cwPSU: Efficient Unbalanced Private Set Union via Constant-weight Codes

Qingwen Li
Xidian University
qingwen@stu.xidian.edu.cn

Song Bian
Beihang University
sbian@buaa.edu.cn

Hui Li
Xidian University
lihui@mail.xidian.edu.cn

***Abstract*—Private Set Union (PSU) allows two parties to compute the union of their private sets without revealing any additional information. While several PSU protocols have been proposed for the unbalanced setting, these constructions still suffer from substantial communication overhead as the size of the larger set increases. Moreover, their reliance on multiple invocations of oblivious pseudo-random functions results in increased communication rounds, which becomes a practical bottleneck.**

In this work, we present cwPSU, a novel unbalanced PSU protocol built upon constant-weight codes and leveled fully homomorphic encryption. To prevent leakage, we introduce a new technique called Batched Ciphertext Shuffle, which enables secure reordering of packed ciphertexts. Additionally, we propose an optimized arithmetic constant-weight equality operator, which reduces the number of non-scalar multiplications to just one-third of those required by the naïve approach. The communication complexity of our protocol scales linearly with the size of the smaller set and remains independent of the larger set. Notably, cwPSU requires only a single round of online communication.

Experimental results demonstrate that our cwPSU outperforms the state-of-the-art protocol in various network conditions, achieving a 5.1–$32.4\times$ reduction in communication and a 3.1–$13.3\times$ speedup in runtime.

I. Introduction

Private Set Union (PSU) is a fundamental cryptographic primitive that enables two parties to jointly compute the union of their private sets, without revealing any additional information. This functionality is crucial in a variety of privacy-preserve applications where data sharing must be limited to the agreed-upon results, such as security risk assessments [1]–[3], fraud detection across financial institutions [4], collaborative contact tracing [5], [6], and cross-institution medical research [7]. Many existing PSU protocols [8]–[11] are designed for the balanced setting, where the two input sets are roughly of equal size. While these protocols offer satisfactory performance under balanced conditions, their efficiency significantly degrades when deployed in unbalanced scenarios. In particular, their communication complexity is typically at least linear in the size of the larger set, making them unsuitable for settings where one party's input set is significantly larger than the other's.

Unbalanced PSU scenarios are common in real-world applications. One representative example is blacklist aggregation [10]–[13], where entities aim to combine their IP blacklists to improve network security and intrusion detection capabilities. In such cases, the sender is often a resource-constrained device, such as a mobile phone or IoT sensor, with limited computational power, storage, and battery life, while the receiver may be a powerful cloud-based server. Moreover, network bandwidth between the parties may be restricted, adding further challenges to existing PSU protocols. According to Ramanathan et al. [14], in a study across 23,483 autonomous systems, over 176 million blacklisted IP addresses were collected, with significant variation in list sizes. Some blacklists included over 500,000 IP addresses, while others contained fewer than 1,000. Balanced PSU protocols perform poorly in such unbalanced settings due to their reliance on linear communication with respect to the larger set.

To address the challenges of unbalanced PSU, two notable works [12], [13] have proposed protocols achieving standard-model security in such unbalanced settings. Tu et al. [12] introduced the Permuted Matrix Private EQuality Test (pm-PEQT) and presented the first unbalanced PSU protocol leveraging leveled Fully Homomorphic Encryption (FHE) [15]–[17]. Their design achieves communication complexity that is linear in the size of the smaller set and logarithmic in the size of the larger set. Building on this, Zhang et al. [13] optimized [12] by incorporating the Oblivious Key-Value Store (OKVS) data structure [18]–[20], which significantly improves efficiency by offloading most of the computational and communication costs to the setup phase. Their protocol demonstrates superior performance during the online phase compared to [12]. Despite these advances, both protocols still maintain a communication complexity that depends at least partially on the size of the larger set. As a result, when the receiver's set grows large (e.g., up to 2^{22} elements), the communication overhead becomes substantial. Additionally, both approaches rely on multiple Oblivious Pseudo-Random Function (OPRF) [21] invocations, making the number of communication rounds another critical bottleneck in practice.

This work aims to design a new unbalanced PSU protocol that further reduces communication complexity and minimizes the number of interaction rounds. Our goal is to provide a more

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.231128
www.ndss-symposium.org

practical and scalable solution by optimizing both computation and communication costs, particularly in scenarios with large-scale input.

A. Contribution

In this work, we propose a novel unbalanced PSU protocol, cwPSU, which is based on constant-weight encoding. Our design achieves significant improvements in communication complexity and round efficiency, while also offering competitive computational performance. The main contributions of this paper are summarized as follows:

cwPSU protocol construction. We design a new unbalanced PSU protocol based on constant-weight encoding, called cwPSU, in which the communication cost depends linearly only on the size of the smaller set and remains independent of the larger set's size. The protocol requires only one round of online communication. However, a direct use of the selection vectors generated by this approach may reveal partial information about the sender's input set.

Batched ciphertext shuffle. To address the potential leakage in the cwPSU protocol, we introduce a novel primitive called Batched Ciphertext Shuffle (BCS), which leverages leveled FHE and the Permute + Share technique. BCS enables the sender to reorder the packed ciphertext of selection vectors according to its permutation, effectively preserving privacy without increasing the number of online communication rounds.

Optimized equality operator. We propose a new arithmetic constant-weight equality operator EEQ, which reduces the number of ciphertext-ciphertext multiplications by two-thirds compared to the naïve operator. This optimization significantly improves computational efficiency in the equality testing step of the protocol.

Implementation and evaluation. We implement our cwPSU protocol and perform a detailed comparison with the state-of-the-art unbalanced PSU protocol [13]. Experimental results show that, in the online phase, our protocol achieves a $5.1-32.4\times$ reduction in communication cost and a $3.1-13.3\times$ speedup in online runtime under various network settings.

B. Related Work

As a special case of secure two-party computation, private set operations primarily include Private Set Intersection (PSI) and PSU.

Over the past decade, PSI [22]–[24] has been extensively studied in both academic and applied cryptography communities. Balanced PSI protocols have achieved significant progress, with most constructions relying on Oblivious Transfer (OT) extension [25]–[28] or various implementations of OPRF [29]–[32]. These techniques enable efficient PSI protocols when both parties hold input sets of comparable size. Chen et al. [17] were the first to propose a fast unbalanced PSI protocol using FHE, achieving sublinear communication complexity with respect to the size of the larger set. This line of work has since been further optimized in follow-up research [33]–[37]. Although PSI and PSU share certain cryptographic building blocks and design principles, PSU introduces unique challenges. As a result, PSU requires tailored constructions to ensure both efficiency and security.

Early research on PSU focused primarily on balanced settings, where both parties' sets are of comparable size. The existing constructions can broadly be classified by their underlying cryptographic primitives. One line of work predominantly uses public-key techniques [38]–[40], which often offer robust security properties at the cost of higher computational and communication overhead. Another line relies on symmetric-key techniques, typically in combination with oblivious transfer (OT) [8]–[10], enabling significantly faster protocols in practice. Chen et al. [41] later proposed an improved PSU protocol from the DDH assumption with linear complexity in the balanced case. Despite these advancements, most balanced PSU protocols suffer from considerable performance degradation once one set becomes significantly larger than the other, since their cost generally scales with the larger set's size.

Motivated by real-world scenarios where one party's set can be orders of magnitude bigger (e.g., a resource-constrained mobile device vs. a high-performance server), researchers have recently begun focusing on unbalanced PSU. Jia et al. [10] were the first to address an unbalanced setting, introducing a PSU protocol that uses cuckoo hashing and a Permute + Share [42], [43] subroutine, followed by multiple invocations of OPRF. However, their protocol incurred linear communication complexity in the size of the larger set and did not achieve standard-model security. Tu et al. [12] later proposed the first unbalanced PSU protocol to offer standard-model security, achieving sublinear communication in the size of the larger set. Their key technique was a new polynomial randomization method for FHE-based unbalanced PSI, combined with the pm-PEQT to reduce leakage. While this design yielded communication complexity linear in the smaller set and logarithmic in the larger set, it still relied on partition techniques that introduced significant constant factors. Subsequently, Zhang et al. [13] refined Tu et al.'s approach by incorporating an OKVS, shifting a substantial portion of computational and communication overhead to the setup phase. They achieved improved performance during the online phase and likewise required only sublinear communication in the larger set's size. Despite these advances, both protocols continue to have a residual dependence on the size of the bigger set, leading to notable overhead for very large input sets (e.g., 2^{22} elements). Furthermore, because they rely on multiple OPRF invocations, the number of communication rounds can also become a practical bottleneck.

II. Technique Overview

We present an overview of our cwPSU protocol as follows. We first design a novel unbalanced PSU protocol that leverages constant-weight encoding and leveled FHE. In our construction, the communication cost depends linearly only on the size of the smaller set and is independent of the size of the larger set. However, a direct application of these optimizations

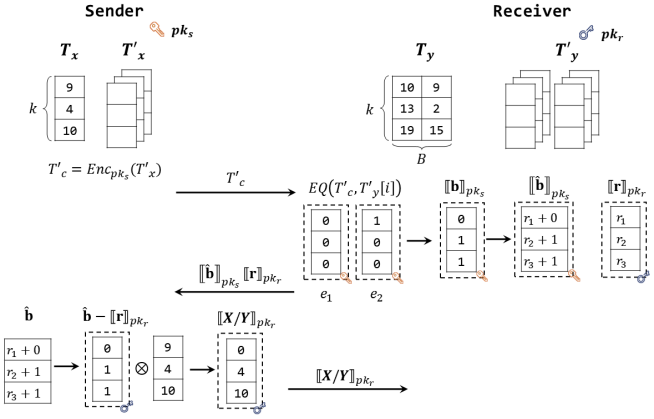

Fig. 1: Overview of our basic protocol.

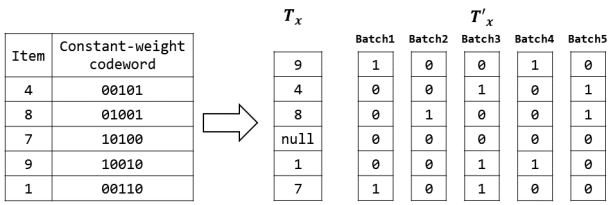

Fig. 2: The example of constant-weight codeword batching, where $l = 5$ and $h = 2$.

may lead to leakage about the sender's input set. To mitigate this issue, we introduce a new cryptographic primitive, called Batched Ciphertext Shuffle (BCS), which securely reorders batched ciphertexts of selection vectors. By integrating BCS, we obtain our full unbalanced PSU protocol that is secure in the semi-honest adversarial model. For clarity, we denote the two parties as the sender $\mathcal{S}$ and the receiver $\mathcal{R}$, holding input sets X and Y, respectively, where $|X| = m$, $|Y| = n$, and $m \ll n$.

A. Basic Protocol

Figure 1 provides an overview of the main steps in our basic protocol. Before the protocol begins, both $\mathcal{S}$ and $\mathcal{R}$ asynchronously preprocess their datasets, including hashing, constant-weight encoding and packing. Specifically, the sender $\mathcal{S}$ inserts the small set X into a 1-dimensional Cuckoo hash table $\mathbb{T}_x$, where each bin $\mathbb{T}_x[i]$ contains exactly one item. Meanwhile, the receiver $\mathcal{R}$ inserts the large set Y into a 2-dimensional hash table $\mathbb{T}_y$ using standard hashing, where each bin has a maximum load of B. This technique is common in PSI and PSU [12], [36], [37]. Inspired by [44], we adopt permutation-based hashing $H : \{0,1\}^{\sigma} \to \{0,1\}^{\sigma'}$ to reduce the effective bit-length of each item. After that, each item is then encoded into a binary string of length l using constant-weight encoding. Dummy items used to fill unused bins are represented as the all-zero binary string of length l. The value of l is determined by the item's effective bit-length δ and the Hamming weight h of the constant-weight codeword, as defined in Equation 2. The constant-weight codeword hash tables are denoted by $\mathbb{T}'_x$ for the sender and $\mathbb{T}'_y$ for the receiver. We now describe how $\mathcal{S}$ prepares its data for homomorphic operations. Given a homomorphic encryption packing parameter N, up to N plaintexts can be packed into a single batch. Since each bit of a constant-weight codeword requires a separate homomorphic computation, we pack the bits at the same position across multiple codewords into a single batch to enable efficient parallel processing. Specifically, for every N bins in $\mathbb{T}'_x$, we extract the bits at the same position from their corresponding codewords and aggregate them into a single batch, as illustrated in Figure 2. The receiver $\mathcal{R}$ follows a similar batching strategy. By treating each of the B columns in its hash table $\mathbb{T}'_y$ as an independent 1-dimensional hash table, $\mathcal{R}$ applies the same packing process to prepare its own encoded dataset for comparison.

After the preprocessing and packing steps, the sender $\mathcal{S}$ encrypts the batched constant-weight codeword and transmits them to the receiver $\mathcal{R}$. For each bin in $\mathbb{T}'_x$, the receiver performs equality comparison between $\mathcal{S}$'s encoded item and all B items in the corresponding bin of $\mathbb{T}'_y$, using Algorithm 1. Thanks to batching, $\mathcal{R}$ can evaluate the equality comparison for up to N bins in parallel. As described in Algorithm 1, if two items are equal, the result is a ciphertext of 1; otherwise, the result is 0. In Section V, we present an optimized equality comparison algorithm that reduces the number of ciphertext–ciphertext multiplications by two-thirds relative to the naïve approach. For each bin i, $\mathcal{R}$ computes the equality between $\mathcal{S}$'s item and all B candidate items from its own bin, yielding B encrypted comparison results $\{e_1, \cdots, e_B\}$. These are then summed to compute a ciphertext $\mathsf{sum} = \sum_{i=1}^{B} e_i$, and the receiver computes the encrypted selection bit as $1 - \mathsf{sum}$. The resulting encrypted selection vector is $1 - \mathsf{sum} = [\![\mathbf{b}]\!]_{\mathsf{pk}_s} = [\![b_1, \cdots, b_k]\!]_{\mathsf{pk}_s}$, where $b_i = 0$ indicates that the item in $\mathcal{S}$'s bin i is already present in $\mathcal{R}$'s set, and $b_i = 1$ indicates that the item should be sent by $\mathcal{S}$.

To preserve privacy and prevent $\mathcal{S}$ from learning this selection vector, $\mathcal{R}$ applies a blinding technique. Specifically, $\mathcal{R}$ samples k random values $r_1, \cdots, r_k$, and computes the blinded selection vector $[\![\hat{\mathbf{b}}]\!]_{\mathsf{pk}_s} = [\![b_1 + r_1, \cdots, b_k + r_k]\!]_{\mathsf{pk}_s}$. After that, $\mathcal{R}$ packs the random values into vectors, which are then encrypted under its public key to produce $[\![\mathbf{r}]\!]_{\mathsf{pk}_r} = [\![r_1, \cdots, r_k]\!]_{\mathsf{pk}_r}$. Finally, $\mathcal{R}$ sends both $[\![\hat{\mathbf{b}}]\!]_{\mathsf{pk}_s}$ and $[\![\mathbf{r}]\!]_{\mathsf{pk}_r}$ to $\mathcal{S}$.

Upon receiving the blinded selection vector $[\![\hat{\mathbf{b}}]\!]_{\mathsf{pk}_s}$, the sender $\mathcal{S}$ first decrypts it and then homomorphically subtracts the encrypted randomness $[\![\mathbf{r}]\!]_{\mathsf{pk}_r}$, resulting in the selection vector encrypted under $\mathcal{R}$'s public key $[\![\mathbf{b}]\!]_{\mathsf{pk}_r} = [\![b_1, \cdots, b_k]\!]_{\mathsf{pk}_r}$. Next, $\mathcal{S}$ performs a plaintext-ciphertext multiplication between each selection bit b_i and the corresponding item $\mathbb{T}_x[i]$, producing $[\![b_1 \cdot \mathbb{T}_x[1], \cdots, b_k \cdot \mathbb{T}_x[k]]\!]_{\mathsf{pk}_r}$. Finally, $\mathcal{S}$ sends the resulting ciphertext vector to $\mathcal{R}$, which allows $\mathcal{R}$ to recover the union set.

Despite the protocol's correctness and efficiency, the above approach introduces a potential privacy leakage. When the receiver $\mathcal{R}$ obtains the vector $(b_1 \cdot \mathbb{T}_x[1], \cdots, b_k \cdot \mathbb{T}_x[k])$, it can

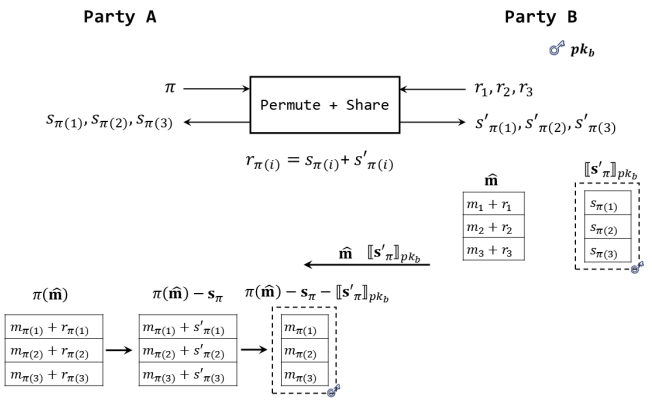

Fig. 3: Batched ciphertext shuffle protocol.

infer the value of each selection bit b_i by checking whether the $b_i \cdot \mathbb{T}_x[i]$ equals zero. Specifically, if the $b_i \cdot \mathbb{T}_x[i]$ is zero, it indicates that the corresponding item in bin i appears in the set of both parties. Although $\mathcal{R}$ does not learn the actual values of any items in $\mathcal{S}$'s input set, it is still able to deduce which bins contain items shared by both parties. This reveals structural information about the sender's set and violates the ideal privacy guarantee of PSU.

B. Batched Ciphertext Shuffle

To overcome the leakage introduced in the previous approach, we propose a new cryptographic primitive called Batched Ciphertext Shuffle, as shown in Figure 3. The goal of BCS is to allow one party, $\mathcal{A}$, to shuffle an encrypted message vector without learning the message contents, while simultaneously ensuring that $\mathcal{B}$ learns nothing about the permutation strategy.

In our BCS protocol, party $\mathcal{A}$ holds a permutation π, while party $\mathcal{B}$ holds a message vector $\mathbf{m} = (m_1, \cdots, m_k)$, along with a corresponding randomness vector $\mathbf{r} = (r_1, \cdots, r_k)$. The two parties first engage in the permute + share procedure. In this phase, $\mathcal{A}$ applies its permutation π to the randomness vector held by $\mathcal{B}$, resulting in two correlated shares. $\mathcal{A}$ receives $\mathbf{s}_\pi$, and $\mathcal{B}$ receives $\mathbf{s}'_\pi$, such that for every index i, the sum $s_{\pi(i)} + s'_{\pi(i)}$ equals $r_{\pi(i)}$.

Following this, $\mathcal{B}$ masks the original messages using a one-time pad approach, adding the randomness to each message entry to obtain a randomized vector $\hat{\mathbf{m}} = (m_1 + r_1, \cdots, m_k + r_k)$. In parallel, $\mathcal{B}$ batches and encrypts its share $\mathbf{s}'_\pi$ under its public key, producing the ciphertext $[\![\mathbf{s}'_\pi]\!]_{\mathsf{pk}_b}$. Both the masked message vector $\hat{\mathbf{m}}$ and the encrypted share $[\![\mathbf{s}'_\pi]\!]_{\mathsf{pk}_b}$ are then transmitted to $\mathcal{A}$.

Upon receiving these, $\mathcal{A}$ reorders the masked messages according to its private permutation π, obtaining the permuted vector $\hat{\mathbf{m}}_\pi = (\hat{m}_{\pi(1)}, \cdots, \hat{m}_{\pi(k)})$. It then subtracts its own share $\mathbf{s}_\pi$ to compute an intermediate vector $\tilde{\mathbf{m}}_\pi = \hat{\mathbf{m}}_\pi - \mathbf{s}_\pi$, which effectively equals the true permuted message vector plus $\mathcal{B}$'s encrypted share. Finally, by performing a homomorphic subtraction between $\tilde{\mathbf{m}}_\pi$ and $[\![\mathbf{s}'_\pi]\!]_{\mathsf{pk}_b}$, $\mathcal{A}$ obtains the permuted batched ciphertext $[\![\mathbf{m}_\pi]\!]_{\mathsf{pk}_b}$, without ever learning the underlying plaintexts or randomness.

C. Full cwPSU Protocol

To summarize, our complete cwPSU protocol proceeds as follows.

Initially, the sender $\mathcal{S}$ and the receiver $\mathcal{R}$ preprocess their respective input sets X and Y, including hashing, constant-weight encoding, and batching. After these transformations, $\mathcal{S}$ encrypts the batched constant-weight codewords and sends them to $\mathcal{R}$. Using these ciphertexts, $\mathcal{R}$ computes the encrypted selection vector $[\![\mathbf{b}]\!]_{\mathsf{pk}_s} = [\![b_1, \cdots, b_k]\!]_{\mathsf{pk}_s}$.

Next, the parties execute the BCS protocol. In this phase, $\mathcal{S}$ acting as party $\mathcal{A}$ provides the permutation π, while $\mathcal{R}$ as party $\mathcal{B}$ inputs the blinded selection vector $[\![\hat{\mathbf{b}}]\!]_{\mathsf{pk}_s} = [\![b_1 + r_1, \cdots, b_k + r_k]\!]_{\mathsf{pk}_s}$, along with the randomness values $r_1, \cdots, r_k$. Unlike the original BCS protocol, here the input from $\mathcal{R}$ is a ciphertext encrypted under $\mathcal{S}$'s public key. This introduces no issues: $\mathcal{S}$ can decrypt the blinded vector using its private key, while $\mathcal{R}$ does not need to know the actual values of $\mathbf{b}$, as the blinding is applied via homomorphic addition. Therefore, this variant remains fully compatible with our BCS construction.

After executing BCS, $\mathcal{S}$ obtains the permuted selection vector encrypted $[\![\mathbf{b}_\pi]\!]_{\mathsf{pk}_r}$. $\mathcal{S}$ then reorders its own dataset according to π, performs plaintext-ciphertext multiplication with the permuted selection vector, and sends the result back to $\mathcal{R}$.

To further optimize protocol efficiency, we follow the design principle proposed in [13] and divide the protocol into setup and online phases. In the setup phase, $\mathcal{R}$ can pre-download the ciphertexts of $\mathcal{S}$'s dataset. This is similar in spirit to digest download in Simple PIR [45], and allows reusability when $\mathcal{R}$'s input set changes and PSU needs to be re-run. Additionally, the Permute + Share part of BCS is independent of the dataset content and can also be precomputed in the setup phase. As a result, the online phase of BCS requires only a half round of communication, and combined with the receiver's query, our cwPSU protocol achieves a total of just one round of online interaction.

III. Preliminary

A. Notations

Throughout this paper, we will use the notation $[\![x]\!]_{\mathsf{pk}}$ to denote the encryption of element x under key pk. We use bolded lowercase letters for vectors. The computational and statistical security parameters will be denoted by κ and λ, respectively. N, q, t represent parameters in FHE, denoting the degree of the modulus polynomial, the ciphertext modulus, and the plaintext modulus, respectively. The entire parameter set of FHE is denoted as params. For $x \in \mathbb{R}$, we use $\lfloor x \rceil$ to denote rounding to the nearest integer, $\lceil x \rceil$ to represent rounding up, and $\lfloor x \rfloor$ for rounding down.

B. Leveled Fully Homomorphic Encryption

Leveled FHE is a cryptographic technique that permits encrypted data to be multiplied and added to a certain depth without decryption. Leveled FHE restricts computation depth by introducing a hierarchical structure that allows for more efficient computations. In this study, we utilize the BFV scheme [15], a specific leveled FHE scheme based on the computational complexity of the Ring Learning With Errors (RLWE) problem [46].

For the BFV scheme, we use t and q to represent the plaintext modulus and the ciphertext modulus, respectively. This implies that the plaintext and ciphertext spaces are represented as rings $R_t = \mathbb{Z}_t[x]/(f(x))$ and $R_q = \mathbb{Z}_q[x]/(f(x))$, respectively. Here, $f(x) = x^n + 1$ is a cyclotomic polynomial, and n is a power of 2.

The BFV scheme is actually a naturally Somewhat Homomorphic Encryption (SHE) scheme, capable of supporting a limited number of homomorphic additions and multiplications. It is worth noting that, through techniques such as key switching and modulus switching, BFV can be extended into a Leveled FHE scheme. Our proposed scheme only requires homomorphic addition and plaintext-ciphertext multiplication, and as a result, does not employ these two techniques. For further details, please refer to [15].

The remarkable computational efficiency of second-generation fully homomorphic encryption schemes is attributed to the key technique of batching. In the RLWE-based scheme, the plaintext space is a polynomial ring denoted as $R_t = \mathbb{Z}_t[x]/(f(x))$. Through batching, a vector over $\mathbb{F}_t$ can be seamlessly encoded into a polynomial over R_t. Utilizing the isomorphic properties of the Chinese Remainder Theorem, polynomial operations over R_t parallelize homomorphic operations, making them ideal for scenarios requiring rapid homomorphic computations.

In our protocol, we rely on basic homomorphic operations supported by leveled FHE, specifically ciphertext-ciphertext and ciphertext-plaintext additions and multiplications. The cost of each homomorphic operation is characterized by two primary metrics: its *runtime* and the *noise* added to the ciphertext. For efficiency analysis, we assume that both ciphertexts and plaintexts are represented in Double-CRT form and that ciphertext noise is measured by its infinity norm. We also assume the use of hybrid key-switching mechanisms for bootstrapping and noise management.

Homomorphic addition is the most lightweight operation, requiring only $\mathcal{O}(n \log q)$ integer additions. The resulting noise is simply the sum of input noises, multiplied by a small factor $\mathcal{O}(t)$.

We distinguish between two types of homomorphic multiplication:

- **Scalar multiplication** (ciphertext-plaintext multiplication): This operation involves coefficient-wise multiplication of an n-dimensional ciphertext vector with a plaintext. It incurs $\mathcal{O}(n \log q)$ integer multiplications and increases the noise by $\mathcal{O}(t\sqrt{n})$.
- **Non-scalar multiplication** (ciphertext-ciphertext multiplication): This is more costly and consists of two steps: vector convolution and key-switching. The convolution phase requires $\mathcal{O}(n \log n)$ integer multiplications, while the key-switching step involves $\mathcal{O}(n \log n \log q + n(\log q)^2)$ integer multiplications. Thus, the overall cost is $\mathcal{O}(n \log n \log q + n(\log q)^2)$, and the output noise grows proportionally to $\mathcal{O}(n \cdot t \cdot \max(V_1, V_2))$, where V_1 and V_2 are the input ciphertext noise levels.

C. Constant-weight Encoding

A constant-weight code, or an l-of-h code, is a class of error-detecting codes in which all codewords have the same Hamming weight. In the special case of binary constant-weight codes, each codeword is a binary string with a fixed number of ones. Constant-weight codes have found important applications in the design of secure multi-party computation (MPC) protocols, including private set intersection (PSI) [44] and private information retrieval (PIR) [47], [48].

The length of a code refers to the number of bits in each codeword, and the size of the code is the number of distinct codewords. For a binary constant-weight code of length l and Hamming weight h, the number of valid codewords is given by the binomial coefficient $\binom{l}{h}$.

To ensure that the code can represent at least n distinct values with fixed weight h, the length l must satisfy:

$$\binom{l}{h} \geq n. \tag{1}$$

The details of how to map numbers from the set $[n]$ into constant-weight codewords can be found in [47].

For the effective bit-length δ of n values, we have $n = 2^\delta$. As an approximation, the length can be bounded by:

$$l \in O\left(\sqrt[h]{h! \cdot 2^\delta} + h\right). \tag{2}$$

We denote the set of all binary constant-weight codewords of length l and weight h as $CW(l, h)$.

Given two codewords $x, y \in CW(l, h)$, it is often necessary to test whether $x = y$. Algorithm 1 shows the arithmetic equality operator [47] over constant-weight codewords that uses arithmetic operations and is compatible with field-based computation.

Algorithm 1 Arithmetic Constant-weight Equality Operator $\text{cwEQ}(x, y)$.

Input: $x, y \in CW(l, h)$

1: $k = \sum_{i \in [l]} x[i] \cdot y[i]$

2: $e = \frac{1}{h!} \cdot \prod_{i \in [h]} (k - i)$

Output: $e \in \{0, 1\}$

The output e takes value in $\{0, 1\}$, where:

$$e = \begin{cases} 1 & \text{if } x = y \text{ (i.e., } k' = k) \\ 0 & \text{if } k' < k \end{cases} \tag{3}$$

This method relies on the fact that the inner product of two identical constant-weight codewords is exactly k, while any mismatch results in a value less than k. As a result, the product in the expression becomes zero unless $k' = k$.

A circuit implementing this operator requires $l + h$ multiplications and achieves a multiplicative depth of $1 + \lceil \log_2 h \rceil$.

D. Permute + Share

We introduce the Permute + Share functionality [42], [43], denoted by $\mathcal{F}_{\text{PS}}$, which involves two parties, denoted as $\mathcal{A}$ and $\mathcal{B}$. Party $\mathcal{A}$ inputs a private permutation π of length n, while party $\mathcal{B}$ inputs a vector $\mathbf{x} = (x_1, \ldots, x_n)$. After executing $\mathcal{F}_{\text{PS}}$, both parties receive respective shuffled shares: $\mathcal{A}$ obtains shuffled shares $\mathbf{s}_\pi = (s_{\pi(1)}, \ldots, s_{\pi(n)})$, and $\mathcal{B}$ obtains complementary shuffled shares $\mathbf{s}'_\pi = (s'_{\pi(1)}, \ldots, s'_{\pi(n)})$. These shares satisfy the relation: $x_{\pi(i)} = s_{\pi(i)} + s'_{\pi(i)}, i \in [n]$, where $+$ denotes a group operation (such as XOR, addition modulo a number, etc.) depending on the underlying construction.

We formally define the functionality as follows:

Parameters: Parties $\mathcal{A}$ and $\mathcal{B}$, vector length n.

Functionality $\mathcal{F}_{\text{PS}}$:

1) Wait for a permutation π on $[n]$ from party $\mathcal{A}$; abort if π is invalid.
2) Wait for input vector $\mathbf{x} = (x_1, \ldots, x_n)$ from party $\mathcal{B}$; abort if $|\mathbf{x}| \neq n$.
3) Output shuffled shares $\mathbf{s}_\pi$ to $\mathcal{A}$, and shuffled shares $\mathbf{s}'_\pi$ to $\mathcal{B}$, where $x_{\pi(i)} = s_{\pi(i)} + s'_{\pi(i)}, i \in [n]$.

Fig. 4: Permute + Share functionality $\mathcal{F}_{\text{PS}}$

E. Private Set Union

PSU is a special case of secure two-party computation. We describe the ideal functionality of PSU in Figure 5.

Parameters: Sender $\mathcal{S}$ and receiver $\mathcal{R}$, set sizes m and n.
Functionality $\mathcal{F}_{\text{PSU}}$:

1) Wait for an input $X = \{x_1, \cdots, x_m\} \subseteq \{0,1\}^\sigma$ from the sender, and an input $Y = \{y_1, \cdots, y_n\} \subseteq \{0,1\}^\sigma$ from the receiver.
2) Output $X \cup Y$ to the receiver.

Fig. 5: Private set union functionality $\mathcal{F}_{\text{PSU}}$

IV. Unbalanced Private Set Union From Constant-weight Encode

A. Batched Ciphertext Shuffle

We formally define the functionality $\mathcal{F}_{\text{BCS}}$ in Figure 6. In this functionality, party $\mathcal{A}$ inputs a permutation π over $[k]$, while party $\mathcal{B}$ inputs a message vector $\mathbf{m} = (m_1, \ldots, m_k)$ and a randomness vector $\mathbf{r} = (r_1, \ldots, r_k)$. The functionality outputs to party $\mathcal{A}$ batched ciphertexts encrypted under $\mathcal{B}$'s public key, encrypting the plaintext vector consisting of messages from $\mathbf{m}$ arranged according to the permutation π. Throughout this process, party $\mathcal{A}$ learns nothing about the contents of $\mathbf{m}$, while party $\mathcal{B}$ learns nothing about the permutation π.

Parameters: Party $\mathcal{A}$ and $\mathcal{B}$, public key pk_b of $\mathcal{B}$, FHE parameter N.
Functionality $\mathcal{F}_{\text{BCS}}$:

- Wait for input $\pi \in S_k$ from $\mathcal{A}$.
- Wait for input $\mathbf{m} = (m_1, \ldots, m_k)$ and $\mathbf{r} = (r_1, \ldots, r_k)$ from $\mathcal{B}$.
- $\mathcal{A}$ output $\gamma = \lceil k/N \rceil$ batched ciphertexts $[\![\mathbf{m}^i_\pi]\!]_{\mathsf{pk}_b} = [\![m_{\pi(i\cdot N+1)}, \ldots, m_{\pi(i\cdot N+k)}]\!]_{\mathsf{pk}_b}, i \in [\gamma]$.
- Reveal nothing to $\mathcal{B}$.

Fig. 6: Batched Ciphertext Shuffle Functionality $\mathcal{F}_{\text{BCS}}$

We realize the BCS functionality using homomorphic encryption and the Permute + Share functionality. The formal description of our protocol construction is given in Figure 7.

Input: The party $\mathcal{A}$ inputs a permutation π over $[k]$; the party $\mathcal{B}$ inputs a message vector $\mathbf{m} = (m_1, \ldots, m_k)$, a randomness vector $\mathbf{r} = (r_1, \ldots, r_k)$ and public key pk_b; FHE parameter N.
Output: $\mathcal{A}$ outputs $\gamma = \lceil k/N \rceil$ batched ciphertexts $[\![\mathbf{m}^i_\pi]\!]_{\mathsf{pk}_b} = [\![m_{\pi(i\cdot N+1)}, \ldots, m_{\pi(i\cdot N+k)}]\!]_{\mathsf{pk}_b}, i \in [\gamma]$; $\mathcal{B}$ outputs $\perp$.

(1) $\mathcal{A}$ and $\mathcal{B}$ invoke the ideal Permute + Share functionality $\mathcal{F}_{\text{ps}}$. $\mathcal{A}$ inputs the permutation π over $[k]$, and learns shuffled shares $\mathbf{s}_\pi$. $\mathcal{B}$ inputs the randomness vector $\mathbf{r} = (r_1, \ldots, r_k)$, and learns shuffled shares $\mathbf{s}'_\pi$.
(2) $\mathcal{B}$ adds the randomness $\mathbf{r}$ to each message entry to obtain a randomized vector $\hat{\mathbf{m}} = (m_1 + r_1, \cdots, m_k + r_k)$. Meanwhile, $\mathcal{B}$ batches and encrypts its shares $\mathbf{s}'_\pi$, producing $\gamma = \lceil k/N \rceil$ ciphertext $[\![\mathbf{s'}^i_\pi]\!]_{\mathsf{pk}_b} = [\![s'_{i\cdot N+1}, \ldots, s'_{i\cdot N+k}]\!]_{\mathsf{pk}_b}, i \in [\gamma]$. Both the $\hat{\mathbf{m}}$ and $[\![\mathbf{s'}_\pi]\!]_{\mathsf{pk}_b}$ are then send to $\mathcal{A}$.
(3) $\mathcal{A}$ first reorders the $\hat{\mathbf{m}}$ according to π, obtaining $\hat{\mathbf{m}}_\pi = (\hat{m}_{\pi(1)}, \cdots, \hat{m}_{\pi(k)})$. It then computes the intermediate vector $\tilde{\mathbf{m}}_\pi = \hat{\mathbf{m}}_\pi - \mathbf{s}_\pi$ by subtracting its own share. Finally, by homomorphically performing $\tilde{\mathbf{m}}_\pi - [\![\mathbf{s}'_\pi]\!]_{\mathsf{pk}_b}$, $\mathcal{A}$ obtains γ batched ciphertexts $[\![\mathbf{m}^i_\pi]\!]_{\mathsf{pk}_b} = [\![m_{\pi(i\cdot N+1)}, \ldots, m_{\pi(i\cdot N+k)}]\!]_{\mathsf{pk}_b}, i \in [\gamma]$.

Fig. 7: Batched Ciphertext Shuffle Protocol

Theorem 1. *The construction given in Figure 7 securely realizes the functionality $\mathcal{F}_{BCS}$ in the $\mathcal{F}_{PS}$-hybrid model against semi-honest adversaries, assuming the underlying fully homomorphic encryption scheme is IND-CPA secure with circuit privacy.*

Proof. We prove security by demonstrating computational indistinguishability between the outputs of the ideal functionality and the real-world execution. Specifically, we construct

polynomial-time simulators $\mathsf{Sim}_{\mathcal{A}}$ and $\mathsf{Sim}_{\mathcal{B}}$ to simulate the views of the corrupted parties as follow.

Corrupt $\mathcal{A}$: Recall that in the ideal world, the party $\mathcal{A}$ inputs a permutation π over the set $[k]$ to the ideal functionality $\mathcal{F}_{\text{BCS}}$, and receives ciphertexts encrypting the permuted message vector $\mathbf{m}^{\pi}$. During the real protocol execution, $\mathcal{A}$ obtains the following intermediate values: the shuffled shares $\mathbf{s}_{\pi}$, the masked message vector $\hat{\mathbf{m}}$, and the encrypted shuffled shares $[\![\mathbf{s}'_{\pi}]\!]_{\mathsf{pk}_b}$.

To simulate the view of $\mathcal{A}$ in the ideal world, we construct a simulator $\mathsf{Sim}_{\mathcal{A}}$ that proceeds as follows: First, $\mathsf{Sim}_{\mathcal{A}}$ samples a random vector $\bar{\mathbf{r}} = (\bar{r}_1, \ldots, \bar{r}_k)$, and interacts with the ideal functionality $\mathcal{F}_{\text{PS}}$ using $\bar{\mathbf{r}}$ as input. It then receives a shuffled share $\bar{\mathbf{s}}_{\pi}$, which is forwarded to $\mathcal{A}$. Owing to the security guarantees of the Permute + Share functionality, $\bar{\mathbf{s}}_{\pi}$ is computationally indistinguishable from the real $\mathbf{s}_{\pi}$, ensuring that the simulated view is indistinguishable from the real one. Next, to simulate the masked message vector $\hat{\mathbf{m}}$, the simulator observes that in the real protocol, each message is randomized via additive masking, i.e., $\hat{m}_i = m_i + r_i$. Hence, $\mathsf{Sim}_{\mathcal{A}}$ samples $\bar{\mathbf{m}}$ uniformly at random from the plaintext domain, which remains computationally indistinguishable from $\hat{\mathbf{m}}$. Finally, for the ciphertexts $[\![\mathbf{s}'_{\pi}]\!]_{\mathsf{pk}_b}$, the simulator generates fresh encryptions of uniformly random plaintexts under the public key pk_b. By the IND-CPA security of the leveled homomorphic encryption scheme, these simulated ciphertexts are computationally indistinguishable from the ciphertexts generated in the real execution. Thus, the complete simulated view is indistinguishable from the real protocol view, as required.

Corrupt $\mathcal{B}$: When party $\mathcal{B}$ is corrupted, it provides the randomness vector $\mathbf{r} = (r_1, \ldots, r_k)$ and receives intermediate shuffled shares $\mathbf{s}'_{\pi}$ from the Permute + Share functionality $\mathcal{F}_{\text{PS}}$. In the ideal functionality $\mathcal{F}_{\text{BCS}}$, $\mathcal{B}$ receives no additional outputs.

To simulate the view of a corrupted $\mathcal{B}$, the simulator $\mathsf{Sim}_{\mathcal{B}}$ works as follows. Given the randomness vector $\mathbf{r}$, the simulator samples a uniformly random permutation π' independent of the true permutation π used by the honest party $\mathcal{A}$. It then invokes the simulator for the Permute + Share functionality $\mathcal{F}_{\text{PS}}$ with inputs $\mathbf{r}$ and π', obtaining shuffled shares $\mathbf{s}'_{\pi'}$. Finally, the simulator delivers $\bar{\mathbf{s}}_{\pi'}$ to the corrupted party $\mathcal{B}$. We argue that this simulated view for corrupted party $\mathcal{B}$ is computationally indistinguishable from the view in the real protocol execution, due to the security of the Permute + Share functionality $\mathcal{F}_{\text{PS}}$.

□

B. Full cwPSU Protocol

Our cwPSU protocol is formally described in Figure 8. It is noted that in our BCS protocol, the receiver's input is a plaintext message vector. However, $\mathcal{R}$ inputs ciphertexts encrypted under pk_s in PSU. $\mathcal{R}$ can homomorphically perform the same operations on ciphertexts as would be done on plaintexts, since the underlying encryption scheme is homomorphic. And $\mathcal{S}$ upon receiving the ciphertexts can decrypt them using its private key to recover the masked selection vector.

Theorem 2. *The protocol described in Figure 8 securely realizes the functionality $\mathcal{F}_{PSU}$ in the $\mathcal{F}_{BCS}$-hybrid model against semi-honest adversaries, provided that the underlying fully homomorphic encryption scheme is IND-CPA secure with circuit privacy.*

Proof. We prove security by constructing simulators $\mathsf{Sim}_{\mathcal{S}}$ and $\mathsf{Sim}_{\mathcal{R}}$ as follow to demonstrate computational indistinguishability between the ideal-world and real-world executions.

Corrupt Sender: The sender $\mathcal{S}$ simply inputs its private set X to the ideal functionality $\mathcal{F}_{\text{PSU}}$ and receives no additional output. During the real-world execution, $\mathcal{S}$ receives only the permuted batched ciphertexts of the selection vector generated via the $\mathcal{F}_{\text{BCS}}$, which are encrypted under the receiver's public key pk_r. The simulator $\mathsf{Sim}_{\mathcal{S}}$ can simulate the view of the corrupted sender by invoking the BCS simulator $\mathsf{Sim}_{\mathcal{A}}$ from the ideal functionality $\mathcal{F}_{\text{BCS}}$, with one minor modification: while $\mathsf{Sim}_{\mathcal{A}}$ simulates masked message vectors by generating random plaintexts, $\mathsf{Sim}_{\mathcal{S}}$ must encrypt these random values using pk_s to simulate the ciphertexts that would appear during the masking process in the real protocol. Since the FHE is IND-CPA secure with circuit privacy and the views of the underlying BCS simulator is indistinguishable. The view of $\mathcal{S}$ in the real execution is computationally indistinguishable from the simulated view in the ideal world.

Corrupt Receiver: In the ideal world, the receiver $\mathcal{R}$ inputs its private set Y and receives the output $X \cup Y$ from the ideal functionality $\mathcal{F}_{\text{PSU}}$. During the real-world protocol execution, $\mathcal{R}$ receives several intermediate messages during the protocol, including the encrypted and batched constant-weight codes $\{[\![\mathcal{X}_{i,z}]\!]_{\mathsf{pk}_s}\}$, and the encrypted items $[\![\mathcal{C}^{\pi}_{i,j}]\!]_{\mathsf{pk}_r}$. To simulate the view of a corrupted receiver, the simulator $\mathsf{Sim}_{\mathcal{R}}$ first generates random ciphertexts $[\![\tilde{\mathcal{X}}_{i,z}]\!]_{\mathsf{pk}_s}$ by encrypting uniformly random messages under pk_s. In order to simulate the final masked ciphertexts $[\![\mathcal{C}^{\pi}_{i,j}]\!]_{\mathsf{pk}_r}$, the simulator uses the known union set $X \cup Y$ to compute the items in $X \setminus Y$, then locally re-encodes and repacks these elements into plaintexts $\mathcal{P}^{\pi}_{i,j}$, using the same format as in the real protocol. The simulator then encrypts these messages under pk_r to obtain the final ciphertexts $[\![\mathcal{C}^{\pi}_{i,j}]\!]_{\mathsf{pk}_r} = \mathsf{FHE.Encrypt}_{\mathsf{pk}_r}(\mathcal{P}^{\pi}_{i,j})$. These simulated ciphertexts are indistinguishable from the real ones, since the underlying homomorphic encryption scheme is IND-CPA secure with circuit privacy.

□

C. Optimization for Small Sender Set

According to our cwPSU protocol, the primary computational overhead stems from invoking the arithmetic Constant-weight Equality Operator, which the receiver $\mathcal{R}$ must perform B times. Therefore, reducing B directly decreases computational cost. As B is upper-bounded by $hn/\mu + \mathcal{O}(\sqrt{hn \log \mu/\mu})$, where h is the number of hash functions, and n is the receiver's set size [49]. A straightforward approach to minimize B is to set the number of bins μ to be a

Input: The receiver $\mathcal{R}$ inputs $Y \subset \{0,1\}^\sigma$ of size $n = |Y|$, the sender $\mathcal{S}$ inputs $x \subset \{0,1\}^\sigma$ of size $m = |X|$. m, n, σ are public. λ denote the statistical security parameter.
Output: The receiver outputs $X \cup Y$, the sender outputs $\perp$.

1. **[Hashing]**
   (a) [Hashing to bins.] Given hashing parameters h, μ, B, $\mathcal{R}$ hashes hn balls into μ bins using simple hashing, resulting in a hash table $\mathbb{T}_y$ with maximum load B per bin. $\mathcal{S}$ hashes m balls into μ bins using Cuckoo hashing, obtaining a hash table $\mathbb{T}_x$.
   (b) [Hashing to shorter strings.] Let $\sigma' = \lambda + \log_2 B$. Both parties $\mathcal{R}$ and $\mathcal{S}$ sample a random hash function $H : \{0,1\}^\sigma \to \{0,1\}^{\sigma'}$, and then apply permutation-based hashing to map each item in their hash tables to a shorter representation of length σ', obtaining $\mathbb{T}^*_y$ and $\mathbb{T}^*_x$.
2. **[Process X]**
   (a) [Constant-weight encoding.] For each item $\mathbb{T}^*_x[i]$ in bin $i \in [\mu]$, $\mathcal{S}$ apply constant-weight encoding to obtain $\mathbb{T}'_x[i][z]$, where $z \in [l]$ and l is the bit-length of the constant-weight codeword.
   (b) [Batching.] For each bit position $j \in [l]$, batch together the same bit position from every N bins in $\mathbb{T}'_x$. This results in batches $\{\mathcal{X}_{i,z}\}$, where $i \in \lceil \mu/N \rceil$, $z \in [l]$.
   (c) [Encrypt.] The sender $\mathcal{S}$ encrypts each batch $\mathcal{X}_{i,z}$ using their public key pk_s with $\mathsf{FHE.Encrypt}$, resulting in ciphertexts $\{[\![\mathcal{X}_{i,z}]\!]_{\mathsf{pk}_s}\}$. $\mathcal{S}$ then sends these ciphertexts to the $\mathcal{R}$.
3. **[Process Y]**
   (a) [Constant-weight encoding.] For each item $\mathbb{T}^*_y[i][j]$ in bin $i \in [\mu]$ and position $j \in [B]$, the receiver $\mathcal{R}$ applies constant-weight encoding to obtain $\mathbb{T}'_y[i][j][z]$, where $z \in [l]$.
   (b) [Batching.] For each bit position $z \in [l]$, batch together the same bit position from every N bins and each of the B columns in $\mathbb{T}'_y$. This yields the batch collection $\{\mathcal{Y}_{i,j,z}\}$, where $i \in \lceil \mu/N \rceil$, $j \in [B]$, and $z \in [l]$.
4. **[Computing Selection Vector]** For each $i \in \lceil \mu/N \rceil$ and $j \in [B]$, the receiver $\mathcal{R}$ invokes the arithmetic constant-weight equality operator to homomorphically compute $[\![\mathbf{e}_{i,j}]\!]_{\mathsf{pk}_s} = \mathsf{cwEQ}([\![\mathcal{X}_i]\!]_{\mathsf{pk}_s}, \mathcal{Y}_{i,j})$. Then, $\mathcal{R}$ computes the sum of the comparison results across all B positions $[\![\mathbf{o}_i]\!]_{\mathsf{pk}_s} = \sum_{j=1}^{B} [\![\mathbf{e}_{i,j}]\!]_{\mathsf{pk}_s}$. Finally, the encrypted selection vector is computed as $[\![\mathbf{b}_i]\!]_{\mathsf{pk}_s} = 1 - [\![\mathbf{o}_i]\!]_{\mathsf{pk}_s}$.
5. **[Performing BCS]** $\mathcal{R}$ inputs the encrypted selection vector $[\![\mathbf{b}_i]\!]_{\mathsf{pk}_s}$ along with its own public key pk_r, while the sender $\mathcal{S}$ inputs a permutation π. As a result, $\mathcal{S}$ obtains the permuted batched ciphertexts of the selection vector $[\![\mathbf{b}^\pi_i]\!]_{\mathsf{pk}_r}$ encrypted under pk_r.
6. **[Output]** $\mathcal{S}$ reorders $\mathbb{T}_x$ according to π to obtain $\mathbb{T}^\pi_x$. Then, for each item $\mathbb{T}^\pi_x[i]$, it splits the data into plaintext slots of $\log_2 t$ bits, and packs them into messages denoted as $\mathcal{P}^\pi_{i,j}$, where $i \in \lceil \mu/N \rceil, j \in \lceil \sigma / \log_2 t \rceil$. $\mathcal{S}$ then homomorphically performs $[\![\mathcal{C}^\pi_{i,j}]\!]_{\mathsf{pk}_r} = [\![\mathbf{b}^\pi_i]\!]_{\mathsf{pk}_r} \cdot \mathcal{P}^\pi_{i,j}$. The results are sent to $\mathcal{R}$. Finally, $\mathcal{R}$ decrypts them to recover the items in $X \setminus Y$, and merges them with its own set Y to obtain the $X \cup Y$.

Fig. 8: Full cwPSU protocol.

multiple of the plaintext slots N, thus fully utilizing all available plaintext slots. However, our BCS protocol incurs a communication complexity of $\mathcal{O}(\mu \log \mu)$. Hence, simply setting μ to N would introduce unnecessary additional communication overhead if the originally required μ is significantly smaller than N.

To resolve this issue, we propose an optimized approach. The high-level idea is illustrated in Figure 9. To avoid additional overhead in the BCS protocol, we maintain the minimum required value of μ. To ensure full utilization of plaintext slots, assume without loss of generality that μ divides N. The receiver $\mathcal{R}$ duplicates its one-dimensional hash table $\tilde{\mathbb{T}}^*_x$ exactly $N/\mu - 1$ times and combines these copies to form an extended hash table of length N, which is used directly in subsequent encoding and batching steps without further modification. Meanwhile, the sender $\mathcal{S}$ evenly partitions the B items within each bin of its two-dimensional hash table $\mathbb{T}^*_y[i]$ into N/μ groups. It then places the j-th group of bin i into the $((j-1)\cdot\mu+i)$-th row of a new two-dimensional hash table $\tilde{\mathbb{T}}^*_y$ with N rows, which becomes the sender's input for subsequent computations.

According to our protocol design, if an item $\tilde{\mathbb{T}}^*_x[i]$ is contained in $\mathbb{T}^*_y[i]$, then exactly one position among $(j-1)\cdot\mu+i$ for $j \in [N/\mu]$ in the encrypted vector $[\![\tilde{\mathbf{o}}]\!]_{\mathsf{pk}_s}$ will be 1, while the others will be 0. Otherwise, all corresponding positions will be 0.

We subsequently modify the BCS protocol accordingly. Receiver $\mathcal{R}$ still inputs μ random values $\mathbf{r} = (r_1, \ldots, r_\mu)$ into the BCS protocol and obtains μ shuffled shares. To align dimensions, $\mathcal{R}$ expands the randomness vector $\mathbf{r}$ into N/μ vectors as follows: for each r_i with $i \in [\mu]$, randomly select $N/\mu - 1$ numbers $r_{i,1}, \ldots, r_{i,N/\mu-1}$, and set the final number

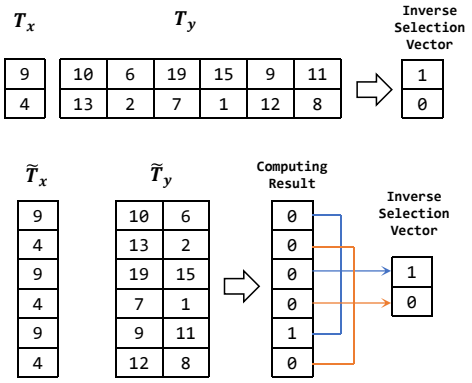

Fig. 9: The high-level idea of our optimization for small sender set.

$r_{i,N/\mu} = r_i - \sum_{j=1}^{N/\mu-1} r_{i,j}$.

Receiver $\mathcal{R}$ then masks the encrypted vector $[\![\tilde{\mathbf{o}}]\!]_{\mathsf{pk}_s}$ by adding corresponding randomness values $r_{i,j}$ to position $(j-1)\cdot\mu+i$, obtaining masked ciphertexts $[\![\tilde{\mathbf{c}}]\!]_{\mathsf{pk}_s}$, and sends these along with the encrypted shuffled shares to sender $\mathcal{S}$.

Upon receiving $[\![\tilde{\mathbf{c}}]\!]_{\mathsf{pk}_s}$, sender $\mathcal{S}$ decrypts them and computes $c_i = \sum_{j=1}^{N/\mu} \tilde{c}_{(j-1)\cdot\mu+i}$, for $i \in [\mu]$. This recovers exactly the original μ bins of the BCS protocol, yielding $c_i = o_i + r_i$ for each bin. Finally, sender $\mathcal{S}$ obtains the permuted inverse selection vector $[\![\tilde{\mathbf{b}}^\pi]\!]_{\mathsf{pk}_r}$ via BCS, and hence it must perform $1 - [\![\tilde{\mathbf{b}}^\pi]\!]_{\mathsf{pk}_r}$ to produce the final encrypted permuted selection vector $[\![\mathbf{b}^\pi]\!]_{\mathsf{pk}_r}$.

In addition, if the number of bins is significantly smaller than the parameter N, many plaintext slots in each ciphertext may remain unused in Step 6. Recall that $\mathcal{S}$ must split each item into $\log_2 t$-bit segments and place them into separate ciphertexts. To fully utilize all available slots, both $\mathcal{R}$ and $\mathcal{S}$ can simply duplicate their shuffled shares and masked selection vectors N/μ times, thereby creating an extended input that matches the slot capacity. As a result, the leftover slots are employed effectively, reducing the total number of ciphertexts need to be sent.

V. Optimization of Equality Operator

As mentioned earlier, the majority of computational overhead in our cwPSU protocol arises from the Arithmetic Constant-weight Equality Operator. Specifically, the protocol must compare two constant-weight codes—each of bit-length l and Hamming weight h—to determine whether they are equal. Since one encoding is in ciphertext form and the other is in plaintext, computing $[\![k]\!]$ in Algorithm 1 only requires plaintext-ciphertext multiplications, which incur negligible cost compared to ciphertext-ciphertext multiplications. By contrast, computing $[\![e]\!]$ requires multiplying h ciphertexts because $[\![k-i]\!]$ is itself a ciphertext. Hence, it incurs $h-1$ ciphertext-ciphertext multiplications with a multiplicative depth on the order of $\sqrt{h}$. To address this, we propose a new Arithmetic Constant-weight Equality Operator, referred to as EEQ, detailed in Algorithm 2.

Algorithm 2 Efficient Arithmetic Constant-weight Equality Operator EEQ$(x, [\![y]\!])$.

Input: $x, [\![y]\!] \in CW(l,h), L, H, \{a_i\}_{i\in[h]}$
1: $[\![k]\!] = \sum_{i\in[l]} x[i] \cdot [\![y[i]]\!]$
2: $z = \lfloor h/2 + 1 \rfloor$
3: $[\![k']\!] = [\![k]\!] - z$
4: Get $[\![k']\!]^2, \cdots, [\![k']\!]^{2L}$ and $[\![k']\!]^{4L}, \cdots, [\![k']\!]^{(H-1)\cdot 2L}$
5: $e = \sum_{i=0}^{H-1} \left([\![k]\!]^{2L}\right)^i \left(\sum_{j=0}^{L-1} a_j \left([\![k]\!]^2\right)^{j+1}\right)$
Output: $e \in \{0,1\}$

First, we analyze the function

$$e = \frac{1}{h!} \prod_{i\in[h]} (k-i),$$

whose core objective is to compute a polynomial

$$f(k) = \begin{cases} 0, & 0 \le k < h, \\ 1, & k = h. \end{cases}$$

Essentially, f is a Lagrange interpolation polynomial that takes the value 0 for all integers from 0 to $h-1$, and 1 at $k=h$. By letting $[\![k]\!] - z$, where $z = \lfloor h/2+1 \rfloor$, center the domain symmetrically around 0 for the first h points, we can rewrite $f(k)$ in the form

$$f(k) = \begin{cases} 0, & \lceil -(h-1)/2 \rceil \le k < \lfloor (h-1)/2 \rfloor, \\ 1, & k = \lfloor (h-1)/2 \rfloor, \end{cases}$$

leading to an equivalent expression:

$$f(k) = \prod_{i=0}^{z} \frac{k^2 - i^2}{h^2 - i^2}.$$

When $[\![k]\!]$ is a ciphertext, this approach reduces the number of ciphertext-ciphertext multiplications from h to $z+1$, effectively halving the overall multiplication cost.

To reduce the multiplicative complexity even more, we employ the Paterson-Stockmeyer algorithm [50]. By expanding

$$f(k) = \prod_{i=0}^{z} \frac{k^2 - i^2}{h^2 - i^2} = \sum_{i=0}^{z} a_i k^{2(i+1)},$$

we can select positive integers L and H to organize powers of $[\![k]\!]$, depending on the Hamming weight h. Concretely, we first compute the low powers $[\![k]\!]^2, [\![k]\!]^4, \ldots, [\![k]\!]^{2L}$ and then the high powers $[\![k]\!]^{4L}, [\![k]\!]^{8L}, \ldots, [\![k]\!]^{(H-1)\cdot 2L}$. Accordingly, the polynomial can be rewritten as

$$f([\![k]\!]) = \sum_{i=0}^{H-1} \left([\![k]\!]^{2L}\right)^i \left(\sum_{j=0}^{L-1} a_j \left([\![k]\!]^2\right)^{j+1}\right).$$

Here, the inner sums rely on scalar multiplications and additions of the low powers, while ciphertext-ciphertext multiplications are only required for multiplying these partial sums by the high powers. Table I provides optimal choices of L and H for different values of h.

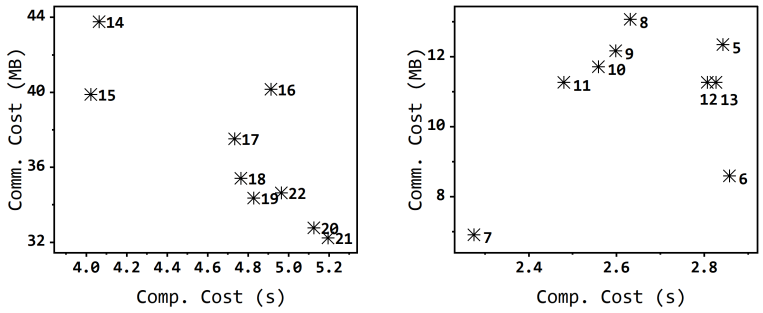

Fig. 10: Communication (MB) and computation (s) overhead under different Hamming weights. The experiments are conducted with $m = 2^{10}$ and $n = 2^{20}$. The left figure reports results for 128-bit items with Hamming weights from 14 to 22, while the right figure shows results for 32-bit items with Hamming weights from 5 to 13.

h	7	8–11	12–15	16–19	20–23
L	2	3	4	5	6
H	2	2	2	2	2
Mult.	3	4	5	6	7

TABLE I: Optimal choices of L and H and required non-scalar multiplications for different h

Our new EEQ operator significantly reduces the number of non-scalar multiplications compared to the original cwEQ algorithm, which requires $h - 1$ such multiplications. By exploiting symmetry in the polynomial expression, we halve the multiplicative cost, and by applying the Paterson–Stockmeyer algorithm, we further minimize the remaining multiplications.

VI. Implementation and Performance

In this section, we experimentally evaluate our cwPSU protocol and EEQ algorithm.

A. Implementation Detail

We implemented our cwPSU protocol in C++ using the Microsoft SEAL library, and we compared its performance against the state-of-the-art protocols [13]. Our measurements focused on both runtime and communication overhead. We set the computational security parameter $\kappa = 128$ and the statistical security parameter $\lambda = 40$. Similar to [12], [13], we realized the Permute + Share functionality following the construction in [43].

We used the Linux `tc` command to emulate different network conditions. Specifically, we considered a LAN environment featuring local connections at 10 Gbps with a round-trip time (RTT) of 0.2 ms. We also evaluated WAN settings with bandwidths of 100 Mbps, 10 Mbps, and 1 Mbps, each with an RTT of 80 ms.

We set the homomorphic encryption parameters to $\log_2 N = 14$, $\log_2 t = 20$, and $\log_2 q = 256$ for 128-bit items, and to $\log_2 N = 13$, $\log_2 t = 20$, and $\log_2 q = 204$ for 32-bit items, with further adjustments made when necessary. As shown in Figure 10, our benchmarking results reveal that when the effective bit-length is $\sigma = 128$, the best balance between communication and computation overhead is achieved at a Hamming weight of $h = 19$. For 32-bit items, the optimal trade-off occurs at $h = 7$.

Following [13], we employ a preprocessing strategy that divides the protocol execution into a one-time setup phase and an online phase. Precomputation is a well-established concept in secure multiparty computation [51]–[53], and is commonly assumed in existing unbalanced PSO protocols [13], [36], [37]. Most of these works allow the sender to anticipate its input set and perform any necessary precomputation based on the set contents before entering the online phase [36], [37]. Likewise, in some PIR schemes, the client can download a digest of the database during the setup phase, allowing a single-server PIR protocol to achieve performance comparable to the two-server counterpart [45], [54]. In our protocol, we make a similar assumption: the sender's input set is known in advance during the setup phase. The sender then generates a digest of its set, specifically encryptions of its elements, and transmits this digest to the receiver.

B. cwPSU Evaluation

Table II presents a detailed benchmark of the setup and online phases, contrasting cwPSU with two protocols from [13], namely $\mathrm{ZLP}^{\mathrm{ps}}_{\mathrm{op}}$ and $\mathrm{ZLP}_{\mathrm{pke}}$. The evaluation is conducted over small set sizes $m \in \{2^8, 2^{10}, 2^{12}\}$, large set sizes $n \in \{2^{18}, 2^{20}, 2^{22}\}$, with efficient bit-length $\sigma = 128$. In addition, Figure 11 includes another protocol from [13], $\mathrm{ZLP}^{\mathrm{ddh}}_{\mathrm{op}}$, for further comparison and illustrates how communication and runtime change with the small set size or varying network bandwidth.

Communication comparison. As shown in Table II and Figure 11, our cwPSU reduces communication overhead by a factor of 5.1–32.4× compared with [13]. Notably, the communication cost of cwPSU remains the same for a fixed small set size m, irrespective of the large set size n. This is because, in the online phase, cwPSU only transmits the selection vector from the receiver and the ciphertext of items in $X \setminus Y$ from the sender, leading to communication complexity linear in m and independent of n.

Param.		Protocol	Comm. (MB)		Running Time (s)							
			Setup	Online	LAN		100Mbps		10Mbps		1Mbps	
n	m				Setup	Online	Setup	Online	Setup	Online	Setup	Online
2^{18}	2^8	$\text{ZLP}^{\text{ps}}_{\text{op}}$	18.01	1.35	45.68	1.23	49.71	2.07	62.99	4.15	200.27	14.41
		ZLP_{pke}	17.99	1.47	119.15	2.83	123.43	3.83	195.27	5.07	272.08	15.70
		Ours	32.28	0.26	3.40	1.42	7.58	1.97	31.42	2.03	268.84	4.70
	2^{10}	$\text{ZLP}^{\text{ps}}_{\text{op}}$	18.01	5.39	46.84	1.46	50.22	4.32	64.48	8.91	201.83	49.31
		ZLP_{pke}	17.99	3.32	268.69	3.93	273.56	6.01	287.18	8.80	421.73	33.41
		Ours	32.53	0.35	3.35	1.69	7.09	2.26	32.12	2.51	271.73	6.09
	2^{12}	$\text{ZLP}^{\text{ps}}_{\text{op}}$	18.01	11.56	95.81	4.41	100.09	6.38	112.77	19.32	249.15	98.87
		ZLP_{pke}	17.99	12.35	386.50	6.74	389.95	8.66	404.71	18.81	539.81	107.52
		Ours	36.01	0.66	3.30	2.12	7.52	2.68	34.69	2.95	299.01	8.53
2^{20}	2^8	$\text{ZLP}^{\text{ps}}_{\text{op}}$	18.01	2.60	85.99	1.53	88.63	2.75	102.58	5.10	239.15	24.10
		ZLP_{pke}	17.99	2.63	175.77	4.08	179.57	5.68	193.44	7.76	328.49	27.37
		Ours	34.54	0.26	3.39	4.94	7.79	5.43	33.74	5.79	286.96	8.43
	2^{10}	$\text{ZLP}^{\text{ps}}_{\text{op}}$	18.01	5.47	127.12	1.93	130.43	4.09	143.69	7.98	281.32	47.87
		ZLP_{pke}	17.99	5.88	302.34	4.08	306.03	5.84	319.21	9.97	455.49	53.22
		Ours	33.72	0.35	3.38	4.94	7.88	5.37	32.89	5.85	278.51	8.77
	2^{12}	$\text{ZLP}^{\text{ps}}_{\text{op}}$	18.01	20.05	128.26	4.37	132.01	7.59	145.76	22.56	281.88	166.76
		ZLP_{pke}	17.99	12.50	670.95	6.76	673.24	8.98	688.26	18.70	824.14	108.92
		Ours	37.77	0.66	3.28	5.91	7.83	6.34	36.40	6.79	312.52	12.47
2^{22}	2^8	$\text{ZLP}^{\text{ps}}_{\text{op}}$	18.01	5.20	331.33	2.55	334.28	4.25	348.35	8.80	482.52	48.31
		ZLP_{pke}	17.99	3.21	643.55	13.30	646.30	14.80	660.66	18.24	795.69	40.63
		Ours	36.67	0.26	3.39	17.50	7.90	18.09	35.48	18.39	304.68	20.56
	2^{10}	$\text{ZLP}^{\text{ps}}_{\text{op}}$	18.01	9.88	422.71	3.00	426.33	5.01	440.25	12.35	575.44	83.99
		ZLP_{pke}	17.99	9.92	692.82	6.93	696.41	8.77	709.83	17.02	846.73	88.70
		Ours	35.95	0.35	3.36	17.90	7.57	18.36	34.86	18.79	298.28	21.91
	2^{12}	$\text{ZLP}^{\text{ps}}_{\text{op}}$	18.01	20.38	614.10	5.16	617.95	8.20	618.68	23.18	766.72	170.57
		ZLP_{pke}	17.99	21.44	1440.05	9.76	1443.32	12.55	1457.33	29.01	1593.03	183.68
		Ours	38.84	0.66	3.32	19.66	7.56	20.23	37.20	20.68	322.12	25.79

TABLE II: Communication cost (in MB) and running time (in seconds) comparing our protocols to $\text{ZLP}^{\text{ps}}_{\text{op}}$ and ZLP_{pke} [13]. The LAN network has 10 Gbps bandwidth and 0.05 ms RTT latency. The best results are marked in cyan.

Running time comparison. Our cwPSU outperforms [13] by achieving a 3.1–13.3× speedup in online runtime under various network conditions. In a LAN environment, while cwPSU is not always faster for every parameter set, certain scenarios (e.g., $m = 2^{12}, n = 2^{18}$) show a runtime of only 2.12 seconds for cwPSU, compared to 4.41 seconds for $\text{ZLP}^{\text{ps}}_{\text{op}}$ (a 2× improvement) and 6.74 seconds for ZLP_{pke} (a 3.1× improvement). Moreover, in LAN settings, our setup phase completes more than two orders of magnitude faster than that of [13]. Under low-bandwidth conditions, the advantage of our lower communication overhead becomes even more pronounced. For instance, with $m = 2^{12}$ and $n = 2^{20}$ at 1 Mbps bandwidth, cwPSU requires just 12.47s , whereas $\text{ZLP}^{\text{ps}}_{\text{op}}$ and ZLP_{pke} take 166.76s and 108.92s, respectively, yielding improvements of approximately 13.3× and 8.7×.

C. EEQ Evaluation

To evaluate how much our EEQ operator improves overall protocol performance compared to the original cwEQ, we measured the time required to compute the selection vector in cwPSU, as well as the resulting ciphertext size for small sets of items at 32-bit length ($h = 7$) and 128-bit length ($h = 19$). As shown in Table III, although the number of non-scalar multiplications is reduced to approximately one-third of its original value, the additional scalar multiplications and unchanged multiplicative depth entail a slightly larger homomorphic parameter configuration for EEQ than cwEQ. Nevertheless, this overhead is offset by a roughly 2 times improvement in runtime, while the increase in ciphertext size remains modest.

D. Scalability

To further evaluate scalability, we tested a larger configuration with a receiver's set of size 2^{24} and a sender's set of size 2^{10}. In this setting, the setup and online communication costs were 38.18 MB and 0.35 MB, respectively, while the LAN execution times were 3.37 s for setup and 70.96 s for the online phase. These results highlight the strong scalability of our protocol in terms of communication, as the online cost remains independent of the large set size. Although computation time naturally increases with input size, the protocol's

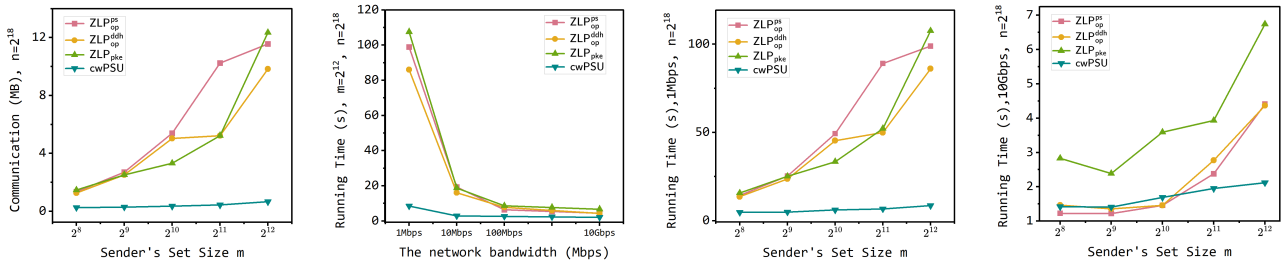

Fig. 11: Communication cost (in MB) and running time (in seconds) comparing our protocols to $\text{ZLP}^{\text{ps}}_{\text{op}}$, $\text{ZLP}^{\text{ddh}}_{\text{op}}$ and ZLP_{pke} [13]. The first figure shows the communication cost increases as the small set size increases. The second figure shows the runtime decreases as the bandwidth increases. The third and fourth figures show the runtime increases as the small set size increases in different bandwidths (i.e., 1 Mbps and 10 Gbps).

Param.		Operator	32 bit		128 bit	
n	m		Time (s)	ct Size (MB)	Time (s)	ct Size (MB)
2^{18}	2^{10}	cwEQ	0.88	4.46	2.44	26.13
		EEQ	0.48	4.82	1.35	27.63
	2^{12}	cwEQ	0.86	4.46	2.35	26.13
		EEQ	0.49	4.84	1.34	27.63
2^{20}	2^{10}	cwEQ	2.20	4.47	6.84	28.38
		EEQ	1.29	4.84	4.03	29.76
	2^{12}	cwEQ	2.24	4.47	6.80	28.38
		EEQ	1.31	4.84	3.86	29.76
2^{22}	2^{10}	cwEQ	6.84	4.47	19.80	30.64
		EEQ	4.10	4.84	11.19	31.52
	2^{12}	cwEQ	6.75	4.47	19.88	30.64
		EEQ	4.10	4.85	11.21	31.52
2^{24}	2^{10}	cwEQ	24.38	4.47	67.61	31.54
		EEQ	14.33	4.86	38.51	32.59
	2^{12}	cwEQ	23.79	4.49	67.59	31.54
		EEQ	14.21	4.86	38.08	32.59

TABLE III: Ciphertext size (in MB) and the time (in seconds) of selection vector computing comparing our EEQ to cwEQ [47].

communication efficiency makes it particularly well-suited for low-bandwidth environments.

VII. Conclusion

In this paper, we proposed cwPSU, a novel unbalanced PSU protocol that achieves low communication complexity and round efficiency by leveraging constant-weight encoding and leveled fully homomorphic encryption. Our protocol achieves communication cost that is linear in the size of the smaller set and independent of the larger set, while requiring only a single round of online communication. Experimental results demonstrate that cwPSU significantly outperforms the state-of-the-art protocols in both communication and runtime under various network settings. Future directions include reducing the ciphertext size transmitted during the setup phase, as well as extending the design to support multi-party settings and other private set operations.

Acknowledgment

We thank all anonymous reviewers for their helpful feedback. This research is supported in part by the Special Funds of the National Natural Science Foundation of China (62441226), National Natural Science Foundation of China (62572020, 62402363), Young Elite Scientists Sponsorship Program by CAST (2023QNRC001), Shaanxi Province Postdoctoral Research Funding Program (2024BSHSDZZ106), Fundamental Research Funds for the Central Universities (XJSJ24066). Hui Li is the corresponding author.

References

[1] K. Hogan, N. Luther, N. Schear, E. Shen, D. Stott, S. Yakoubov, and A. Yerukhimovich, "Secure multiparty computation for cooperative cyber risk assessment," in *2016 IEEE Cybersecurity Development (SecDev)*. IEEE, 2016, pp. 75–76.

[2] D. Landoll, *The security risk assessment handbook: A complete guide for performing security risk assessments*. CRC press, 2021.

[3] A. Shameli-Sendi, R. Aghababaei-Barzegar, and M. Cheriet, "Taxonomy of information security risk assessment (isra)," *Computers & security*, vol. 57, pp. 14–30, 2016.

[4] J. West and M. Bhattacharya, "Intelligent financial fraud detection: a comprehensive review," *Computers & security*, vol. 57, pp. 47–66, 2016.

[5] H. Li, Y. Zhu, and Y. Niu, "Contact tracing research: a literature review based on scientific collaboration network," *International Journal of Environmental Research and Public Health*, vol. 19, no. 15, p. 9311, 2022.

[6] E. Osmanlliu, E. Rafie, S. Bédard, J. Paquette, G. Gore, M.-P. Pomey *et al.*, "Considerations for the design and implementation of covid-19 contact tracing apps: scoping review," *JMIR mHealth and uHealth*, vol. 9, no. 6, p. e27102, 2021.

[7] T. Yum, T. Zhou, Q. Ye, H. Peng, and J. Chen, "Cross-institution online problem based learning in chinese medicine education," in *Asia Pacific Conference on Advanced Research, APCAR 2016 Proceedings*, 2016.

[8] V. Kolesnikov, M. Rosulek, N. Trieu, and X. Wang, "Scalable private set union from symmetric-key techniques," in *International conference on the theory and application of cryptology and information security*. Springer, 2019, pp. 636–666.

[9] G. Garimella, P. Mohassel, M. Rosulek, S. Sadeghian, and J. Singh, "Private set operations from oblivious switching," in *IACR international conference on public-key cryptography*. Springer, 2021, pp. 591–617.

[10] Y. Jia, S.-F. Sun, H.-S. Zhou, J. Du, and D. Gu, "Shuffle-based private set union: Faster and more secure," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 2947–2964.

[11] C. Zhang, Y. Chen, W. Liu, M. Zhang, and D. Lin, "Linear private set union from {Multi-Query} reverse private membership test," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 337–354.

[12] B. Tu, Y. Chen, Q. Liu, and C. Zhang, "Fast unbalanced private set union from fully homomorphic encryption," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 2959–2973.

[13] C. Zhang, Y. Chen, W. Liu, L. Peng, M. Hao, A. Wang, and X. Wang, "Unbalanced private set union with reduced computation and communication," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 1434–1447.

[14] S. Ramanathan, J. Mirkovic, and M. Yu, "Blag: Improving the accuracy of blacklists," in *NDSS*, 2020.

[15] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.

[16] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.

[17] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in cryptology–ASIACRYPT 2017: 23rd international conference on the theory and applications of cryptology and information security, Hong kong, China, December 3-7, 2017, proceedings, part i 23*. Springer, 2017, pp. 409–437.

[18] G. Garimella, B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "Oblivious key-value stores and amplification for private set intersection," in *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part II 41*. Springer, 2021, pp. 395–425.

[19] M. Hao, W. Liu, L. Peng, H. Li, C. Zhang, H. Chen, and T. Zhang, "Unbalanced {Circuit-PSI} from oblivious {Key-Value} retrieval," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 6435–6451.

[20] A. Bienstock, S. Patel, J. Y. Seo, and K. Yeo, "{Near-Optimal} oblivious {Key-Value} stores for efficient {PSI},{PSU} and {Volume-Hiding}{Multi-Maps}," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 301–318.

[21] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold, "Keyword search and oblivious pseudorandom functions," in *Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings 2*. Springer, 2005, pp. 303–324.

[22] C. Meadows, "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, 1986, pp. 134–134.

[23] B. A. Huberman, M. Franklin, and T. Hogg, "Enhancing privacy and trust in electronic communities," in *Proceedings of the 1st ACM conference on Electronic commerce*, 1999, pp. 78–86.

[24] E. De Cristofaro, J. Kim, and G. Tsudik, "Linear-complexity private set intersection protocols secure in malicious model," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2010, pp. 213–231.

[25] M. Orrù, E. Orsini, and P. Scholl, "Actively secure 1-out-of-n ot extension with application to private set intersection," in *Topics in Cryptology–CT-RSA 2017: The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14–17, 2017, Proceedings*. Springer, 2017, pp. 381–396.

[26] P. Benny, S. Thomas, and Z. Michael, "Faster private set intersection based on ot extension," in *Usenix security*, vol. 14, 2014, pp. 797–812.

[27] B. Pinkas, T. Schneider, G. Segev, and M. Zohner, "Phasing: Private set intersection using permutation-based hashing," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 515–530.

[28] P. Rindal and M. Rosulek, "Malicious-secure private set intersection via dual execution," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1229–1242.

[29] M. Chase and P. Miao, "Private set intersection in the internet setting from lightweight oblivious prf," in *Annual International Cryptology Conference*. Springer, 2020, pp. 34–63.

[30] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient batched oblivious prf with applications to private set intersection," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 818–829.

[31] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "Spot-light: lightweight private set intersection from sparse ot extension," in *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*. Springer, 2019, pp. 401–431.

[32] ——, "Psi from paxos: Fast, malicious private set intersection," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2020, pp. 739–767.

[33] B. Tu, X. Zhang, Y. Bai, and Y. Chen, "Fast unbalanced private computing on (labeled) set intersection with cardinality," *Cryptology ePrint Archive*, 2023.

[34] M. Wu and T. H. Yuen, "Efficient unbalanced private set intersection cardinality and user-friendly privacy-preserving contact tracing," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 283–300.

[35] Y. Son and J. Jeong, "Psi with computation or circuit-psi for unbalanced sets from homomorphic encryption," in *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*, 2023, pp. 342–356.

[36] K. Cong, R. C. Moreno, M. B. da Gama, W. Dai, I. Iliashenko, K. Laine, and M. Rosenberg, "Labeled psi from homomorphic encryption with reduced computation and communication," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1135–1150.

[37] H. Chen, Z. Huang, K. Laine, and P. Rindal, "Labeled psi from fully homomorphic encryption with malicious security," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1223–1237.

[38] A. Davidson and C. Cid, "An efficient toolkit for computing private set operations," in *Information Security and Privacy: 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3–5, 2017, Proceedings, Part II 22*. Springer, 2017, pp. 261–278.

[39] C. Hazay and K. Nissim, "Efficient set operations in the presence of malicious adversaries," in *International Workshop on Public Key Cryptography*. Springer, 2010, pp. 312–331.

[40] L. Kissner and D. Song, "Privacy-preserving set operations," in *Annual International Cryptology Conference*. Springer, 2005, pp. 241–257.

[41] Y. Chen, M. Zhang, C. Zhang, M. Dong, and W. Liu, "Private set operations from multi-query reverse private membership test," in *IACR international conference on public-key cryptography*. Springer, 2024, pp. 387–416.

[42] M. Chase, E. Ghosh, and O. Poburinnaya, "Secret-shared shuffle," in *Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part III 26*. Springer, 2020, pp. 342–372.

[43] P. Mohassel and S. Sadeghian, "How to hide circuits in mpc an efficient framework for private function evaluation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2013, pp. 557–574.

[44] R. A. Mahdavi, N. Lukas, F. Ebrahimianghazani, T. Humphries, B. Kacsmar, J. Premkumar, X. Li, S. Oya, E. Amjadian, and F. Kerschbaum, "{PEPSI}: Practically efficient private set intersection in the unbalanced setting," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 6453–6470.

[45] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, and V. Vaikuntanathan, "One server for the price of two: Simple and fast {Single-Server} private information retrieval," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 3889–3905.

[46] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29*. Springer, 2010, pp. 1–23.

[47] R. A. Mahdavi and F. Kerschbaum, "Constant-weight {PIR}: Single-round keyword {PIR} via constant-weight equality operators," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 1723–1740.

[48] J. Liu, J. Li, D. Wu, and K. Ren, "Pirana: Faster multi-query pir via constant-weight codes," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 4315–4330.

[49] M. Raab and A. Steger, ""balls into bins"—a simple and tight analysis," in *International Workshop on Randomization and Approximation Techniques in Computer Science*. Springer, 1998, pp. 159–170.

[50] M. S. Paterson and L. J. Stockmeyer, "On the number of nonscalar multiplications necessary to evaluate polynomials," *SIAM Journal on Computing*, vol. 2, no. 1, pp. 60–66, 1973.
[51] F. Kerschbaum, E.-O. Blass, and R. A. Mahdavi, "Faster secure comparisons with offline phase for efficient private set intersection," *arXiv preprint arXiv:2209.13913*, 2022.
[52] Á. Kiss, J. Liu, T. Schneider, N. Asokan, and B. Pinkas, "Private set intersection for unequal set sizes with mobile applications," in *Privacy Enhancing Technologies Symposium*. De Gruyter, 2017, pp. 177–197.
[53] P. Rindal and P. Schoppmann, "Vole-psi: fast oprf and circuit-psi from vector-ole," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2021, pp. 901–930.
[54] M. Zhou, A. Park, W. Zheng, and E. Shi, "Piano: extremely simple, single-server pir with sublinear server computation," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 4296–4314.