size n.
- Linear 3H-GCG scheme (Encode, Decode), mapping n items to m slots.
- Computational and statistical security parameters κ and λ.
- Random oracles H_1:$\{0,1\}^* \to \{0,1\}^t$ and H_2:$\{0,1\}^* \to \{0,1\}^l$ and H_3:$\{0,1\}^{l_1} \to \{0,1\}^{l_2}$.

Protocol:
1. Sender samples $w^s \leftarrow \mathbb{F}$ and sends $s = H_3(w^s)$ to receiver.
2. Receiver computes D with Y and sends the randomly generated w^r to sender. Note that the data structure of D is consistent with the output data structure of 3H-GCG.
3. The parties invoke the VOLE functionality and neither party has any input. The receiver obtains C and A', while the sender obtains B and Δ, so that $C = B + \Delta A'$. Note that C, $A'$and B are all OKVS structures of 3H-GCG, like D.
4. Receiver computes $A := A' + D$ and sends A to the sender.
5. Sender sends w^s to receiver, who aborts if $H_3(w^s) \neq s$. Both parties then compute $w = w^s + w^r$.
6. Sender computes $P = \Delta A + B$ and then outputs $M_1 = \{H_2(x, Decode_1(P, x) - \Delta H_1(x) + w) \mid x \in X\}$ and $M_2 = \{H_2(x, Decode_2(P, x) - \Delta H_1(x) + w) \mid x \in X\}$.

Figure 3. OPRF based on the 3H-GCG.

The receiver receives C and A' and the sender receives B and scalar Δ where C, A', and B, the same as D, are OKVS structures output by the 3H-GCG encoding algorithm. The receiver assigns $A' + D$ to A, which is then sent to the sender. The sender sends the w^s generated in the first step to the receiver, who checks the w^s against the s previously received. If s is not equal to $H_3(w^s)$, then the agreement aborts; if the two values are equal, then both parties can calculate $w = w^s + w^r$, respectively. Finally, the sender computes $P = \Delta A + B$ and uses P to compute two decodes of 3H-GCG for the elements in the set X, and the obtained results are integrated into the M_1 and M_2 sets, respectively. The reason for having two sets here is that 3H-GCG has two decryption algorithms. The security of this OPRF protocol is demonstrated in the next subsection.

4.2. PSI Protocol Description

We can adapt the OPRF protocol discussed in the previous section to create a PSI protocol in the same security environment. This adaptation is illustrated in Figure 4. Notably, the PSI protocol in Figure 4 primarily differs from the OPRF protocol in Figure 3 in just two key aspects.

Parameters:
- There exists two parties, a sender with set $X \subset \mathbb{F}$ of size n, and a receiver with a set $Y \subset \mathbb{F}$ of size n.
- Linear 3H-GCG scheme (Encode, Decode), mapping n items to m slots.
- Computational and statistical security parameters κ and λ.
- Random oracles H_1: $\{0,1\}^* \to \{0,1\}^t$, H_2: $\{0,1\}^* \to \{0,1\}^l$, H_3: $\{0,1\}^{l_1} \to \{0,1\}^{l_2}$ and H_3:$\{0,1\}^{l_1} \to \{0,1\}^{l_2}$.

Protocol:
1. Sender samples $w^s \leftarrow \mathbb{F}$ and sends s = $H_3(w^s)$ to receiver.
2. Receiver computes $D = Encode(\{(y, H_1(y)) \mid y \in Y\}) = (L_D, R_D)$ using the Encode algorithm of 3H-GCG and sends the randomly generated w^r to sender.
3. The parties invoke the VOLE functionality and neither party has any input. The receiver obtains C and A', while the sender obtains B and Δ, so that $C = B + \Delta A'$. Note that C, A', and B are all OKVS structures of 3H-GCG, the same as D.
4. Receiver computes $A := A' + D$ and sends A to the sender.
5. Sender sends w^s to receiver who aborts if $H_3(w^s) \neq s$. Both parties then compute $w = w^s + w^r$.
6. Sender computes $P = \Delta A + B$ and then computes $M_1 = \{H_2(x, Decode_1(P, x) - \Delta H_1(x) + w) \mid x \in X\}$ and $M_2 = \{H_2(x, Decode_2(P, x) - \Delta H_1(x) + w) \mid x \in X\}$, then sends M_1, M_2 to receiver.
7. Receiver outputs $\{y \in Y \mid (H_2(y, Decode_1(C, y)) + w \in M_1) \vee (H_2(y, Decode_2(C, y)) + w \in M_2)\}$

Figure 4. The PSI protocol based on 3H-GCG.

The first place is where step 2 of the protocol accounts for the calculation of D, which is derived by applying the coding algorithm of 3H-GCG to the receiver's set. The second place is the addition of a receiver output intersection at the end of the protocol. There

is a practical detail in this intersection result calculation that reduces the receiver's final intersection matching effort. When constructing D, the results of sorting elements with the queue can be left in place and elements not in the queue can be looked up directly in M_1. In the case of elements in the queue, a lookup is required in both M_1 and M_2. The whole process is illustrated in Figure 5. During the encoding process, the participants in Figure 5a completed the encoding themselves. During decoding, the encoder sends the encoded data S to the decoder. The decoder decodes with its own value and sends the decoded value to the encoder for matching. This decoding algorithm is more like probing whether the decoder's value is encoded into S or not.

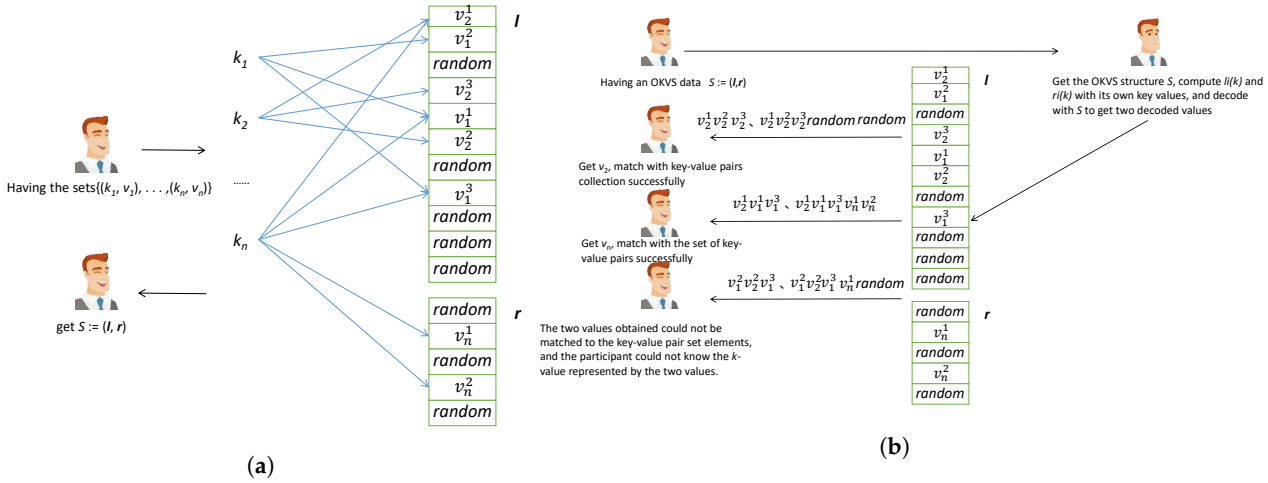

Figure 5. Demonstration of 3H-GCG encoding and decoding process. (**a**) Encoding process; (**b**) decoding process.

5. Theoretical Analysis

The main work in this section is to analyze the performance of the PSI protocol scheme in Section 4. In Section 5.1, we analyze the correctness of our PSI protocol. In Section 5.1 we analyze the security of our PSI protocol.

5.1. Correctness Analysis

In the following we give a correctness analysis of the PSI protocol in Section 4.2. For the intersection part $x = y$, we only need to prove that $Decode(C, y) = Decode(P, x) - \Delta H_1(x)$. The detailed derivation process is as follows:

$$\begin{aligned} Decode(C, y) &= Decode(P, x) - \Delta H_1(x) \\ &= Decode(\Delta A' + \Delta D + B, x) - \Delta H_1(x) \\ &= Decode(C + \Delta D, x) - \Delta H_1(x) \\ &= Decode(C, x) + Decode(\Delta D, x) - \Delta H_1(x) \\ &= Decode(C, x) + \Delta Decode(D, x) - \Delta H_1(x) \\ &= Decode(C, x) \end{aligned}$$

The Decode algorithm in the derivation of the above equation is generally applicable to the decryption algorithms $Decode_1$ and $Decode_2$ of 3H-GCG in Section 3.1.

For the non-intersection part elements put into the $Decode_1$ and $Decode_2$ algorithms, the value decrypted is a random value and the receiver cannot match this random value with its own elements. Therefore, our proposed PSI protocol is correct.

5.2. Security Analysis

Our PSI protocol is resistant to brute force decryption attacks, statistical attacks, and man-in-the-middle attacks due to the masking of encoded data in the protocol. Since

it is stated in the literature [18] that a protocol secure in a malicious environment is not necessarily secure in a semi-honest environment, our security analysis is divided into two main parts. The first part is to demonstrate that the OPRF protocol proposed in Section 4.1 is resistant to malicious adversaries. The second part proves that the PSI protocol we proposed in Section 4.2 is resistant to attacks by malicious adversaries. The security proof satisfies the semi-honest security model and the malicious security model introduced in Section 2.4.

Theorem 1. *The PSI protocol proposed in Figure 4 has semi-honest security under the model where H_1 and H_2 are random prediction machines and VOLE variant models.*

Proof. We prove Theorem 1 by the following two lemmas. □

Lemma 1. *The PSI protocol proposed in Figure 4 can resist semi-honest sender under the model where H_1 and H_2 are random prediction machines and VOLE variants.*

Proof. We construct the simulator S_1 as follows. Given the set of receivers Y, the S_1 runs the honest sender's protocol to generate its perspective, but with the following differences: for the VOLE machine, S_1 runs the VOLE simulator, which generates a sender-side perspective. Finally, the simulator S_1 outputs the sender's view. We prove $(view_1^{\Pi}(X,Y), out^{\Pi}(X,Y)) \overset{c}{\approx} (S_1(1^n, X, n^2), f(X,Y))$ by the following mixed arguments:

Hybrid 0: Sender's view and receiver's output under the real protocol.

Hybrid 1: Same as Hybrid 0, except that instead of running the VOLE mechanism, S_1 runs the VOLE simulator, which generates the structures A', B, C, and the scalar Δ, satisfying the equation $C = B + \Delta A'$. The VOLE simulator sends B and Δ to the sender. This Hybrid is exactly the same as Hybrid 0.

Hybrid 2: Same as Hybrid 1, except that in this variant, S_1 selects "w_s" based on specific conditions. This variation holds statistical equivalence to Hybrid 1.

Hybrid 3: Same as Hybrid 2, except that the protocol terminates when there exists $x_1, x_2 \in X \cup Y$ with $x_1 \neq x_2$ such that $H_1(x_1) = H_1(x_2)$. This termination probability is negligible because H_1 is collision resistant.

Hybrid 4: Same as Hybrid 3, except that there exist $x_1, x_2 \in X \cup Y$ with $x_1 \neq x_2$ such that one of the following equations is satisfied, then the agreement aborts.

$$H_2(x_1, Decode_1(P, x_1) - \Delta H_1(x_1) + w) = H_2(x_2, Decode_1(P, x_2) - \Delta H_1(x_2) + w) \quad (7)$$

$$H_2(x_1, Decode_2(P, x_1) - \Delta H_1(x_1) + w) = H_2(x_2, Decode_2(P, x_2) - \Delta H_1(x_2) + w) \quad (8)$$

$$H_2(x_1, Decode_1(C, x_1) + w) = H_2(x_2, Decode_1(C, x2) + w) \quad (9)$$

$$H_2(x_1, Decode_2(C, x_1) + w) = H_2(x_2, Decode_2(C, x2) + w) \quad (10)$$

The protocol is designed to terminate immediately upon the occurrence of any of these specified conditions. This rapid termination mechanism is an integral part of the protocol's efficiency and security measures. It ensures that in the event of certain predefined situations, the protocol can gracefully and swiftly exit without proceeding further.

The termination probability in these cases is intentionally kept at an extremely low level. This is primarily attributed to the inherent collision resistance of H_2, a fundamental cryptographic primitive used within the protocol. The reliance on H_2's collision resistance properties adds an additional layer of security to the termination process, further reducing the likelihood of any undesired or unexpected outcomes.

It is important to emphasize that this termination mechanism is a crucial component of the protocol's robustness, ensuring that it can withstand various adversarial scenarios while maintaining its integrity and security.

Hybrid 5: Same as Hybrid 4, except that the output of P_2 is replaced with $f(X, Y)$. The final output will change when and only when $x \in X, y \in Y, x \neq y$ but $H_2(x, Decode1(P, x) - H_1(x) + w) = H_2(y, Decode1(C, y) + w)$. Since H_2 is collision resistant and the parameters of l_2 are chosen large enough, the probability of output change is negligible.

Hybrid 6: Same as Hybrid 5, except that the protocol will no longer abort. The indistinguishability between Hybrid 5 and Hybrid 6 is based on the collision resistance of H_1 and H_2 and the conversion of the VOLE. □

Lemma 2. *The PSI protocol proposed in Figure 4 can resist the semi-honest receiver under the model where H_1 and H_2 are random prediction machines and VOLE variants.*

Proof. We construct the simulator S_2 as follows. Given the set of senders X, S_2 runs the honest receiver's protocol to generate its perspective, but with the following differences: S_2 randomly generates 3H-GCG coding structure D. For each element $x \in I$, x is decrypted according to the decoding method of 3H-GCG ($Decode_1$, $Decode_2$) to obtain the two values $M_1(x)$ and $M_2(x)$. S_2 runs the VOLE simulator to generate the vectors C, A', B, and the scalar Δ, and sends vectors C and A' to the receiver. We prove $view_2^{\Pi}(X, Y) \overset{c}{\approx} S_2(1^n, X, n^2)$ by the following mixed arguments:

Hybrid 0: Receiver perspective under the real protocol.

Hybrid 1: Same as Hybrid 0, except that the protocol aborts when there exists $x_1, x_2 \in X \cup Y$ and $x_1 \neq x_2$ such that $H_1(x_1) = H_1(x_2)$. This abort probability is negligible because H_1 is collision-resistant.

Hybrid 2: Same as Hybrid 1, except that when there exists $x_1, x_2 \in X \cup R$, R is the set consisting of random values, and $x_1 \neq x_2$ such that one of Equations (7)–(10) is satisfied, then the protocol aborts .

Hybrid 3: Same as Hybrid 2, except that instead of running the VOLE mechanism, S_2 runs the VOLE simulator, which generates the structures A', B, C and the scalar Δ, satisfying the equation $C = B + \Delta A'$. The VOLE simulator sends A' and C to the receiver. This Hybrid is identical to Hybrid 2.

Hybrid 4: Same as Hybrid 3, Hybrid 4 introduces a slight variation in the protocol, specifically involving the actions of S_2. In this modified hybrid, S_2 plays a unique role by considering the intersection set I, which represents the common elements between the two parties' datasets. Additionally, S_2 generates a set of $n_1 - |I|$ random values and employs the sophisticated decoding algorithm of 3H-GCG, utilizing both $Decode1$ and $Decode2$ procedures.

The outcome of this process yields two sets of decoded values, denoted as M_1 and M_2, which S_2 subsequently transmits to the receiver. It is essential to highlight that the randomly generated values remain entirely concealed from the receiver throughout this operation, ensuring data privacy and security.

One noteworthy aspect of Hybrid 4 is the statistical indistinguishability it shares with Hybrid 3. This property signifies that an external observer or attacker would find it exceptionally challenging to differentiate between the two hybrids based on the information available to them. This inherent similarity adds an extra layer of cryptographic strength and resilience to the protocol, contributing to its overall security.

Hybrid 5: Same as Hybrid 4, except that the protocol no longer aborts. Hybrid 4 and Hybrid 5 are indistinguishable due to the collision resistance of H_1 and H_2 and the convertibility of the VOLE. □

Theorem 2. *The PSI protocol proposed in Figure 4 has malicious security under the model where H_1 and H_2 are random prediction machines and VOLE variant models.*

Proof. First assume that the sender is malicious and sets up simulator M. The interaction between the malicious sender and simulator M is as follows:

1. Before the participating parties start executing the PSI protocol, the simulator M waits for the messages y sent by the malicious sender instead of the OPRF model, and all the y sent from the set Y'.
2. After the malicious sender sends M_1 and M_2 to the simulator M, M computes $Y_1 := \{y \in Y' \wedge y' \notin Y' \text{ s.t. } y \neq y' \wedge F(y) = F(y')\}$ and then execute the PSI protocol with Y_1. Since the simulator has ruled out the collision case $F(y) = F(y')$, the above simulation is correct and indistinguishable. However, there exists the case $F(y) = F(y')$ where the probability of having an element x in the set X of senders satisfying $F(y') = F(x)$ is $2^{-\lambda}$, which is overall indistinguishable. We prove it using the following hybrid argument:

Hybrid 0: Same protocol as the original, but the simulator M interacts with the malicious sender instead of the VOLE.

Hybrid 1: Similar to Hybrid 0, except that A is generated by sampling under a uniform distribution instead of summing over A' and D. In the perspective of the malicious sender, A' is also uniformly distributed, so Hybrid 1 is indistinguishable from Hybrid 0.

Hybrid 2: Similar to Hybrid 1, except that the simulator M no longer calls the Encode algorithm, so the protocol does not terminate even if the Encode algorithm fails to encode. Since the cuckoo hash graph OKVS generated by 3H-GCG has not been queried and is therefore also randomly sampled, the upper bound on the probability of protocol termination is $2^{-\lambda}$. So, Hybrid 2 is statistically indistinguishable from Hybrid 1. It is worth noting that any input from the receiver is no longer used in this hybrid.

Hybrid 3: Similar to Hybrid 2, except that the protocol terminates if the query $H_2(x, Decode(P, x) - \Delta H_1(x) + w)$ has been queried before. Otherwise, the simulator M invokes the decoding step of the OPRF protocol to respond to the query. Since the queries $H_2(x, Decode(P, x) - \Delta H_1(x) + w)$ are all similarly distributed and the malicious sender has negligible probability to query $H_2(x, Decode(P, x) - \Delta H_1(x) + w)$, Hybrid 3 is indistinguishable from Hybrid 2.

Assuming the recipient is malicious and sets up simulator M, the interaction between the malicious recipient and simulator M is recorded as follows:

1. The simulator M replaces the OPRF model and when the malicious recipient sends a query message to M, the simulator records the set X of messages sent by the malicious recipient and sends the response message after the query to the malicious recipient.
2. The simulator executes the PSI protocol with the collected X and the Y held by itself as input to obtain the intersection set Z.
3. The simulator uses the elements in Z and the consistent values of the non-intersecting elements in Y to input into OPRF for calculation, the calculated values are assembled into Y', and Y' is sent to the malicious recipient.

Consistent values for the non-intersecting elements of Y are added throughout the third step of the simulation, and only this differs from the real protocol. However, since such consistent values occur with probability $2^{-\lambda}$, this change is indistinguishable. We prove it using the following hybrid argument:

Hybrid 0: Same protocol as the original, but the simulator M interacts with the malicious sender instead of the VOLE.

Hybrid 1: Similar to Hybrid 0, except that A is generated by sampling under a uniform distribution instead of summing over A' and D. In the perspective of the malicious sender, A' is also uniformly distributed, so Hybrid 1 is indistinguishable from Hybrid 0.

Hybrid 2: Similar to Hybrid 1, except that the simulator M no longer calls the Encode algorithm, so the protocol does not terminate even if the Encode algorithm fails to encode. Since the cuckoo hash graph OKVS generated by 3H-GCG has not been queried and is therefore also randomly sampled, the upper bound on the probability of protocol termina-

tion is $2^{-\lambda}$. So Hybrid 2 is statistically indistinguishable from Hybrid 1. It is worth noting that any input from the receiver is no longer used in this hybrid.

Hybrid 3: Similar to Hybrid2, except that the protocol terminates if the query $H_2(x, Decode(P, x) - \Delta H_1(x) + w)$ has been queried before. Otherwise, the simulator M invokes the decoding step of the OPRF protocol to respond to the query. Since the queries $H_2(x, Decode(P, x) - \Delta H_1(x) + w)$ are all similarly distributed and the malicious sender has negligible probability to query $H_2(x, Decode(P, x) - \Delta H_1(x) + w)$, Hybrid 3 is indistinguishable from Hybrid 2. □

6. Theoretical Efficacy Analysis and Experimental Comparison

The main work in this section is to analyze the efficacy of the 3H-GCG designed in Section 3 and the PSI protocol designed in Section 4 of this paper in terms of theoretical efficacy and the efficacy of the experimental implementation results.

The experiments were conducted on a host machine with the specifications of a LAPTOP-7CRPMU9N, featuring an AMD Ryzen 7 5800H processor clocked at 3.20 GHz. This system ran a 64-bit operating system and utilized an x64-based processor. For experimentation purposes, a virtual machine running Ubuntu 22.04 was employed. The virtual machine was allocated 8 GB of RAM, creating an isolated environment for conducting the research.All of our experiments are implemented in C/C++, including our work as well as replications of existing work. The VOLE mechanism was used in an extended version of Schoppmann et al. [23].

6.1. OKVS Part

Figure 6 illustrate the bit sizes of the post-coding structures of the individual OKVS structures. Figure 6 shows a comparison of the data in the semi-honest setting and in the malicious setting where the horizontal coordinate represents the logarithmic value of the set size owned by the participant with a base of two, and the vertical coordinate is the number of bits of the OKVS structure. The independent variable for both line graphs is the growth in the size of the number of elements of the set, and it grows exponentially. The random matrix is the method of reference [24] and the 3H-GCT is the method of reference [5]. It is evident that the construction size of our 3H-GCG scales more favorably with set size compared to the structural growth exhibited by the random matrix method. Moreover, it remains competitive with 3H-GCT.

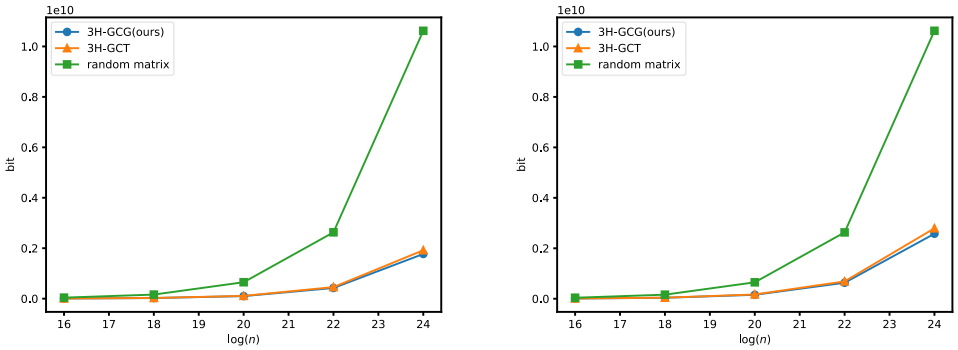

Figure 6. Schematic illustration of the size of each OKVS coding structure.

In Table 2, we list the existing constructs that meet the OKVS definition and analyze the type separately, the ratio of the original data to the constructed data, the encoding overhead and the batch decoding overhead. These analyses are based on constructing OKVS with a failure probability of $2^{-\lambda}$. We developed the 3H-GCG protocol by extending the principles of 3H-GCT and introducing specific constraints during its construction. As a result, we observed an approximately 7.5% improvement in the ratio between raw and processed data, while the encoding and decoding overhead maintained a consistent order of magnitude. Although the overall ratio is slightly lower than that achieved by the PaXoS

structure, 3H-GCG stands out due to its ability to handle both binary and linear data types, making it more versatile when compared to the PaXoS structure, which is primarily designed for binary data types.

Table 2. Different OKVS constructions and their properties with an error probability of $2^{-\lambda}$.

OKVS	Encoding Cost	(Batch) Decoding Cost	Rate
polynomial	$O(nlog^2 n)$	$O(nlog^2 n)$	1
random matrix (binary)	$O(n^3)$	$O(n^2)$	1
random matrix (linear)	$O(n^3)$	$O(n^2)$	$1/(1+\lambda)$
garbled bloom filter	$O(n\lambda)$	$O(n\lambda)$	$O(1/\lambda)$
PaXoS [17]	$O(n\lambda)$	$O(n\lambda)$	0.4-$O(1)$
3H-GCT [18] (binary)	$O(n\lambda)$	$O(n\lambda)$	0.8-$O(1)$
3H-GCT [18] (linear)	$O(n\lambda)$	$O(n\lambda)$	0.8-$O(1)$
3H-GCG(ours) (binary)	$O(n\lambda)$	$O(n\lambda)$	0.86-$O(1)$
3H-GCG(ours) (linear)	$O(n\lambda)$	$O(n\lambda)$	0.86-$O(1)$

6.2. PSI Part

We compare the traffic of the PSI protocol in [3–7,19,25,26] with that of our protocol in a theoretical analysis. Note that these analyses are based on a semi-honest security environment and a malicious security environment. The specific results are presented in Table 3, where it can be seen that the communication overhead of our protocol outperforms other protocols when the set size is 2^{16}.

Table 3. Comparison of the theoretical communication overheads of various PSI protocols.

Protocals	Communication	$n = n_x = n_y$ 2^{16}	2^{20}	2^{24}
Semi-Honest PSI				
DH-PSI [24]	$4\kappa n_x + (\lambda + log(n_x n_y))n_y$	584n	592n	600n
KKRT16 [25]	$6\kappa n_x + 3(\lambda + log(n_x n_y))n_y$	984n	1008n	1032n
PRTY19 low-Comm [13]	$3.5\kappa n_x + 1.02(2 + \lambda + log(n_x))n_y$	509n	513n	517n
PRTY19 fast [13]	$3.5(1 + 1/\lambda)\kappa n_x + 2(\lambda + log(n_x n_y))n_y$	603n	619n	635n
PRTY20 [17]	$9.3\kappa n_x + (\lambda + log(n_x n_y))n_y$	1208n	1268n	1302n
CM20 [27]	$4.8\kappa n_x + (\lambda + log(n_x n_y))n_y$	678n	694n	702n
RS21 w = 2 [15]	$2^{24} n_x^{0.05} + 307 n_x + 40 n_y + log(n_x n_y) n_y$	914n	**426n**	**398n**
GPR+21 star [18]	$5.6\kappa n_x + (\lambda + log(n_x n_y))n_y$	780n	788n	804n
OURS	$3.4\kappa n_x + (\lambda + log(n_x n_y))n_y$	**507n**	515n	523n
Malicious PSI				
PRTY20 [17]	$11.8\kappa n_x + 2\kappa n_y$		1766n	
GPR+21 star [18]	$8.6\kappa n_x + 2\kappa n_y$		1357n	
OURS	$7.1\kappa n_x + 2\kappa n_y$		**1165n**	

Some protocols have additional parameters, which are approximated by κ, λ. Note that the coefficients shown below often vary (non-linearly) as a function of n, κ, λ. The second column contains overheads for fixed λ = 40, κ = 128 (representatively), while the last three columns also fix the size of the set. More importantly, our protocols are resistant to malicious adversary attacks on a semi-honestly secure basis, and have a good competitive edge in terms of security.

We have undertaken reproduction work on existing PSI protocols and compared these protocols experimentally with our protocol, and the comparison results are shown in Table 4. All protocols operate in a LAN environment at less than 1 Gbps with sub-millisecond latency.

The experimental data are divided into two parts, runtime and traffic, in milliseconds (ms) and megabytes (MB), respectively. We classify these protocols into two categories, semi-honest sets and malicious sets, and compare them separately. From the results, we can see that we make a size limit on the encoded data, so our scheme has an optimal communication overhead. Based on the data in Table 4, it can be seen that in the semi-honest setting, our protocol can compete strongly with the work of Kolesnikov et al. [25] in terms of runtime. The communication aspect is not able to surpass the dominance of Pinkas et al. [13], but our work is not far from it in terms of communication volume. Under the malicious setting, our protocol has a significant advantage over other existing protocols in terms of runtime, especially reaching the fastest at set sizes of 2^{18}. Communication is also minimized compared to other existing protocols, with an approximately 10% improvement over existing protocols. Analyzed from an overall perspective, our PSI protocol compared to the PSI protocols proposed in the literature [27] is more advantageous in terms of both communication and computational overhead, which fully justifies the relevance of our work.

Table 4. Comparison of experimental data, with optimal data categorized by semi-honest and malicious settings marked in the figure in bold.

Protocol	Setting	Running Time (Set Size n) (ms)				Communication (MB)			
		2^{16}	2^{18}	2^{20}	2^{22}	2^{16}	2^{18}	2^{20}	2^{22}
DH-PSI [24]	se	31,378	127,893	518,620	2,063,949	4.69	19	76	308
KKRT16 [25]	se	**99**	406	**1170**	**7916**	7.73	31.88	128.5	530.1
PRTY20 (w = 2) [17]	se	534	910	2673	18,806	13.21	54.56	224.5	861.9
CM20 [27]	se	409	1217	6515	29,148	5.343	21.75	87.66	357.6
SpOT-low [21]	se	878	3476	14,558	90,144	**3.91**	**15.6**	**63.2**	**253**
SpOT-fast [21]	se	561	2354	6190	36,150	4.61	18.9	76.4	314
GPR+21 [18]	se	129	492	1365	8574	5.63	22.8	96.71	385.63
OURS	se	117	**392**	1176	8120	5.14	20.97	85.45	344.27
RR17a [3]	ma	1366	5619	-	-	154.17	616.55	-	-
RR17b [4]	ma	716	2727	12,010	-	95.78	383.03	1616.03	-
PRTY20 [17]	ma	476	1803	3128	18,679	14.71	60.33	247.6	950
GPR+21 [18]	ma	**146**	577	1674	**10,134**	8.68	35.59	136.66	532.97
OURS	ma	153	**563**	**1652**	12,304	**8.26**	**30.15**	**116.38**	**459.7**

7. Conclusions

In this paper, we proposed a new OKVS scheme, 3H-GCG, which improves on existing schemes and is more suitable for scenarios where decoding results are already expected. Based on this, we used 3H-GCG to construct an OPRF protocol and combine it with the VOLE approach to design a PSI protocol that can withstand malicious adversaries. And we give detailed security proofs for the PSI protocol. Finally, we compared the designed OKVS scheme and PSI protocol with other existing schemes. When the set size is $2^{18} \sim 2^{20}$, our PSI protocol is less computationally intensive than other existing protocols. The final results show that for our 3H-GCG, the ratio of raw data to constructed data is improved by about 7.5% over other existing OKVSs. Under the semi-honest security setting, our PSI protocol achieves the fastest runtime with a set size of 2^{18}. Under the malicious security setting, our PSI protocol has about 10% improvement in communication compared to other existing protocols.

With the increasing number of real-life privacy protection scenarios, such as smart homes, smart cities, remote healthcare, smart transportation, and smart education, the need for privacy protection is becoming more stringent. Furthermore, it is worth noting that PSI protocols may also find application in the field of Computer-Aided Vehicle (CAV) traffic behavior research [28], further expanding the horizons of their utility. Specifically, the results of current research developments in PSI protocols in terms of reducing communication and computational overhead are sufficient for practical needs. As the various applications

become smarter, there are higher requirements for privacy computing protocols, such as security, scalability, and adaptability. For example, there is a greater need for privacy computing protocols that are resistant to malicious adversaries, and a greater need for privacy computing protocols that can be operated collaboratively by multiple participants. This will be the goal of our future research.

Author Contributions: Conceptualization, Z.J.; Methodology, X.G.; Software, X.G. and J.W.; Validation, T.Y., H.Z., J.W. and Z.W.; Formal analysis, X.G.; Investigation, H.Z., J.W. and Z.W.; Writing—original draft, X.G.; Writing—review & editing, Z.J.; Visualization, T.Y., H.Z. and Z.W.; Supervision, T.Y. and J.W.; Funding acquisition, Z.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Beijing Municipal Natural Science Foundation (M22002, 4212019), National Natural Science Foundation of China (62172005), Fundamental Research Funds of Communication University of China (CUC22GP006).

Data Availability Statement: No new data were created for this study.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Areekijseree, K.; Tang, Y.; Chen, J. Secure and Efficient Multi-Party Directory Publication for Privacy-Preserving Data Sharing. In Proceedings of the SecureComm, Singapore, 8–10 August 2018; Volume 254, pp. 71–94.
2. Karl, R.; Takeshita, J.; Mohammed, A.; Striegel, A.; Jung, T. Cryptonomial: A Framework for Private Time-Series Polynomial Calculations. In Proceedings of the SecureComm, Virtual, 6–9 September 2021; Volume 398, pp. 332–351.
3. Rindal, P.; Rosulek, M. Improved Private Set Intersection Against Malicious Adversaries. In Proceedings of the EUROCRYPT, Paris, France, 4 May 2017; Volume 10210, pp. 235–259.
4. Rindal, P.; Rosulek, M. Malicious-secure private set intersection via dual execution. In Proceedings of the CCS, Dallas, TX, USA, 3 November 2017; pp. 1229–1242.
5. Garimella, G.; Rosulek, M.; Singh, J. Structure-Aware Private Set Intersection, with Applications to Fuzzy Matching. In Proceedings of the CRYPTO, Santa Barbara, CA, USA, 15–18 August 2022; pp. 323–352.
6. Freedman, M. J.; Hazay, C.; Nissim, K.; Pinkas, B. Efficient Set Intersection with Simulation-Based Security. In Proceedings of the CRYPTO, Santa Barbara, CA, USA, 14–18 August 2016; pp. 115–155.
7. Blanton, M.; Aguiar, E. Private and oblivious set and multiset operations. *Int. J. Inf. Sec.* **2016**, *15*, 493–518.
8. Hazay, C. Oblivious Polynomial Evaluation and Secure Set-Intersection from Algebraic PRFs. *J. Cryptol.* **2018**, *31*, 537–586.
9. Nevo, O.; Trieu, N.; Yanai, A. Simple, fast malicious multiparty private set intersection. In Proceedings of the CCS, Virtual, 15–19 November 2021; pp. 1151–1165.
10. Zhu, K.; Chen, W.; Jiao, L.; Wang, J.; Peng, Y.; Zhang, L. Online training data acquisition for federated learning in cloud-edge networks. *J. Comput. Netw.* **2023**, *223*, *109556*. [CrossRef]
11. Freedman, M.J.; Nissim, K.; Pinkas, B. Efficient Private Matching and Set Intersection. In Proceedings of the EUROCRYPT, Interlaken, Switzerland, 2–6 May 2004; pp. 1–19.
12. Pinkas, B.; Schneider, T.; Weinert, C.; Wieder, U. Efficient Circuit-Based PSI via Cuckoo Hashing. In Proceedings of the EUROCRYPT, Tel Aviv, Israel, 29 April–3 May 2018; pp. 125–157.
13. Pinkas, B.; Rosulek, M.; Trieu, N.; Yanai, A. SpOT-Light: Lightweight Private Set Intersection from Sparse OT Extension. In Proceedings of the CRYPTO, Santa Barbara, CA, USA, 18–22 August 2019; pp. 401–431.
14. Orrù, M.; Orsini, E.; Scholl, P. Actively Secure 1-out-of-N OT Extension with Application to Private Set Intersection. In Proceedings of the CT-RSA, San Francisco, CA, USA, 14–17 February 2017; pp. 381–396.
15. Rindal, P.; Schoppmann, P. VOLE-PSI: Fast OPRF and Circuit-PSI from Vector-OLE. In Proceedings of the EUROCRYPT, Zagreb, Croatia, 17–21 October 2021; pp. 901–930.
16. Rindal, P.; Raghuraman, S. Blazing Fast PSI from Improved OKVS and Subfield VOLE. In Proceedings of the CCS, Los Angeles, CA, USA, 7–11 November 2022; pp. 2505–2517.
17. Pinkas, B.; Rosulek, M.; Trieu, N.; Yanai, A. PSI from PaXoS: Fast, malicious private set intersection. In Proceedings of the EUROCRYPT, Virtual, 11–15 May 2020; pp. 739–767.
18. Garimella, G.; Pinkas, B.; Rosulek, M.; Yanai, A. Oblivious Key-Value Stores and Amplification for Private Set Intersection. In Proceedings of the CRYPTO, Virtual, 16–20 August 2021; pp. 395–425.
19. Manulis, M.; Pinkas, B.; Poettering, B. Privacy-Preserving Group Discovery with Linear Complexity. In Proceedings of the ACNS, Beijing, China, 22–25 June 2010; pp. 420–437.
20. Kolesnikov, V.; Matania, N.; Pinkas, B. Practical multi-party private set intersection from symmetric-key techniques. In Proceedings of the CCS, Dallas, TX, USA, 30 October–3 November 2017; pp. 1257–1272.

21. Pinkas, B.; Schneider, T.; Tkachenko, O.; Yanai, A. Efficient Circuit-Based PSI with Linear Communication. In Proceedings of the EUROCRYPT, Darmstadt, Germany, 19–23 May 2019; pp. 122–153.
22. Kolesnikov, V.; Rosulek, M.; Trieu, N.; Wang, B. Scalable Private Set Union from Symmetric-Key Techniques. In Proceedings of the ASIACRYPT, Kobe, Japan, 8–12 December 2019; pp. 636–666.
23. Schoppmann, P.; Gascon, A.; Reichert, L.; Raykova, M. Distributed Vector-OLE: Improved Constructions and Implementation. In Proceedings of the CCS, New York, NY, USA, 11–15 November 2019; pp. 1055–1072.
24. Meadows, C. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In Proceedings of the S&P, Oakland, CA, USA, 7–9 April 1986; pp. 134–137.
25. Kolesnikov, V.; Kumaresan, R.; Rosulek, M.; Trieu, N. Efficient batched oblivious PRF with applications to private set intersection. In Proceedings of the CCS, Vienna, Austria, 24–28 October 2016; pp. 818–829.
26. Changyu, D.; Liqun, C.; Zikai, W. When private set intersection meets big data: An efficient and scalable protocol. In Proceedings of the CCS, Berlin, Germany, 4–8 November 2013; pp. 789–800.
27. Chase, M.; Miao, P. Private Set Intersection in the Internet Setting from Lightweight Oblivious PRF. In Proceedings of the CRYPTO, Santa Barbara, CA, USA, 17–21 August 2020; pp. 34–63.
28. Wang, P.; Wu, X.; He, X. Vibration-Theoretic Approach to Vulnerability Analysis of Nonlinear Vehicle Platoons. *J. IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 11334–11344. [CrossRef]