Unbalanced Private Set Intersection from Client-Independent Relaxed Oblivious PRF

Xiaodong Wang
Tsinghua University
Beijing Institute of Mathematical Sciences and Applications
wangxd22@mails.tsinghua.edu.cn

Zijie Lu
Beijing Institute of Mathematical Sciences and Applications
luzijie@bimsa.cn

Bei Liang
Beijing Institute of Mathematical Sciences and Applications
lbei@bimsa.cn

Shengzhe Meng
Tsinghua University
Beijing Institute of Mathematical Sciences and Applications
msz22@mails.tsinghua.edu.cn

Abstract

Private Set Intersection (PSI) enables parties to compute the intersection of their input sets while preserving privacy. While most PSI protocols are designed for balanced scenarios with sets of similar sizes, unbalanced PSI addresses situations where a server with a large database (e.g., millions of records) performs PSI with multiple clients, each with a set of only a few hundred elements. In this scenario, it is desirable for the server's computation on its large set to be performed offline and reusable, which we refer to as the "Client-Independent" property. However, existing offline/online unbalanced PSI protocols rely on less efficient OPRF constructions, which involve either computationally expensive exponential operations or communication-intensive garbled circuits.

In this work, we present a framework for offline/online unbalanced PSI, with its core component being a novel functionality called *Client-Independent Relaxed OPRF* (ci-rOPRF). The key insight behind ci-rOPRF is to capture the requirements for OPRF in offline/online scenarios. To realize this functionality, we propose two constructions of ci-rOPRF, inspired by the top-performing CM-OPRF (CRYPTO '20) and VOLE-OPRF (EUROCRYPT '21), respectively. Leveraging these efficient ci-rOPRF constructions, we design highly efficient offline/online unbalanced PSI protocols. Furthermore, we extend this framework with two enhancements: one supports set updates, while the other reduces offline communication costs.

Our C++ implementation demonstrates highly efficient performance. For instance, in the online phase, our fastest unbalanced PSI protocol computes the intersection of a client set with 2^{12} elements and a server set with 2^{28} elements in just 0.55 seconds and 0.62 MiB of communication on a 100 Mbps WiFi connection. Comparisons with state-of-the-art unbalanced PSI protocols show that our protocols significantly outperform existing solutions in the semi-honest model on most metrics.

Proceedings on Privacy Enhancing Technologies 2025(3), 475–493

https://doi.org/10.56553/popets-2025-0109

Keywords

unbalanced PSI, OPRF, secure multiparty computation

1 Introduction

Private Set Intersection (PSI) enables two parties, the sender and the receiver, with input sets X and Y, respectively, to compute the intersection $X \cap Y$ without revealing any information about items that are not part of the intersection. PSI has numerous applications, including private contact discovery [23, 26], privacy-preserving location sharing [35], and botnet detection [33].

The first and simplest PSI protocols, which elegantly follow from the Diffie-Hellman (DH) key agreement scheme, were introduced by Huberman et al. [24] and can be traced back to the 1980s [32]. The primary advantage of DH-based protocols lies in their low communication cost. However, these protocols require a number of exponentiations proportional to the size of the input set, making them less practical due to their high computational overhead. Another category of PSI protocols is based on oblivious transfer (OT) [18, 37], where PSI computation is reduced to multiple instances of OT. This approach has been proven to be highly efficient in practice, due to the OT extension (OTe) protocols [25, 30] that use only a small constant number of public-key operations. As is well known, OT-based protocols incur higher communication costs compared to DH-based protocols but are significantly faster. The connection between oblivious pseudorandom functions (OPRFs) and PSI was first pointed out in [20]. Building on this insight, Kolesnikov et al. [31] demonstrated that the aforementioned OT-based protocols could be reframed in terms of OPRF. Expanding on this perspective, Pinkas et al. [36] proposed a PSI protocol based on a multi-point OPRF, achieving a better balance between computation and communication. Chase and Miao [12] later introduced a new multi-point OPRF protocol for PSI based on lightweight operations, which serves as one of the foundations of our work.

Recently, a correlation called Vector Oblivious Linear Evaluation (VOLE), which can be viewed as a generalization of OT, has been extensively studied. Significant progress has been made in developing VOLE generators with sublinear communication. Boyle et al. [8] introduced the first such protocols based on the LPN assumption, with subsequent work improving efficiency and offering practical

implementations [9, 44, 48]. Due to the highly sublinear cost of generating VOLE, this correlation has been quickly adopted to enhance the efficiency of PSI. VOLE-based PSI protocols, combined with oblivious key-value store (OKVS) [21], were proposed in [43] and further refined in [40]. These protocols can be seen as VOLE-based variants of the OT-based PSI protocol in [21]. Similarly, many recent VOLE-based protocols [10] can be viewed as extensions of existing OT-based protocols [31].

However, the PSI protocols aforementioned are designed for scenarios where both sets are of similar size and the parties involved have comparable computational and storage capabilities. In real-world applications, it is common for one party's set to be significantly larger than the other's, as seen in typical *Client-Server* settings. In such cases, the *Client* might be a mobile device with limited computing power and storage, while the *Server* is a high-performance machine. Additionally, bandwidth between the *Server* and *Client* may also be constrained. Below, we illustrate application scenarios where the *Client*'s input set is much smaller than the *Server*'s.

Mobile Private Contact Discovery. In [23, 26], the server represents a messaging service hosting a large database of user information. When a mobile user installs the app, it performs contact discovery, enabling the user to connect with existing users whose phone numbers are in their address book. The user's contact list is typically much smaller than the server's database. Moreover, the messaging service must handle dynamic updates, such as new users registering or old users unregistering.

Discovery of Leaked Passwords. A database storing compromised passwords may offer a service for users to check if their passwords are included in the breached data. Here, the user's password set is far smaller than the server's database, which is regularly updated with new leaks. In [46], a protocol is proposed that preserves user privacy while hiding the server's list of compromised credentials. This ensures that users can identify compromised passwords without revealing their own passwords, enhancing security.

In these examples, when the *Server* interacts with multiple *Clients* or processes repeated requests from a single *Client*, it is desirable for the *Server*'s computation on its large set to be reusable. Motivated by this, Kiss et al. [28] proposed PSI protocols with separate offline and online phases. In the offline phase, the *Server* precomputes OPRF values for its large set, independent of the *Client*'s input, and sends them to the *Client*. During the online phase, the *Client* computes OPRF values for its small set (the cost is linear in small set size), and identifies the intersection.

Unfortunately, state-of-the-art PSI protocols, such as VOLE-based protocols [6, 40] and OT-based protocols [12, 31], cannot be applied to this setting. This is because the OPRF keys in all protocols depend on the *Client*'s input set, so that it is infeasible for *Server* to precomputes OPRF on its set offline, which implies that they are not kind of "Client-Independent" OPRFs. Existing offline/online PSI protocols [26, 28] rely on less efficient OPRF constructions, such as the computation-heavy NR-OPRF [34] or communication-intensive Garbled-Circuit-based OPRF [38]. This raises the question:

Can we leverage state-of-the-art OPRF protocols to construct "Client-Independent" OPRF protocols and apply them to PSI in the offline/online setting, thereby resulting in more efficient unbalanced PSI protocols?

1.1 Our Contributions

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

- As the keystone of our methodology, we introduce a new functionality called *"Client-Independent Relaxed OPRF"* (ci-rOPRF), which captures the requirements for OPRF in offline/online scenarios. We analyze its relationship with existing OPRF notions.
- We provide two constructions of ci-rOPRF, abbreviated as CI-CM and CI-VOLE Relaxed OPRF, inspired by CM-OPRF [12] and VOLE-OPRF [43] respectively. Both constructions are secure against semi-honest adversaries.
- Based on the *Client-Independent Relaxed OPRF*, we construct the offline/online unbalanced PSI protocol and provide two extended versions, one of which is to support set updates while the other is to reduce the offline communication overhead.
- We implement our protocol in C++ and conduct detailed experiments with sets of varying sizes to evaluate its performance. The experiments show that our protocol achieves a highly efficient online phase, requiring only 0.55 seconds and 0.62 MiB of communication for our fastest protocol when computing the intersection between a *Client* with a set size of 2^{12} and a *Server* with a large dataset of 2^{28} entries, measured on a WiFi connection with 100 Mbps bandwidth. Moreover, we perform a comprehensive comparison with state-of-the-art unbalanced PSI protocols. The experimental results demonstrate that our protocol significantly outperforms existing protocols in the semi-honest model.

1.2 Related Work

In this section, we introduce some related works on unbalanced PSI. The work related to PSI in balanced scenarios was briefly introduced above. Existing unbalanced PSI protocols can be mainly categorized into two types: those based on the OPRF paradigm and those constructed using Fully Homomorphic Encryption (FHE). Generally speaking, in the offline/online setting, OPRF-based PSI protocols have lower computational costs compared to FHE-based protocols, and the computation and communication costs in the online phase are almost unaffected by the size of the *Server*'s large set. Our work also falls within the OPRF paradigm. On the other hand, FHE-based protocols offer advantages such as lower communication and *Client* storage costs, but their major disadvantage is that the computation and communication costs in the online phase scale linearly with the size of the *Server*'s set. As a result, FHE-based protocols are better suited for scenarios with smaller *Server* sets or when the *Client*'s storage capacity is limited.

OPRF based protocols. Kiss et al. [28] introduced the idea of performing an offline precomputation phase, which is only related to the *Server*'s large input set, in unbalanced PSI scenarios to enable an efficient online intersection computation. They transform four existing OPRF-based PSI protocols, secure in the semi-honest model, into a precomputation form, where the cost in the offline phase is linear only in the size of the larger input set, and the cost in the online phase is linear in the size of the smaller input set. This offline/online paradigm serves as the foundation for subsequent work.

Resende et al. [41] use techniques similar to [28], but replace Bloom filters [7] with the more efficient and versatile Cuckoo filters

[19] to efficiently store the OPRF values. However, in their implementation, they set very aggressive Cuckoo filter parameters for a false positive probability (FPP) of $\varepsilon \approx 2^{-13}$. In contrast, in our work, we use the Cuckoo filter parameters as in [26], with a FPP of $\varepsilon \approx 2^{-29}$.

For the purpose of mobile private contact discovery, Kales et al. [26] made further optimizations to the NR-PSI [22] and GC-PSI [38] protocols presented in [28]. Their improvements include the use of more efficient OT preprocessing techniques, better Cuckoo filter parameters, and a specialized cipher (LowMC) [1] for the GC-PSI protocol. The optimized protocols are referred to as LowMC-GC and ECC-NR, respectively.

Most recently, Sun et al. [45] constructed a maliciously secure unbalanced PSI protocol using the Dodis-Yampolskiy PRF [17]. Their key technique involves encoding the *Server*'s set with a oblivious verifiable pseudorandom function (OVRF) and sending it to the *Client*. The *Client* can then execute the PSI protocol with the *Server*, with complexity linear in the size of the *Client*'s set. The advantage of this approach, compared to directly using OPRF-based protocols, lies in achieving malicious security through verifiability. However, the trade-off is a higher computational and communication cost. Specifically, when the *Client*'s set size is 2^{10}, the communication cost of the online phase reaches 63.23 MB, while our protocols requires less than 1 MB.

Directly using OPRF-based protocols requires sending the *Server*'s OPRF values to the *Client*. When the *Client*'s storage capacity is limited, a variant that combines private information retrieval (PIR) can be considered. The combination of two-server PIR and PSI for private contact discovery was first proposed in [16] with PIR-PSI. Their protocol achieves sublinear communication complexity in the *Server*'s set size but requires online computation that is linear in the *Server*'s set size for each query. By leveraging PIR preprocessing, Hetz et al. [23] present a new mobile private contact discovery protocol based on unbalanced PSI [26] and offline/online PIR [29], which achieves both sublinear communication cost and online runtime in the *Server*'s set size. The advantage of these PIR-based protocols [16, 23] is that the *Client* does not need to store the *Server*'s OPRF values. However, this comes at the cost of increased online runtime.

FHE based protocols. Chen et al. [14] present a PSI protocol with low communication complexity based on Fully Homomorphic Encryption, where the majority of the tasks are handled by the *Server*, while the *Clients* only perform encryption and decryption. This work was later refined in [13], improving both performance and security, and extending the protocol to the labeled PSI setting with arbitrary length labels. Later, Cong et al. [15] introduced several optimizations to the protocols in [13, 14], resulting in improved running time and communication complexity in terms of the *Server*'s set size. Most recently, Wu and Yuen [47] propose new techniques called "VBF" and "POL" to resolve the long item issue in FHE-based PSI protocols. Compared to [15], their unbalanced PSI protocol can save 42.04% to 58.85% in communication costs and accelerate the receiver's query time.

As mentioned above, the advantage of the protocols presented in [13–15, 47] is that their communication complexity is sublinear, rather than linear, in the size of the *Server*'s set. However, these protocols require longer computation times when the *Server*'s set is large.

2 Preliminaries

2.1 Notation

Throughout the paper we use the following notation: We denote the parties as *Server* and *Client*, and their respective input sets as X and Y with $|X| = N_s$ and $|Y| = N_c$. The intersection $X \cap Y$ is denoted as I. We use κ, λ to denote the computational and statistical security parameters, respectively. We use $[m]$ to denote the set $\{1, 2, \ldots, m\}$. For a vector v of length ℓ, we use $v[i]$ to denote the i-th element of the vector. For a matrix M of dimension $m \times w$, we use M_i to denote its i-th column vector ($i \in [w]$). For some set S, the notation $s \xleftarrow{\$} S$ means that s is assigned a uniformly random element from S. By $\mathrm{negl}(\kappa)$ we denote a negligible function, i.e., a function f such that $f(\kappa) < 1/p(\kappa)$ holds for any polynomial $p(\cdot)$ and sufficiently large κ.

2.2 Oblivious Transfer

Oblivious Transfer (OT) is a central cryptographic primitive in the area of secure computation, which was introduced by Rabin [39]. 1-out-of-2 OT refers to the setting where a sender has two input strings (m_0, m_1) and a receiver has an input choice bit $b \in \{0, 1\}$. As the result of the OT protocol, the receiver learns m_b without learning anything about m_{1-b} while the sender learns nothing about b. This primitive requires expensive public-key operations. However, with OT extension (OTe) protocol [25], a small number (e.g., 128) of "base OTs" can be extended to a large number of OTs using only efficient symmetric cryptographic operations.

There exist other flavors of OTe with reduced communication complexity [3]. In random OT (R-OT), neither party inputs any values, but the inputs of sender and receiver are randomly chosen by the protocol. It is feasible to precompute oblivious transfers in such a way that all computationally intensive tasks are executed in advance using R-OTs [4]. Subsequently, the random values obtained through R-OTs are used to mask the actual inputs, requiring only inexpensive XOR operations in the style of one-time-pad encryption.

2.3 Vector Oblivious Linear Evaluation

The random subfield Vector Oblivious Linear Evaluation (VOLE) functionality $\mathcal{F}_{\mathrm{r-sVOLE}}$ is presented in Fig. 1. Let $\mathbb{F}$ be an extension field over base field $\mathbb{B}$. The *Client* obtains random vectors $\overrightarrow{A} \xleftarrow{\$} \mathbb{F}^m$ and $\overrightarrow{B} \xleftarrow{\$} \mathbb{B}^m$. The *Server* obtains a random element $\Delta \xleftarrow{\$} \mathbb{F}$ and the vector $\overrightarrow{C} = \overrightarrow{A} + \Delta \overrightarrow{B}$.

A naive implementation of a VOLE generator would involve running a two-party multiplication protocol with communication linear in m. Recently, significant advances have been made in developing VOLE generators with sub-linear communication. Boyle et al. [8] presented the first protocols in that direction based on the LPN assumption. The subsequent work further improved efficiency and provided effective implementations [9, 44, 48].

In this paper, we will use a variant of subfield VOLE, where the *Server* can choose the input Δ, and the *Client* can choose the inputs $\overrightarrow{A}, \overrightarrow{B}$, and finally the *Server* obtains $\overrightarrow{C} = \overrightarrow{A} + \Delta \overrightarrow{B}$. The chosen-input variant VOLE protocol (depicted in Fig. 2) can be constructed from the random VOLE protocol [8].

Parameters: There are two parties, a *Server* and a *Client*. Let $\mathbb{F}$ be an extension field over base field $\mathbb{B}$. Let m denote the size of the output vectors.
Functionality: Upon input (Server,sid) from the *Server* and (Client,sid) from the *Client*.
Sample $\overrightarrow{A} \xleftarrow{\$} \mathbb{F}^m$, $\overrightarrow{B} \xleftarrow{\$} \mathbb{B}^m$, $\Delta \xleftarrow{\$} \mathbb{F}$ and compute $\overrightarrow{C} := \overrightarrow{A} + \Delta\overrightarrow{B}$. The functionality sends $\Delta, \overrightarrow{C}$ to the *Server* and $\overrightarrow{A}, \overrightarrow{B}$ to the *Client*.

Figure 1: Ideal functionality for random subfield VOLE $\mathcal{F}_{\text{r-sVOLE}}$.

Inputs: The *Server*'s input $\Delta \in \mathbb{F}$ and the *Client*'s input $\overrightarrow{A} \in \mathbb{F}^m, \overrightarrow{B} \in \mathbb{B}^m$.
Outputs: The *Client* has no output; the *Server*'s output is given by $\overrightarrow{C} := \overrightarrow{A} + \Delta\overrightarrow{B}$.
Protocol Steps: Upon input (Server,sid) from the *Server* and (Client,sid) from the *Client*, the protocol specifies the following:
(1) The *Server* and *Client* invoke the functionality $\mathcal{F}_{\text{r-sVOLE}}$. The *Server* obtains $\Delta' \in \mathbb{F}$ and $\overrightarrow{C}' = \overrightarrow{A}' + \Delta\overrightarrow{B}'$. The *Client* obtains $\overrightarrow{A}' \in \mathbb{F}^m$ and $\overrightarrow{B}' \in \mathbb{B}^m$.
(2) The *Server* sends $m_\Delta := \Delta - \Delta'$ to the *Client*.
(3) The *Client* sends $m_B := \overrightarrow{B} - \overrightarrow{B}'$ and $m_A := \overrightarrow{A} - \overrightarrow{A}' + m_\Delta \overrightarrow{B}$ to the *Server*.
(4) The *Server* outputs $\overrightarrow{C} := \overrightarrow{C}' + m_B\Delta' + m_A$.

Figure 2: Random VOLE to VOLE.

The correctness follows from

$$\begin{aligned}\overrightarrow{C} &= \overrightarrow{C}' + m_B\Delta' + m_A \\ &= (\overrightarrow{A}' + \Delta'\overrightarrow{B}') + (\overrightarrow{B} - \overrightarrow{B}')\Delta' + (\overrightarrow{A} - \overrightarrow{A}' + (\Delta - \Delta')\overrightarrow{B}) \\ &= \overrightarrow{A} + \Delta\overrightarrow{B}\end{aligned}$$

The security proof can be found in [8] (Proposition 10).

2.4 Oblivious Key-Value Store

The concept of an oblivious key-value store (OKVS) was introduced by Garimella et al. [21]. At a high level, an OKVS enables the encoding of n pairs of key-value pairs in such a way that an adversary cannot reverse-engineer the original input keys from the encoding, assuming the input values are random.

Definition 2.1. A **oblivious key-value store (OKVS)** is parameterized by a set $\mathcal{K}$ of keys, a set $\mathcal{V}$ of values, and consists of two algorithms:

- Encode: Takes as input a set of (k_i, v_i) key-value pairs and outputs an object T (or, with statistically negligible probability, an error indicator $\perp$).
- Decode: Takes as input an object T, a key k, and outputs a value v.

An OKVS is correct if, for all $A \subseteq \mathcal{K} \times \mathcal{V}$ with distinct keys:

$$(k, v) \in A \text{ and } \perp \neq T \leftarrow \mathsf{Encode}(A) \implies \mathsf{Decode}(T, k) = v$$

An OKVS is computationally oblivious, if for all distinct $\{k_1^0, \ldots, k_n^0\}$ and distinct $\{k_1^1, \ldots, k_n^1\}$, if Encode does not output $\perp$ for $(k_1^0, \ldots, k_n^0)$ or $(k_1^1, \ldots, k_n^1)$, then the output of $\mathsf{Exp}^{\mathcal{A}}(\mathcal{K} = (k_1^0, \ldots, k_n^0))$ is computationally indistinguishable to that of $\mathsf{Exp}^{\mathcal{A}}(\mathcal{K} = (k_1^1, \ldots, k_n^1))$, where:

$\mathsf{Exp}^{\mathcal{A}}(\mathcal{K} = (k_1, \ldots, k_n))$:
(1) for $i \in [n]$: choose uniform $v_i \leftarrow \mathcal{V}$
(2) return $\mathcal{A}(\mathsf{Encode}(\{(k_1, v_1), \ldots (k_n, v_n)\}))$

Additional property. In our construction, we need an OKVS with homomorphic properties. Specifically, we need $\mathsf{Decode}(\cdot, k)$ to be a linear function for all k.

Definition 2.2. An OKVS is **linear** (over a field $\mathbb{F}$) if $\mathcal{V} = \mathbb{F}$ ("values" are elements of $\mathbb{F}$), the output of Encode is a vector T in $\mathbb{F}^m$, and the Decode function is defined as:

$$\mathsf{Decode}(T, k) = \langle \mathrm{d}(k), T \rangle \stackrel{\text{def}}{=} \sum_{j=1}^{m} \mathrm{d}(k)_j T_j$$

for some function $\mathrm{d} : \mathcal{K} \to \mathbb{F}^m$. Hence Decode $(\cdot, k)$ is a linear map from $\mathbb{F}^m$ to $\mathbb{F}$.

2.5 Cuckoo Filter

A Cuckoo filter (CF) [19] is a space-efficient probabilistic data structure designed for fast membership testing, like a Bloom filter (BF) [7] does. It has a controllable false positive probability (FPP), but false negatives do not occur. The Cuckoo filter was used for constructing unbalanced PSI protocols in [23, 26, 41].

The use of Cuckoo filters has several advantages:

- The Cuckoo filter supports adding and removing items dynamically, but Bloom filters do not.
- It provides higher lookup performance than traditional Bloom filters.
- It uses less space than Bloom filters in many practical applications.

A Cuckoo filter uses a hash table based on Cuckoo hashing to store the tags (fingerprints) of items, where each tag is located in a bucket and each bucket contains up to b tags.

To insert the tag of an item x, we need to find the two potential buckets $p_1(x)$ and $p_2(x)$ using the technique called "partial-key Cuckoo hashing", it can derive an item's alternate location only based on its tag and current position. More specifically, these positions of the buckets are calculated using the formula:

$$\begin{aligned}p_1(x) &= H(x) \\ p_2(x) &= p_1(x) \oplus H(t_x)\end{aligned}$$

Note that we can determine the other candidate bucket p_j just from knowing its tag t_x and the current position p_i using the relation: $p_j(x) = p_i(x) \oplus H(t_x)$. The tag of x is placed into one of the two buckets, if both buckets are full, then randomly select one of the two buckets, evict one of the b tags from that bucket using Cuckoo hashing and place it into the other bucket where it can go.

To check whether an item is contained in the Cuckoo filter, one calculates its tag and checks both potential bucket locations, comparing the stored tags for equality. To delete the item, the corresponding tag is removed from the filter.

3 Client-independent Relaxed OPRF

3.1 Ideal Functionality

Oblivious Pseudorandom Function (OPRF) is a central primitive for constructing protocols such as private set intersection (PSI), oblivious keyword search (KS), and password-protected secret sharing (PPSS), and many variants have emerged based on different application requirements. However, many references that utilize the concept of OPRF do not explicitly provide a formal definition, leading to ambiguity. Therefore, before providing our definition of Client-Independent Relaxed OPRF, we will first clarify the meaning of "Relaxed". Then we analyze the demand for OPRF in the unbalanced PSI scenario and introduce the concept of Client-Independence. Finally, we formalize the novel functionality of "Client-Independent Relaxed OPRF" and explain its relationship with existing concepts related to OPRF.

3.1.1 Strong OPRF and Relaxed OPRF. The concept of oblivious pseudorandom function (OPRF) was formally introduced by Freedman et al. in [20]. In the strongest definition of OPRF, an OPRF protocol for the PRF $f_k(\cdot)$ is a secure computation of the functionality $(k, Y) \rightarrow (\perp, \{f_k(y)|y \in Y\})$. In other words, an OPRF protocol allows the *Client* to receive the output of a PRF $f_k(\cdot)$ on the input set Y, using a key k held by the *Server*, while keeping the input Y hidden from the *Server*. At the same time, the *Client* learns nothing about the PRF key k beyond what is revealed by $\{f_k(y)|y \in Y\}$.

However, some natural and efficient OPRF protocols do not satisfy the strong definition. Here, we illustrate with the example from [20]. Consider the following pseudo-random function based on the Naor-Reingold construction: The key r consists of two sets $x_1, \ldots, x_m$ and $y_1, \ldots, y_m$; the function is defined for inputs (i, j) such that $1 \leq i, j \leq m$, and its value is $f_r(i, j) = g^{x_i y_j}$ in a group where the DDH assumption holds and g is a generator. Consider the OPRF protocol where a client whose input is (i, j) learns x_i and y_j and uses them to compute $f_r(i, j)$. Although these values reveal part of the key r to the *Client*, the other outputs of the function remain pseudorandom.

In most application scenarios, we only need the evaluation of the PRF $f_k(\cdot)$ on all other inputs to remain pseudorandom in the view of the *Client* (rather than k). Based on the above observations, Freedman et al. (Definition 6 in [20]) introduced the concept of Relaxed OPRF. Roughly speaking, the *Client* obtains no additional information about the outputs of a PRF $f_k(\cdot)$ beyond what follows from a legitimate set of queries. (Recall that the strong definition requires that no information be learned about the PRF key k.)

Nevertheless, in many works that construct PSI protocols from OPRF, the definition of the underlying OPRF are not explicitly specified. In fact, the OPRF schemes [12, 31, 36, 43] particularly developed for building PSI merely satisfy the definition of Relaxed OPRF. In the contrary, OPRFs based on algebraic constructions [20, 24] can satisfy the definition of Strong OPRF. (For more details on existing OPRF constructions and variants, please refer to [11] for a systematic summary.)

3.1.2 Client-Independent OPRF. To introduce the requirement of Client-Independence for OPRF, let's first recall the framework for constructing PSI from OPRFs, which mainly consists of following steps (here we omit the step of hashing). The *Client* with inputs set Y and the *Server* with no input invoke the functionality of OPRF and as a result *Client* learns $f_k(y)$ for $y \in Y$ while *Server* learns a random PRF seed k. The *Server* can then compute and send the OPRF values $\{f_k(x)|x \in X\}$ which allows the *Client* to identify the common items.

Table 1: Comparison of different OPRF schemes designed for PSI.

Reference	Client-Independent	Strong/Relaxed
KKRT16 [31]	×	Relaxed
KLSAP17 [28]	✓	Strong
PRTY19 [36]	×	Relaxed
KRSSW19 [26]	✓	Strong
CM20 [12]	×	Relaxed
RS21 [43]	×	Relaxed

Parameters: There is a *Server* $\mathcal{S}$ with input set X and *Clients* $C_1, C_2, \ldots$ with input sets $Y_1, Y_2, \ldots$, respectively. Let $\mathbb{F}$ be a field. Let $out \in \mathbb{Z}$ be the output bit length.
Functionality: Upon input (Server,sid, X) from the *Server* $\mathcal{S}$, the functionality samples $F : \mathbb{F} \rightarrow \{0,1\}^{out}$ and sends $\Psi_X := \{F(x) \mid x \in X\}$ to $\mathcal{S}$.
Subsequently, upon input (Server,sid) and (Client,sid, Y_i) from the *Client* C_i, the functionality sends $\Psi_{Y_i} := \{F(y) \mid y \in Y_i\}$ to the C_i.

Figure 3: Ideal functionality for Client-Independent Relaxed OPRF $\mathcal{F}_{\text{ci-rOPRF}}$.

In the case of unbalanced PSI where the parties have unequal set sizes: Often the *Server* that has a database of millions of records performs PSI with multiple *Clients*, each with a set of a few hundred elements. In this setting, to avoid the computational overhead that scales linearly with the size of the *Server*'s set, it is desirable for the OPRF values obtained by the *Server* to be reusable with different *Clients*. Therefore, this requires that the PRF key k is independent of the *Client*'s input sets. After selecting the PRF key in a Client-Independent manner, the *Server* only needs to compute the OPRF values once on its large set X during the offline phase, which can be securely reused by multiple clients to identify the intersection.

We notice that the Relaxed OPRF schemes in [12, 31, 36, 43] are not Client-Independent. In contrast, Strong OPRFs based on algebraic constructions [20, 24] are Client-Independent.

In Tab. 1, We survey most widely used OPRF schemes constructed for the PSI protocol, and summaries the properties they achieve, including the "Client-Independent" property and security level, i.e., satisfying "Relaxed" or "Strong" security conditions.

We observe that the existing relaxed OPRFs employed in constructing top-performing PSI schemes [12, 36, 40, 43] do not satisfy the "Client-Independent" property, making it impossible to compute the *Server*'s OPRF values offline. In this paper, we aim to propose a Client-Independent Relaxed OPRF.

3.1.3 Client-Independent Relaxed OPRF. We formalize the functionality of Client-Independent Relaxed OPRF $\mathcal{F}_{\text{ci-rOPRF}}$ in Fig. 3. It can

be seen as a variant of the OPRF functionality of Rindal and Schoppmann [43]. The main difference with their functionality is that in our ideal functionality, the *Server* can independently generate its OPRF values before the *Client* participates in the protocol. Specifically, at the start of the protocol, the *Server* only needs to input (Server,sid, X), after which the *Server* will obtain the OPRF values for set X. Subsequently, when the *Client* C_i inputs (Client,sid, Y_i) and the *Server* inputs (Server,sid) (this process is independent of the *Server*'s set X, but requires the *Server*'s random tape), the *Client* C_i obtains the OPRF values for his set Y_i.

Currently, the "Client-Independent" OPRF schemes designed for unbalanced PSI scenarios [26, 28] rely on techniques that involve computationally expensive exponential operations or the construction of garbled circuits, leading to significant communication costs. If we consider that the *Server* and *Client* only execute the PSI protocol only once, the protocol in [26, 28] will have much poorer performance compared to the top-performing PSI schemes.

To this end, in the next two subsections, we utilize the structures of the best-performing OPRF protocols, e.g. CM-OPRF [12] and VOLE-OPRF [43] respectively, to construct efficient "Client-Independent Relaxed OPRF" protocols.

3.2 CI-CM Relaxed OPRF

Our first Client-Independent Relaxed OPRF construction is an adaption of the the OT-based Relaxed OPRF of Chase and Miao [12]. We describe our CI-CM Relaxed OPRF protocol in Fig.4. In their original protocol (see Appendix B), the choice of the key used in the OPRF protocol is determined by the *Client*'s input set and *Server*'s random choice in the OT correlation, which does not meet the condition for "Client-Independent".

To address the issue, at a high level, we use a random binary matrix $R \in \{0,1\}^{m\times w}$ selected by the *Server* as the OPRF key. For each element x in its input set X, the *Server* computes the OPRF value as $\mathsf{H}(R_1[v[1]]\| \ldots \|R_w[v[w]])$, where v is the "position vector" computed as $v = F_k(x)$[1]. The *Server* can perform this process locally with no need to interact with the *Client*.

To enable the *Client* to compute the OPRF values for the input set Y, the *Server* masks the matrix R as $P = R \oplus C$ and sends P to the *Client*. The construction of matrix C (in steps 4, 5, and 6) is the same as in CM-OPRF. Specifically, for *Server*'s random string $s \xleftarrow{\$} \{0,1\}^w$, define a matrix S (with dimension $m \times w$), the i-th column $S_i = 1^m$ if $s[i] = 1$, otherwise $S_i = 0^m$. We have the relationship: $C = A \oplus (D \wedge S)$ through oblivious transfer. Therefore, it holds that $P = R \oplus A \oplus (D \wedge S)$.

The *Client* can eliminate the matrix A in P (in step 8) to compute

$$\begin{aligned} Q &= A \oplus P = A \oplus R \oplus C \\ &= A \oplus R \oplus (A \oplus (D \wedge S)) = R \oplus (D \wedge S) \end{aligned}$$

and calculate its OPRF values as $H(Q_1[v[1]]\| \ldots \|Q_w[v[w]])$ for $y \in Y$ and $v = F_k(y)$.

The correctness follows from the fact that for $y \in Y$ and $v = F_k(y)$, it holds that $D_i[v[i]] = 0$, thus $\mathsf{H}(Q_1[v[1]]\| \ldots \|Q_w[v[w]]) = \mathsf{H}(R_1[v[1]]\| \ldots \|R_w[v[w]])$.

[1] F is a pseudorandom function $F : \{0,1\}^\kappa \times \{0,1\}^* \to [m]^w$.

Roughly speaking, the obliviousness follows from the fact that the *Client* does not know the bit string s chosen by the *Server* in OT.

Formally, we state the following theorem, with the proof provided in Appendix A.2.

Theorem 1. *Assuming that the Clients do not collude. if F is a PRF, and H is modeled as a random oracle, the oblivious transfer protocol is secure in the semi-honest model, then the protocol in Fig. 4 securely realizes $\mathcal{F}_{\text{ci-rOPRF}}$ in the semi-honest model when parameters m, w are chosen such that for $x \in X$, there are at least κ 1's in $D_1[v[1]], \ldots, D_w[v[w]]$ where $v = F(x)$ and F is a random function, with all but negligible probability.*

3.3 CI-VOLE Relaxed OPRF

We propose another Client-Independent Relaxed OPRF construction based on VOLE. Our approach is inspired by the VOLE-OPRF [43], which uses VOLE to generate correlations (instead of OT) and coordinates with OKVS to embed the *Client*'s set information. In [43], Rindal and Schoppmann first constructed an OPRF using VOLE and PaXoS (a specialized OKVS, which was further abstracted as an OKVS data structure in [21] by Garimella et al.) and applied it to PSI. Subsequent works have built on such a paradigm, improving components such as VOLE and OKVS [6, 40]. In [40], Raghuraman and Rindal constructed an improved OKVS and employed subfield VOLE to reduce communication, which is also used in our implementation. In the aforementioned VOLE-OPRF construction, the OPRF key depends on the *Client*'s input, which means it does not satisfy the "client-independent" property. We describe our CI-VOLE Relaxed OPRF protocol in Fig.5.

In the CI-VOLE protocol, we use a random vector $\overrightarrow{R} \in \mathbb{F}^m$ as the OPRF key, rather than a binary random matrix in the CI-CM protocol. The *Server* chooses a random element $\Delta \in \mathbb{F}$ and computes the OPRF value for $x \in X$ as $\mathsf{H}^o\left(\mathsf{Decode}(\overrightarrow{R}, x) + \Delta \mathsf{H}^{\mathbb{B}}(x)\right)$. This procedure doesn't require any interaction with the *Client*.

In the CI-CM protocol, the matrix A is a random matrix and the matrix B is generated as $B := A \oplus D$. In our CI-VOLE protocol, the matrix B is the encoded vector of OKVS. Specifically, B is an OKVS generated by encoding key-value pairs $(y, \mathsf{H}(A[y]))$[2], for $y \in Y$. The decode algorithm can be seen as $\mathsf{Decode}(B, y') = \mathsf{H}(B[y'])$. More precisely, in the CI-VOLE protocol, we choose $\overrightarrow{A}$ as a random vector, while $\overrightarrow{B}$ is taken as the OKVS encoded by key-value pairs $(y, \mathsf{H}^{\mathbb{B}}(y))$ for $y \in Y$. (We use the subfield $\mathbb{B}$ here to further reduce the communication cost.) Through the subfield VOLE, where the *Server* inputs Δ and the *Client* inputs $\overrightarrow{A}$ and $\overrightarrow{B}$, the *Server* will obtain $\overrightarrow{C} = \overrightarrow{A} + \Delta\overrightarrow{B}$. Using $\overrightarrow{C}$ as a mask, the *Server* sends $\overrightarrow{P} = \overrightarrow{R} + \overrightarrow{C}$ to the *Client*. It holds that $\overrightarrow{P} = \overrightarrow{R} + \overrightarrow{C} = \overrightarrow{R} + \overrightarrow{A} + \Delta\overrightarrow{B}$, thus the *Client* can eliminate $\overrightarrow{A}$ to obtain $\overrightarrow{Q} = \overrightarrow{P} - \overrightarrow{A} = \overrightarrow{R} + \Delta\overrightarrow{B}$. Then the *Client* can compute the OPRF value for $y \in Y$ as $\mathsf{H}^o\left(\mathsf{Decode}(\overrightarrow{Q}, y)\right)$.

The correctness follows from that for $y \in Y$, $\mathsf{H}^o\left(\mathsf{Decode}(\overrightarrow{Q}, y)\right) = \mathsf{H}^o\left(\mathsf{Decode}(\overrightarrow{R} + \Delta\overrightarrow{B}, y)\right) = \mathsf{H}^o\left(\mathsf{Decode}(\overrightarrow{R}, y) + \Delta\mathsf{H}^{\mathbb{B}}(y)\right)$. (This utilizes the linear properties of OKVS.)

[2] For convenience, we abbreviate $\mathsf{H}(A_1[v[1]]\| \ldots \|A_w[v[w]])$ as $\mathsf{H}(A[y])$, where $v = F_k(y)$.

Inputs: *Server*'s set $X = \{x_i\}_{i\in[N_s]}$ and *Client*'s set $Y = \{y_j\}_{j\in[N_c]}$.
Parameters: Computational and statistical security parameter κ, λ, the set sizes N_s and N_c, the protocol parameters m, w, out, random oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{out}$, pseudorandom function $F : \{0,1\}^\kappa \times \{0,1\}^* \to [m]^w$.
Protocol Steps: Upon input (Server,sid, X) from the *Server*, the protocol specifies the following:

(1) The *Server* samples a binary random matrix $R \xleftarrow{\$} \{0,1\}^{m\times w}$ and a random PRF key $k \xleftarrow{\$} \{0,1\}^\kappa$.
(2) The *Server* outputs $\Psi_X := \{\mathsf{H}(R_1[v[1]] \| \ldots \| R_w[v[w]]) \mid v = F_k(x), x \in X\}$.

Subsequently, upon input (Server,sid) and (Client,sid, Y) from the *Client*, the protocol specifies the following:

(3) The *Server* sends the PRF key k to *Client*.
(4) The *Client* initialize an $m \times w$ binary matrix D to all 1's. Denote its column vectors by $D_1, \ldots, D_w$. Then, for each $y \in Y$, computes $v = F_k(y)$ and sets $D_i[v[i]] = 0$ for all $i \in [w]$.
(5) The *Client* randomly samples an $m \times w$ binary matrix $A \xleftarrow{\$} \{0,1\}^{m\times w}$ and defines the matrix $B := A \oplus D$.
(6) The *Server* samples a random string $s \xleftarrow{\$} \{0,1\}^w$ and run w oblivious transfers where the *Client* is the sender with inputs $\{A_i, B_i\}_{i\in[w]}$ and the *Server* is the receiver with inputs $s[1], \ldots, s[w]$. As a result, the *Server* obtains w number of m-bit strings as the column vectors of matrix C.
(7) The *Server* defines matrix $P := R \oplus C$ and sends it to the *Client*.
(8) The *Client* defines matrix $Q := A \oplus P$. Then the *Client* outputs $\Psi_Y := \{\mathsf{H}(Q_1[v[1]] \| \ldots \| Q_w[v[w]]) \mid v = F_k(y), y \in Y\}$.

Figure 4: The CI-CM Relaxed OPRF protocol.

Inputs: *Server*'s set $X = \{x_i\}_{i\in[N_s]} \subseteq \mathbb{F}$ and *Client*'s set $Y = \{y_j\}_{j\in[N_c]} \subseteq \mathbb{F}$.
Parameters: Computational and statistical security parameter κ, λ. The set sizes N_s and N_c. Let $\mathbb{B}$ be a field with extension $\mathbb{F}$ such that $|\mathbb{F}| = O(2^\kappa)$. Let $\mathsf{H}^{\mathbb{B}} : \{0,1\}^* \to \mathbb{B}$, $\mathsf{H}^o : \{0,1\}^* \to \{0,1\}^{out}$ be random oracles. The linear OKVS scheme denoted as (Encode,Decode).
Protocol Steps: Upon input (Server,sid, X) from the *Server*, the protocol specifies the following:

(1) The *Server* samples a random vector $\overrightarrow{R} \xleftarrow{\$} \mathbb{F}^m$ and an random element $\Delta \xleftarrow{\$} \mathbb{F}$. ($m$ is determined by N_c and the OKVS scheme)
(2) The *Server* outputs $\Psi_X := \left\{\mathsf{H}^o\left(\mathsf{Decode}(\overrightarrow{R}, x) + \Delta\mathsf{H}^{\mathbb{B}}(x)\right) \mid x \in X\right\}$.

Subsequently, upon input (Server,sid) and (Client,sid, Y) from the *Client*, the protocol specifies the following:

(3) The *Client* randomly samples an vector $\overrightarrow{A} \xleftarrow{\$} \mathbb{F}^m$ and computes the vector $\overrightarrow{B} := \mathsf{Encode}(Y, \mathsf{H}^{\mathbb{B}}(Y))$.
(4) The *Server* and the *Client* run the subfield VOLE protocol, where the *Client* has inputs $\overrightarrow{A} \in \mathbb{F}^m$ and $\overrightarrow{B} \in \mathbb{B}^m$, and the *Server* has input Δ. As a result, the *Server* obtains a vector $\overrightarrow{C} := \overrightarrow{A} + \Delta\overrightarrow{B}$.
(5) The *Server* defines vector $\overrightarrow{P} := \overrightarrow{R} + \overrightarrow{C}$ and sends it to the *Client*.
(6) The *Client* defines vector $\overrightarrow{Q} := \overrightarrow{P} - \overrightarrow{A}$. Then the *Client* outputs $\Psi_Y := \left\{\mathsf{H}^o\left(\mathsf{Decode}(\overrightarrow{Q}, y)\right) \mid y \in Y\right\}$.

Figure 5: The CI-VOLE Relaxed OPRF protocol.

Similar to the previous subsection, obliviousness arises from the randomness of Δ, which is unknown to the *Client*.

Formally, we state the following theorem, with the proof provided in Appendix A.3.

Theorem 2. *Assuming that the Clients do not collude. If H is modeled as a random oracle, the OKVS scheme is linear, the parameters* $|\mathbb{B}| \geq 2^{\lambda+\log_2 N_s+\log_2 N_c}$ *and* $|\mathbb{F}| \geq 2^\kappa$*, the subfield VOLE protocol is secure in the semi-honest model, then the protocol in Fig. 5 securely realizes* $\mathcal{F}_{\text{ci-rOPRF}}$ *in the semi-honest model.*

Remarks.

- Since our CI-CM protocol enables the *Server* to choose s during interacting with the *Client*, the *Server* can select different s values for multiple *Clients* to enhance security. In contrast, the CI-VOLE protocol requires Δ to be determined during the precomputation phase. Further security discussions on these two protocols can be found in Appendix E.
- In our CI-VOLE protocol, we use subfield VOLE to reduce communication, the method is derived from [40].

4 Unbalanced PSI in the Offline/Online Setting

In this section, we use the $\mathcal{F}_{\text{ci-rOPRF}}$ functionality from the previous section to construct the unbalanced PSI protocol in the offline/online setting. We also propose two extended versions, one of which is to support set updates while the other is to reduce the offline communication overhead.

Before presenting the protocol, we first analyze the challenges faced when performing PSI in an unbalanced scenario. The ideal functionality for PSI is given in Fig. 6. When performing the PSI protocol, if the *Server*'s dataset is significantly larger than that of

Parameters: *Server*'s input set size $|X| = N_s$ and *Client*'s input set size $|Y| = N_c$.
Functionality: Upon input (Server,sid, X) from the *Server* and (Client,sid, Y) from the *Client*, the functionality outputs $I = X \cap Y$ to the *Client*.

Figure 6: Ideal functionality for Private Set Intersection $\mathcal{F}_{\text{PSI}}$.

the *Client*, we have to consider the limited computational power and storage capacity of the *Client* compared to the *Server*. It is desirable to design PSI protocols that share following common properties:

- The *Server* performs the computationally expensive tasks;
- All computationally expensive and communication intensive tasks are performed only once;
- The actual intersection computation is very fast and also allows for efficient updates.

The traditional approach to constructing unbalanced PSI [26, 28, 47] which can achieve above requirements is to divide the protocol into two phases: the *offline phase* and the *online phase*. The *offline phase* is a pre-computation stage conducted before interacting with the *Client* and is client-independent. In this phase, the computational and communication costs are dependent on the *Server*'s set size. Since the *Server*'s set is much larger than that of the *Client*'s, the cost of this stage is typically quite high. In the *online phase*, after the *Client* provides its input set, the pre-computed results from the offline phase enable the efficient calculation of the intersection, which is then output to the *Client*. The computational and communication costs in this phase mainly depend on the *Client*'s set size. [3]

In practical applications such as private contact discovery, the offline stage corresponds to the process of downloading the application software, which can be completed during idle time. The online stage corresponds to using the software to calculate the private set intersection, which requires highly efficient performance.

4.1 Unbalanced PSI Protocol from ci-rOPRF

By leveraging $\mathcal{F}_{\text{ci-rOPRF}}$ and the Cuckoo filter, we present our unbalanced PSI protocol in Fig. 7. In the offline phase, the *Server* obtains the OPRF values Ψ_X for its set X from $\mathcal{F}_{\text{ci-rOPRF}}$[4], inserts them into a Cuckoo filter CF, and sends the filter CF to the *Client*. Then, in the online phase, the *Client* inputs its set Y and obtains the OPRF values Ψ_Y for set Y by running $\mathcal{F}_{\text{ci-rOPRF}}$ with the *Server*. The *Client* then uses the Cuckoo filter to look up and obtain the output $I := \{y \mid F(y)$ is contained in $CF, y \in Y\}$. The correctness and privacy of the protocol are straightforward.
Remarks.

- The advantage of using a Cuckoo filter is that, compared to directly transmitting Ψ_X to the *Client*, it allows the *Server* to incur a slightly higher computational cost (computing the Cuckoo filter for Ψ_X), thereby reducing the cost of a single query for the *Client* from $O(N_s)$ to $O(1)$. Additionally, since the false positive probability (FPP) of the Cuckoo filter can be adjusted based on different application scenarios, in some less sensitive contexts, a larger FPP can be chosen in benefit for a Cuckoo filter that occupies less space.
- The main purpose of dividing unbalanced PSI into an offline and an online phase is that, when handling multiple queries from the *Clients*, this offline stage only needs to be performed once, so only the efficient online phase is executed for each query. However, it should be noted that when facing multiple queries, an increased number of queries will lead to a reduction in the security parameter. Therefore, after a certain number of queries, the offline phase needs to be re-executed.

Inputs: *Server*'s set $X = \{x_i\}_{i\in[N_s]} \subseteq \mathbb{F}$ and *Client*'s set $Y = \{y_j\}_{j\in[N_c]} \subseteq \mathbb{F}$.
Parameters: Let $\mathcal{F}_{\text{ci-rOPRF}}$ be the Client-Independent Relaxed OPRF functionality with $out = \lambda + \log_2(N_s) + \log_2(N_c)$. Let CF be the Cuckoo filter with false positive probability ε.
Protocol:

I. **Offline phase.**
   (a) The *Server* sends (Server,sid, X) to $\mathcal{F}_{\text{ci-rOPRF}}$ and receives back $\Psi_X := \{F(x) \mid x \in X\}$.
   (b) The *Server* inserts Ψ_X into the Cuckoo filter CF. Upon receiving (Client,sid) from the *Client*, the *Server* sends CF to the *Client*.

II. **Online phase.**
   (a) The *Server* sends (Server,sid) and the *Client* sends (Client,sid, Y) to $\mathcal{F}_{\text{ci-rOPRF}}$. The *Client* receives back $\Psi_Y := \{F(y) \mid y \in Y\}$.
   (b) For each $y \in Y$, the *Client* checks if the Cuckoo filter CF contains $F(y)$, and then outputs $I := \{y \mid F(y)$ is contained in $CF, y \in Y\}$.

Figure 7: The unbalanced PSI protocol in offline/online setting from ci-rOPRF.

4.2 Extensions

In practical applications of unbalanced PSI, we need to consider the requirements of different scenarios and extend the basic protocol accordingly. In this subsection, we propose two extensions: one introduces an update phase to handle scenarios with frequent set updates, while the other leverages Private Information Retrieval (PIR) to reduce offline communication costs for scenarios involving very large *Server* sets.

4.2.1 Unbalanced PSI with Update Phase. In many applications of unbalanced PSI, the sets of the *Server* and *Client* may undergo frequent updates, which introduces new challenges for protocol design. For example, in private contact discovery, the messaging service should allow new users to register and old users to unregister. Since the *Client*'s set is relatively small, rerunning the online phase after updates to the *Client*'s set can quickly yield new results. However, when the *Server*'s set is updated, an efficient solution requires that the computational and communication costs during the

[3] In some protocols, a sublinear complexity term may depend on the *Server*'s set size in the online phase.
[4] The OPRF output length is set to $out = \lambda + \log_2(N_s) + \log_2(N_c)$. Using the union bound, it is straightforward to show that the probability of spurious collisions between Ψ_X and Ψ_Y is bounded by $2^{-\lambda}$.

update phase be linear with respect to the size of the updates to the *Server*'s set, without needing to rerun the offline phase. To achieve this, we follow the approach of Kiss et al. [28], further dividing the unbalanced PSI protocol into the following four phases.

Base phase includes the data-independent precomputation and must be performed in order to setup the underlying primitives within the protocol.

Setup phase only needs *Server*'s input data. In our protocol, it involves calculating *Server*'s OPRF values and inserting them into a Cuckoo filter for compact representation, which is sent to the *Client*.

Online phase is the stage where the intersection is computed. In our protocol, it involves OT and calculating *Client*'s OPRF values. The *Client* then looks up its OPRF values in the Cuckoo filter to determine the intersection.

Update phase is used to modify the results obtained during the previous *setup phase*. In our protocol, it involves updating the Cuckoo filter.

The distinction between these four phases and the protocol in Fig. 7 is that the offline phase is further divided into a base phase and a setup phase, while the online phase remains unchanged, and an update phase is added. Leveraging the functionality of $\mathcal{F}_{\text{ci-rOPRF}}$ and the dynamic add-and-remove capabilities of the Cuckoo filter, our protocol can easily incorporate an update phase. In this update phase, since the *Server*'s OPRF key is "Client-Independent", adding elements is straightforward: the *Server* simply computes the OPRF value for each new element and sends the corresponding tag, position in the Cuckoo filter, and an "Insert" command to the *Client*, who can then update the Cuckoo filter for subsequent operations. To remove elements, the *Server* provides the tag and position of the element's OPRF value in the Cuckoo filter along with a "Delete" command, allowing the *Client* to adjust the filter accordingly.

In Appendix D.1, we provide the functionality of the unbalanced PSI with an update phase. Then, in Appendix D.2, we present detailed protocol diagrams for unbalanced PSI with an update phase, implemented based on CI-CM-rOPRF and CI-VOLE-rOPRF, respectively.

4.2.2 Reducing Offline Communication. In some application scenarios, the *Server*'s set size can reach tens of millions or even hundreds of millions, as seen in the user base of social media platforms like WhatsApp. In such cases, directly applying the protocol in Fig. 7 would generate and send a Cuckoo filter during the offline phase that requires substantial storage. For instance, with a *Server* set size of $N_s = 2^{28}$ and a FPP of $\varepsilon \approx 2^{-29}$, the size of the Cuckoo filter would be 1072 MiB, which imposes significant transmission and storage demands on the *Client*.

In these scenarios, it becomes essential to balance computational and communication costs. The significant communication cost arises because the *Server* must send the Cuckoo filter containing the OPRF values for its set X, denoted as Ψ_X, to the *Client*. Only then can the *Client* check which elements of Ψ_Y are present in Ψ_X to determine the intersection. A straightforward approach would be to employ PIR for querying, rather than directly transmitting the Cuckoo filter.

In [23], Hetz et al. proposed a solution for the private contact discovery scenario, leveraging offline-online PIR (OO-PIR) [29] combined with the unbalanced PSI approach from Kales et al. [26]. This scheme achieves a total communication overhead that is sublinear relative to the *Server*'s set size. This approach is compatible with any Client-Independent OPRF, allowing our protocol to similarly leverage OO-PIR to reduce communication costs during the offline phase. However, this benefit comes with trade-offs: it incurs higher computational costs and requires the assumption of two non-colluding servers. Below, we briefly introduce the concept of OO-PIR and explain how it can be applied to our protocol.

Protocols for PIR allow a *Client* to privately retrieve a record from a public database without revealing the requested item to the *Server*. It is well-known that the *Server*'s computational cost must inherently scale linearly with the size of the database [5]. Therefore, to achieve sublinear online complexity relative to the database size, we need to shift these linear-complexity computations to the offline phase. In the OO-PIR protocol in [29], two *Servers*, S_{off} and S_{on}, each store a copy of the database DB. During the offline phase, S_{off} randomly selects $\lambda\sqrt{|DB|}$ sets, each containing $\sqrt{|DB|}$ database indices, computes the parity of each subset, and sends these sets (compressed using a puncturable PRF to save space) along with the parities as "hints" to the *Client*. This process ensures with overwhelming probability that every database index is included in at least one set, and it requires one offline phase per unique *Client*. In the online phase, for each queried index id_y, the *Client* locates the hint containing id_y and uses a "puncturing" process to remove id_y from this subset, and then send the punctured set to S_{on}. S_{on} returns the parity of this received set. By XORing the parities, the *Client* retrieves $DB[id_y]$. Please note that many details were omitted in the above protocol description. For a comprehensive explanation, refer to [23, 29].

In our scheme, the database is the *Server*'s Cuckoo filter CF containing the OPRF values Ψ_X, and the *Client*'s queries correspond to the positions in the Cuckoo filter associated with elements in Ψ_Y. By embedding OO-PIR into the unbalanced PSI protocol as demonstrated in [23], we can reduce the offline phase communication to $O(\sqrt{N_s})$, at the cost of an additional $O(\sqrt{N_s})$ online phase computation. Specifically, with a *Server* set size of $N_s = 2^{28}$ and a FPP of $\varepsilon \approx 2^{-29}$, the offline phase communication required is only 66 MiB—representing a 24× reduction compared to the original approach. This enhancement is particularly valuable in scenarios with limited network bandwidth.

5 Implementation and Performance Comparison

5.1 Concrete Parameter Choices

We use a computational security parameter of $\kappa = 128$ and a statistical security parameter of $\lambda = 40$.

Choice of m, w in CI-CM PSI. The parameters m, w in our CM-OPRF based protocol are chosen such that for each $x \in X$, there are at least κ 1's in $D_1[v[1]], \ldots, D_w[v[w]]$ where $v = F(x)$ and F is a random function, with all but negligible probability. The same analytical approach as in [12] is put to use here. First, we fix m, and then determine w as follows. Consider each column D_i, initialized as 1^m. Then, for each $y \in Y$, the *Client* computes $v = F(y)$ and

sets $D_i[v[i]] = 0$. After this, since F is a random function, the probability $\Pr[D_i[j] = 1] = \left(1 - \frac{1}{m}\right)^{N_c}$ (denoted as p) is the same for all $j \in [m]$. Thus, for any $x \in X$ and $v = F_K(x)$, the number of 1's in $D_1[v[1]], \ldots, D_w[v[w]]$ follows a binomial distribution, and the probability of having fewer than κ 1's is $\sum_{k=0}^{\kappa-1} \binom{w}{k} p^k (1-p)^{w-k}$. By the union bound, it is sufficient for the following probability to be negligible:

$$N_s \cdot \sum_{k=0}^{\kappa-1} \binom{w}{k} p^k (1-p)^{w-k} \leq negl(\lambda)$$

where $p = \left(1 - \frac{1}{m}\right)^{N_c}$. From this we can derive a proper w for some m.

We list different choices of the parameters in Appendix C (Tab. 5). The protocol is flexible in that we can set these parameters differently to trade-off between computation and communication. Intuitively, for fixed set size N_s and N_c, the bigger m will lead to smaller w and requires less computation of the pseudorandom function F_k, thus speeding up the offline phase. However, simultaneously, the size of the matrix $m \times w$ may increase, leading to higher communication costs and a longer online phase time. In our experiment, we will set $m = N_c$ for all settings, since it achieves nearly optimal communication and has the fastest online phase among all choices of m.

Choice of $\mathbb{B}, \mathbb{F}$ in CI-VOLE PSI. As shown in [40], when using subfield VOLE to construct the PSI protocol, the parameter choice that results in the fastest runtime in practice is: $\mathbb{B} = GF(2^\kappa)$, and $\mathbb{F} = \mathbb{B}$. However, to minimize communication costs, $\mathbb{B}$ can be set as the smallest field such that $|\mathbb{B}| \geq 2^{\lambda + \log_2 N_s + \log_2 N_c}$ and $|\mathbb{F}| \geq 2^\kappa$. In our implementation, to achieve faster runtime, we choose $\mathbb{B} = \mathbb{F} = GF(2^\kappa)$.

Choice of Cuckoo Filter. We use the parameter recommendations in [26] with bucket size $b = 3$ and tag size $f = 32$ for a false positive probability of $\varepsilon \approx 2^{-29}$. This false positive probability is sufficient to meet the requirements of most realistic scenarios.

Once the above parameter choices are made and N_s, N_c is determined, the remaining parameters required are explicitly provided in the protocols.

5.2 Implementation Details

We implement our protocols in C++ using libOTe [42] and the Cuckoo filter from [26]. We employ IKNP [25] for OT extension and BLAKE2 as the hash function. For the implementation of the pseudorandom function, to improve concrete efficiency, we apply the technique by Chase and Miao [12] to parallelize the computation over multiple elements as much as possible, thereby making optimal use of the hardware-optimized AES-ECB mode implementation. Specifically, we require $F : \{0,1\}^\kappa \times \{0,1\}^\ell \to [m]^w$. For elements of length $\ell = 128$ (longer elements need to be hashed first), let $G : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$ be a pseudorandom function (instantiated by AES) and $\mathrm{PRG} : \{0,1\}^\kappa \to \{0,1\}^{t \cdot \kappa}$ be a pseudorandom generator where $t = \left\lceil \frac{w \cdot \log m}{\kappa} \right\rceil$. On a key k and input x,

$$F_k(x) = G_{k_1}(x) || G_{k_2}(x) || \ldots || G_{k_t}(x)$$

Table 2: Offline phase computation cost (in seconds) and communication cost (in MiB) for different *Server* set size N_s in the CI-CM PSI and CI-VOLE PSI protocols. Best results are marked in bold.

N_s	Protocol	Offline Comp. [s]	Offline Comm. [MiB]
2^{20}	CI-CM PSI	2.43	4.19
	CI-VOLE PSI	**0.87**	
2^{24}	CI-CM PSI	40.16	67.01
	CI-VOLE PSI	**13.49**	
2^{28}	CI-CM PSI	641.60	1072.14
	CI-VOLE PSI	**219.55**	

where $k_1 || k_2 || \ldots || k_t \longleftarrow \mathrm{PRG}(k)$.[5] Now PRG (instantiated by AES-CTR mode) is only applied once on the key k, and $G_{k_i}(\cdot)$ are all parallelizable by AES-ECB mode. We use a binary OKVS with a weight of 3 from [40] and silent VOLE from [42]. The implementation is single-threaded, but most computations can be parallelized. Therefore, we expect that the offline time can achieve approximately linear speedup when using multi-threading. [6]

5.3 Benchmark

In this subsection, we present benchmarks for the offline and online phases of the PSI protocol implemented using two different ci-rOPRFs from Sec. 3 (denoted as CI-CM PSI and CI-VOLE PSI). The experiments were run on a desktop computer with AMD 3950X CPU and 32GB RAM. We considered localhost environment and simulated different WAN network settings using the Linux tc command.

In Tab. 2, we present a benchmark of the computational and communication costs for the offline phase. Note that this phase is client-independent, we evaluate it for different *Server* set sizes $N_s \in \{2^{20}, 2^{24}, 2^{28}\}$. The offline computation includes the time required to compute the *Server*'s OPRF values Ψ_X and the time spent constructing the Cuckoo filter CF, while the offline communication refers to the cost of sending CF to the *Client*. From the experimental results, we observe that in all settings, the CI-VOLE PSI protocol achieves approximately a 3× speedup in computation time compared to the CI-CM PSI protocol. Since the Cuckoo filter size is the same across different protocols for the same N_s, the offline communication cost remains identical.

In Tab. 3, we present a benchmark of the computational and communication costs for the online phase. We fix $N_s = 2^{28}$ and conduct experiments for both protocols with $N_c \in \{2^8, 2^{12}, 2^{16}\}$ under different network bandwidths, with Round-trip time (RTT) set to 200 ms, reflecting latency close to that of intercontinental communication in the real world. The computation time includes the time to calculate the *Client*'s OPRF values Ψ_Y, as well as the time for looking up values in the Cuckoo filter. The cost of the online phase primarily depends on N_c, with the size of N_s having

[5] The construction we use is a simplified version of the one presented in [12]. The detailed security proof can be found in Appendix A.4.

[6] Our implementation is available at https://github.com/lzjluzijie/upsi.

Table 3: Online phase communication cost (in MiB) and running time (in seconds) for varying *Client* set sizes N_c and network bandwidths in the CI-CM PSI and CI-VOLE PSI protocols. In all experiments, $N_s = 2^{28}$ and RTT is set to 200 ms. Best results are marked in bold.

N_c	Protocol	Comm. [MiB]	Time [s]			
			localhost	1000Mbps	100Mbps	10Mbps
2^8	CI-CM PSI	**0.05**	**0.01**	**0.61**	**0.61**	**0.65**
	CI-VOLE PSI	0.54	0.09	1.47	1.50	1.61
2^{12}	CI-CM PSI	**0.62**	**0.03**	**1.25**	**1.27**	**1.65**
	CI-VOLE PSI	0.75	0.24	1.49	1.53	1.80
2^{16}	CI-CM PSI	9.72	1.16	3.38	3.79	11.37
	CI-VOLE PSI	**3.14**	**0.61**	**2.01**	**2.17**	**3.97**

little impact (only influencing the size of matrix B in the CI-CM protocol or the size of field $\mathbb{B}$ in the CI-VOLE protocol logarithmically). From the experimental results, we observe that when N_c is small, the CI-CM PSI protocol has an evident advantage in both communication and runtime over the CI-VOLE PSI protocol. However, as N_c becomes larger, the CI-VOLE PSI protocol is more advantageous. When $N_c = 2^{12}$, the performance of both protocols is not significantly different. This phenomenon occurs because the cost of generating VOLE is highly sublinear, and its amortized overhead quickly diminishes as N_c increases.

5.4 Comparison to Related Work

In this subsection, we compare the performance of our protocol with several state-of-the-art unbalanced PSI protocols, including LowMC-GC and ECC-NR based on OPRF from [26], and VBF+slicing and PoL based on FHE from [47]. Since the code for the protocol in [47] is not open-source, we performed our comparison on input sizes $N_s \in \{2^{24}, 2^{20}\}$ and $N_c \in \{11041, 5535\}$ to match the parameters used in [47]. We used the same RTT as in [47], and all experiments were conducted using a single thread. While the experiments in [47] were run on an Intel(R) Xeon(R) Gold 6226R CPU with 236GB of RAM, we ran our protocol on our own hardware with an AMD 3950X CPU and 32GB of RAM. Our CPU performance is slightly better than that of [47], but their RAM is significantly larger than ours. Regardless, as observed from the subsequent experiments, the performance gap between the protocols is much larger than the differences in CPU and RAM between the hardware used in the experiments. The detailed results of the experiments are provided in Tab. 4.

Comparisons with OPRF based protocols [26]. Our protocols (CI-CM and CI-VOLE), as well as LowMC-GC and ECC-NR from [26], are all based on the OPRF paradigm for PSI. Therefore, under the same security parameters, the length of the OPRF values is identical, resulting in the same offline phase communication cost. In all other aspects, our protocol significantly outperforms both LowMC-GC and ECC-NR. This can be attributed to the fact that the LowMC-GC protocol requires building a garbled circuit during the offline phase, which leads to a high communication cost. In contrast, the ECC-NR protocol requires multiple exponentiations in both the offline and online phases, resulting in a high computational cost. Compared to these protocols, our approach is based on top-performing PSI protocols, where most operations are lightweight, and the communication cost is lower.

However, it is important to note that in terms of security, our protocol only supports security in the semi-honest setting for both parties, while LowMC-GC and ECC-NR can easily achieve security against a malicious *Client* [7] (but not against a malicious *Server*) by replacing the OT extension protocol with a maliciously secure OT extension protocol. Since OT extension contributes only a small percentage to the total runtime of the PSI protocols, and today's maliciously secure OT extension [2, 27] protocols are only slightly less efficient than their semi-honest counterparts, replacing the OT extension protocol does not result in a noticeable increase in the total runtime of the PSI protocols.

Regarding the different levels of security mentioned above, we would like to argue the following:

- Many realistic scenarios do correspond to semi-honest attack behavior. One such example is computing with players who are trusted to act honestly, but cannot fully guarantee that their storage might not be compromised in the future. Therefore, in many application scenarios, semi-honest security is sufficient to meet the requirements.
- Even in the face of more powerful adversaries, by applying hash functions to the elements before running the protocol, the information leaked to the adversary reveals no more than what would be exposed through a naïve hashing approach for PSI.
- Semi-honest protocols often serve as a basis for protocols in more robust settings with more powerful attackers.

Comparisons with FHE based protocols [47]. The protocols in [47] (VBF+slicing and PoL) are unbalanced PSI protocols based on fully homomorphic encryption, with the main advantage of not requiring any offline communication. However, this comes at the cost of the *Server* needing to perform computations that are linear in the size of the large database for each client during the online phase. Additionally, due to the high computational cost of FHE, the offline phase of the protocol takes significantly longer. As observed in Tab. 4, as the *Server* set size N_s increases, both the communication and computation costs in the online phase of the protocol from [47] increase accordingly, while the costs in the online phase of OPRF-based protocols are minimally affected by N_s. Therefore, FHE-based protocols are more suitable for scenarios where the client's storage capability is limited, and the size of N_s is not too large. In terms of security, [47] claims to provide one-sided simulatability against a malicious adversary, which is a relaxed level of security where one corruption case is simulatable, and for the other party, only privacy is guaranteed. However, no formal security proof is provided.

Overall, in the semi-honest setting, the performance of the online phase of our protocol significantly outperforms all known existing unbalanced PSI protocols, including those not explicitly compared in this section, such as [16, 28, 41]. Therefore, in scenarios where both parties have a high level of trust or efficiency is a key concern, our protocol and the extensions presented in Sec. 4 (including the update phase or the reduction of offline communication) are

[7] A malicious adversary may arbitrarily deviate from the prescribed protocol in an attempt to break the security.

Table 4: Comparison results between our unbalanced PSI protocols and other protocols in various settings. The best results are highlighted in bold. All experiments were conducted in a single-threaded environment, with network latency set to 0.05 ms for consistency with [47].

Cardinality		Protocol	Online	Online Time [s]				Offline	Offline
N_s	N_c		Comm. [MiB]	10 Gbps	100 Mbps	10 Mbps	1 Mbps	Comp. [s]	Comm. [MiB]
2^{24}	11041	LowMC-GC [26]	253.75	12.43	53.46	413.13	4248.98	85.49	67.01
		ECC-NR [26]	65.24	4.85	12.09	114.12	1154.91	2048.49	67.01
		VBF+slicing [47]	11.76	21.35	21.42	22.01	34.58	660.13	**0**
		PoL [47]	6.15	16.43	16.42	17.22	33.65	5143.96	**0**
		Ours (CI-CM PSI)	1.68	**0.07**	**0.19**	1.49	14.19	41.41	67.01
		Ours (CI-VOLE PSI)	**1.03**	0.29	0.37	**1.11**	**9.10**	**13.90**	67.01
	5535	LowMC-GC [26]	127.22	5.13	25.82	215.32	2133.24	87.85	67.01
		ECC-NR [26]	32.71	2.64	8.99	86.23	871.78	2418.46	67.01
		VBF+slicing [47]	6.86	16.73	16.83	17.17	25.75	683.48	**0**
		PoL [47]	4.29	14.51	14.55	15.00	22.16	4918.41	**0**
		Ours (CI-CM PSI)	0.85	**0.03**	**0.11**	**0.73**	7.20	41.41	67.01
		Ours (CI-VOLE PSI)	**0.81**	0.25	0.30	0.87	**7.19**	**13.90**	67.01
2^{20}	11041	LowMC-GC [26]	253.75	11.61	52.79	430.21	4359.08	5.48	4.19
		ECC-NR [26]	65.24	4.52	12.09	114.12	1151.91	151.05	4.19
		VBF+slicing [47]	8.84	5.01	5.09	5.64	25.18	26.56	**0**
		PoL [47]	4.50	3.85	3.99	4.66	24.46	88.04	**0**
		Ours (CI-CM PSI)	1.65	**0.06**	**0.20**	1.46	13.94	2.53	4.19
		Ours (CI-VOLE PSI)	**1.03**	0.29	0.37	**1.10**	**9.10**	**0.81**	4.19
	5535	LowMC-GC [26]	127.22	4.89	25.56	215.23	2136.56	5.35	4.19
		ECC-NR [26]	32.71	2.35	8.99	86.23	871.78	150.67	4.19
		VBF+slicing [47]	5.30	3.63	3.68	3.94	18.68	26.27	**0**
		PoL [47]	3.13	3.15	3.21	3.63	18.30	93.09	**0**
		Ours (CI-CM PSI)	0.83	**0.03**	**0.10**	**0.73**	**7.05**	2.53	4.19
		Ours (CI-VOLE PSI)	**0.81**	0.24	0.30	0.87	7.17	**0.81**	4.19

the better choice. On the other hand, in the malicious setting, if the *Server*'s set is large and the *Client* has sufficient storage, the protocol in [26] (LowMC-GC and ECC-NR) should be considered. If the *Client*'s storage is limited, the PIR-based version in [23] that reduces offline communication can be a good alternative. When the *Server*'s set is not very large, the FHE based protocols in [47] outperforms the protocols in [26], and it does not require additional storage from the *Client*.

6 Conclusion and Future Work

In this paper, we propose the "Client-Independent Relaxed OPRF" functionality, which captures the requirements for OPRF in unbalanced PSI within the offline/online setting. We then construct the CI-CM and CI-VOLE Relaxed OPRF protocols, which securely realizes $\mathcal{F}_{\text{ci-rOPRF}}$ in the semi-honest model. We apply these protocols to unbalanced PSI and analyze extensions with an update phase and in combination with PIR. Finally, we implement the protocols and conduct a detailed comparison with related work. The experimental results show that our protocols significantly outperform others in the semi-honest model. However, our protocols cannot be directly extended to the malicious model, so improving their security will be part of our future work. Additionally, we note that recent advances in more efficient offline-online PIR protocols, when combined with our unbalanced PSI protocols, can lead to more efficient contact discovery schemes. The selection of suitable PIR protocols and their implementation will also be an important area for future exploration.

Acknowledgments

This work is supported by National Key R&D Program of China No. 2023YFC3305501.

References

[1] Martin R Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. 2015. Ciphers for MPC and FHE. In *EUROCRYPT 2015, Proceedings, Part I*. Springer, 430–454.

[2] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. 2015. More efficient oblivious transfer extensions with security for malicious adversaries. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 673–701.

[3] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. 2017. More efficient oblivious transfer extensions. *Journal of Cryptology* 30 (2017), 805–858.

[4] Donald Beaver. 1995. Precomputing oblivious transfer. In *CRYPTO 1995*. Springer, 97–109.

[5] Amos Beimel, Yuval Ishai, and Tal Malkin. 2000. Reducing the servers computation in private information retrieval: PIR with preprocessing. In *Advances in Cryptology—CRYPTO 2000*. Springer, 55–73.

[6] Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. 2023. {Near-Optimal} Oblivious {Key-Value} Stores for Efficient {PSI},{PSU} and {Volume-Hiding}{Multi-Maps}. In *32nd USENIX Security Symposium (USENIX Security 23)*. 301–318.

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

[8] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. 2018. Compressing vector OLE. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 896–912.

[9] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. 2019. Efficient two-round OT extension and silent non-interactive secure computation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 291–308.

[10] Dung Bui and Geoffroy Couteau. 2023. Improved private set intersection for sets with small entries. In *IACR International Conference on Public-Key Cryptography*. Springer, 190–220.

[11] Silvia Casacuberta, Julia Hesse, and Anja Lehmann. 2022. SoK: oblivious pseudorandom functions. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 625–646.

[12] Melissa Chase and Peihan Miao. 2020. Private set intersection in the internet setting from lightweight oblivious PRF. In *CRYPTO 2020, Proceedings, Part III 40*. Springer, 34–63.

[13] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. 2018. Labeled PSI from fully homomorphic encryption with malicious security. In *Proceedings of the ACM CCS 2018*. 1223–1237.

[14] Hao Chen, Kim Laine, and Peter Rindal. 2017. Fast private set intersection from homomorphic encryption. In *Proceedings of the ACM CCS 2017*. 1243–1255.

[15] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. 2021. Labeled PSI from homomorphic encryption with reduced computation and communication. In *Proceedings of the ACM CCS 2021*. 1135–1150.

[16] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. 2018. PIR-PSI: scaling private contact discovery. *Proceedings on Privacy Enhancing Technologies* (2018).

[17] Yevgeniy Dodis and Aleksandr Yampolskiy. 2005. A verifiable random function with short proofs and keys. In *International Workshop on Public Key Cryptography*. Springer, 416–431.

[18] Changyu Dong, Liqun Chen, and Zikai Wen. 2013. When private set intersection meets big data: an efficient and scalable protocol. In *Proceedings of the ACM CCS 2013*. 789–800.

[19] Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. 2014. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. 75–88.

[20] Michael J Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. 2005. Keyword search and oblivious pseudorandom functions. In *TCC 2005*. Springer, 303–324.

[21] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2021. Oblivious key-value stores and amplification for private set intersection. In *CRYPTO 2021, Proceedings, Part II 41*. Springer, 395–425.

[22] Carmit Hazay and Yehuda Lindell. 2010. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. *Journal of cryptology* 23, 3 (2010), 422–456.

[23] Laura Hetz, Thomas Schneider, and Christian Weinert. 2023. Scaling mobile private contact discovery to billions of users. In *European Symposium on Research in Computer Security*. Springer, 455–476.

[24] Bernardo A Huberman, Matt Franklin, and Tad Hogg. 1999. Enhancing privacy and trust in electronic communities. In *Proceedings of the 1st ACM conference on Electronic commerce*. 78–86.

[25] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending oblivious transfers efficiently. In *CRYPTO 2003*. Springer, 145–161.

[26] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. 2019. Mobile private contact discovery at scale. In *USENIX Security 19*. 1447–1464.

[27] Marcel Keller, Emmanuela Orsini, and Peter Scholl. 2015. Actively secure OT extension with optimal overhead. In *Annual Cryptology Conference*. Springer, 724–741.

[28] Ágnes Kiss, Jian Liu, Thomas Schneider, N Asokan, and Benny Pinkas. 2017. Private set intersection for unequal set sizes with mobile applications. In *Privacy Enhancing Technologies Symposium*. De Gruyter, 177–197.

[29] Dmitry Kogan and Henry Corrigan-Gibbs. 2021. Private blocklist lookups with checklist. In *30th USENIX security symposium (USENIX Security 21)*. 875–892.

[30] Vladimir Kolesnikov and Ranjit Kumaresan. 2013. Improved OT extension for transferring short secrets. In *CRYPTO 2013, Proceedings, Part II*. Springer, 54–70.

[31] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. 2016. Efficient batched oblivious PRF with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 818–829.

[32] Catherine Meadows. 1986. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *1986 IEEE Symposium on Security and Privacy*. IEEE, 134–134.

[33] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. 2010. {BotGrep}: Finding {P2P} Bots with Structured Graph Analysis. In *USENIX Security 10*.

[34] Moni Naor and Omer Reingold. 2004. Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM (JACM)* 51, 2 (2004), 231–262.

[35] Arvind Narayanan, Narendran Thiagarajan, Mugdha Lakhani, Michael Hamburg, Dan Boneh, et al. 2011. Location privacy via private proximity testing.. In *NDSS*, Vol. 11.

[36] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2019. SpOT-light: lightweight private set intersection from sparse OT extension. In *CRYPTO 2019, Proceedings, Part III 39*. Springer, 401–431.

[37] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. 2015. Phasing: Private set intersection using permutation-based hashing. In *24th USENIX Security Symposium (USENIX Security 15)*. 515–530.

[38] Benny Pinkas, Thomas Schneider, Nigel P Smart, and Stephen C Williams. 2009. Secure two-party computation is practical. In *Advances in Cryptology–ASIACRYPT 2009: 15th International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 250–267.

[39] Michael O Rabin. 2005. How to exchange secrets with oblivious transfer. *Cryptology ePrint Archive* (2005).

[40] Srinivasan Raghuraman and Peter Rindal. 2022. Blazing fast PSI from improved OKVS and subfield VOLE. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2505–2517.

[41] Amanda C Davi Resende and Diego F Aranha. 2018. Faster unbalanced private set intersection. In *Financial Cryptography and Data Security, FC 2018, Revised Selected Papers 22*. Springer, 203–221.

[42] Peter Rindal and Lance Roy. 2024. libOTe: An Efficient, Portable, and Easy to Use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe.

[43] Peter Rindal and Phillipp Schoppmann. 2021. VOLE-PSI: fast OPRF and circuit-PSI from vector-OLE. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 901–930.

[44] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. 2019. Distributed vector-OLE: Improved constructions and implementation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1055–1072.

[45] Yunqing Sun, Jonathan Katz, Mariana Raykova, Phillipp Schoppmann, and Xiao Wang. 2024. Actively Secure Private Set Intersection in the Client-Server Setting. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 1478–1492.

[46] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, et al. 2019. Protecting accounts from credential stuffing with password breach alerting. In *USENIX Security 19*. 1556–1571.

[47] Mingli Wu and Tsz Hon Yuen. 2023. Efficient unbalanced private set intersection cardinality and user-friendly privacy-preserving contact tracing. In *32nd USENIX Security Symposium (USENIX Security 23)*. 283–300.

[48] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. 2020. Ferret: Fast extension for correlated OT with small communication. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 1607–1626.

A Security Proofs

A.1 Security Model

A semi-honest adversary is the corrupted party that follows the protocol as specified. In other words, the corrupt party runs the protocol honestly but may try to learn as much as possible from the messages it receives from the other party.

The protocol is secure against a semi-honest adversary means that it guarantees the corrupt party can never learn any information about the other party's input, other than what is revealed through its own output, as long as it follows the protocol.

Definition 1. Let Π be a two-party protocol computing $f = (f_1, f_2)$ and $\text{view}_i^{\Pi}(x, y)$ be the view of P_i (the entire distribution that P_i can see), $\text{out}^{\Pi}(x, y) = (\text{out}_1^{\Pi}(x, y), \text{out}_2^{\Pi}(x, y))$ be the output of the protocol where x and y are inputs of P_1 and P_2, respectively. We say Π has semi-honest security if there exist PPT simulators S_1, S_2, and the following holds for all inputs x, y:

$$\left(\text{view}_1^{\Pi}(x, y), \text{out}^{\Pi}(x, y)\right) \overset{c}{\approx} \left(S_1\left(1^{\lambda}, x, f_1(x, y)\right), f(x, y)\right)$$

$$\left(\text{view}_2^{\Pi}(x, y), \text{out}^{\Pi}(x, y)\right) \overset{c}{\approx} \left(S_2\left(1^{\lambda}, y, f_2(x, y)\right), f(x, y)\right)$$

A.2 The Security Proof for CI-CM Relaxed OPRF

Theorem 1. *Assuming that the Clients do not collude, if F is a PRF, and H is modeled as a random oracle, the oblivious transfer protocol is secure in the semi-honest model, then the protocol in Fig. 4 securely realizes $\mathcal{F}_{\text{ci-rOPRF}}$ in the semi-honest model when parameters m, w are chosen such that for $x \in X$, there are at least κ 1's in $D_1[v[1]], \ldots, D_w[v[w]]$ where $v = F(x)$ and F is a random function, with all but negligible probability.*

Proof. Since the *Clients* do not collude, we only need to consider the interaction between the *Server* and one of the *Clients*.
Security against corrupt *Server*. We construct $\mathcal{S}_1$ as follows. It is given *Server*'s input set X and corresponding output $\{F(x) \mid x \in X\}$. $\mathcal{S}_1$ runs the honest *Server*'s protocol to generate its view with the following exceptions: In Step 2, for the query $q = R_1[v[1]] \| \ldots \| R_w[v[w]]$ where $v = F_k(x), x \in X$, $\mathcal{S}_1$ programs $\mathsf{H}(q)$ to $F(x)$ (the output from functionality). Otherwise H responds normally. For the oblivious transfer in Step 6, The simulator generates *Server*'s random string $s \xleftarrow{\$} \{0,1\}^w$ and chooses a random matrix $C \in \{0,1\}^{m \times w}$, and runs the OT simulator to simulate the view for an OT receiver with inputs $s[1], \ldots, s[w]$ and outputs $C_1, \ldots, C_w$. Finally $\mathcal{S}_1$ outputs *Server*'s view.

Denote the output of *Server* and *Client* in the functionality $\mathcal{F}_{\text{ci-rOPRF}}$ as $f(X, Y) = (f_1(X, Y), f_2(X, Y))$. We prove that

$$\{\mathcal{S}_1(1^\kappa, X, N_c, f_1(X, Y)), f(X, Y)\} \overset{c}{\approx} \left\{\mathsf{view}_1^\Pi(X, Y), \mathsf{out}^\Pi(X, Y)\right\}$$

via the following hybrid argument:

- Hybrid_0: The *Server*'s view and *Client*'s output in the real protocol.
- Hybrid_1: Same as Hybrid_0 except that on *Client*'s side, for each $i \in [w]$, if $s[i] = 0$, then sample $A_i \xleftarrow{\$} \{0,1\}^m$ and compute $B_i = A_i \oplus D_i$; otherwise sample $B_i \xleftarrow{\$} \{0,1\}^m$ and compute $A_i = B_i \oplus D_i$. This hybrid is identical to Hybrid_0.
- Hybrid_2: Same as Hybrid_1 except that $\mathcal{S}_1$ samples the matrix C and runs the OT simulator to simulate the view of an OT receiver for *Server*. This modification is indistinguishable from Hybrid_1 by security of the OT protocol.
- Hybrid_3: The simulated view of $\mathcal{S}_1$ and $f(X, Y)$ (i.e. $\{F(x)|x \in X\}$ and $\{F(y)|x \in Y\}$, for a random function F). The only difference from Hybrid_2 is that the output $\mathsf{out}^\Pi(X, Y)$ is replaced by $f(X, Y)$. Note that as mentioned above, for $y \in Y$, if $v = F_k(y)$, then $\mathsf{H}(Q_1[v[1]] \| \ldots \| Q_w[v[w]]) = \mathsf{H}(R_1[v[1]] \| \ldots \| R_w[v[w]])$. Therefore, for the elements in $X \cup Y$, the output can be considered as computed through the random matrix R in the real execution. Specifically, we have $\Psi_X^\Pi := \{\mathsf{H}(R_1[v[1]] \| \ldots \| R_w[v[w]]) \mid v = F_k(x), x \in X\}$ and $\Psi_Y^\Pi := \{\mathsf{H}(R_1[v[1]] \| \ldots \| R_w[v[w]]) \mid v = F_k(y), y \in Y\}$ [8], the real output $\mathsf{out}^\Pi(X, Y) = (\Psi_X^\Pi, \Psi_Y^\Pi)$. Due to the randomness of R and H as a random oracle, if Ψ_X^Π is programmed to $f_1(X, Y)$, then $\mathsf{out}^\Pi(X, Y)$ is computationally indistinguishable from $f(X, Y)$. Therefore, this hybrid is computationally indistinguishable from Hybrid_2.

Security against corrupt *Client*. We construct $\mathcal{S}_2$ as follows. It is given *Client*'s input set Y and corresponding output $\{F(y) \mid y \in Y\}$. $\mathcal{S}_2$ runs the honest *Client*'s protocol to generate its view with the following exceptions: In Step 3, $\mathcal{S}_2$ sends a uniformly random PRF key k to *Client*. For the oblivious transfer in Step 6, $\mathcal{S}_2$ computes the matrices A and B honestly and run the OT simulator to produce a simulated view for the OT sender. In Step 7, $\mathcal{S}_2$ samples a random string $s \xleftarrow{\$} \{0,1\}^w$ and computes the matrix S as follows: if $s[i] = 1$, then the i-th column $S_i = 1^m$, otherwise $S_i = 0^m$. $\mathcal{S}_2$ samples the random matrix R and computes $P = R \oplus A \oplus (D \wedge S)$ and sends it to the *Client*. In Step 8, for the query $q = Q_1[v[1]] \| \ldots \| Q_w[v[w]]$ where $v = F_k(y), y \in Y$, $\mathcal{S}_2$ programs $\mathsf{H}(q)$ to $F(y)$ (the output from functionality). Finally $\mathcal{S}_2$ outputs *Client*'s view.

We prove that

$$\{\mathcal{S}_2(1^\kappa, Y, N_s, f_2(X, Y)), f(X, Y)\} \overset{c}{\approx} \left\{\mathsf{view}_2^\Pi(X, Y), \mathsf{out}^\Pi(X, Y)\right\}$$

via the following hybrid argument:

- Hybrid_0: The *Client*'s view and *Server*'s output in the real protocol.
- Hybrid_1: Same as Hybrid_0 except that $\mathcal{S}_2$ (instead of *Server*) chooses the random PRF key k. This hybrid is statistically identical to Hybrid_0.
- Hybrid_2: Same as Hybrid_1 except that $\mathcal{S}_2$ runs the OT simulator to produce a simulated view of an OT sender for *Client*. This hybrid is computationally indistinguishable from Hybrid_1 by security of the OT protocol.
- Hybrid_3: Same as Hybrid_2 but the protocol aborts if there exists $x \in X$ such that, for $v = F_k(x)$, there are fewer than κ 1's in $D_1[v[1]], \ldots, D_w[v[w]]$. The parameters m, w are chosen such that if F is a random function, then the aborting probability is negligible. If the aborting probability in Hybrid_3 is non-negligible, then we can construct a probabilistic polynomial-time distinguisher $\mathcal{D}$ to distinguish the PRF from a random function. More specifically, given the sets X and Y, it computes the matrix D as in Step 4. For the input function f, the distinguisher $\mathcal{D}$ computes $v = f(x)$ and $D_1[v[1]], \ldots, D_w[v[w]]$ for each $x \in X \setminus I$. If there exists x such that, there are fewer than κ 1's in $D_1[v[1]], \ldots, D_w[v[w]]$, then output "$f$ is PRF", otherwise output "f is random function". The output is correctly with probability $\frac{1}{2}$ + non-negl, it breaks the security of PRF. Hence, the protocol aborts with negligible probability in Hybrid_3.
- Hybrid_4: Same as Hybrid_3 except that $\mathcal{S}_2$ samples random string s and random matrix R and computes $P = R \oplus A \oplus (D \wedge S)$ in Step 7. Due to the randomness of R, this hybrid is statistically indistinguishable from the previous.
- Hybrid_5: The simulated view of $\mathcal{S}_2$ and $f(X, Y)$. The only difference from Hybrid_4 is that the output $\mathsf{out}^\Pi(X, Y)$ is replaced by $f(X, Y)$. Note that H is a random oracle, and Ψ_Y^Π is programmed to output $f_2(X, Y)$. We only need to show that Ψ_X^Π is computationally indistinguishable from $\{F(x)|x \in X\}$. As mentioned above, it holds that $R = Q \oplus (D \wedge S)$. Since s is randomly sampled and unknown to the *Client*, and, at the same time, for $x \in X$, if $v = F_k(x)$, there will be more than κ 1's in $D_1[v[1]], \ldots, D_w[v[w]]$. For

[8] In the following proof, we denote the output in the real protocol as Ψ_X^Π for the set X and ψ_x^Π for the element x.

$x \notin Y$, ψ_x^Π (the output for x in the real protocol) appears random to the *Client*, and for $x \in Y$, $\psi_x^\Pi \in \Psi_Y^\Pi$. Therefore, Ψ_X^Π is computationally indistinguishable from $\{F(x)|x \in X\}$.

□

A.3 The Security Proof for the CI-VOLE Relaxed OPRF

Theorem 2. *Assuming that the Clients do not collude, if H is modeled as a random oracle, the OKVS scheme is linear, the parameters* $|\mathbb{B}| \geq 2^{\lambda+\log_2 N_s+\log_2 N_c}$ *and* $|\mathbb{F}| \geq 2^\kappa$*, the subfield VOLE protocol is secure in the semi-honest model, then the protocol in Fig. 5 securely realizes* $\mathcal{F}_{\text{ci-rOPRF}}$ *in the semi-honest model.*

Proof. **Security against corrupt *Server*.** We construct $\mathcal{S}_1$ as follows. It is given *Server*'s input set X and corresponding output $\{F(x) \mid x \in X\}$. $\mathcal{S}_1$ runs the honest *Server*'s protocol to generate its view with the following exceptions: In Step 2, for the query $q = \text{Decode}(\overrightarrow{R}, x) + \Delta \mathsf{H}^{\mathbb{B}}(x)$, $\mathcal{S}_1$ programs $\mathsf{H}(q)$ to $F(x)$. Otherwise H responds normally. For the subfield VOLE in Step 4, The simulator generates *Server*'s random element Δ and chooses a random vector $\overrightarrow{C}$, and runs the subfield VOLE simulator to simulate the view with inputs Δ and output $\overrightarrow{C}$. Finally $\mathcal{S}_1$ outputs *Server*'s view. We prove that

$$\left\{\mathcal{S}_1\left(1^\kappa, X, N_c, f_1(X,Y)\right), f(X,Y)\right\} \overset{c}{\approx} \left\{\text{view}_1^\Pi(X,Y), \text{out}^\Pi(X,Y)\right\}$$

via the following hybrid argument:

- Hybrid_0: The *Server*'s view and *Client*'s output in the real protocol.
- Hybrid_1: Same as Hybrid_0 except that $\mathcal{S}_1$ samples the vector $\overrightarrow{C}$ and runs the subfield VOLE simulator to simulate the view. Note that in the real execution, $\overrightarrow{A}$ is a random vector, so $\overrightarrow{C}$ is also a random vector. This modification is indistinguishable from Hybrid_0 due to the security of the subfield VOLE protocol.
- Hybrid_2: The simulated view of $\mathcal{S}_1$ and $f(X,Y)$. The only difference from Hybrid_1 is that the output $\text{out}^\Pi(X,Y)$ is replaced by $f(X,Y)$. Note that H is a random oracle, and Ψ_X^Π is programmed to output $f_1(X,Y)$. We only need to show that Ψ_Y^Π is computationally indistinguishable from $\{F(y)|y \in Y\}$. Note that B is encoded by $\mathsf{H}^{\mathbb{B}}(Y)$ and is unknown to the *Server*. Therefore, for $y \notin X$, $\psi_y^\Pi := \text{Decode}(\overrightarrow{Q}, y) = \text{Decode}(\overrightarrow{R} + \Delta\overrightarrow{B}, y) = \text{Decode}(\overrightarrow{R}, y) + \Delta\mathsf{H}^{\mathbb{B}}(y)$ appears random to the *Server*, and for $y \notin X$, $\psi_y^\Pi \in \Psi_X^\Pi$, so Ψ_Y^Π is computationally indistinguishable from $\{F(y)|x \in Y\}$.

Security against corrupt *Client*. We construct $\mathcal{S}_2$ as follows. It is given *Client*'s input set Y and corresponding output $\{F(y) \mid y \in Y\}$. $\mathcal{S}_2$ runs the honest *Client*'s protocol to generate its view with the following exceptions: For the subfield VOLE in Step 4, $\mathcal{S}_2$ computes the vectors $\overrightarrow{A}$ and $\overrightarrow{B}$ honestly and run the OT simulator to produce a simulated view. In Step 5, $\mathcal{S}_2$ samples a random vector $\overrightarrow{P}$ and sends it to the *Client*. In Step 6, for the query $q = \text{Decode}(\overrightarrow{Q}, y)$, $\mathcal{S}_2$ programs $\mathsf{H}^o(q)$ to $F(y)$. Finally $\mathcal{S}_2$ outputs *Client*'s view.

We prove that

$$\left\{\mathcal{S}_2\left(1^\kappa, Y, N_s, f_2(X,Y)\right), f(X,Y)\right\} \overset{c}{\approx} \left\{\text{view}_2^\Pi(X,Y), \text{out}^\Pi(X,Y)\right\}$$

via the following hybrid argument:

- Hybrid_0: The *Client*'s view and *Server*'s output in the real protocol.
- Hybrid_1: Same as Hybrid_0 except that $\mathcal{S}_2$ runs the subfield VOLE simulator with input $\overrightarrow{A}$ and $\overrightarrow{B}$ to produce a simulated view for the *Client*. This hybrid is computationally indistinguishable from Hybrid_0 by security of the subfield VOLE protocol.
- Hybrid_2: Same as Hybrid_1 but the protocol aborts if there exists $x \in X$, $y \in Y$, and $x \neq y$ such that $\mathsf{H}^{\mathbb{B}}(x) = \mathsf{H}^{\mathbb{B}}(y)$. The probability of this is at most $\frac{N_s \cdot N_c}{|\mathbb{B}|}$, and the parameter $\mathbb{B}$ is chosen such that $|\mathbb{B}| \geq 2^{\lambda+\log_2 N_s+\log_2 N_c}$. Therefore, the probability of aborting is at most $2^{-\lambda}$, which is negligible with respect to the statistical security parameter.
- Hybrid_3: Same as Hybrid_2 but the protocol aborts if $\text{Encode}(Y, \mathsf{H}^{\mathbb{B}}(Y)) = \perp$. The OKVS scheme guarantees that the probability of this aborting is negligible with respect to the statistical security parameter.
- Hybrid_4: Same as Hybrid_3 except that $\mathcal{S}_2$ samples the random vector $\overrightarrow{P}$ in Step 5. Due to the randomness of $\overrightarrow{R}$, this hybrid is statistically indistinguishable from the previous.
- Hybrid_5: The simulated view of $\mathcal{S}_2$ and $f(X,Y)$. The only difference from Hybrid_4 is that the output $\text{out}^\Pi(X,Y)$ is replaced by $f(X,Y)$. Note that H^o is a random oracle, and Ψ_Y^Π is programmed to output $f_2(X,Y)$. We only need to show that Ψ_X^Π is computationally indistinguishable from $\{F(x)|x \in X\}$. Note that in the real protocol, it holds that $\psi_x^\Pi := \text{Decode}(\overrightarrow{R}, x) + \Delta\mathsf{H}^{\mathbb{B}}(x) = \text{Decode}(\overrightarrow{Q}, x) + \Delta(\mathsf{H}^{\mathbb{B}}(x) - \text{Decode}(\overrightarrow{B}, x))$. Since Δ is randomly sampled and unknown to the *Client*, and $\mathsf{H}^{\mathbb{B}}, \mathsf{H}^o$ are random oracles, for $x \notin Y$, ψ_x^Π appears random to the *Client*, and for $x \in Y$, $\psi_x^\Pi \in \Psi_Y^\Pi$. Therefore, Ψ_X^Π is computationally indistinguishable from $\{F(x)|x \in X\}$.

□

A.4 The Security Proof for the PRF F

The pseudorandom function F we used is a simplified version of the construction by Chase and Miao [12]. Since [12] considers the case of a malicious server, the adversary's input is extracted through the use of a random oracle, i.e., $F_k(H(x))$, where H is a random oracle (instantiated by hash function) and F_k is the pseudorandom function. Therefore, they set the input length of the pseudorandom function (i.e., the output length of $H(x)$) to be 256 bits to ensure collision resistance. In contrast, when only considering semi-honest adversaries, there is no need to extract the input, so our scheme directly computes $F_k(x)$, with an input length of 128 bits, which can be used directly as input for AES.

The security proof of our construction is similar to that in [12]. For completeness, we provide the formal security proof below.

Theorem 3. *Let* $G : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$ *be a pseudorandom function and* $\text{PRG} : \{0,1\}^\kappa \to \{0,1\}^{t\cdot\kappa}$ *be a pseudorandom generator. On a key* k *and input* x*, define*

$$F_k(x) := G_{k_1}(x)||G_{k_2}(x)||\ldots||G_{k_t}(x)$$

where $k_1||k_2||\ldots||k_t \longleftarrow \text{PRG}(k)$*. Then* $F_k(x)$ *is also a pseudorandom function.*

Proof. We show that any PPT adversary $\mathcal{A}$ cannot distinguish $F_k(x)$ from a random function via a sequence of hybrids:

- Hybrid_0: The adversary $\mathcal{A}$ has access to $F_k(x)$.
- Hybrid_1: The adversary $\mathcal{A}$ has access to the following function

$$G_{k_1}(x)||G_{k_2}(x)||\dots||G_{k_t}(x)$$

where $k_1, \dots, k_t \xleftarrow{\$} \{0,1\}^\kappa$ are sampled uniformly at random.
If $\mathcal{A}$ can distinguish between Hybrid_0 and Hybrid_1, then we can construct another PPT adversary $\mathcal{B}$ that breaks the security of PRG. In particular, $\mathcal{B}$ first gets $k_1||k_2||\dots||k_t$ from the PRG challenger. On query x from $\mathcal{A}$, $\mathcal{B}$ responds with $G_{k_1}(x)||G_{k_2}(x)||\dots||G_{k_t}(x)$. Finally $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.
If the PRG challenger generates $k_1||k_2||\dots||k_t$ from PRG, then $\mathcal{A}$ is accessing Hybrid_0; otherwise, the challenger generates $k_1||k_2||\dots||k_t$ uniformly at random, then $\mathcal{A}$ is accessing Hybrid_0. Hence, if $\mathcal{A}$ can distinguish between Hybrid_0 and Hybrid_1, then $\mathcal{B}$ can break the PRG security.
- Hybrid_2: The adversary $\mathcal{A}$ has access to the following function

$$G_1(x)||G_2(x)||\dots||G_t(x)$$

where $G_1, \dots, G_t$ are all independent random functions. We argue that Hybrid_2 is computationally indistinguishable from Hybrid_1 via a sequence of hybrids, where $\text{Hybrid}_{2,0} = \text{Hybrid}_1$ and $\text{Hybrid}_{2,t} = \text{Hybrid}_2$:
$\text{Hybrid}_{2,i}$: The adversary $\mathcal{A}$ has access to the following function

$$G_1(x)||\dots||G_i(x)||G_{k_{i+1}}(x)||\dots||G_{k_t}(x)$$

where $G_1, \dots, G_i$ are independent random functions, and k_{i+1}, $\dots$, k_t are sampled uniformly at random.
If $\mathcal{A}$ can distinguish between $\text{Hybrid}_{2,i-1}$ and $\text{Hybrid}_{2,i}$ for any $1 \le i \le t$, then we can construct another PPT adversary $\mathcal{B}$ that breaks the PRF security of G_i. In particular, $\mathcal{B}$ first randomly samples $k_{i+1}, \dots, k_t$, and then starts the experiment with $\mathcal{A}$. On query x from $\mathcal{A}$, $\mathcal{B}$ computes $G_{k_{i+1}}(x)||\dots||G_{k_t}(x)$ and randomly samples the outputs of $G_1(x), \dots, G_{i-1}(x)$. Note that if x already appears as an input to $G_1, \dots, G_{i-1}$ before, $\mathcal{B}$ uses the previous outputs. Then $\mathcal{B}$ queries the PRF challenger on input x for an output s, and sends the following back to $\mathcal{A}$:

$$G_1(x)||\dots||s||G_{k_{i+1}}(x)||\dots||G_{k_t}(x).$$

Finally $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.
If the PRF challenger chooses PRF, then $\mathcal{A}$ is accessing $\text{Hybrid}_{2,i-1}$; otherwise $\mathcal{A}$ is accessing $\text{Hybrid}_{2,i}$. Hence, if $\mathcal{A}$ can distinguish between $\text{Hybrid}_{2,i-1}$ and $\text{Hybrid}_{2,i}$, then $\mathcal{B}$ can distinguish PRF from a random function.
- Hybrid_3: The adversary $\mathcal{A}$ has access to a random function $F(x)$. Let the queries from $\mathcal{A}$ be $x_1, \dots, x_n$, and assume WLOG that they are all distinct queries. We argue that Hybrid_2 is computationally indistinguishable from Hybrid_3 via a sequence of hybrids, where $\text{Hybrid}_{3,0} = \text{Hybrid}_2$ and $\text{Hybrid}_{3,n} = \text{Hybrid}_3$:
$\text{Hybrid}_{3,i}$: For the first i queries $x_1, \dots, x_i$ from $\mathcal{A}$, choose the outputs $r_1, \dots, r_i$ independently at random. For each $j \in [i]$, store the implied table for $G_1, \dots, G_t$, namely store $G_1(x)||G_2(x)||\dots||G_t(x) = r_j$. After the first i queries, compute the output according to this $G_1, \dots, G_t$.
The $\text{Hybrid}_{3,i}$ is identical to $\text{Hybrid}_{3,i-1}$ unless the i-th query from $\mathcal{A}$ collides with a previous query x_j. However, the probability that $\mathcal{A}$ can find such a collision is negligible.

□

B CM-OPRF

Chase and Miao [12] present an efficient relaxed OPRF protocol that is based on lightweight operations, in which the PRF key is an $m \times w$ binary matrix C where m and w is depend on the *Client*'s input size and security parameters.

At the beginning of the protocol, the *Client* generates two $m \times w$ binary matrices A and B, where A is a random matrix while $B = A \oplus D$ is generated according to matrix A and the *Client*'s input set Y. Specifically, for each element $y \in Y$, let $v = F_k(y)$ in which $F : \{0,1\}^\kappa \times \{0,1\}^* \to [m]^w$ is a pseudorandom function and k is known to both parties, the matrix D is constructed such that $D_i[v[i]] = 0$ for all $i \in [w]$ and the remaining positions are 1. After that, the *Server* can obtain the matrix C (which is the key of this OPRF) where C_i is either equal to A_i or B_i by using oblivious transfer. When the *Server* wants to do OPRF evaluation on some x, it computes $u = F_k(x)$ and its OPRF value is $H(C_1[u[1]]\| \dots \|C_w[u[w]])$ where H is a hash function. For each $y \in Y$, the *Client* can get its OPRF value $H(A_1[v[1]]\| \dots \|A_w[v[w]])$ where $v = F_k(y)$.

The correctness of this protocol is follows from that for each element $y \in Y$ and $v = F_k(y)$, it holds that $A_i[v[i]] = B_i[v[i]] = C_i[v[i]]$ for $i \in [w]$. It can be seen that the *Client* can only compute OPRF values on its own input elements since the *Client* knows nothing about the *Server*'s random chooses about A_i or B_i, while the *Server* cannot learn anything about *Client*'s input set.

C Choice of Parameters

In this section, we present the specific parameter choices for CI-CM PSI under different values of N_s, N_c. The following table is derived from the inequality

$$N_s \cdot \sum_{k=0}^{\kappa-1} \binom{w}{k} p^k (1-p)^{w-k} \le negl(\lambda)$$

where $p = \left(1 - \frac{1}{m}\right)^{N_c}$.

Table 5: Parameters for *Server*'s set size N_s, *Client*'s set size N_c, matrix height m, matrix width w, and output length out in bits of the hash function H.

N_s	N_c	m	w	out
2^{20}	2^8	N_c	623	68
	2^{12}	N_c	621	72
	2^{16}	N_c	621	76
2^{24}	2^8	N_c	635	72
	2^{12}	N_c	633	76
	2^{16}	N_c	633	80
2^{28}	2^8	N_c	646	76
	2^{12}	$0.5N_c$	1887	80
	2^{12}	N_c	645	80
	2^{12}	$2N_c$	355	80
	2^{16}	N_c	645	84

D The Unbalanced PSI Protocol with Update Phase

D.1 Ideal Functionality

In Fig. 8, we present the ideal functionality of the unbalanced PSI protocol with an update phase. We only consider updates to the *Server*'s set, as updates to the *Client*'s set can be efficiently handled through the online phase of the unbalanced PSI protocol by rerunning it. In the scenario we consider, after the *Server* and *Client* obtain the intersection of their initial sets X_0 and Y_0 via PSI, the *Server* can add or remove elements on day d, denoted as X_d^{add} and X_d^{del}, respectively. The functionality computes the intersection of the updated elements with the *Client*'s set and outputs I_d^{add} and I_d^{del} to the *Client*. From this, the *Client* can compute the intersection $I_d = (I_{d-1} \cup I_d^{add}) \setminus I_d^{del}$ for day d.

Inputs: *Server*'s initial input set X_0 and *Client*'s initial input set Y_0. On day d, the set of elements added by the Server is X_d^{add}, and the set of elements deleted is X_d^{del}.
Functionality:

I. **Initial intersection.**
Upon input (Server,sid, X_0) from the *Server* and (Client,sid, Y_0) from the *Client*, the functionality outputs $I_0 = X_0 \cap Y_0$ to the *Client*.

II. **Update phase.**
Day d:
Upon input (Server,sid, (X_d^{add}, X_d^{del})) from the *Server* and (Client,sid) from the *Client*, the functionality outputs $I_d^{add} = X_d^{add} \cap Y_0$ and $I_d^{del} = X_d^{del} \cap Y_0$ to the *Client*.
The *Client* computes the intersection as $I_d = (I_{d-1} \cup I_d^{add}) \setminus I_d^{del}$.

Figure 8: The ideal functionality for unbalanced PSI protocol with update phase.

D.2 Protocols

In Fig. 9 and Fig. 10, we present the protocol diagrams for the extended unbalanced PSI protocol with an update phase, built upon our ci-rOPRF protocols (CI-CM and CI-VOLE).

E Discussion

The two ci-rOPRF constructions in our paper assume that the clients do not collude, and we provide formal security proofs for this setting. In this section, we discuss the scenario where the clients may collude. [9]

When considering the scenario where multiple *Clients* collude, we expect the CI-CM OPRF to remain secure, whereas directly using the CI-VOLE OPRF would no longer be secure. The key point, as mentioned in Sec. 3.3, is that the CI-CM OPRF allows the *Server* to select s during the online phase, enabling different values of s to be chosen for different *Clients*. In contrast, the CI-VOLE OPRF requires the *Server* to choose Δ in the offline phase, and using the same Δ for colluding *Clients* would leak the *Server*'s privacy. Specifically, after the two colluding clients, C_1 and C_2, execute the CI-VOLE OPRF protocol with the *Server*, C_1 will obtain $R + \Delta B_1$ and C_2 will obtain $R + \Delta B_2$, where B_i ($i \in \{1, 2\}$) is the OKVS encoding of the elements of *Client* C_i's set. As a result, the adversary could deduce $(R + \Delta B_1) - (R + \Delta B_2) = \Delta(B_1 - B_2)$. Since the adversary knows $B_1 - B_2$, it can then compute the *Server*'s secret Δ. In contrast, the CI-CM OPRF can avoid this issue by using different values of s for each *Client*. Therefore, we expect CI-CM OPRF to remain secure even in the presence of multiple colluding clients. However, a rigorous security proof is still required, and we consider this as part of future work.

Although directly using CI-VOLE OPRF is not secure when multiple clients collude, we can address this issue by selecting different Δ values for each *Client*. However, this would require re-executing the offline phase for each *Client*. Nonetheless, the offline phase remains client-independent, meaning it does not depend on the *Client*'s data. It is worth noting that this process does not incur significant costs. On one hand, the offline phase of the CI-VOLE OPRF is already quite efficient (taking only around 13 seconds for a set of size 2^{24}). More importantly, in this case, we can further significantly reduce the computational cost by reusing the computed results. Specifically, during the offline phase, the OPRF values for the *Server*'s set X are computed using the following expression:

$$\Psi_X := \left\{ \mathsf{H}^o\left(\mathsf{Decode}(\overrightarrow{R}, x) + \Delta \mathsf{H}^{\mathbb{B}}(x)\right) \mid x \in X \right\}.$$

Each computation requires: one OKVS decoding, two hash operations, one field addition, and one field multiplication. The primary computational cost comes from the OKVS decoding. However, when different values of Δ are selected and the expression is recomputed, the OKVS decoding and the hash operation $\mathsf{H}^{\mathbb{B}}(x)$ can be reused, meaning that only a minimal additional computational cost is incurred.

[9] By the way, collusion between the *Server* and a *Client* clearly does not leak any information about other *Clients*. This is because there is no interaction between the *Clients*, so the *Server* cannot obtain any additional information about the other *Clients* from the *Client* it colludes with.

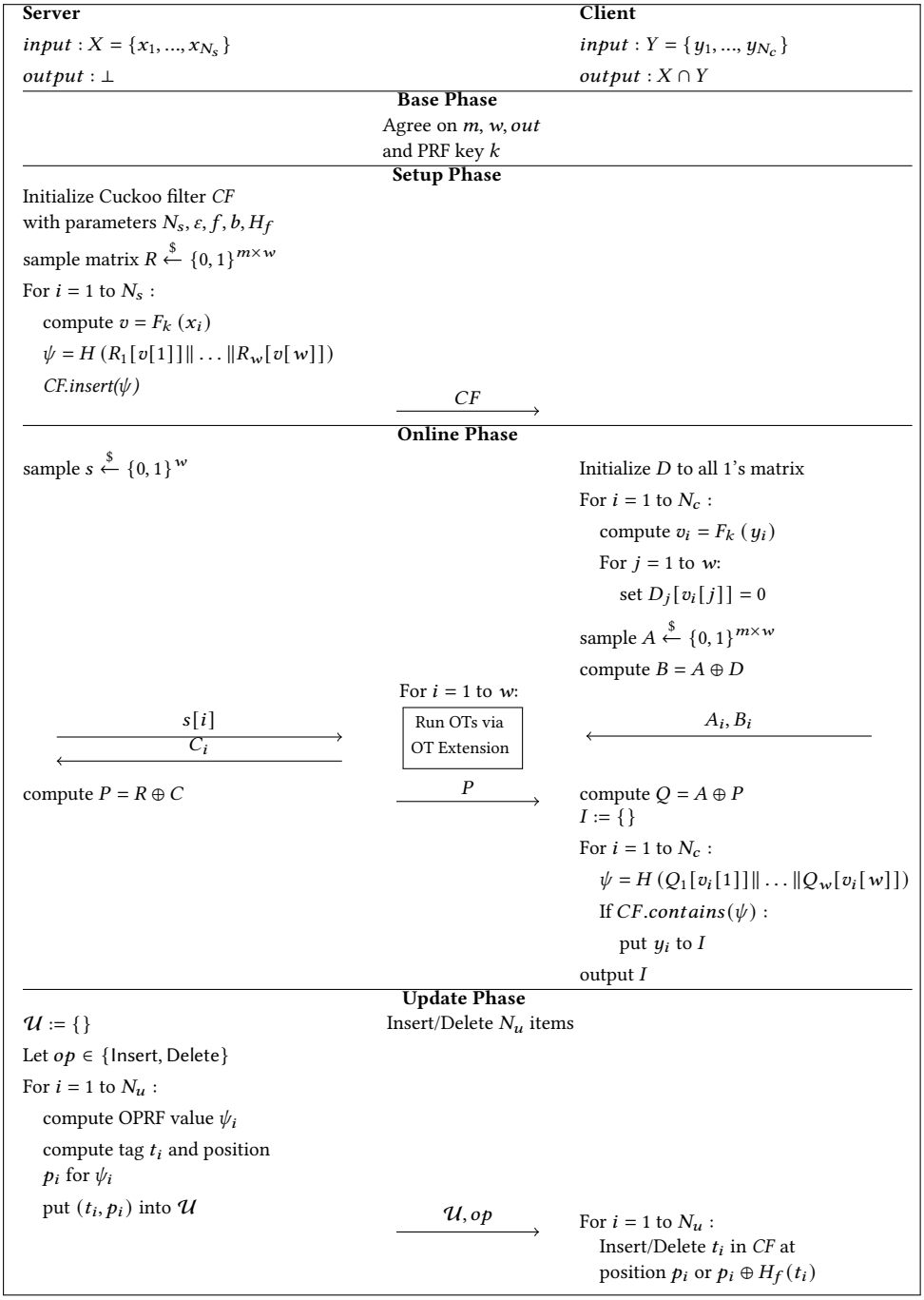

Figure 9: Unbalanced PSI protocol with update phase from CI-CM Relaxed OPRF.

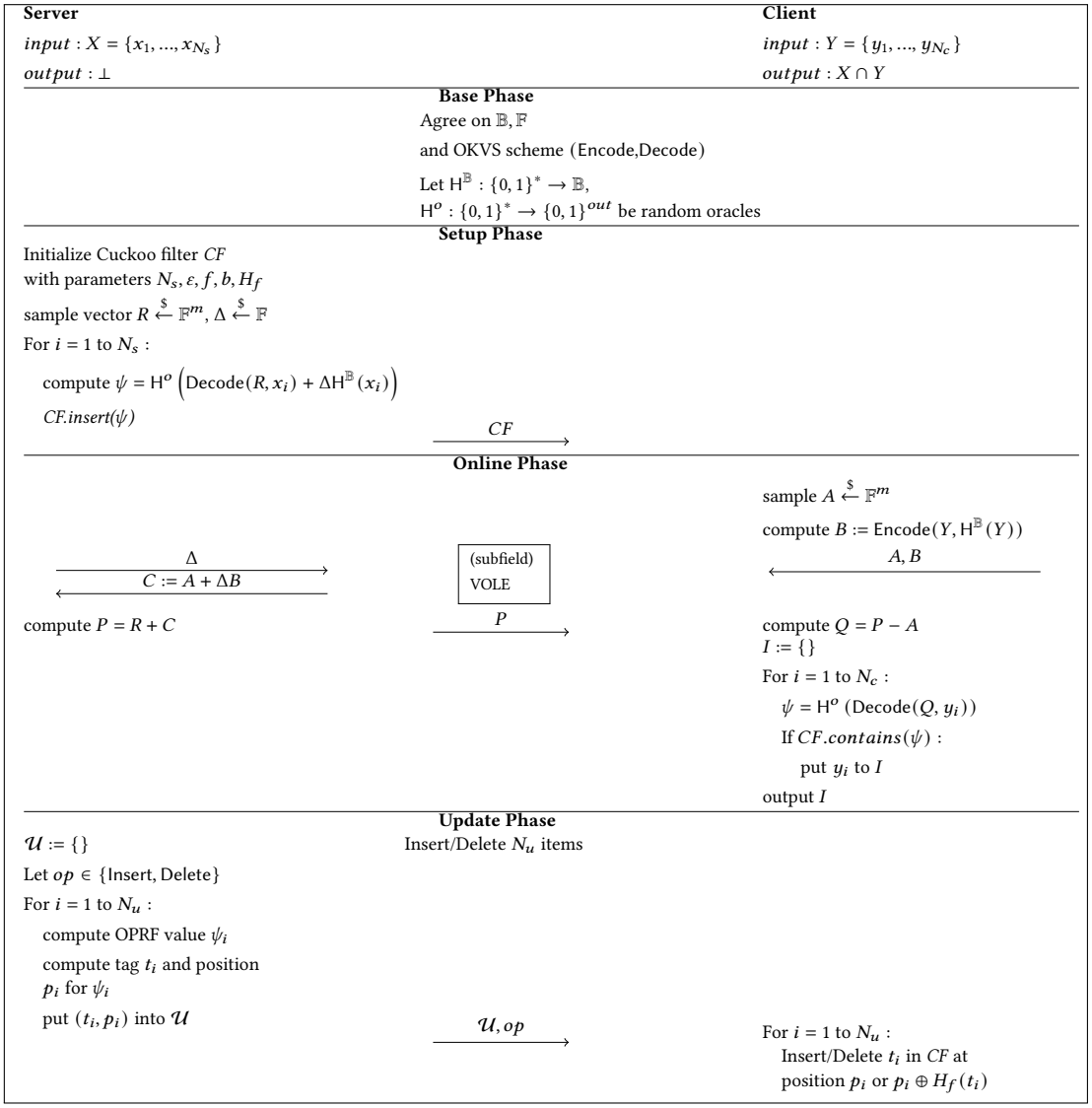

Figure 10: Unbalanced PSI protocol with update phase from CI-VOLE Relaxed OPRF.