Unbalanced Circuit-PSI from Oblivious Key-Value Retrieval

Meng Hao[1], Weiran Liu[2], Liqiang Peng[2], Hongwei Li[3, ✉], Cong Zhang[4], Hanxiao Chen[1], and Tianwei Zhang[1]

[1]Nanyang Technological University
[2]Alibaba Group
[3]Peng Cheng Laboratory
[4]Institute for Advanced Study, BNRist, Tsinghua University
{menghao303, chenhanxiao.chx, hongweili.hli}@gmail.com, {weiran.lwr, plq270998}@alibaba-inc.com, zhangcong@mail.tsinghua.edu.cn, tianwei.zhang@ntu.edu.sg

Abstract

Circuit-based Private Set Intersection (circuit-PSI) empowers two parties, a client and a server, each with input sets X and Y, to securely compute a function f on the intersection $X \cap Y$ while preserving the confidentiality of $X \cap Y$ from both parties. Despite the recent proposals of computationally efficient circuit-PSI protocols, they primarily focus on the balanced scenario where $|X|$ is similar to $|Y|$. However, in many practical situations, a circuit-PSI protocol may be applied in an unbalanced context, where $|X|$ is significantly smaller than $|Y|$. Directly applying existing protocols to this scenario poses notable efficiency challenges due to the communication complexity of these protocols scaling at least linearly with the size of the larger set, i.e., $\max(|X|,|Y|)$.

In this work, we put forth efficient constructions for unbalanced circuit-PSI, demonstrating sublinear communication complexity in the size of the larger set. Our key insight lies in formalizing unbalanced circuit-PSI as the process of obliviously retrieving values corresponding to keys from a set of key-value pairs. To achieve this, we propose a new functionality named Oblivious Key-Value Retrieval (OKVR) and design the OKVR protocol based on a new notion termed sparse Oblivious Key-Value Store (sparse OKVS). We conduct comprehensive experiments and the results showcase substantial improvements over the state-of-the-art circuit-PSI schemes, i.e., $1.84 \sim 48.86\times$ communication improvement and $1.50 \sim 39.81\times$ faster computation. Compared to a very recent unbalanced circuit-PSI work, our constructions outperform them by $1.18 \sim 15.99\times$ and $1.22 \sim 10.44\times$ in communication and computation overhead, respectively, depending on set sizes and network environments.

1 Introduction

In Private Set Intersection (PSI) [20, 23, 32, 34, 42–44, 48, 51], two parties, the client with a set X and the server with a set Y, securely compute the intersection $X \cap Y$, without leaking the information of the items that are not in the intersection. Depending on the output, PSI can be broadly classified into two categories, plain PSI and circuit-based PSI (denoted as circuit-PSI) [28]. Notably, circuit-PSI has garnered significant attention in recent times [10, 45, 51]. Unlike plain PSI, which directly reveals the plaintext of the intersection $X \cap Y$, circuit-PSI offers the capability to securely compute a function f on the intersection and exclusively outputs the result $f(X \cap Y)$ without leaking $X \cap Y$. This expands the applicability of PSI to a broader range of scenarios. Formally, for each element $x \in X$, the two parties acquire secret shares of b, where $b = 1$ if $x \in X \cap Y$ and $b = 0$ otherwise. These shares can then be leveraged to securely compute any desired function using generic secure two-party computation protocols [26, 57]. Recent advancements in circuit-PSI protocols [6, 10, 45, 48, 51] have demonstrated efficient computation and achieved linear communication complexity proportional to the size of the parties' sets.

While circuit-PSI protocols offer notable advantages, their current emphasis is primarily on scenarios where the sets held by the involved parties are of similar size (i.e., the balanced setting). However, practical applications, particularly in client-server scenarios, often demand the execution of circuit-PSI protocols in an *unbalanced* setting. In such situations, the client typically possesses a smaller set, compared to the server's considerably larger set. A relevant example is private contact tracing for infectious diseases [9, 55], where a client aims to privately check if the collected tracing data aligns with traces of diagnosed patients. Achieving this involves computing the cardinality function on the set intersection [55], with only the client gaining knowledge of the count of matching traces. In practical terms, the server may hold a set comprising several million items, vastly outnumbering the client's set, which might only consist of a few hundred items.

Furthermore, Google [29] employed a private application for measuring ad conversion rates, specifically examining the revenues from ad viewers who later engage in related transactions. In this scenario, a client (such as a holder of transaction data) possesses records of customer transactions, while

✉Corresponding author.

a server (like an advertising company) maintains records of customers who have viewed ads. The task involves intersecting identifiers of those who viewed a specific ad with those who completed a transaction, followed by the computation of an aggregation function on the resulting intersection. Notably, the sets involved in this process are highly unbalanced, given that the number of ad impressions typically exceeds the number of transactions by orders of magnitude. Moreover, the transaction records and ad views are privacy-sensitive, potentially containing personal interests, preferences, and even health-related information. Consequently, there is an imperative need for such an unbalanced circuit-PSI protocol.

Nevertheless, the current landscape lacks effective techniques tailored for unbalanced circuit-PSI scenarios. Existing solutions encounter challenges, as they either compromise the confidentiality of private intersections or impose substantial communication overhead. On one hand, some studies have introduced unbalanced PSI protocols [11, 12, 16, 32, 50] addressing scenarios where the server's set is much larger than the client's. Despite their relevance, extending these protocols to unbalanced circuit-PSI settings proves challenging, as they inherently disclose the intersection in plaintext. On the other hand, applying state-of-the-art circuit-PSI protocols [6, 10, 45, 48, 51] directly to unbalanced settings is also problematic. These protocols are specifically designed for parties' sets of similar size. When used in unbalanced settings, these protocols incur at least linear communication complexity on the larger set, leading to a notable performance decline, especially in bandwidth-constrained network environments. Refer to Section 1.3 for more details. The above discussion raises the following natural question:

Can we construct concretely efficient unbalanced circuit-PSI protocols with sublinear communication in the size of the larger set?

1.1 Our Contributions

In this paper, we make an affirmative answer to the above question. Our contributions can be summarized as follows:

- We construct highly efficient unbalanced circuit-PSI protocols, exhibiting communication overhead that scales sublinearly with the size of the larger set. In line with previous works, our protocols are secure against semi-honest adversaries.
- At the heart of our methodology is a newly introduced functionality called Oblivious Key-Value Retrieval (OKVR). We provide its construction based on a novel notion termed sparse Oblivious Key-Value Store (sparse OKVS).
- We implement our protocols alongside state-of-the-art circuit-PSI protocols in a unified framework. The results demonstrate the significant enhancements of our protocols in both communication and computation overhead. Specifically, our protocol achieves a communication cost $1.84 \sim 48.86\times$ lower and a running time $1.50 \sim 39.81\times$ faster compared to the state-of-the-art protocols. Very recently, Son and Jeong [54] also present unbalanced circuit-PSI protocols, and our constructions show reductions in communication and computation overhead by $1.18 \sim 15.99\times$ and $1.22 \sim 10.44\times$, respectively, depending on set sizes and network environments.

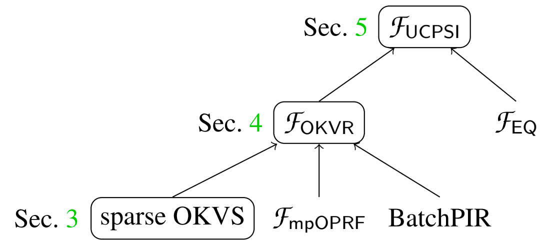

Figure 1: Overview of our unbalanced circuit-PSI framework. Rounded rectangles are contributions of this work.

1.2 Overview of Our Techniques

The main insight driving our framework is that unbalanced circuit-PSI can be formalized as obliviously retrieving the values corresponding to keys from a key-value store[1], followed by an equality test. We elaborate on this insight below. For simplicity, we consider the scenario where the client's set comprises only a single element x and the server's set is represented as $Y = \{y_1, \dots, y_n\}$. Specifically, the server initiates the process by constructing a key-value store with key-value pairs $\{(y_1, r), \dots, (y_n, r)\}$, where each item y_i serves as a key and the corresponding value is a randomly sampled r. Subsequently, the client employs x as the query key and obliviously retrieves a value r^* from the server. This retrieved value should satisfy $r^* = r$ if $x \in Y$ and r^* is uniformly random otherwise. Finally, the unbalanced circuit-PSI evaluation concludes with the computation of a secret-shared output, derived from a private equality test conducted on r^* and r. To achieve the fully-fledged construction based on the above insight, we undertake three intermediate steps detailed in Figure 1, and overview them in a bottom-up manner.

1. We introduce a new notion termed sparse Oblivious Key-Value Store (sparse OKVS), designed to encode key-value pairs into a compact representation while supporting efficient retrieval. Within this sparse OKVS abstraction, we identify and detail efficient instantiations that align with our goals.

[1]A key-value store for $\mathcal{K} \times \mathcal{V}$ is a data structure that represents a desired mapping $k_i \to v_i$, where $(k_i, v_i) \in \mathcal{K} \times \mathcal{V}$.

2. We present a new functionality called Oblivious Key-Value Retrieval (OKVR). This serves the purpose of obliviously retrieving the values corresponding to keys from a key-value store. We construct an efficient OKVR protocol, which is facilitated by utilizing the property of sparse OKVS.

3. We construct fully-fledged unbalanced circuit-PSI protocols characterized by sublinear communication complexity in the size of the larger set. The cornerstone of our construction is a specialized padding-free variant of OKVR. This specifically addresses an efficiency concern arising from commonly employed hash-to-bin techniques in circuit-PSI.

Below, we give more technical details to illustrate the above three intermediate steps.

Retrieving key-value pairs efficiently. The central objective of our framework is the efficient and oblivious retrieval of key-value pairs. Thus, the starting point involves transforming the server's key-value pairs into a compact representation conducive to efficient retrieval.

Although this functionality can be accomplished by leveraging Oblivious Key-Value Store (OKVS) [23, 43], a primary challenge lies in that the retrieval efficiency can not be well guaranteed in existing OKVS. Specifically, OKVS comprises two algorithms, namely $(\mathsf{Encode}, \mathsf{Decode})$. The Encode algorithm takes a set of key-value pairs $\{(k_i, v_i)\}_{i \in [n]}$ as input and produces a vector D. The Decode algorithm takes D and any key k as input, providing an output that corresponds to v_i if k matches some k_i used to generate D. In our framework, the Decode algorithm aligns with our retrieval procedure and will be implemented in an oblivious manner. Unfortunately, prior works utilize OKVS instances in a black-box way, e.g., accessing the whole vector D for Decode, causing linear communication complexity in the size of the key-value pairs. This problem stems from the absence of explicit requirements for decoding efficiency in existing OKVS.

We introduce a new OKVS notion, termed *sparse* OKVS, designed to enhance decoding efficiency. A key difference from prior OKVS constructions [23] lies in that the structure of the output D from Encode can be organized as $D_0 \| D_1$ with $|D_0| = \omega(|D_1|)$, and Decode exclusively accesses a constant number (e.g., $2 \sim 3$) of elements within the larger D_0 and an arbitrary number of elements within the smaller D_1. Consequently, we refer to D_0 as the sparse part and D_1 as the dense part. Importantly, accessing a constant number of elements on the larger sparse part is a pivotal property for achieving efficient oblivious key-value retrieval. Later, we will make a large number of retrievals on the dense part nearly cost-free. We identify existing constructions that satisfy the sparse OKVS abstraction, such as Garbled Cuckoo Tables (GCT) [23, 43]. Refer to Section 3 for more details.

Retrieving key-value pairs obliviously. Building upon the efficient retrieval strategy outlined above, our next objective is to facilitate the client to obliviously retrieve the server's key-value pairs with the following security guarantee. The server should remain unaware of which value was retrieved, and the client should not acquire knowledge of other key-value pairs within the server's set.

Unfortunately, it is challenging to achieve the above goal while ensuring concrete efficiency. A possible solution to oblivious key-value retrieval is employing Private Information Retrieval (PIR)[2] [5, 38–40], in which the client privately retrieves the relevant items from the server's OKVS encoding. These retrieved items are used in the decoding for the reconstruction of the corresponding value for the query. However, it requires a substantial number of PIR queries, causing a huge loss of efficiency. Moreover, PIR alone cannot safeguard the privacy of the server's key-value pairs.

We formalize the above objective as a new functionality named Oblivious Key-Value Retrieval (OKVR). In OKVR, the server possesses a large set of key-value pairs denoted as $L = \{(k_1, v_1), \ldots, (k_n, v_n)\}$ and the client holds a smaller query set $Q = \{q_1, \ldots, q_t\}$ with $n \gg t$. After the execution, the client obtains a value z_i for each $q_i \in Q$, where $z_i = v_j$ if q_i matches some k_j and $(k_j, v_j) \in L$. Otherwise, z_i is a uniformly random value. To achieve concretely efficient and asymptotically communication-sublinear OKVR, we present the construction derived from sparse OKVS and a hybrid PIR strategy. Specifically, we first utilize sparse OKVS to encode the server's key-value pairs into $D_0 \| D_1$. Subsequently, PIR is exclusively applied to the sparse part D_0 to retrieve the desired items, while the dense part D_1, owing to its small size, is directly transmitted from the server to the client. This ensures the retrieval of a constant number of items on D_0 via PIR, while the client can locally obtain a considerable number of items in D_1 with minimal cost. To enhance PIR efficiency further, given the requirement of multiple retrievals at once, we exploit recent BatchPIR schemes [5, 40] that substantially reduce communication and computation overhead in batch retrievals. Moreover, similar to prior works [11, 45], we address the PIR's privacy concern regarding the server's key-value pairs by invoking an Oblivious Pseudorandom Function (OPRF) [22] before the execution of sparse OKVS. Refer to Section 4 for more details.

Constructing fully-fledged unbalanced circuit-PSI. We propose an efficient construction of unbalanced circuit-PSI from OKVR, achieving sublinear communication complexity on the size of the larger set.

A key challenge is the PIR efficiency issue in OKVR caused by the commonly employed hash-to-bin technique. In constructing efficient circuit-PSI protocols [10, 45], hash-to-bin techniques are integral for reducing computational costs. Specifically, the client employs Cuckoo hashing [41] to map the set X of size-t into a table T_X of size $m = (1+\varepsilon) \cdot t$, where $\varepsilon > 0$ is a constant. In this mapping, each bin of T_X contains

[2] The communication complexity of PIR is sublinear to the database size.

at most one item. Simultaneously, the server maps its items in Y to a table T_Y of size m using simple hashing, allowing each bin to accommodate multiple items. After that, the client has to conceal the mapped positions by padding dummy points into empty bins of T_X [44, 45]. The evaluation of unbalanced circuit-PSI invokes the OKVR functionality, where the client inputs T_X and the server inputs $\{P_i\}_{i\in[m]}$ with $P_i := \{(y', r_i)\}$ for all $y' \in T_Y[i]$ and a random r_i. Nevertheless, these dummy points introduce additional overhead in terms of PIR queries within the OKVR procedure, which dominates the overhead of our unbalanced circuit-PSI protocol.

To mitigate this issue, we introduce a specialized *padding-free* OKVR solution where the client only inputs the key set of size t rather than $(1+\varepsilon)\cdot t$. The key insight here is that, prior to PIR queries, the client is already aware that these dummy points do not belong to the intersection. Consequently, random values are obtained according to the functionality of OKVR. Recall that OKVR outputs a random value when a query is not present in the server's key set. As a result, our padding-free OKVR protocol invokes OKVR solely on the bins of T_X that correspond to the elements of X and locally samples uniformly random values for the dummy points. Notably, the confidentiality of the mapped positions is guaranteed because we take the whole client's query set as input to OKVR. Refer to Section 5 for more details.

1.3 Related Works

We elaborate on existing balanced circuit-PSI protocols as well as related techniques in the unbalanced setting, and further discuss the essential differences between these works and our protocols.

Circuit-PSI. Circuit-PSI was initially introduced by Huang et al. [28], leveraging generic secure computation techniques, i.e., garbled circuits [57]. They employed optimized sort-compare-shuffle circuits to reduce circuit size, resulting in a circuit that contains $O(n \log n)$ comparisons, where n represents the set size. Following this, several subsequent works [15, 44, 46] aimed to reduce the number of comparisons and hence optimize asymptotic computation and communication complexity. Recently, Pinkas et al. [45] achieved the first circuit-PSI protocol with linear communication complexity by introducing a novel batch Oblivious Programmable Pseudorandom Function (OPPRF) [35]. Similar to an oblivious PRF that provides the sender with a key of PRF and the receiver with the outputs of the PRF on points of his choice, an OPPRF enables the sender to additionally send the receiver a hint that can program the PRF to output specific values on certain input points privately chosen by the sender. The main bottleneck of this protocol is that the computation complexity is super-linear in the size of the server's set [10, 45]. This is caused by the expensive polynomial interpolation in their OPPRF construction. Recent works [6, 33, 48, 51] addressed this problem leveraging state-of-the-art OKVS such as Garbled Cuckoo Tables (GCT) [23] that achieves linear computation complexity. Chandran et al. [10] also proposed specialized circuit-PSI protocols with both linear computation and communication complexity. The main technique is a new primitive called relaxed batch OPPRF. Moreover, Chandran et al. [10] and Han et al. [27] presented specialized equality test protocols aimed at further improving the concrete overhead of circuit-PSI.

A potential limitation of the above state-of-the-art circuit-PSI approaches is the communication complexity, which scales at least linearly with the size of the two parties' sets. The fundamental reason is that these circuit-PSI utilize existing OKVS [23] in a black-box way and send the whole encoded vector (i.e., the hint) to the client. In our work, to avoid linear communication, we present sparse OKVS and particularly focus on the decoding efficiency, which could be used to construct sublinear circuit-PSI protocols.

Unbalanced PSI-related works. There are some related PSI protocols in the unbalanced setting. We discuss the main differences between these works and ours. Several works proposed computation and communication efficient schemes for unbalanced PSI [12, 16, 32, 50] and labeled PSI [11, 16] from fully homomorphic encryption (FHE). However, these approaches face challenges when extending to our circuit-PSI setting because they focus on directly outputting the intersection to the client. While the work on labeled PSI [11] theoretically discussed the possibility of extending their protocol to unbalanced circuit-PSI, concrete constructions were lacking. Subsequently, Lepoint et al. [36] proposed an unbalanced private join and compute (PJC) scheme for the inner product, achieving sublinear communication cost in the size of the larger set. This scheme focuses on computing the inner product of associated values within the intersection and illustrates how to extend to compute arbitrary functions. However, their construction relies on (garbled) Bloom filters [7, 19], which encode n items into a vector of length $O(\lambda n)$ with a statistical security parameter λ. This leads to the communication and computation complexity additionally depending on λ. In comparison, our unbalanced circuit-PSI protocols can realize a $\lambda\times$ improvement on both the computation and communication overhead thanks to our concretely and asymptotically efficient OKVR protocols.

Very recently, Son and Jeong [54] (SJ23) provided two concrete constructions to instantiate the theoretical extension from labeled PSI to unbalanced circuit-PSI [11]. The first construction follows the labeled PSI using FHE on polynomial evaluation along with several optimizations, while the second construction proposes a novel recursive solution. However, their two constructions have a significant trade-off between communication and computation due to balancing the expensive homomorphic multiplication and the number of communicated FHE ciphertexts. Different from their works, we formalize circuit-PSI as obliviously retrieving the value corresponding to a key from a key-value store, and provide efficient constructions from sparse OKVS and BatchPIR. As shown in Section 6.2, the main advantage is that our protocols are more suitable for the unbalanced setting. Rather, SJ23 works well for relatively large client's set sizes, e.g., at least thousands of elements.

2 Preliminaries

2.1 Notations

We denote the two parties as client $\mathcal{C}$ and server $\mathcal{S}$. We use κ and λ to denote the computational and statistical security parameters, respectively. For two distributions X and Y, we write $X \approx_c Y$ and $X \approx_s Y$ if X and Y are computationally and statistically indistinguishable, respectively. We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$ and $D[i]$ to represent the i-th element of a vector D. By $a \leftarrow A$, we denote that a is randomly selected from the set A. $a \leftarrow \mathrm{A}(x)$ denotes that a is the output of the randomized algorithm A on input x, and $a := b$ denotes that a is assigned by b. $\langle x, y\rangle$ denotes the inner product of x and y. $\langle x\rangle_0^B$ and $\langle x\rangle_1^B$ denote Boolean shares of x such that $x = \langle x\rangle_0^B \oplus \langle x\rangle_1^B$. $1\{x = y\}$ outputs 1 if $x = y$ and 0 otherwise.

2.2 Threat Model

Similar to prior circuit-PSI schemes [10, 45, 48, 51, 54], in this work, we consider static semi-honest probabilistic polynomial-time (PPT) adversaries. Namely, a PPT adversary $\mathcal{A}$ passively corrupts either the client $\mathcal{C}$ or the server $\mathcal{S}$ at the beginning of the protocol and honestly follows the protocol specification. We use the standard simulation-based security definition for secure two-party computation [25]. Like these previous works, our construction invokes multiple sub-protocols, and we use the *hybrid model* to describe them. By convention, a protocol invoking a functionality $\mathcal{F}$ is referred to as the $\mathcal{F}$-hybrid model. We give the formal security definition as follows.

Definition 1. *Let* $\mathrm{view}_{\mathcal{C}}^{\Pi}(x,y)$ *and* $\mathrm{view}_{\mathcal{S}}^{\Pi}(x,y)$ *be the views (including input, random tape, and all received messages) of* $\mathcal{C}$ *and* $\mathcal{S}$ *in a protocol* Π*, respectively, where* x *is the input of* $\mathcal{C}$ *and* y *is the input of* $\mathcal{S}$*. Let* $\mathrm{out}(x,y)$ *be the protocol's output of both parties and* $\mathcal{F}(x,y)$ *be the functionality's output.* Π *is said to securely compute a functionality* $\mathcal{F}$ *in the semi-honest model if for every PPT adversary* $\mathcal{A}$ *there exists PPT simulators* $\mathrm{Sim}_{\mathcal{C}}$ *and* $\mathrm{Sim}_{\mathcal{S}}$ *such that for all inputs* x *and* y*,*

$$\{\mathrm{view}_{\mathcal{C}}^{\Pi}(x,y), \mathrm{out}(x,y)\} \approx_c \{\mathrm{Sim}_{\mathcal{C}}(x, \mathcal{F}_{\mathcal{C}}(x,y)), \mathcal{F}(x,y)\},$$
$$\{\mathrm{view}_{\mathcal{S}}^{\Pi}(x,y), \mathrm{out}(x,y)\} \approx_c \{\mathrm{Sim}_{\mathcal{S}}(x, \mathcal{F}_{\mathcal{S}}(x,y)), \mathcal{F}(x,y)\}. \quad (1)$$

2.3 Oblivious Key-Value Store

A key-value store [23, 43] is simply a data structure that maps a set of keys to the corresponding values. The definition is as follows:

Functionality $\mathcal{F}_{\mathsf{mpOPRF}}$

Parameters: A PRF $F : \{0,1\}^{\kappa} \times \{0,1\}^{\ell} \to \{0,1\}^{\ell}$.

Functionality:

1. Wait for input $\{x_1, \ldots, x_n\} \in (\{0,1\}^{\ell})^n$ from the receiver.
2. Wait for input $\mathsf{k} \in \{0,1\}^{\kappa}$ from the sender.
3. Output $\{F(\mathsf{k}, x_1), \ldots, F(\mathsf{k}, x_n)\}$ to the receiver.

Figure 2: Ideal functionality for multi-point OPRF

Definition 2. *A Key-Value Store (KVS) is parameterized by a key space* $\mathcal{K}$*, a value space* $\mathcal{V}$*, a set of randomized functions* H*, input length* n *and output length* m*, and consists of two algorithms:*

- Encode_H*: on input key-value pairs* $\{(k_i, v_i)\}_{i\in[n]} \in (\mathcal{K} \times \mathcal{V})^n$*, outputs an object* $D \in \mathcal{V}^m$ *(or an error indicator* $\perp$ *with statistically small probability).*
- Decode_H*: on input* $D \in \mathcal{V}^m$ *and a key* $k \in \mathcal{K}$*, outputs a value* $v \in \mathcal{V}$*.*

Correctness. A KVS is correct if, for all $L \in (\mathcal{K} \times \mathcal{V})^n$ with distinct keys such that $\mathsf{Encode}_H(L) \neq \perp$, it holds that $\mathsf{Decode}_H(\mathsf{Encode}_H(L), k) = v$, where $(k, v) \in L$.

Obliviousness [23]. A KVS is an Oblivious KVS (OKVS) if, for all distinct $\{k_1^0, \ldots, k_n^0\} \in \mathcal{K}^n$ and $\{k_1^1, \ldots, k_n^1\} \in \mathcal{K}^n$, Encode_H does not output $\perp$ on $\{k_1^0, \ldots, k_n^0\}$ and $\{k_1^1, \ldots, k_n^1\}$, and then

$$\{\mathsf{Encode}_H(\{(k_1^0, v_1), \ldots, (k_n^0, v_n)\}) \mid v_i \leftarrow \mathcal{V} \text{ for } i \in [n]\} \approx_s$$
$$\{\mathsf{Encode}_H(\{(k_1^1, v_1), \ldots, (k_n^1, v_n)\}) \mid v_i \leftarrow \mathcal{V} \text{ for } i \in [n]\}. \quad (2)$$

Double obliviousness [48, 51]. An OKVS is doubly oblivious if, for all distinct $\{k_1, \ldots, k_n\} \in \mathcal{K}^n$ and n values $v_1, \ldots, v_n$ each sampled uniformly at random from $\mathcal{V}$ such that Encode_H does not output $\perp$ for $\{k_1, \ldots, k_n\}$, $\mathsf{Encode}_H(\{(k_1, v_1), \ldots, (k_n, v_n)\})$ is statistically indistinguishable from an uniformly random element in $\mathcal{V}^m$. Note that double obliviousness directly implies obliviousness [6, 48].

Binary OKVS [23]. An OKVS is binary if, for any k, $\mathsf{Decode}_H(D, k)$ can be expressed as a binary linear combination of D, i.e., $\mathsf{Decode}_H(D, k) := \langle D, h(k)\rangle$ where $h : \mathcal{K} \to \{0,1\}^m$ is some public function defined by H. In this work, we restrict ourselves to binary OKVS schemes.

2.4 Batch Private Information Retrieval

In a Batch Private Information Retrieval (BatchPIR) scheme [5, 13, 30, 37, 40], the client wants to privately download a batch of b entries from the server's dataset D of size n. A

BatchPIR consists of three routines, all taking the computational security parameter κ as an implicit input:

- $\mathsf{Query}(\{i_1,\ldots,i_b\}) \to (qu, st)$: on input a set of distinct indexes $\{i_1,\ldots,i_b\} \in ([n])^b$, outputs a query qu and a private state st including the index set.
- $\mathsf{Response}(D, qu) \to res$: on input the database D and the query qu, outputs a response res.
- $\mathsf{Extract}(st, res) \to \{y_1,\ldots,y_b\}$: given the state st and the response res, outputs a batch of entries $\{y_1,\ldots,y_b\}$.

A BatchPIR protocol should satisfy the following properties:

Correctness. A BatchPIR is correct if for any dataset D and all distinct inputs $I = \{i_1,\ldots,i_b\}$, it holds that

$$\mathsf{Extract}(st, \mathsf{Response}(D, qu)) = D[i_1],\ldots,D[i_b], \tag{3}$$

where $(st, qu) \leftarrow \mathsf{Query}(I)$.

Client query privacy. The client's query should reveal no information about the query indexes. Formally, a BatchPIR scheme satisfies *client query privacy* if, for all PPT adversaries $\mathcal{A}$ and all distinct batch query sets I_1, I_2 with $|I_1| = |I_2|$,

$$\begin{aligned} &\Pr[\mathcal{A}(qu) = 1 \mid (st, qu) \leftarrow \mathsf{Query}(I_1)] \\ &- \Pr[\mathcal{A}(qu) = 1 \mid (st, qu) \leftarrow \mathsf{Query}(I_2)] \le \mathsf{negl}(\kappa). \end{aligned} \tag{4}$$

Note that (Batch)PIR does not aim to protect the privacy of the server's dataset D [5].

2.5 Multi-point Oblivious Pseudorandom Function

An Oblivious Pseudorandom Function (OPRF) [22] is a protocol involving two parties, i.e., the sender and the receiver. The sender inputs the key $\mathsf{k} \in \{0,1\}^\kappa$ of a PRF F and obtains nothing, while the receiver takes $x \in \{0,1\}^\ell$ as input and obtains $F(\mathsf{k}, x) \in \{0,1\}^\ell$. In a multi-point OPRF (mpOPRF) [31], the receiver takes as input $\{x_1,\ldots,x_n\} \in (\{0,1\}^\ell)^n$ rather than a single point and obtains $\{F(\mathsf{k},x_1),\ldots,F(\mathsf{k},x_n)\} \in (\{0,1\}^\ell)^n$. The mpOPRF functionality is shown in Figure 2.

2.6 Secret Sharing and Equality Test

Our construction uses 2-out-of-2 boolean secret sharing techniques [53]. The boolean shares of $x \in \{0,1\}$ are $\langle x\rangle_0^B$ and $\langle x\rangle_1^B$, held by the client $\mathcal{C}$ and the server $\mathcal{S}$ respectively, satisfying $x = \langle x\rangle_0^B \oplus \langle x\rangle_1^B$. In our protocols, $\langle x\rangle^B$ denotes that $\mathcal{C}$ and $\mathcal{S}$ hold boolean shares of x. Our construction also invokes a private equality test, which takes as input two ℓ-bit values x, y and outputs the boolean shares of b, where $b := 1\{x = y\}$. We present the functionality in Figure 3. We use the state-of-the-art construction [10] to instantiate this functionality.

Functionality $\mathcal{F}_{\mathsf{EQ}}$

Parameters: Client $\mathcal{C}$ and server $\mathcal{S}$.

Functionality:

1. Wait for input $x \in \{0,1\}^\ell$ from $\mathcal{C}$.
2. Wait for input $y \in \{0,1\}^\ell$ from $\mathcal{S}$.
3. Sample $\langle b\rangle_0^B$, $\langle b\rangle_1^B$ uniformly at random from $\{0,1\}$ such that $b := 1\{x = y\}$.
4. Output $\langle b\rangle_0^B$, $\langle b\rangle_1^B$ to $\mathcal{C}$ and $\mathcal{S}$, respectively.

Figure 3: Ideal functionality for private equality test

2.7 Cuckoo Hashing

Cuckoo hashing [41] uses α random hash functions $h_1,\ldots,h_\alpha : \{0,1\}^* \to [m]$ to map n elements into m bins, where $m = (1+\varepsilon)\cdot n$ with the constant $\varepsilon > 0$. The mapping procedure is as follows. An element x is inserted into the bin $h_i(x)$, if this bin is empty for some $i \in [\alpha]$. Otherwise, we pick a random $i \in [\alpha]$, insert x in $h_i(x)$, evict the item currently in $h_i(x)$ and recursively insert the evicted item. The recursion proceeds until no more evictions are necessary or a threshold number of re-allocations are done. If the recursion stops for the latter reason, it is considered a failure event, meaning there exists at least an element that is not mapped to any bins. Some variants of Cuckoo hashing maintain a set called the stash, to store such elements. Stash-less Cuckoo hashing is where no special stash is maintained. In this work, we only leverage stash-less Cuckoo hashing.

3 Sparse Oblivious Key-Value Store

3.1 Definition

We present a new notion called sparse Oblivious Key-Value Store (sparse OKVS), which incorporates a sparsity property into OKVS introduced in Section 2.3. A formal definition is provided below.

Definition 3. *An OKVS is sparse if (1) the output D of Encode_H can be structured as $D = D_0 \| D_1$ with $|D_0| = \omega(|D_1|)$, and (2) for any k, $\mathsf{Decode}_H(D,k) := \langle l(k)\|r(k), D_0\|D_1\rangle$, where two mappings l, r are defined by H such that $l : \mathcal{K} \to \{0,1\}^{|D_0|}$ outputs a sparse binary vector with a constant weight α and $r : \mathcal{K} \to \{0,1\}^{|D_1|}$ outputs a dense binary vector.*

In other words, the sparsity property allows Decode to only access a constant number of elements in large D_0 and an arbitrary number of elements in small D_1. Thus, we refer D_0 and D_1 to the sparse and dense parts, respectively. Besides, the correctness and (double) obliviousness properties follow those of OKVS in Section 2.3. For convenience in our construction, we use α mappings $\{l_i : \mathcal{K} \to [|D_0|]\}_{i\in[\alpha]}$ to equivalently represent the mapping $l : \mathcal{K} \to \{0,1\}^{|D_0|}$, namely the output of $\{l_i\}_{i\in[\alpha]}$ are α non-zero positions of the mapping l's output. We emphasize that the sparsity property latter will be importantly utilized to facilitate the efficiency of our oblivious key-value retrieval constructions in Section 4.

Compared to OKVS [23], our sparse OKVS has the following technical advantage. Specifically, sparse OKVS encodes key-value pairs into a compact representation for efficient retrieval with a constant number of accesses on the large sparse part. This is an important property for applications in the unbalanced setting. However, many OKVS [23], e.g., polynomial and random matrix-based solutions, do not have this property. Building on sparse OKVS, the unbalanced circuit-PSI in Section 5 achieves asymptotically sublinear communication with the larger server's set. In contrast, existing circuit-PSI protocols based on OKVS suffer from linear communication complexity due to using OKVS in a black-box manner.

3.2 Instantiation

Before giving instantiations that fall within our definition of sparse OKVS, it is crucial to note that many recent instances of OKVS do not meet this criterion. Counterexamples include those based on polynomials or random matrices [23, 43], where Decode requires accessing all positions of the output of Encode. Besides, although RB-OKVS [6] and Garbled Bloom Filter (GBF) [19] satisfy the definition of sparse OKVS, they require accessing $O(\lambda)$ positions of the Encode's output. This is concretely inefficient in our following constructions.

In this work, we instantiate sparse OKVS using Garbled Cuckoo Table (GCT) [23, 43]. In a GCT, a set of n key-value pairs is encoded into a vector represented as $D_0\|D_1$, where the sparse part D_0 is of size $s = O(n)$ and the dense part D_1 contains very few $d = \lambda + O(\log n)$ items. Generally speaking, for a set of key-value pairs $L = \{(k_1, v_1), \dots, (k_n, v_n)\}$, the encoding algorithm constructs a matrix A based on the keys $k_1, \dots, k_n$, where the i-th row of A equals $l(k_i)\|r(k_i)$ for two mappings $l : \{0,1\}^* \to \{0,1\}^s$ with constant weight-α output and $r : \{0,1\}^* \to \{0,1\}^d$. Here, l can be represented by $\{l_i : \{0,1\}^* \to [s]\}_{i\in[\alpha]}$, where α is a small constant (e.g., 2 or 3). Namely, $l(k)$ is 1 only in α positions, i.e., $l_1(k)$, ..., $l_\alpha(k)$. Thus, it holds the sparsity property as defined in the above. The output $D_0\|D_1$ satisfies that $A \cdot (D_0\|D_1)^T = (v_1, \dots, v_n)$. To solve this equation, there are several novel methods such as Cuckoo graph strategies [23, 43] and triangulation-based techniques [48]. We refer the interested readers to their works [23, 43, 48].

Functionality $\mathcal{F}_{\mathsf{OKVR}}$

Parameters: Client $\mathcal{C}$ and server $\mathcal{S}$. The input sizes of $\mathcal{C}$ and $\mathcal{S}$ are t and n, respectively, where $n \gg t$. The output size of $\mathcal{C}$ is t. The key space $\mathcal{K}$ and the value space $\mathcal{V}$.

Functionality:

1. Wait for input a set of keys $Q = \{q_1, \dots, q_t\} \in \mathcal{K}^t$ from $\mathcal{C}$.
2. Wait for input a set of key-value pairs $L = \{(k_1, v_1), \dots, (k_n, v_n)\} \in (\mathcal{K} \times \mathcal{V})^n$ from $\mathcal{S}$.
3. Output $Z := \{z_1, \dots, z_t\} \in \mathcal{V}^t$ to $\mathcal{C}$, where $z_i = v_j$ if $q_i = k_j$ and $(k_j, v_j) \in L$, otherwise z_i is uniformly sampled from $\mathcal{V}$.

Figure 4: Ideal functionality for oblivious key-value retrieval

4 Oblivious Key-Value Retrieval

4.1 Definition

We propose a new functionality called Oblivious Key-Value Retrieval (OKVR). The formal definition of OKVR is given in Figure 4. Generally speaking, the server has a large set of key-value pairs $L = \{(k_1, v_1), \dots, (k_n, v_n)\}$ and the client holds a small set of keys $Q = \{q_1, \dots, q_t\}$, where $n \gg t$. For each $q_i \in Q$, the client wants to obtain the corresponding value v_j if $q_i = k_j$ and $(k_j, v_j) \in L$, and a uniformly random value otherwise. After the protocol execution, neither party should gain any additional information. In particular, the server should not learn which value was retrieved, while the client should not learn the other key-value pairs in the server's set. Moreover, the client even does not know whether the query is in the server's key-value pairs, when the server's values are uniformly random.

A similar functionality is Batch Oblivious Programmable Pseudorandom Function (B-OPPRF) [45]. We explain the differences between our OKVR and B-OPPRF as follows. (1) Unlike B-OPPRF, we particularly focus on the unbalanced setting and aim to achieve sub-linear communication on the larger set. (2) While B-OPPRF requires a specific distribution (e.g., related but uniform distribution) of values, our OKVR functionality supports arbitrary values. Given these features, OKVR itself may be of independent interest and can be employed for key-value retrieval scenarios to ensure privacy for both parties in the client-server setting. We also note that our OKVR can be viewed as a batched variant of symmetric keyword PIR [14] with the following exception: our OKVR requires responding to a random value when the query is not in the key-value pairs, but symmetric keyword PIR returns a special symbol for this situation. Below, we give an efficient construction of OKVR, which is built on sparse OKVS in Section 3 and crucially exploits its sparsity to facilitate efficiency.

4.2 Construction

We introduce the construction of OKVR in Figure 5 based on sparse OKVS, which achieves *sublinear* communication cost in the size of the server's large set. The core idea of our OKVR protocol is that the server first exploits a sparse OKVS to encode the key-value pairs $L = \{(k_1, v_1), \ldots, (k_n, v_n)\}$ into a compact vector $D_0 \| D_1$. Then, the client employs a Batch Private Information Retrieval[3] (BatchPIR) protocol [5, 40] to secretly retrieve the desirable items from $D_0 \| D_1$ that are used to compute the corresponding values of the query $Q = \{q_1, \ldots, q_t\}$. Note that this requires $t \cdot (\alpha + |D_1|)$ PIR queries, where α is the access number on the sparse part D_0 required for decoding. Such a large number of PIR queries on the dense part D_1, e.g., $|D_1| = \lambda + O(\log n)$ for GCT [23], will dominate the overhead of this solution.

To further improve performance, we present a hybrid PIR strategy exploiting the sparsity property of sparse OKVS. That is the decoding process requires only accessing a small constant number α of positions from the sparse part D_0, while accessing a large number of positions from the dense part D_1 of small size. In our hybrid PIR strategy, the two parties invoke PIR only on D_0, while D_1 is directly sent to the client by the server. As a result, this strategy takes $\alpha \cdot t$ PIR queries in total, significantly saving $t \cdot |D_1|$ PIR invocations.

Besides, to hide information of the server's key-value pairs L, we additionally invoke a multi-point Oblivious Pseudorandom Function (mpOPRF) protocol [31]. It takes the query $Q = \{q_1, \ldots, q_t\}$ from the client and the PRF key k from the server, and returns $F(\mathsf{k}, q_i)$ to the client for $q_i \in Q$. We let the server compute $D_0 \| D_1$ on PRF-masked key-value pairs, denoted as $L' = \{(k_i, v_i + F(\mathsf{k}, k_i))\}_{i \in [n]}$. After PIR, the client can de-mask retrieved items using $\{F(\mathsf{k}, q_i)\}_{i \in [t]}$ to obtain the desired values.

We analyze the asymptotic communication complexity, which consists of three parts. (1) The mpOPRF protocol requires $O(t)$ communication cost. (2) $\alpha \cdot t$ PIR queries on D_0 consume $O(t \cdot \sqrt{n})$ or $O(t \cdot \log n)$ depending on PIR schemes [5,38,40]. (3) Taking GCT as an example, sending D_1 requires $\lambda + O(\log n)$ communication cost. Therefore, the asymptotic communication cost of OKVR scales sublinearly with the server's set size n.

Theorem 1. *Given a BatchPIR scheme with client query privacy and a sparse OKVS algorithm, the protocol in Figure 5 securely computes $\mathcal{F}_{\mathrm{OKVR}}$ in Figure 4 against semi-honest adversaries in the $\mathcal{F}_{\mathrm{mpOPRF}}$-hybrid model.*

Proof. We prove the following two properties.

Correctness. There are two cases for the correctness analysis. (1) In the case $q_i \in \{k_1, \ldots, k_n\}$ and $q_i = k_j$, according to the correctness of mpOPRF, sparse OKVS, and BatchPIR, $z_i = \mathsf{Decode}_H(D_0 \| D_1, q_i) - F(\mathsf{k}, q_i) = \mathsf{Decode}_H(D_0 \| D_1, k_j) - F(\mathsf{k}, k_j) = v_j$ with overwhelming probability. (2) In the case $q_i \notin \{k_1, \ldots, k_n\}$, due to the pseudorandomness of the underlying PRF, $z_i = \mathsf{Decode}_H(D_0 \| D_1, q_i) - F(\mathsf{k}, q_i)$ is pseudo-random.

Security. We exhibit simulators $\mathsf{Sim}_{\mathcal{C}}$ and $\mathsf{Sim}_{\mathcal{S}}$ for simulating the view of the corrupt client $\mathcal{C}$ and server $\mathcal{S}$, respectively, and prove that the simulated view is indistinguishable from the real one via standard hybrid arguments.

Corrupt client. $\mathsf{Sim}_{\mathcal{C}}(Q, Z)$ simulates the view of the corrupt $\mathcal{C}$. It executes as follows:

- $\mathsf{Sim}_{\mathcal{C}}$ samples uniform values $q'_i \leftarrow \mathbb{F}$ for $i \in [t]$. Then, it invokes the mpOPRF receiver's simulator $\mathsf{Sim}^R_{\mathrm{mpOPRF}}(Q, Q')$, where $Q' := \{q'_1, \ldots, q'_t\}$, and appends the output to the view.
- $\mathsf{Sim}_{\mathcal{C}}$ uniformly samples $D_0 \| D_1 \leftarrow \mathbb{F}^s \times \mathbb{F}^d$ such that for $q_i \in Q$, $\mathsf{Decode}(D_0 \| D_1, q_i) = z_i + q'_i$, where $q'_i \in Q'$. Moreover, $\mathsf{Sim}_{\mathcal{C}}$ follows the real protocol to generate qu and computes $\mathsf{Response}(D_0, qu) \rightarrow res$, and appends res and D_1 to the view.

We argue that the view output by $\mathsf{Sim}_{\mathcal{C}}$ is indistinguishable from the real one. We first define three hybrid transcripts T_0, T_1, T_2, where T_0 is the real view of $\mathcal{C}$, and T_2 is the output of $\mathsf{Sim}_{\mathcal{C}}$.

1. Hybrid_0. The first hybrid is the real interaction described in Figure 5. Here, an honest $\mathcal{S}$ uses real inputs and interacts with the corrupt $\mathcal{C}$. Let T_0 denote the real view of $\mathcal{C}$.
2. Hybrid_1. Let T_1 be the same as T_0, except that the mpOPRF execution is replaced by the mpOPRF receiver's simulator $\mathsf{Sim}^R_{\mathrm{mpOPRF}}(Q, Q')$, where Q' has t elements $\{q'_1, \ldots, q'_t\}$, each of which is uniformly sampled from $\mathbb{F}$. The simulator security of mpOPRF and pseudorandomness of the underlying PRF guarantee this view is indistinguishable from T_0.
3. Hybrid_2. Let T_2 be the same as T_1, except that $D_0 \| D_1$ is sampled uniformly from $\mathbb{F}^s \times \mathbb{F}^d$ such that for $q_i \in Q$, $\mathsf{Decode}(D_0 \| D_1, q_i) = z_i + q'_i$, where $q'_i \in Q'$. Moreover, $\mathsf{Sim}_{\mathcal{C}}$ follows the real protocol to generate qu and $\mathsf{Response}(D_0, qu) \rightarrow res$ is executed locally by $\mathsf{Sim}_{\mathcal{C}}$. By the double obliviousness property of sparse OKVS, the simulated $D_0 \| D_1$ has the same distribution as it would in the real protocol. Consequently, both distributions of $D_0 \| D_1$ are randomly uniform with the constraint $\mathsf{Decode}(D_0 \| D_1, q_i) = z_i + q'_i$. Hence, T_1 and T_2 are statistically indistinguishable. This hybrid is exactly the view output by the simulator.

Corrupt server. $\mathsf{Sim}_{\mathcal{S}}(L, \perp)$ simulates the view of the corrupt $\mathcal{S}$. It executes as follows:

[3] Compared to PIR, BatchPIR [5,40] has concretely lower communication and computation overhead when performing batch retrievals.

Protocol Π_{OKVR}

Parameters: Client $\mathcal{C}$ and server $\mathcal{S}$. Functionality $\mathcal{F}_{\text{mpOPRF}}$ for $F : \{0,1\}^\kappa \times \{0,1\}^* \to \mathbb{F}$ in Figure 2. A sparse OKVS scheme $(\mathsf{Encode}_H, \mathsf{Decode}_H)$ with $\mathcal{K} = \{0,1\}^*$ and $\mathcal{V} = \mathbb{F}$ over a field $\mathbb{F}$, where H consists of $\alpha+1$ random mappings $\{l_i : \{0,1\}^* \to [s]\}_{i\in[\alpha]}$ and $r : \{0,1\}^* \to \{0,1\}^d$. A BatchPIR scheme $(\mathsf{Query}, \mathsf{Response}, \mathsf{Extract})$.

Inputs: $\mathcal{C}$ inputs a set of keys $Q = \{q_1,\dots,q_t\} \in (\{0,1\}^*)^t$. $\mathcal{S}$ inputs a set of key-value pairs $L = \{(k_1,v_1),\dots,(k_n,v_n)\} \in (\{0,1\}^* \times \mathbb{F})^n$.

Protocol execution:

1. $\mathcal{C}$ and $\mathcal{S}$ invoke $\mathcal{F}_{\text{mpOPRF}}$, in which $\mathcal{C}$ acts as the receiver with the input Q and $\mathcal{S}$ acts as the sender with the input $\mathsf{k} \leftarrow \{0,1\}^\kappa$. Then, $\mathcal{C}$ obtains $Q' := \{q'_1,\dots,q'_t\}$, where $q'_i := F(\mathsf{k}, q_i) \in \mathbb{F}$ for $i \in [t]$.
2. $\mathcal{S}$ invokes sparse OKVS to compute $D_0 \| D_1 := \mathsf{Encode}_H(L') \in \mathbb{F}^s \times \mathbb{F}^d$, where $L' := \{(k_1, v_1 + F(\mathsf{k},k_1)),\dots,(k_n, v_n + F(\mathsf{k},k_n))\}$.
3. $\mathcal{C}$ computes a set $I := \{l_j(q_i)\}_{i\in[t],j\in[\alpha]}$ of size αt. When there are collisions, I needs to be filled with distinct values to size αt. $\mathcal{C}$ executes $\mathsf{Query}(I) \to (qu, st)$ of BatchPIR and sends qu to $\mathcal{S}$.
4. $\mathcal{S}$ executes $\mathsf{Response}(D_0, qu) \to res$ on the sparse part D_0 and sends res to $\mathcal{C}$. In parallel, $\mathcal{S}$ also sends the dense part D_1 to $\mathcal{C}$.
5. $\mathcal{C}$ invokes $\mathsf{Extract}(st, res)$ to learn $\{D_0[l_j(q_i)]\}_{i\in[t],j\in[\alpha]}$. $\mathcal{C}$ computes and outputs $Z := \{z_1,\dots,z_t\}$ with $z_i := \mathsf{Decode}_H(D_0\|D_1, q_i) - q'_i$, where $\mathsf{Decode}_H(D_0\|D_1, q_i) := \sum_{j\in[\alpha]} D_0[l_j(q_i)] + \langle r(q_i), D_1\rangle$ over $\mathbb{F}$.

Figure 5: Protocol for oblivious key-value retrieval

- $\text{Sim}_\mathcal{S}$ generates a random key k. Then, it invokes the mpOPRF sender's simulator $\text{Sim}^S_{\text{mpOPRF}}(\mathsf{k}, \perp)$ and appends the output to the view.
- $\text{Sim}_\mathcal{S}$ samples uniformly random $I = \{i_1,\cdots,i_{\alpha t}\} \leftarrow [s]^{\alpha t}$ with distinct values and invokes $\mathsf{Query}(I) \to (qu, st)$. $\text{Sim}_\mathcal{S}$ appends qu to the view.

We argue that the view output by $\text{Sim}_\mathcal{S}$ is indistinguishable from the real one. We first define three hybrid transcripts T_0, T_1, T_2 where T_0 is the real view of $\mathcal{S}$, and T_2 is the output of $\text{Sim}_\mathcal{S}$.

1. Hybrid_0. The first hybrid is the real interaction described in Figure 5. Here, an honest $\mathcal{C}$ uses real inputs Q and interacts with the corrupt $\mathcal{S}$. Let T_0 denote the real view of $\mathcal{S}$.
2. Hybrid_1. Let T_1 be the same as T_0, except that the mpOPRF execution is replaced by its sender's simulator $\text{Sim}^S_{\text{mpOPRF}}(\mathsf{k}, \perp)$ where k is uniformly sampled by $\text{Sim}_\mathcal{S}$. The security of the mpOPRF functionality guarantees the view is indistinguishable from the real execution.
3. Hybrid_2. Let T_2 be the same as T_1, except that the query index set I is replaced by uniformly random $i_1,\cdots,i_{\alpha t} \in [n]^{\alpha t}$ with distinct values. This hybrid is computationally indistinguishable from T_1 by the client query privacy of the BatchPIR scheme. Specifically, if there is a distinguisher $\mathcal{D}$ that can distinguish T_1 and T_2 with non-negligible probability, then we can construct a PPT adversary $\mathcal{A}$ to break the client query privacy of the BatchPIR scheme as follows: $\mathcal{A}$ sends I_0 and I_1 as the challenge message to the challenger, where I_0 is the query index generated using $\mathcal{C}$'s real inputs and I_1 is uniformly sampled. Then, $\mathcal{A}$ receives the query ciphertext $\mathsf{Query}(I_b) \to qu$ from the challenger, where b is uniformly sampled. Then, $\mathcal{A}$ executes as $\mathcal{C}$ in Hybrid_1 with the corrupt $\mathcal{S}$ except the BatchPIR query generation step. After that, $\mathcal{A}$ invokes $\mathcal{D}$ with $\mathcal{S}$'s view in the above interaction and outputs $\mathcal{D}$'s output. Note that if $qu \leftarrow \mathsf{Query}(I_0)$, the view of the corrupt $\mathcal{S}$ is exactly T_1. If $qu \leftarrow \mathsf{Query}(I_1)$, the resulting view corresponds to T_2. Therefore, $\mathcal{A}$ can break the client query privacy of the BatchPIR scheme with the same advantage as $\mathcal{D}$.

□

5 Unbalanced Circuit-PSI

5.1 Definition

We give the formal definition of the unbalanced circuit-PSI (UCPSI) functionality in Figure 6. This functionality closely follows prior circuit-PSI works [45, 51], except with a particular focus on the unbalanced setting. Generally speaking,

Functionality $\mathcal{F}_{\mathsf{UCPSI}}$

Parameters: Client $\mathcal{C}$ and server $\mathcal{S}$. The input sizes of $\mathcal{C}$ and $\mathcal{S}$ are t and n, respectively, where $n \gg t$. The output size is $m = (1+\varepsilon)\cdot t$ with a constant $\varepsilon > 0$. A function $\mathsf{Reorder} : (\{0,1\}^*)^t \to (\pi : [t] \to [m])$, which on input a set of size t outputs an injective mapping π.

Functionality:

1. Wait for input a set $X = \{x_1, \ldots, x_t\} \in (\{0,1\}^*)^t$ from $\mathcal{C}$.
2. Wait for input a set $Y = \{y_1, \ldots, y_n\} \in (\{0,1\}^*)^n$ from $\mathcal{S}$.
3. Compute $\pi \leftarrow \mathsf{Reorder}(X)$.
4. For $i \in [m]$, sample $\langle b_i \rangle_0^B, \langle b_i \rangle_1^B \in \{0,1\}$ uniformly such that $\langle b_i \rangle_0^B \oplus \langle b_i \rangle_1^B = 1$ if $\exists x_{i'} \in X, y_j \in Y$ s.t. $x_{i'} = y_j$ where $i = \pi(i')$, and $\langle b_i \rangle_0^B \oplus \langle b_i \rangle_1^B = 0$ otherwise.
5. Output $(\{\langle b_i \rangle_0^B\}_{i\in[m]}, \pi)$ to $\mathcal{C}$ and $\{\langle b_i \rangle_1^B\}_{i\in[m]}$ to $\mathcal{S}$.

Figure 6: Ideal functionality for unbalanced circuit-PSI

it allows the client to input a small set X of size t and the server to input a large set Y of size n, where $n \gg t$. After the protocol, the two parties will learn secret shares of whether an element of X lies in the intersection. These secret-shared outputs along with X can then be used in any subsequent secure computation [36, 52].

5.2 Construction

We propose the construction of UCPSI from OKVR in Figure 7. We emphasize that the main difference from previous circuit-PSI works [10, 45, 48, 51] is that our UCPSI achieves sublinear communication complexity on the size of the larger set. Below, we explain our idea by first giving a basic construction from our OKVR functionality, followed by an optimized version employing a *padding-free* OKVR technique tailored to UCPSI.

Basic construction. Following circuit-PSI, we utilize hash-to-bin techniques [44, 45], which reduce the intersection evaluation on all items to only execute on bins with fewer items. The construction contains the following three steps.

(1) *Hash-to-bin.* The hash-to-bin technique employs Cuckoo hashing [41] and simple hashing as follows. Given the set X of size t, the client creates a Cuckoo hash table T_X of size $m = (1+\varepsilon)\cdot t$ with β hash functions $h_1, \ldots, h_\beta : \{0,1\}^* \to [m]$, where $\varepsilon > 0$ is a constant. It ensures that for each $x_i \in X$, $x_i \| j$ is stored at $T_X[h_j(x_i)]$ for some $j \in [\beta]$. Due to $m > t$, there are some empty bins and we require padding these bins with dummy items that are marked as $\perp$. On the other hand, the server constructs a simple hash table T_Y of size m using the same hash functions as the above. Specifically, given the set Y of size n, for each $y_i \in Y$, this table stores $y_i \| j$ at all locations $T_Y[h_j(y_i)]$ for $j \in [\beta]$. In T_Y, each bin may hold more than one item. Unlike other works [16, 36] that pad all bins to a pre-defined maximum size with dummy items, we do not need padding since our protocol will combine the items of all T_Y's bins into a single set for subsequent steps. Obviously, the total number of items in all T_Y's bins is fixed, i.e., βn.

(2) *OKVR.* Since both parties use the same hash functions, our scheme only requires checking whether the item placed in a bin by the client is among the items placed in this bin by the server. We employ our OKVR to accomplish this functionality. Specifically, for each i-th bin, the server samples a random value r_i and constructs a set of key-value pairs $P_i := \{(y', r_i)\}$ for all $y' \in T_Y[i]$. The client and the server invoke the OKVR protocol with input T_X and $\{P_i\}_{i\in[m]}$, respectively. After the protocol, the client obtains r_i^* for $i \in [m]$. Note that r_i^*'s are uniformly random, because in each bin of T_Y the client only queries once.

(3) *Private equality test.* In this step, the client and server check $r_i = r_i^*$, and compute boolean secret shares of b_i that indicates whether the client's i-th element is in the server's set. This operation is achieved by the private equality test (EQ) functionality $\mathcal{F}_{\mathrm{EQ}}$ in Figure 3. This operation causes a small overhead in our unbalanced case since we only require $O(t)$ invocations, where $t \ll n$.

Optimization from padding-free OKVR. The dominant cost of the basic construction stems from the OKVR step, which requires BatchPIR with $m = (1+\varepsilon)\cdot t$ queries on the server's dataset $\{P_i\}_{i\in[m]}$. To reduce this overhead, we propose padding-free OKVR that only consumes t BatchPIR queries. We emphasize this reduction is important, since as shown in Section 6.1, ε is usually chosen as 0.27.

The main insight is that OKVR outputs a random value when a query is not found in the server's key set, which is actually the case for dummy items $\perp$ in the table T_X. In detail, before BatchPIR queries, the client has known that these $\perp$ items are not in the intersection, and hence a random r_i^* will be obtained according to the functionality of OKVR. Therefore, we present an efficient padding-free strategy for invoking OKVR. In particular, let $\pi : [t] \to [m]$ be an injective mapping that maps an element index of X to the position in T_X, i.e., $\pi(i) = h_j(x_i)$ such that $T_X[h_j(x_i)] = x_i \| j$. Then, the client invokes $\mathcal{F}_{\mathrm{OKVR}}$ only with$\{T_X[\pi(i)]\}_{i\in[t]}$ and learns $\{r^*_{\pi(i)}\}_{i\in[t]}$, and for $j \in [m] \backslash \{\pi(i)\}_{i\in[t]}$, r_j^* is sampled uniformly from $\{0,1\}^\ell$ by the client. One may question whether it leaks to the server the mapping of the client's Cuckoo hashing table, which depends on the client's private input set. We note that this is not the case because the client combines all queries into a batch to perform BatchPIR on a single database from key-value pairs of all server's bins. We give the formal description of our UCPSI protocol in Figure 7, and the correctness and security are analyzed below.

Protocol Π_{UCPSI}

Parameters: Client $\mathcal{C}$ and server $\mathcal{S}$. β hash functions $\{h_i : \{0,1\}^* \to [m]\}_{i\in[\beta]}$ used in Cuckoo hashing, where $m = (1+\varepsilon)\cdot t$ with a constant $\varepsilon > 0$. Ideal functionalities $\mathcal{F}_{\mathrm{EQ}}$ in Figure 3 with input bitlength $\ell := \lambda + \log m$ and $\mathcal{F}_{\mathrm{OKVR}}$ in Figure 4.

Inputs: $\mathcal{C}$ inputs a set $X = \{x_1, \ldots, x_t\} \in (\{0,1\}^*)^t$. $\mathcal{S}$ inputs a set $Y = \{y_1, \ldots, y_n\} \in (\{0,1\}^*)^n$.

Protocol execution:

1. $\mathcal{C}$ maps X into a Cuckoo hash table T_X of m bins, such that for any $x \in X$, there exists an $j \in [\beta]$ satisfying $T_X[h_j(x)] = x\|j$. Define a function $\pi : [t] \to [m]$ that maps an element index of X to the position in T_X, i.e., $\pi(i) = h_j(x_i)$ such that $T_X[h_j(x_i)] = x_i\|j$.
2. $\mathcal{S}$ maps Y into a simple hash table T_Y of m bins, such that for any $y \in Y$ and all $j \in [\beta]$, it holds that $y\|j \in T_Y[h_j(y)]$.
3. For $i \in [m]$, $\mathcal{S}$ samples random $r_i \in \{0,1\}^\ell$, and defines $P_i := \{(y', r_i)\}$ for all $y' \in T_Y[i]$.
4. $\mathcal{C}$ and $\mathcal{S}$ invoke $\mathcal{F}_{\mathrm{OKVR}}$ with input $\{T_X[\pi(i)]\}_{i\in[t]}$ and $\{P_i\}_{i\in[m]}$, respectively. $\mathcal{C}$ initializes empty $R^* := \{r_1^*, \ldots, r_m^*\}$ and assigns the output of $\mathcal{F}_{\mathrm{OKVR}}$ to $\{r^*_{\pi(i)}\}_{i\in[t]}$. For $j \in [m]\backslash\{\pi(i)\}_{i\in[t]}$, r_j^* is sampled uniformly from $\{0,1\}^\ell$.
5. For $i \in [m]$, $\mathcal{C}$ and $\mathcal{S}$ invoke $\mathcal{F}_{\mathrm{EQ}}$ with input r_i^* and r_i, respectively. As a result, they learn boolean shares $\langle b_i\rangle_0^B$ and $\langle b_i\rangle_1^B$, respectively, where $b_i = 1$ if $r_i^* = r_i$ and $b_i = 0$ otherwise.

Figure 7: Protocol for unbalanced circuit-PSI

Theorem 2. *The protocol in Figure 7 securely computes $\mathcal{F}_{\mathrm{UCPSI}}$ in Figure 6 against semi-honest adversaries in the $(\mathcal{F}_{\mathrm{OKVR}}, \mathcal{F}_{\mathrm{EQ}})$-hybrid model.*

Proof. We prove the following two properties.

Correctness. There are three cases for the correctness analysis. (1) In the case where $x \in X \cap Y$, there exists a unique j such that $T_X[h_j(x)] = x\|j$ and $y\|j \in T_Y[h_j(y)]$ and $x = y$ with an overwhelming probability according to the statistical analysis of Cuckoo hashing. From the correctness of OKVR, when $\mathcal{S}$ samples r, $\mathcal{C}$ will obtain r^* such that $r^* = r$. From the correctness of EQ, $\mathcal{C}$ and $\mathcal{S}$ obtain secret shares of $b = 1$. (2) In the case where $x \in X \setminus Y$, there exists a unique j such that $T_X[h_j(x)] = x\|j$ but $T_X[h_j(x)] \notin T_Y[h_j(x)]$. According to the correctness of OKVR, $\mathcal{C}$ will obtain a random r^*. The collision may occur namely $r^* = r \wedge x \notin Y$, which violates the correctness. The false positive error probability in each bin equals $2^{-\ell}$. By setting $\ell = \lambda + \log m$, a union bound shows the overall probability of false positive is $2^{-\lambda}$, which is negligible. Due to $r^* \neq r$ with overwhelming probability and the correctness of EQ, $\mathcal{C}$ and $\mathcal{S}$ obtain secret shares of $b = 0$. (3) In the case of dummy points, $\mathcal{C}$ directly samples r^* uniformly at random. The correctness is the same as the second case.

Security. We exhibit simulators $\mathrm{Sim}_{\mathcal{C}}$ and $\mathrm{Sim}_{\mathcal{S}}$ for simulating corrupt $\mathcal{C}$ and $\mathcal{S}$, respectively, and argue the indistinguishability of the produced transcript from the real execution.

Corrupt client. $\mathrm{Sim}_{\mathcal{C}}(X, (\pi, \{\langle b_i\rangle_0^B\}_{i\in[m]}))$ simulates the view of the corrupt $\mathcal{C}$. It executes as follows:

- $\mathrm{Sim}_{\mathcal{C}}$ uniformly samples $R^* := \{r_1^*, \ldots, r_m^*\}$, and invokes the OKVR's client simulator $\mathrm{Sim}^{\mathcal{C}}_{\mathrm{OKVR}}(\{T_X[\pi(i)]\}_{i\in[t]}, \{r^*_{\pi(i)}\}_{i\in[t]})$. $\mathrm{Sim}_{\mathcal{C}}$ appends the output to the view.
- $\mathrm{Sim}_{\mathcal{C}}$ invokes the EQ's client simulator $\mathrm{Sim}^{\mathcal{C}}_{\mathrm{EQ}}(r_i^*, \langle b_i\rangle_0^B)$ for $i \in [m]$, where $r_i^* \in R^*$, and appends the output to the view.

The view simulated by $\mathrm{Sim}_{\mathcal{C}}$ is computationally indistinguishable from the real one by the underlying simulators' indistinguishability. It is worth noting that although for all keys in each set P_i of key-value pairs, the corresponding value is always a uniformly random r_i, the client's output $\{r_i^*\}_{i\in[m]}$ of OKVR is still uniformly random. The reason is that for each P_i, at most one value is retrieved by the client according to the property of Cuckoo hashing.

Corrupt server. $\mathrm{Sim}_{\mathcal{S}}(Y, \{\langle b_i\rangle_1^B\}_{i\in[m]})$ simulates the view of the corrupt $\mathcal{S}$. It executes as follows:

- $\mathrm{Sim}_{\mathcal{S}}$ samples uniformly random $\{r_1, \ldots, r_m\}$ and generates $\{P_i\}_{i\in[m]}$ like Figure 7, and invokes the OKVR's server simulator $\mathrm{Sim}^{\mathcal{S}}_{\mathrm{OKVR}}(\{P_i\}_{i\in[m]}, \perp)$. $\mathrm{Sim}_{\mathcal{S}}$ appends the output to the view.
- For $i \in [m]$, $\mathrm{Sim}_{\mathcal{S}}$ invokes the EQ's server simulator $\mathrm{Sim}^{\mathcal{S}}_{\mathrm{EQ}}(r_i, \langle b_i\rangle_1^B)$ where r_i is sampled as above, and appends the output to the view.

It is straightforward to see that the view simulated by $\mathrm{Sim}_{\mathcal{S}}$ is computationally indistinguishable from the real one by the OKVR and EQ simulators' indistinguishability.

□

6 Implementation and Evaluation

We implement our protocols in Java, and run the experiments on a single Intel Core i9-9900K with 3.6GHz and 128GB RAM. All evaluations are performed using 8 threads. We simulate the network connection using the Linux tc command. The simulated network settings include LAN (10Gbps bandwidth and 0.05ms RTT latency) and WAN (50Mbps bandwidth with 80ms RTT latency). Given that we focus on the unbalanced setting, where the client may only have constrained resources (e.g., very low bandwidth), we also test our protocols on a simulated mobile network setting (1Mbps bandwidth and 80ms RTT latency). Our source code is available at https://github.com/alibaba-edu/mpc4j.

6.1 Implementation Details

A unified circuit-PSI framework. We provide a comprehensive circuit-PSI framework that serves as a unified platform for implementing both our proposed protocols and state-of-the-art circuit-PSI schemes. This framework involves a full re-implementation of existing circuit-PSI schemes, such as the general circuit-PSI protocols[4] (PSTY19 [45], CGS22 [10] and RR22 [48]) and the unbalanced circuit-PSI protocol SJ23 [54]. The necessity for a large-scale re-implementation arises from the original implementations being conducted under different underlying protocols and experimental settings. This diversity poses challenges for fair comparisons. Specifically, some prior implementations, like CGS22, focus solely on balanced circuit-PSI, lacking support for unbalanced cases[5]. (2) There are variations in performance evaluations due to different thread settings: PSTY19, CGS22, and SJ23 evaluate their protocols in the single-thread setting, while RR22 employs multi-thread execution. (3) Different protocols instantiate underlying cryptographic primitives. For instance, PSTY19 and CGS22 utilize IKNP OT for the equality test, while RR22 and SJ23 use different silent OT variants, namely Silver OT [18] and Ferret OT [56], respectively. To address these challenges, we provide unified implementations and conduct fair comparisons within our framework. The details are elaborated in the subsequent sections.

Implementation details of our protocols. We set the computational security parameter $\kappa = 128$ and the statistical security parameter $\lambda = 40$. The used dataset is 128-bit random strings. We note that our implementation is generic and the dataset can be set as arbitrary values of any length. Following existing circuit-PSI protocols [10, 45, 48], we set the size of the server's set as $2^{20} \sim 2^{22}$. We choose the smaller size of the client's set as $2^4 \sim 2^{12}$, since the client is defined to use a smaller set in our focused unbalanced setting. To complete our constructions, we implement the following components.

Stash-less Cuckoo hashing. We use stash-less Cuckoo hashing with 3 hash functions that store n elements into $\lceil (1+\varepsilon)\cdot n \rceil$ bins, where $\varepsilon > 0$. Based on the analysis of PSZ18 [47], ε is usually chosen as 0.27, which has been widely used in prior works [10, 45, 51].

Blazing-fast OKVS. We implement the blazing-fast OKVS with clustering as our sparse OKVS, as proposed by RR22 [48], following their open-source code [2] as a reference. The underlying field $\mathbb{F}$ is instantiated as $GF(2^\ell)$. Specifically, the elements are distributed into several buckets through a hash function. Each bucket independently generates an OKVS, and these individual OKVS are subsequently merged. Notably, the OKVS encoding for each bucket operates independently and is amenable to parallel processing, leading to a substantial enhancement in efficiency. The blazing-fast OKVS offers two variants employing 2 or 3 hashing functions, both of which are included in our evaluation for a comprehensive assessment.

BatchPIR. We implement the state-of-the-art vectorized BatchPIR [40], utilizing their open-source code [4] as a reference. We specifically opt for the BatchPIR variant due to its support for batched PIR queries with very low communication costs. We adopt the JNI technique to invoke the BFV homomorphic encryption [8, 21] provided by Microsoft SEAL library v4.1 [3]. Our implementation aligns with the chosen parameter settings and also leverages the modulus switching technique to effectively reduce the size of response ciphertexts.

mpOPRF. We implement OMG-DH-based OPRF [31, 50], enabling the server to encode inputs and initialize PIR protocols in the setup phase (specific details of the setup setting will be elaborated later). For better efficiency, we use Four$\mathbb{Q}$ elliptic curve [17], renowned for its rapid scalar multiplication of random points and an efficient implementation of hash-to-curve operations. We note that existing unbalanced PSI protocols integrate Four$\mathbb{Q}$ and similar techniques to expedite the setup phase [11, 12, 16].

Equality test. We implement the private equality test (EQ) protocol in CGS22 [10]. This protocol has low communication costs and is rooted in the solution to the Millionaires' problem within the CrypTFlow2 framework [49]. It recursively performs equality tests on substrings organized in a tree structure. To conduct tests on leaf substrings, a 1-out-of-2^s OT scheme is employed, where s is a constant and we set $s = 4$, aligning with the choice made in [10]. Our OT implementation utilizes a silent OT variant, namely, Ferret OT [56].

Pre-processing setting. Similar to unbalanced PSI protocols [11, 12, 16], we establish our framework within the pre-processing setting. This configuration allows the server to pre-process its input set during the setup phase, enhancing online performance. Specifically, the pre-processing in our

[4] Bienstock et al. [6] recently propose nearly-optimal OKVS and also apply it in circuit-PSI (called BPSY23). Currently, we do not include this baseline because (to the best of our knowledge) we cannot find a way of supporting multi-thread encoding for their OKVS construction without increasing the encoding rate. Note that BPSY23 has a similar performance as RR22 (Refer to Figure 7 of BPSY23).

[5] See Issue #4 of the CGS22 open-source implementation for details.

Table 1: Performance of our OKVR.

Protocol	Param.		Comm. (MB)		Encoding Length		Time LAN (s)		Time WAN (s)		Time Mobile (s)	
	n	t	Setup	Online	Sparse	Dense	Setup	Online	Setup	Online	Setup	Online
OKVR 2-Hash	2^{20}	2^4	17.875	0.672	2,222,080	5,120	34.465	5.968	37.999	6.460	184.990	12.373
		2^8		1.950			45.037	1.578	48.936	2.338	195.781	18.729
		2^{12}		12.924			103.210	3.681	106.693	6.874	254.439	113.133
	2^{22}	2^4	17.875	0.789	8,902,656	20,480	97.594	9.414	98.013	9.667	243.289	16.110
		2^8		3.331			126.995	2.320	129.698	3.384	277.929	31.115
		2^{12}		13.041			324.424	5.999	332.950	9.313	484.606	116.570
OKVR 3-Hash	2^{20}	2^4	17.875	0.656	1,406,976	3,072	22.724	2.804	26.462	3.345	173.364	9.088
		2^8		2.566			33.377	1.094	36.544	2.046	184.136	23.381
		2^{12}		9.750			108.453	4.126	109.869	6.709	258.608	86.654
	2^{22}	2^4	17.875	0.726	5,638,144	12,288	67.583	4.336	74.997	5.054	218.303	10.949
		2^8		2.636			131.976	2.690	145.952	3.414	280.235	25.437
		2^{12}		18.664			217.321	5.866	287.687	10.726	400.932	163.956

protocol encompasses tasks such as hashing the input set into bins, encoding elements within these bins using sparse OKVS, and initializing BatchPIR. Notably, these pre-processing tasks are independent of the client's inputs and can be efficiently completed during the setup phase [12].

Implementation details of baselines. We re-implement state-of-the-art circuit-PSI schemes including PSTY19 [45], CGS22 [10], RR22 [48], and SJ23 [54]. Our implementations and comparisons are fair for the following reasons. (1) All their implementations support multi-thread execution and are compatible with both balanced and unbalanced settings. We test both our protocols and prior works under the same and commonly used experimental setting (e.g., thread numbers and security parameters) and the same 128-bit dataset. (2) We instantiate the underlying building blocks using the same state-of-the-art techniques. For example, all protocols invoke Ferret OT [56] as the OT extension. We also introduce the advanced optimizations in prior works. Below, we give more detailed descriptions.

SJ23. We re-implement the two constructions of SJ23 [54] due to the absence of an available open-source implementation. We set the polynomial degree and modulus coefficient based on the analysis provided by SJ23, while also determining the query power according to APSI parameters [1]. In the case of their first construction, we integrate the noise flooding technique [24], introducing an encryption of zero with 40-bit noise during the response generation phase. For the second construction, we adopt SEAL's default configurations for the ciphertext modulus to ensure a sufficient noise budget for the multiplication of a multiplicative mask. Notably, partition numbers per bin in SJ23 are fixed for specific set sizes. To accommodate arbitrary set sizes, we employ the lookup table method, as identified in the RR22 implementation [2], to estimate partition numbers per bin. This approach allows us to support a broader range of set sizes in our re-implementation.

PSTY19. We re-implement PSTY19 while introducing two crucial optimizations to enhance performance. Firstly, we address the computational bottleneck identified in PSTY19 [10,45], specifically associated with polynomial interpolation (MegaBin). To mitigate this, we substitute the MegaBin operation with the 3-Hash Garbled Cuckoo Table (3H-GCT) [23]. Secondly, to further optimize communication costs, we incorporate the state-of-the-art equality test protocol from CGS22 into PSTY19 and instantiate it with Ferret OT.

CGS22 and RR22. We re-implement CGS22 and RR22 and adhere to their original experimental settings. A notable adjustment is made by employing the equality test protocol of CGS22, integrated with Ferret OT, to decrease communication costs in our implementations.

6.2 Experimental Results

We compare our protocols with the state-of-the-art balanced circuit-PSI, i.e., PSTY19 [45], CGS22 [10], and RR22 [48], and unbalanced circuit-PSI SJ23 [54], which consists of two constructions, called SJ23-C1 and SJ23-C2. The runtime and communication costs of our constructions and previous works are shown in Tables 1 and 2. Note that in the setup phase, the communication is dominated by sending Galois and relinearization keys of the underlying homomorphic encryption in BatchPIR to the server. As these keys remain independent of the server's set, they can be sent only once and cached for repeated protocol executions [12, 16]. Consequently, this communication cost can be amortized over multiple client requests.

Performance of OKVR. We first test the performance of our main building block, i.e., OKVR, in Table 1. We report the communication and computation costs to show the scalability of OKVR. For example, for the server's set size 2^{22} and the client's set size 2^{12}, it only requires 13.041 MB and 9.313 seconds in the WAN setting. Moreover, we also test the length of the encoded vector and observe that the length of the dense part is much smaller than that of the sparse part. This is consistent with our construction in Section 4. We note that the size of the dense part can be further reduced when utilizing sparse OKVS without clustering. For instance, with

Table 2: Communication and runtime comparison of our protocols with previous works. The best result is marked in green, and the second best result is marked in blue.

Param.		Protocol	Comm. (MB)		Time LAN (s)		Time WAN (s)		Time Mobile (s)	
n	t		Setup	Online	Setup	Online	Setup	Online	Setup	Online
2^{20}	2^4	PSTY19 [45]	0.308	32.859	0.100	7.020	2.224	18.035	5.162	356.548
		CGS22 [10]	0.317	30.978	0.117	10.866	2.581	21.736	5.377	340.301
		RR22 [48]	1.882	33.087	0.137	7.941	3.386	19.778	21.067	358.152
		SJ23-C1 [54]	1.994	5.938	7.996	4.232	10.380	8.567	18.581	58.258
		SJ23-C2 [54]	3.958	18.608	2.750	2.252	7.599	9.669	40.451	164.546
		Our UCPSI 2-Hash	18.172	1.872	82.666	3.655	87.271	7.310	236.127	23.341
		Our UCPSI 3-Hash	18.172	2.457	58.875	1.937	64.045	5.610	214.787	26.816
	2^8	PSTY19 [45]	0.405	33.057	0.077	6.965	2.188	17.816	5.787	358.192
		CGS22 [10]	0.415	31.196	0.075	11.674	2.575	22.412	6.235	343.088
		RR22 [48]	1.980	33.278	0.134	8.028	3.386	19.117	21.935	360.385
		SJ23-C1 [54]	1.994	5.938	8.265	3.843	9.812	8.752	18.284	57.751
		SJ23-C2 [54]	3.957	18.609	2.828	2.021	7.565	9.247	41.127	165.253
		Our UCPSI 2-Hash	18.270	3.336	113.585	2.616	117.690	6.843	267.438	35.475
		Our UCPSI 3-Hash	18.269	3.921	80.066	1.521	83.482	6.012	235.190	39.270
	2^{12}	PSTY19 [45]	0.687	34.028	0.175	8.221	2.350	18.510	8.380	365.269
		CGS22 [10]	0.697	32.537	0.110	12.426	2.678	23.398	8.767	353.945
		RR22 [48]	2.352	34.080	0.186	8.521	3.459	20.147	25.007	369.325
		SJ23-C1 [54]	4.854	9.669	6.393	11.932	8.955	17.296	41.982	96.145
		SJ23-C2 [54]	3.958	18.607	2.715	1.917	7.960	9.164	40.486	164.379
		Our UCPSI 2-Hash	18.551	17.616	191.363	5.157	199.714	12.317	350.308	158.070
		Our UCPSI 3-Hash	18.551	25.149	161.069	6.254	165.943	14.949	315.529	222.685
2^{22}	2^4	PSTY19 [45]	0.308	130.148	0.090	31.202	2.237	62.721	5.067	1,405.807
		CGS22 [10]	0.318	122.418	0.095	61.104	2.591	91.548	5.343	1,352.874
		RR22 [48]	1.882	130.376	0.133	34.840	3.343	67.732	20.567	1,407.752
		SJ23-C1 [54]	5.227	6.618	102.531	10.102	103.283	15.881	112.038	70.543
		SJ23-C2 [54]	3.958	42.674	11.998	3.728	16.224	15.202	44.846	369.253
		Our UCPSI 2-Hash	18.172	2.856	287.176	7.330	295.251	11.241	443.559	36.117
		Our UCPSI 3-Hash	18.172	2.668	323.034	7.627	316.533	11.544	483.251	35.354
	2^8	PSTY19 [45]	0.405	130.346	0.077	31.955	2.255	64.689	5.838	1,407.101
		CGS22 [10]	0.415	122.636	0.080	64.442	2.670	93.478	6.361	1,357.350
		RR22 [48]	1.980	130.567	0.318	35.146	3.331	64.838	21.478	1,410.698
		SJ23-C1 [54]	5.226	6.618	103.195	9.663	101.296	15.717	111.341	70.589
		SJ23-C2 [54]	3.957	42.673	11.075	3.555	16.572	15.720	45.625	368.650
		Our UCPSI 2-Hash	18.269	5.583	398.922	5.785	399.003	10.471	557.363	57.775
		Our UCPSI 3-Hash	18.269	7.291	287.933	5.425	297.677	10.449	446.333	71.604
	2^{12}	PSTY19 [45]	0.687	131.322	0.281	35.768	2.355	66.032	8.363	1,416.060
		CGS22 [10]	0.697	123.982	0.112	66.434	2.633	101.694	8.766	1,369.970
		RR22 [48]	2.352	131.369	0.162	39.705	3.584	69.666	25.332	1,422.936
		SJ23-C1 [54]	5.227	6.618	101.144	10.897	104.207	16.169	112.113	71.405
		SJ23-C2 [54]	3.957	42.675	11.639	3.464	16.575	15.108	44.739	368.319
		Our UCPSI 2-Hash	18.551	33.130	623.735	13.086	631.618	20.526	795.836	295.674
		Our UCPSI 3-Hash	18.551	25.361	672.840	12.123	674.243	19.188	843.413	230.016

the server's set size 2^{20}, the sizes of the dense part are 72 and 56 in our 2-Hash and 3-Hash OKVR protocols, respectively.

Communication comparison. The comparison between our protocols and recent balanced circuit-PSI techniques, i.e., PSTY19, CGS22, and RR22, is reported in Table 2. We can observe that our protocols achieve the lowest online communication cost, outperforming them by $1.84 \sim 48.86\times$. Moreover, our protocols exhibit greater advantages when the difference between the set sizes of the client and server is significant. In addition, when comparing with SJ23, the communication of our constructions is $1.18 \sim 15.99\times$ better than their protocols in the extremely unbalanced setting, e.g., the client set of size 2^4 and 2^8. For the client's set size 2^{12}, our protocols still outperform the second construction of SJ23.

Runtime comparison. As evidenced by Table 2, our protocols consistently outperform balanced circuit-PSI works PSTY19, CGS22, and RR22 across all set sizes and network setups. Specifically, the runtime efficiency of our constructions surpasses these works by $1.50 \sim 39.81\times$. On the other hand, when compared with SJ23, the runtime of our constructions is $1.22 \sim 10.44\times$ better in the extremely unbalanced setting, e.g., the client set of size 2^4 and 2^8. Moreover, we observe that SJ23 works well for relatively large client's set sizes, e.g., 2^{12}, and has a similar high overhead even when the set size decreases. The reason is that SJ23 requires packing a large number of the client's elements into a ciphertext and amortizing the overhead of expensive FHE operations. Besides, the two SJ23 protocols have varying runtime results between LAN and Mobile settings due to large communication differences, which will affect the performance in bandwidth-limited settings. Rather, our two UCPSI constructions with 2-Hash and 3-Hash have a similar communication cost and particularly perform better in these constraint network settings.

7 Conclusion

In this work, we introduce unbalanced circuit-PSI constructions that achieve sublinear communication complexity in the size of the larger set. The core idea is the functionality and construction of Oblivious Key-Value Retrieval (OKVR), which may be of independent interest. We fully implement our constructions and state-of-the-art circuit-PSI protocols in a unified framework for fair comparisons. We evaluate the efficiency of our constructions and the results show that our scheme achieves up to 48.86× communication improvement and 39.81× runtime reduction over previous works.

Acknowledgments

We would like to express our deepest gratitude for the invaluable help provided by our shepherd as well as all the reviewers for their constructive comments. This work is supported by the National Key R&D Program of China under Grant 2022YFB3103500, the National Natural Science Foundation of China under Grant 62020106013, the Sichuan Science and Technology Program under Grant 2023ZYD0142, the National Research Foundation, Singapore, the Cyber Security Agency under its National Cybersecurity R&D Programme (NCRP25-P04-TAICeN), the Singapore Ministry of Education (MOE) AcRF Tier 2 MOE-T2EP20121-0006, the Nanyang Technological University (NTU)-DESAY SV Research Program under Grant 2018-0980, and the Major Programs of the National Social Science Foundation of China (No. 22&ZD147).

References

[1] Apsi. https://github.com/microsoft/APSI/tree/main/parameters.

[2] Blazing-fast okvs. https://github.com/Visa-Research/volepsi.

[3] Microsoft seal library v4.1. https://github.com/Microsoft/SEAL.

[4] Vectorized batch pir. https://github.com/mhmughees/vectorized_batchpir.

[5] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. Pir with compressed queries and amortized query processing. In *Proceedings of IEEE S&P*, 2018.

[6] Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. Near-optimal oblivious key-value stores for efficient psi, psu and volume-hiding multi-maps. In *Proceedings of USENIX Security*, 2023.

[7] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[8] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Proceedings of CRYPTO*, 2012.

[9] Justin Chan, Dean Foster, Shyam Gollakota, Eric Horvitz, Joseph Jaeger, Sham Kakade, Tadayoshi Kohno, John Langford, Jonathan Larson, Puneet Sharma, et al. Pact: Privacy sensitive protocols and mechanisms for mobile contact tracing. *arXiv preprint arXiv:2004.03544*, 2020.

[10] Nishanth Chandran, Divya Gupta, and Akash Shah. Circuit-psi with linear complexity via relaxed batch opprf. *Proceedings on Privacy Enhancing Technologies*, 1:353–372, 2022.

[11] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled psi from fully homomorphic encryption with malicious security. In *Proceedings of ACM CCS*, 2018.

[12] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In *Proceedings of ACM CCS*, 2017.

[13] B Chor, O Goldreich, E Kushilevitz, and M Sudan. Private information retrieval. In *Proceedings of FOCS*, pages 41–41, 1995.

[14] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. 1997.

[15] Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. In *Proceedings of SCN*, pages 464–482, 2018.

[16] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled psi from homomorphic encryption with reduced computation and communication. In *Proceedings of ACM CCS*, 2021.

[17] Craig Costello and Patrick Longa. Four: four-dimensional decompositions on a-curve over the mersenne prime. In *Proceedings of ASIACRYPT*, 2015.

[18] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: silent vole and oblivious transfer from hardness of decoding structured ldpc codes. In *Proceedings of CRYPTO*, pages 502–534, 2021.

[19] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *Proceedings of ACM CCS*, 2013.

[20] Thai Duong, Duong Hieu Phan, and Ni Trieu. Catalic: Delegated psi cardinality with applications to contact tracing. In *Proceedings of ASIACRYPT*, 2020.

[21] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive 2012/144*, 2012.

[22] Michael J Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *Proceedings of TCC*, volume 3378, pages 303–324, 2005.

[23] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In *Proceedings of CRYPTO*, 2021.

[24] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of STOC*, 2009.

[25] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. 2009.

[26] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game, or a completeness theorem for protocols with an honest majority. In *Proceedings of STOC*, 1987.

[27] Kyoohyung Han, Dukjae Moon, and Yongha Son. Improved circuit-based psi via equality preserving compression. In *Proceedings of SAC*, 2021.

[28] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *Proceedings of NDSS*, 2012.

[29] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. On deploying secure computing: Private intersection-sum-with-cardinality. In *Proceedings of EuroS&P*, 2020.

[30] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In *Proceedings of STOC*, pages 262–271, 2004.

[31] Stanisław Jarecki and Xiaomin Liu. Fast secure computation of set intersection. In *Proceedings of SCN*, 2010.

[32] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In *Proceedings of USENIX Security*, 2019.

[33] Ferhat Karakoç and Alptekin Küpçü. Linear complexity private set intersection for secure two-party protocols. In *Proceedings of CANS*, 2020.

[34] Florian Kerschbaum, Erik-Oliver Blass, and Rasoul Akhavan Mahdavi. Faster secure comparisons with offline phase for efficient private set intersection. In *Proceedings of NDSS*, 2023.

[35] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *Proceedings of ACM CCS*, 2017.

[36] Tancrede Lepoint, Sarvar Patel, Mariana Raykova, Karn Seth, and Ni Trieu. Private join and compute from pir with default. In *Proceedings of ASIACRYPT*, 2021.

[37] Jian Liu, Jingyu Li, Di Wu, and Kui Ren. Pirana: Faster multi-query pir via constant-weight codes. In *Proceedings of IEEE S&P*, 2024.

[38] Samir Jordan Menon and David J Wu. Spiral: Fast, high-rate single-server pir via fhe composition. In *Proceedings of IEEE S&P*, pages 930–947, 2022.

[39] Muhammad Haris Mughees, Hao Chen, and Ling Ren. Onionpir: Response efficient single-server pir. In *Proceedings of ACM CCS*, pages 2292–2306, 2021.

[40] Muhammad Haris Mughees and Ling Ren. Vectorized batch private information retrieval. In *Proceedings of IEEE S&P*, 2023.

[41] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In *Proceedings of Algorithms—ESA*, 2001.

[42] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: lightweight private set intersection from sparse ot extension. In *Proceedings of CRYPTO*, 2019.

[43] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Psi from paxos: fast, malicious private set intersection. In *Proceedings of EUROCRYPT*, 2020.

[44] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *Proceedings of USENIX Security*, 2015.

[45] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based psi with linear communication. In *Proceedings of EUROCRYPT*, 2019.

[46] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based psi via cuckoo hashing. In *Proceedings of EUROCRYPT*, 2018.

[47] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on ot extension. *ACM Transactions on Privacy and Security*, 21(2):1–35, 2018.

[48] Srinivasan Raghuraman and Peter Rindal. Blazing fast psi from improved okvs and subfield vole. In *Proceedings of ACM CCS*, 2022.

[49] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In *Proceedings of ACM CCS*, 2020.

[50] Amanda C Davi Resende and Diego F Aranha. Faster unbalanced private set intersection. In *Proceedings of FC*, 2018.

[51] Peter Rindal and Phillipp Schoppmann. Vole-psi: fast oprf and circuit-psi from vector-ole. In *Proceedings of EUROCRYPT*, 2021.

[52] Phillipp Schoppmann, Adrià Gascón, Mariana Raykova, and Benny Pinkas. Make some room for the zeros: data sparsity in secure distributed machine learning. In *Proceedings of ACM CCS*, 2019.

[53] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[54] Yongha Son and Jinhyuck Jeong. Psi with computation or circuit-psi for unbalanced sets from homomorphic encryption. In *Proceedings of AsiaCCS*, pages 342–356, 2023.

[55] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. Epione: Lightweight contact tracing with strong privacy. *arXiv:2004.13293*, 2020.

[56] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated ot with small communication. In *Proceedings of ACM CCS*, 2020.

[57] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of FOCS*, 1986.

A Unbalanced Circuit-PSI with Payloads

In some scenarios, each input item of the parties has an associated payload. The task is to compute a function of the payloads of the items in the intersection. In this section, we extend our unbalanced circuit-PSI protocol to support computing functions on the intersection of input sets along with associated payloads.

Formally, the client inputs a small set X of size t along with an associated value set $\tilde{X}$, and the server inputs a large key set Y of size n along with an associated value set $\tilde{Y}$. After the protocol, the two parties will compute a function on the intersection and the corresponding payloads. This construction is significantly built on our unbalanced circuit-PSI protocols in Figures 7 except for the following parts. (1) In step 3 of the unbalanced circuit-PSI protocol, the client samples random r_i, w_i, and defines $P_i := \{(y', r_i \| (\tilde{y} - w_i))\}$, where $y' = y \| j \in T_Y[i]$ for some $j \in [\beta]$ and $\tilde{y}$ is the payload of y. (2) In step 4, the two parties invoke $\mathcal{F}_{\mathrm{OKVR}}$ on this new key-value pairs P_i, and the client obtains $r_i^* \| w_i^*$. After $\mathcal{F}_{\mathrm{OKVR}}$, the generic 2PC functionality will then take as input $(r_i^*, w_i^*, x_i, \tilde{x}_i)$ from the client and (r_i, w_i) from the server.