MinBucket MPSI: Breaking the Max-Size Bottleneck in Multi-Party Private Set Intersection

Binbin Tu*†, Boyudong Zhu*†, Yang Cao*† and Yu Chen*†‡(✉)
*School of Cyber Science and Technology, Shandong University, Qingdao 266237, China
†State Key Laboratory of Cryptography and Digital Economy Security, Shandong University, Qingdao 266237, China
‡State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China
Email: {tubinbin,yuchen}@sdu.edu.cn, {boyudongzhu,caoyang24}@mail.sdu.edu.cn

***Abstract*—Multi-Party Private Set Intersection (Cardinality) protocol enables T $(T > 2)$ parties, each holding a private set, to jointly compute the intersection (or its cardinality) without revealing any additional information to other parties. To date, all known MPSI (MPSI-Card) protocols require communication complexity that scales linearly with the size of the large set, fundamentally precluding their efficient deployment in real-world applications with heterogeneous input scales.**

In this work, we present a new framework for MPSI based on newly proposed protocols: batched membership conditional randomness generation and joint private equality test. By instantiating this framework, we develop two MPSI protocols with communication complexities that are linear in the size of the small set and logarithmic in the size of the large set. One protocol offers security against an arbitrary number of colluding parties, while the other secures against $(T-2)$ colluding parties. Additionally, we develop a protocol called the joint permuted private equality test and propose the MPSI-Card framework. By instantiating this framework, we derive an MPSI-Card protocol with similar communication efficiency: linear in the small set and logarithmic in the large set, providing security against an arbitrary number of colluding parties.

We implement our protocols and conduct extensive experiments over both LAN and WAN networks. Experimental results demonstrate that our protocols achieve significantly better performance as the size difference between the sets or the number of participants holding the small set increases. For the setting, where 5 parties holding large set (size 2^{20}) and 5 parties holding small set (size 2^{10}) with a single thread and a 10 Mbps bandwidth, our MPSI (MPSI-Card) protocol requires only 12.2 (12.2) MB of communication and 129.86 (130.05) seconds of runtime. Compared with the state-of-the-art MPSI by Wu et al. (USENIX Security 2024) and MPSI-Card by Gao et al. (PETS 2024), our protocol achieves a $157\times$ $(76\times)$ reduction in communication cost and a $12.7\times$ $(3.1\times)$ speedup in runtime.

I. Introduction

Private Set Intersection (PSI) enables a group of mutually distrustful parties, each holding a private set, to compute the intersection of their sets without revealing any information beyond the intersection itself. PSI and its variants have been widely applied in various domains, such as evaluating the effectiveness of online advertising campaigns [1], [2], facilitating private contact discovery [3], [4], and more. Over the last decade, there has been a significant amount of work on two-party private set intersection (PSI), including both balanced case [5], [6], [7], [8], [9], [10], [11], [12], [13], [14] and unbalanced case [15], [16], [17], [18], [19]. In the balanced setting, numerous PSI protocols achieve linear complexity, and the current state-of-the-art (SOTA) PSI [20] is almost as efficient as the naive insecure hash-based protocol. In the unbalanced setting, several protocols [15], [17], [19] have broken the linear communication barrier of the large set, achieving logarithmic complexity in the size of the large set and linear complexity in the size of the small set.

Although two-party PSI is valuable for numerous applications, many real-world scenarios, including medical data integration [21], [22], cache sharing in edge computing [23], network intrusion detection [24], and identification of high-risk individuals during disease outbreaks [25], are better addressed in the multi-party setting. As noted in [26], approximately 176 million blacklisted IP addresses were aggregated across 23,483 autonomous systems, with list sizes varying sharply: some exceeding 500,000 entries, while others contain under 1,000. Thus, computing intersection (cardinality) among multiple IP blacklists exhibiting large size disparities is a real-world scenario for unbalanced MPSI/MPSI-Card. However, existing multi-party private set intersection (MPSI) and MPSI cardinality (MPSI-Card) protocols [27], [28], [29], [30], [31] are primarily designed for balanced scenarios, where the input set sizes of the participants are relatively similar. These protocols struggle to handle common unbalanced scenarios in real-world applications efficiently. For instance, some participants may be mobile devices with limited resources (e.g., battery, computing power, storage) and small datasets, while others are high-performance servers with large datasets. Furthermore, the available bandwidth between the parties may also be limited, further complicating protocol performance in such settings. Yang et al. [32] first propose an unbalanced quorum PSI protocol, but their construction is limited to scenarios where only a single client holds the small set. To the best of our knowledge, all existing MPSI/MPSI-Card protocols require communication complexity that scales at least linearly with the size of the largest set, fundamentally limiting their efficient deployment in real-world applications with heterogeneous

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.2026.240182
www.ndss-symposium.org

input sizes. Motivated by the above discussion, we ask the following questions.

Is it possible to design efficient MPSI and MPSI-Card protocols with communication complexity that breaks the linear bound of the largest set, achieving logarithmic communication in the size of the largest set?

A. Our Contribution

In this paper, we give an affirmative answer to the above questions through the following results.

New frameworks of MPSI/MPSI-Card. We begin by formalizing two ideal functionalities: batched Membership Conditional Randomness Generation (bMCRG) and Joint Private Equality Test (J-PEQT). Building upon these, we introduce a novel MPSI framework based on bMCRG and J-PEQT. Then, we extend J-PEQT and develop a new ideal functionality named Joint Permuted Private Equality Test (JP-PEQT). Based on bMCRG and JP-PEQT, we further present an MPSI-Card framework.

Instantiations. We present two generic constructions of bMCRG: One from batched oblivious pseudorandom function (bOPRF) and oblivious key-value store (OKVS) in the balanced setting, and the other from bOPRF and fully homomorphic encryption (FHE) in the unbalanced setting following [17], [19]. Then, we propose two constructions of J-PEQT: The first, based on joint zero secret sharing (JZSS), is secure against $T-2$ colluding parties. The second, based on threshold additive homomorphic encryption (TAHE), is secure against an arbitrary number of colluding parties. Additionally, we provide a construction of JP-PEQT based on threshold additive homomorphic encryption, which is secure against an arbitrary number of colluding parties. In summary, by instantiating our frameworks, we obtain two MPSI protocols and one MPSI-Card protocol, whose communication complexity is *linear* in the size of the small set and *logarithmic* in the large set. Meanwhile, JZSS-based MPSI satisfies the security against $T-2$ colluding parties. TAHE-based MPSI/MPSI-Card enjoys security against an arbitrary number of colluding parties.

Evaluations. We implement and compare our MPSI/MPSI-Card with the SOTA MPSI [29] and MPSI-Card [31].

- **Comparisons of MPSI.** The experimental results show that our protocol achieves a 1.37 - 607.7× reduction in communication cost and a 2.5 - 64.1× speedup in running time, depending on the network environment when the large set size is $\geq 2^{18}$ and the small set size is 2^{10}. In particular, for the setting where 5 parties hold a large set (size 2^{20}) and 5 parties hold a small set (size 2^{10}) with a single thread and a 10 Mbps bandwidth, our protocol requires only 12.2 MB of communication and 129.86 seconds of runtime. Compared with MPSI [29], our MPSI achieves a 157× reduction in communication cost and a 12.7× speedup in runtime.
- **Comparisons of MPSI-Card.** The experimental results show that our protocol achieves a 1.3 - 170.4× reduction in communication cost and a 1.03 - 26.24 × speedup in running time, depending on the network environment. In particular, for the setting where 5 parties hold a large set (size 2^{20}) and 5 parties hold a small set (size 2^{10}) with a single thread and a 10 Mbps bandwidth, our protocol requires only 12.2 MB of communication and 130.05 seconds of runtime. Compared with MPSI-Card [31], our MPSI-Card achieves a 76× reduction in communication cost and a 3.1× speedup in runtime.

Overall, our protocols demonstrate significant advantages in unbalanced multi-party scenarios with limited communication and computation resources, especially when there is a large disparity between set sizes or when more participants hold the small sets.

B. Technical Overview

We provide a technical overview of our MPSI and MPSI-Card depicted in Figure 1. First, we formalize the following ideal functionalities: batched membership conditional randomness generation (bMCRG), joint private equality test (J-PEQT), and joint permuted private equality test (JP-PEQT). Then, we propose frameworks of MPSI from bMCRG and J-PEQT, and MPSI-Card from bMCRG and JP-PEQT. In the instantiations, balanced bMCRG can be derived from bOPRF and OKVS, and unbalanced bMCRG can be constructed from bOPRF and FHE. Furthermore, we build J-PEQT from JZSS or TAHE, and construct JP-PEQT based on TAHE.

1) Core ideas: align-then-compare: The core ideas of the constructions of our MPSI and MPSI-Card could be viewed as two steps "**align-then-compare**":

- **Align**: The participant with the **smallest input set size** is selected as the **Leader** P_1. P_1 performs alignment operations with all other participants P_k, $k \in [2,T]$, respectively, where P_1 inputs $Y^1 = \{y_i^1\}_{i\in[m_1]}$ and P_k inputs $Y^k = \{y_j^k\}_{j\in[m_k]}$: Pairwise executions of the bMCRG functionality are conducted to transform membership relations into equality relations of characteristic values and non-membership relations into non-equality relations of characteristic values.
  - **bMCRG between P_1 and P_k** : For membership: If $y_i^1 \in Y^k$, $i \in [m_1]$, $k \in [2,T]$, P_1 and P_k obtain equal random values $s_i^k = t_i^k$. For non-membership: If $y_i^1 \notin Y^k$, P_1 and P_k obtain unequal random values $s_i^k \neq t_i^k$.

  Therefore, for each item $y_i^1 \in Y^1$, P_1 obtains $T-1$ characteristic values s_i^k, $k \in [2,T]$, and each participant P_k obtains one characteristic value t_i^k. If $y_i \in \bigcap_{k=2}^{T} Y^k$, we have $(\bigwedge_{k=2}^{T}(s_i^k = t_i^k)) = 1$, otherwise, we have $(\bigwedge_{k=2}^{T}(s_i^k = t_i^k)) = 0$.
- **Compare**: All participants P_k, $k \in [T]$, invoke comparison operations and output the intersection to the receiver in MPSI or the cardinality of the intersection to the receiver in MPSI-Card.
  - **J-PEQT**: All participants P_k, $k \in [T]$ input their characteristic values and invoke J-PEQT functionality to let P_1 obtain indication bit-vector $\mathbf{b} = [b_i]_{i\in[m_1]}$, where if

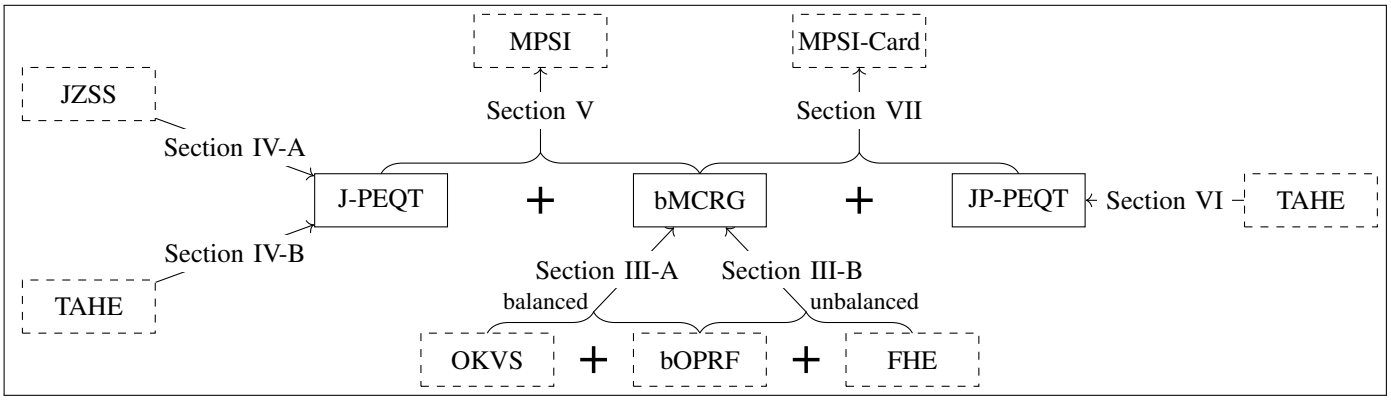

Fig. 1. Technical overview of our frameworks. The rectangle with solid (dotted) lines denotes the new (previous) notions.

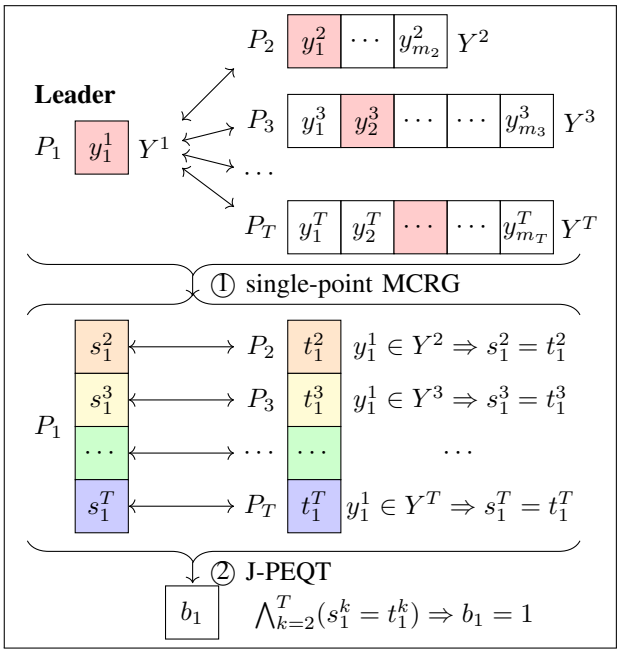

Fig. 2. Illustration of the single-point MPSI from single-point MCRG and J-PEQT. Equal values are shown in the same color; white indicates unequal values.

$\bigwedge_{k=2}^{T}(s_i^k = t_i^k)$, we have $b_i = 1$, otherwise, $b_i = 0$. For $i \in [m_i]$ if $b_i = 1$, P_1 outputs the intersection item y_i^1.

– **JP-PEQT**: All participants P_k, $k \in [T]$ input their characteristic values and invoke JP-PEQT functionality to let P_1 obtain permuted indication bit-vector $\mathbf{b} = [b_i]_{i \in [m_1]}$, where if $\bigwedge_{k=2}^{T}(s_{\pi(i)}^k = t_{\pi(i)}^k)$[1], we have $b_i = 1$, otherwise, $b_i = 0$. P_1 outputs the cardinality of intersection $\sum_{i=1}^{m_1} b_i$.

- **Communication complexity analysis**: In the alignment phase, we select the party with the smallest set as the anchor and perform pairwise alignment operations between this anchor set and the sets of all other participants. Depending on the size of the anchor set, there are two cases: **in the balanced case (where both parties have small sets, like $P_1 \leftrightarrow P_2$ in Figure 2)**, we propose a bMCRG construction with linear communication complexity; **in the unbalanced case (where another party has a large set, like $P_1 \leftrightarrow P_3$ in Figure 2)**, we use a bMCRG construction with communication complexity that is linear with the small set size and logarithmic with the large set size. In the comparison phase, the communication complexity is **linear with the small anchor set size**, as all outputs from the alignment phase depend linearly only on the size of the anchor set. Therefore, the overall communication complexity of our MPSI and MPSI-Card is logarithmic in the size of the large set and linear in the size of the small set.

In summary, our protocols are highly efficient in unbalanced multi-party settings, where M parties hold small sets and N parties hold large sets. In contrast, existing protocols primarily target balanced multi-party scenarios, in which all $T = M+N$ parties possess input sets of comparable sizes. Their communication complexity grows at least linearly with the size of the largest set, making them inefficient for deployment in unbalanced multi-party scenarios.

2) A single-point MPSI: We start with a special case of single-point MPSI depicted in Figure 2. in which P_1 has only one item y_1^1, and each other party P_k has a set $Y^k = \{y_1^k, \cdots, y_{m_k}^k\}$, $k \in [2, T]$.

First, we formalize the functionality of single-point MCRG (two parties): P_1 inputs an item y_1^1 and P_2 inputs a set Y^2, the result is that P_1 and P_2 obtain their random characteristic values s_1^2 and t_1^2, respectively, such that if $y_1^1 \in Y$, $s_1^2 = t_1^2$, otherwise, $s_1^2 \neq t_1^2$. Therefore, P_1 invokes single-point MCRG with each other party $P_k, k \in [2, T]$. The result is that P_1 obtains a random characteristic vector $[s^2, \cdots, s^T]$ and $P_k, k \in [2, T]$ obtains a random characteristic value t^k. If $\bigwedge_{k=2}^{T}(s^k = t^k)$, we have $y_1^1 \in \bigcap_{k=2}^{T} Y^k$. Otherwise, $y_1^1 \notin \bigcap_{k=2}^{T} Y^k$. Then, we formalize the functionality of J-PEQT (T parties): P_1 inputs a vector $[s^2, \cdots, s^T]$ and $P_k, k \in [2, T]$ inputs a value t^k, the result is that P_1 obtains the indicated bit b, such that if $\bigwedge_{k=2}^{T}(s^k = t^k)$, we have $b = 1$, otherwise, $b = 0$. Finally, P_1 outputs the intersection $\{y_1^1\}$ if the output $b = 1$ of J-PEQT, otherwise outputs $\emptyset$. Figure 3 provides an

[1] π over $[m_1]$ is an implicit random permutation

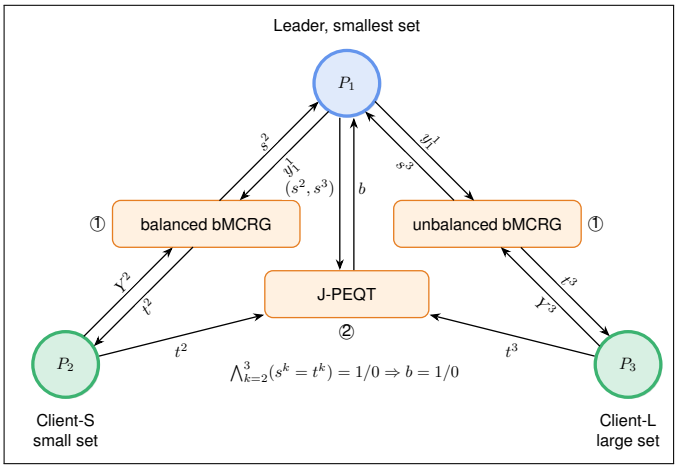

Fig. 3. A single-point MPSI under three parties. Step ① denotes that P_1 interacts with other parties and invokes (unbalanced/balanced) $\mathcal{F}_{\text{bMCRG}}$; Step ② denotes that all parties invoke $\mathcal{F}_{\text{J-PEQT}}$.

illustration of our single-point three-party PSI protocol, where Leader (P_1) holds one item, and Client-S (P_2)/Client-L (P_3) denotes that the parties hold small/large sets.

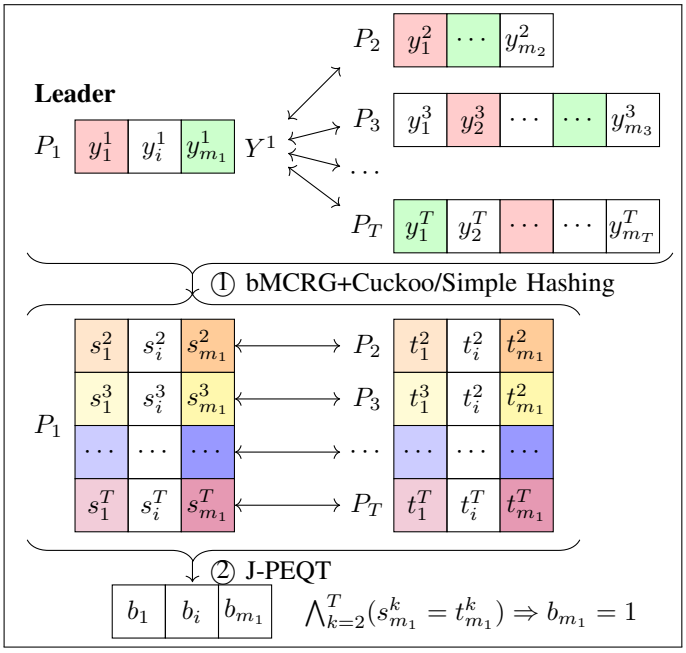

Fig. 4. Illustration of MPSI from bMCRG and J-PEQT, where $m_1 \approx m_2$ represents the balanced case, while $m_1 \ll m_3$ and $m_1 \ll m_T$ represent unbalanced cases. Equal values are shown in the same color; white indicates unequal values.

3) MPSI from bMCRG and J-PEQT: Now, we demonstrate how to extend the special case to a general case depicted in Figure 4. where $|Y^1| = m_1 > 1$. Intuitively, one might execute the above process for each $y_i^1 \in Y^1$ individually. However, this naive approach requires comparing each item y_i^1 to the entire set Y^k, resulting in significant overhead. To mitigate this, we employ the standard hash-to-bin technique to reduce computational costs. Specifically, P_1 assigns each of its items $y_i^1, i \in [m_1]$ (concatenated with the hash function index) to one of the bins $h_1(y_i^1), h_2(y_i^1), h_3(y_i^1)$ using cuckoo hashing [33]. P_k, $k \in [2,T]$ assigns each of its items $y_i^k, i \in [m_k]$ (concatenated with hash function index) to all of the bins $h_1(y_i^k), h_2(y_i^k), h_3(y_i^k)$ by simple hashing. Here, both parties align two input sets Y^1 and Y^k: The same items are inserted into the same bins, and all bins Y_i^k, $i \in [m_c]$ are mutually exclusive. Then, P_1 performs the above single-point MCRG with P_k on each bin. This method greatly reduces the input size of P_k from the entire set $|Y^k|$ to a small hash bin.

We extend the single-point MCRG to the functionality of batched MCRG (two parties: P_1 and P_k), defined in Figure 11. Specifically, P_1 inputs a set $Y^1 = \{y_i^1\}_{i\in[m_1]}$, and P_k inputs m_1 mutually exclusive sets $\{Y_i^k\}_{i\in[m_1]}$[2]. The result is that P_1 and P_k obtain their respective random characteristic vectors $\mathbf{s}^k = [s_i^k]_{i\in[m_1]}$ and $\mathbf{t}^k = [t_i^k]_{i\in[m_1]}$, such that for each $i \in [m_1]$, if $y_i^1 \in Y_i^k$, then $s_i^k = t_i^k$, otherwise, $s_i^k \neq t_i^k$.

We further generalize the J-PEQT functionality for T parties, defined in Figure 14. Specifically, P_1 inputs $T-1$ vectors $\mathbf{s}^k = [s_i^k]_{i\in[m_1]}$, $k \in [2,T]$, and interacts with $T-1$ parties $P_k, k \in [2,T]$, where each party P_k inputs a vector $\mathbf{t}^k = [t_i^k]_{i\in[m_1]}$. The result is that P_1 obtains a random bit string $\mathbf{b} = [b_i]_{i\in[m_1]}$, such that for each $i \in [m_1]$, if $\bigwedge_{k=2}^{T}(s_i^k = t_i^k)$, we have $b_i = 1$; otherwise, $b_i = 0$.

In summary, we obtain an MPSI protocol based on bMCRG and J-PEQT. Specifically, P_1 inserts Y^1 into a cuckoo hash table $Y_c^1[i], i \in [m_c]$, while each $P_k, k \in [2,T]$ inserts Y^k into a simple hash table $\{Y_1^k, \ldots, Y_{m_c}^k\}$ using the same hash functions, resulting in m_c mutually exclusive sets (each bin is treated as a set). Next, P_1 invokes bMCRG with each other party $P_k, k \in [2,T]$, using $Y_c^1[i], i \in [m_c]$ and $\{Y_1^k, \ldots, Y_{m_c}^k\}$ as inputs. As a result, P_1 obtains $T-1$ vectors $\mathbf{s}^k = [s_i^k]_{i\in[m_c]}, k \in [2,T]$, while each $P_k, k \in [2,T]$ obtains a vector $\mathbf{t}^k = [t_i^k]_{i\in[m_c]}$. For each $i \in [m_c]$, if $\bigwedge_{k=2}^{T}(s_i^k = t_i^k)$, $Y_c^1[i]$ belongs to the intersection. Otherwise, $Y_c^1[i]$ does not belong to the intersection. Subsequently, P_1 invokes J-PEQT with each $P_k, k \in [2,T]$, using $\mathbf{s}^k = [s_i^k]_{i\in[m_c]}$ and $\mathbf{t}^k = [t_i^k]_{i\in[m_c]}$ as inputs. As a result, P_1 obtains a bit string $\mathbf{b} = [b_i]_{i\in[m_c]}$. Finally, P_1 outputs the intersection $I = \{Y_c^1[i^*]\}$, where for $i^* \in [m_c]$, $b_{i^*} = 1$.

4) MPSI-Card from bMCRG and JP-PEQT: A single-point MPSI-Card is functionally equivalent to a single-point MPSI. However, in the general case of $|Y^1| = m_1 > 1$, MPSI-Card offers stronger privacy by revealing only the intersection size rather than the intersection itself. Consequently, J-PEQT is not directly suitable for constructing MPSI-Card, as it leaks the indicated bits of the intersection items to P_1, causing the intersection to be revealed. To address this issue, we need to shuffle the indicated bits without P_1 knowing the permutation, ensuring that P_1 only receives a permuted indicated bits. P_1 can compute the Hamming weight of the indicated bit vector to obtain the intersection size.

[2]The reason for the mutual exclusivity of the set is that OKVS requires all the keys to be distinct.

Based on the above observation, we further extend the J-PEQT with permutation, a new functionality named joint permuted private equality test (JP-PEQT), defined in Figure 18. Specifically, P_1 inputs $T-1$ characteristic vectors $\mathbf{s}^k = [s_i^k]_{i\in[m_1]}, k \in [2,T]$ interacts with $T-1$ parties $P_k, k \in [2,T]$ where each party inputs a vector $\mathbf{t}^k = [t_i^k]_{i\in[m_1]}, k \in [2,T]$ and a permutation π_k, $k \in [2,T]$ over $[m_1]$. As a result, JP-PEQT generates a random bit string $\mathbf{b} = [b_i]_{i\in[m_1]}$ to P_1 such that for $i \in [m_1]$, if $\bigwedge_{k=2}^{T}(s_{\pi(i)}^k = t_{\pi(i)}^k)$, we have $b_i = 1$, otherwise, $b_i = 0$, where $\pi = \pi_2 \circ \pi_3 \circ \cdots \circ \pi_T$.

Therefore, we obtain an MPSI-Card protocol based on bMCRG and JP-PEQT. The construction is similar to the MPSI protocol derived from bMCRG and J-PEQT, except for replacing J-PEQT with JP-PEQT, which can shuffle the indicated bit vector, hiding the intersection from P_1.

5) Two kinds of constructions of bMCRG: We focus on the unbalanced multi-party scenario, where during the alignment phase, the party holding the smallest input set engages in pairwise bMCRG executions with all other parties. Depending on the relative sizes between both parties, scenarios can be classified into two cases: (1) Balanced case: The input set sizes of the two parties are similar. (2) Unbalanced case: The input set sizes of the two parties differ significantly.

bMCRG in the balanced setting. The balanced construction of bMCRG is based on bOPRF and OKVS: First, both parties P_1 and P_2 input $(y_1^1, \cdots, y_m^1)$ and mutually exclusive sets $(Y_1^2, \cdots, Y_m^2)$, and then invoke $\mathcal{F}_{\text{bOPRF}}$. The result is that P_1 obtains all PRF values $F(k_i, y_i^1)$ and P_2 obtains all PRF keys $k_1, k_2, \cdots, k_m$. P_2 then computes all PRF values $F(k_i, Y_i^2[j])$ by the PRF key k_i, where $Y_i^2[j]$ denotes the j-th item in the i-th set Y_i^2. Subsequently, P_2 chooses m random characteristic values t_i^2, $i \in [m]$ and encodes all key-value pairs $\{(Y_i^2[j], t_i \oplus F(k_i, Y_i^2[j]))\}_{i\in[m], j\in[|Y_i^2|]}$ into a OKVS data structure D. P_2 sends D to P_1. For each $i \in [m]$, P_1 inputs $F(k_i, y_i^1)$ and runs Decode to output the characteristic values $s_i^2 = F(k_i, y_i^1) \oplus \mathsf{Decode}(y_i^1)$. According to the correctness of bOPRF and OKVS, we obtain a bMCRG in the balanced case: For each $i \in [m]$, if $y_i^1 \in Y_i^2$, we have $s_i^2 = F(k_i, y_i^1) \oplus \mathsf{Decode}(y_i^1) = F(k_i, y_i^1) \oplus t_i^2 \oplus F(k_i, y_i^1) = t_i^2$, otherwise, $s_i^2 \neq t_i^2$. Here, our bMCRG inherits the **linear complexity** from both bOPRF and OKVS, achieving linear communication in the size of Y^1 and $Y_i^k, i \in [m]$.

bMCRG in the unbalanced setting. The unbalanced construction is based on bOPRF and FHE following [15], [17], [19]: First, both parties P_1 and P_2 input $(y_1^1, \cdots, y_m^1)$ and mutually exclusive sets $(Y_1^2, \cdots, Y_m^2)$, respectively. P_2 uses the polynomial randomization method like [17], [34] to encode each set Y_i^2, $i \in [m]$ of P_2, so that $f_i(x) = \Pi_{j=1}^{B_i}(x - Y_i^2[j]) + r_i$, where $B_i = |Y_i^2|$ and r_i is a random value. Next, P_1 sends an FHE ciphertext of encrypting y_i^1, denoted as $[\![y_i^1]\!]$ to P_2. Then, P_2 homomorphically computes and returns $[\![f_i(y_i^1)]\!]$. Finally, P_1 decrypts $[\![f_i(y_i^1)]\!]$ and outputs the characteristic values $s_i^2 = f_i(y_i^1)$. P_2 outputs the characteristic values $t_i^2 = r_i$[3]. As discussed above, we obtain a bMCRG in the unbalanced case: For each $i \in [m]$, if $y_i^1 \in Y_i^2$, we have $f_i(y_i^1) = \Pi_{j=1}^{B_i}(y_i^1 - Y_i^2[j]) + r_i = r_i$, and $s_i^2 = r_i = t_i^2$, otherwise, $s_i^2 \neq t_i^2$. Our unbalanced bMCRG follows the constructions of [15], [17], [19] and achieves logarithmic communication complexity with the large sets size Y_i^k, $i \in [m]$.

Relationship to the constructions of [17], [27]. We formally abstract the construction of [17] as a functionality called bMCRG, which transforms membership relations into equality relations of characteristic values, thereby enabling multiple parties to jointly perform equality comparisons on these values via J-PEQT/JP-PEQT to realize the functionalities of MPSI and MPSI-Card. Conceptually, bMCRG can be regarded as a weak notion of the Programmable Oblivious Pseudorandom Function (OPPRF) [27]: An OPPRF implies a bMCRG; the converse does not hold, as bMCRG does not output the PRF key. In the instantiation, we adapt the constructions of [17] by decoupling the hash-to-bin. This allows bMCRG to inherit various optimization techniques to reduce the depth of the homomorphic circuit in [15], [17], [19]. However, when applying the partitioning optimization, fine-grained processing is required to aggregate multiple pairs of characteristic values into a single pair (details are provided in Section III-B), thereby ensuring compatibility with the subsequent J-PEQT/JP-PEQT protocols.

6) Constructions of J-PEQT and JP-PEQT: Here, we give constructions of joint private equality test (J-PEQT) and joint permuted private equality test (JP-PEQT).

J-PEQT from JZSS. We generate secret shares of zero for all parties using JZSS. Then, each of the $T-1$ parties publishes its input, hiding it with the corresponding zero share, similar to a one-time pad. Finally, P_1 can check whether the sums are equal to determine if all corresponding positions are equal. Since JZSS satisfies security against $T-2$ colluding parties[4], our J-PEQT inherits this security.

The construction of J-PEQT from JZSS is as follows: First, P_1 and $P_k, k \in [2,T]$ encode their all strings $\mathbf{s}^k$ and $\mathbf{t}^k$ into $\mathbb{Z}_q$. Then, $P_k, k \in [T]$ invoke the functionality of JZSS, such that P_1 obtains $\mathbf{e} = [e_i]_{i\in[m]}$ and $P_k, k \in [2,T]$ obtains $\mathbf{d}^k = [d_i^k]_{i\in[m]}$, where for each $i \in [m]$, $e_i + \sum_{k=2}^{T} d^k = 0 \bmod q$. Furthermore, $P_k, k \in [2,T]$ computes and sends $c_i^k = d_i^k + t_i^k \bmod q$ to P_1, and P_1 computes $p_i = \sum_{k=2}^{T} c_i^k + e_i - (\sum_{k=2}^{T} s_i^k) \bmod q$. For each $i \in [m]$, P_1 sets $b_i = 1$, if $p_i = 0$, otherwise, $b_i = 0$. Finally, P_1 outputs the bit vector $\mathbf{b} = [b_i]_{i\in[m]}$.

We observe that $\sum_{k=2}^{T} s_i^k = \sum_{k=2}^{T} t_i^k \bmod q$ can be used to represent $\bigwedge_{k=2}^{T}(s_i^k = t_i^k)$ in the large group $\mathbb{Z}_q$ with q being a prime number of λ-bit length, according to Lemma 1. In other words, for two random vectors $\mathbf{s} = [s_i]_{i\in T}$ and $\mathbf{t} = [t_i]_{i\in T}$

[3]Following [17], [19], we can use a bOPRF to compute the items on both sides before engaging in the bMCRG, which prevents P_1 from learning anything about the original items and allows efficient FHE parameters.

[4]The shared secret is zero, so any $T-1$ parties can reconstruct the share of the other party.

in $\mathbb{Z}_q$, the probability that $\sum_{k=2}^{T} s_i^k = \sum_{k=2}^{T} t_i^k \bmod q$ holds, but $\bigwedge_{k=2}^{T}(s_i^k = t_i^k)$ does not hold, is negligible.

J-PEQT from TAHE. We use TAHE to homomorphically compute $\sum_{k=2}^{T} s_i^k - \sum_{k=2}^{T} t_i^k$ under ciphertexts. Then, all parties decrypt the ciphertext jointly, and P_1 checks whether the plaintext is zero to determine if all corresponding positions are equal. Since TAHE provides security against an arbitrary number of colluding parties, our J-PEQT inherits this security.

The construction of J-PEQT from TAHE is as follows: First, $P_k, k \in [T]$ run $\mathsf{TKeyGen}\ (1^\lambda) \rightarrow (pk, [sk_k]_{k\in[T]})$. $P_k, k \in [T]$ obtain sk_k, respectively. Then, P_k, $k \in [2,T]$ encrypts $\mathbf{t}^k = [t_i^k]_{i\in[m]}$, $k \in [2,T]$: $c_i^k = \mathsf{TEnc}\ (pk, t_i^k)$, and send all ciphertexts to P_1. P_1 computes $\bar{c}_i^1 = (\boxplus_{k=2}^{T} c_i^k) \boxplus \mathsf{TEnc}(pk, \sum_{k=2}^{T}(-s_i^k))$[5], and sends $\bar{c}_i^1$, $i \in [m]$ to P_2. Subsequently, from $k = 2$ to $k = T$, $P_k, k \in [2,T]$ chooses m random values α_i^k, $i \in [m]$ and computes $\bar{c}_i^k = \alpha_i^k \boxtimes \bar{c}_i^{k-1}$, and sends $\bar{c}_i^k$ to P_{k+1}. P_T sends $\bar{c}_i^T$ to P_1 and $P_k, k \in [2, T-1]$. Additionally, P_k, $k \in [2,T]$ decrypts the plaintext share $p_i^k = \mathsf{TDec}\ (sk_k, \bar{c}_i^T)$, and sends the plaintext share to P_1. Finally, P_1 decrypts the plaintext share $p_i^1 = \mathsf{TDec}(sk_1, \bar{c}_i^T)$ and combine the plaintext $p_i = \mathsf{Combine}(p_i^1, p_i^2, \cdots, p_i^T)$. From $i = 1$ to $i = m$, P_1 sets $b_i = 1$, if $p_i = 0$, otherwise, $b_i = 0$. P_1 outputs the bit vector $\mathbf{b} = [b_i]_{i\in[m]}$.

JP-PEQT from TAHE. The construction of JP-PEQT from TAHE is similar to J-PEQT from TAHE, except that after obtaining $\bar{c}_i^1 = (\boxplus_{k=2}^{T} c_i^k) \boxplus \mathsf{TEnc}(pk, \sum_{k=2}^{T}(-s_i^k))$ by P_1, all parties $P_k, k \in [2,T]$ choose random permutation π_k over $[m]$ and shuffle $[\bar{c}_i^k]_{i\in m}$, $k \in [2,T]$. Therefore, P_1 only obtains a shuffled indicated bit vector which hides the intersection. Similarly, JP-PEQT enjoys security against an arbitrary number of colluding parties.

Security against collusion. During the bMCRG phase, pairwise interactions do not suffer from collusion attacks, so our MPSI/MPSI-Card inherits the security of J-PEQT/JP-PEQT. TAHE-based MPSI/MPSI-Card protocols enjoy security against an arbitrary number of colluding parties. JZSS-based MPSI enjoys security against $T-2$ colluding parties.

J-PEQT and JP-PEQT vs. PEQT. J-PEQT and JP-PEQT can be regarded as extensions of the Private Equality Test (PEQT) [12]. The PEQT enables two parties to determine whether their input strings are equal: P_1, holding input strings $y_1^1, \cdots, y_m^1$, interacts with P_2, holding input strings $y_1^2, \cdots, y_m^2$. The result is that P_2 learns an indication bit vector whether $y_i^1 = y_i^2$, $i \in [m]$ or not, while P_2 learns nothing.

Our J-PEQT and JP-PEQT extend this functionality to a multi-party setting. The J-PEQT allows multiple parties to collaboratively determine whether the corresponding inputs of P_1 and $T-1$ other parties are equal. Specifically, P_1 holds $T-1$ input sets $\mathbf{s}^2, \cdots, \mathbf{s}^T$, where $\mathbf{s}^k = \{s_1^k, \cdots, s_m^k\}$, and interacts with $T-1$ parties P_k, each holding an input string $\mathbf{t}^k = \{t_1^k, \cdots, t_m^k\}$. The result is that P_1 obtains the indication bit vector specifying whether $\bigwedge_{k=2}^{T}(s_i^k = t_i^k) = 1$ for each $i \in [m]$, while the other parties P_k, $k \in [2,T]$, learn nothing.

The JP-PEQT further extends the functionality by outputting an implicitly permuted indication bit vector. This ensures that P_1 does not learn the equal strings, but only the number of equal strings.

[5]In this work, ciphertexts must be re-randomized before being published by any party. We omit the details for convenience.

C. Related Work

In this section, we introduce some efficient related MPSI protocols. The computation and communication complexities of these protocols and our protocols are shown in Table I, where T is the number of participants. t is the maximum number of corrupted participants. m and n denote the sizes of the small set and large set, respectively, where $m \ll n$. Client-L/Client-S denotes that the participants hold Large/Small sets.

Protocols	Communication			Computation			Corrupt Threshold
	Leader	Client-L	Client-S	Leader	Client-L	Client-S	
MPSI [27]	$O(Tn\log n)$	$O(tn\log n)$		$O(Tn)$	$O(tn)$		$t<T$
MPSI [28]	$O(Tn\log n)$	$O(n\log n)$		$O(Tn)$	$O(tn)$		$t<T$
O-Ring [29]	$O(Tn\log n)$	$O(tn\log n)$		$O(Tn)$	$O(tn)$		$t<T$
K-Star [29]	$O(Tn\log n)$	$O(tn\log n)$		$O(Tn)$	$O(tn)$		$t<T$
Our MPSI	$O(Tm\log n)$	$O(m\log n)$	$O(m)$	$O(T(m\log n))$	$O(T(n+m\log n))$	$O(m)$	$t<T$

TABLE I
COMPARISON OF COMMUNICATION AND COMPUTATION COMPLEXITY BETWEEN OUR MPSI AND RELATED WORKS.

In 2017, Kolesnikov et al. [27] introduced the first OT-based MPSI protocol, which was built upon two newly proposed primitives: the oblivious programmable pseudo-random function (OPPRF) and conditional zero-sharing. They began by integrating OPRF from [35] with three distinct data structures (garbled bloom filters, polynomials, and hash tables), resulting in three variants of OPPRF. Subsequently, they designed conditional zero-sharing protocols with security guarantees under both the semi-honest and augmented semi-honest models. By combining OPPRF and zero-sharing, they developed two MPSI protocols tailored to these respective security models.

In 2021, Nevo et al. [28] proposed two MPSI protocols leveraging an OKVS and OPPRF, building upon the work of [27] with malicious security guarantees. These protocols address two scenarios: one assuming no participant collusion and the other accommodating potential collusion. The protocol designed for non-colluding participants relies solely on symmetric cryptographic primitives, while the collusion-resistant protocol employs ZeroXOR based on OPPRF.

Recently, Wu et al. [29] introduced two efficient MPSI protocols, O-Ring and K-Star, both secure against an arbitrary number of colluding parties. In the O-Ring protocol, they leverage a ring network topology, allowing the party with the largest workload to incur lower communication costs compared with other MPSI protocols that utilize a star topology. In contrast, the K-Star protocol adopts a star topology, enabling enhanced concurrency and thereby improving the protocol's speed and performance. Their core idea is to utilize OKVS to build a ring to filter out non-common items, and then further exploit the OPRF to achieve security against an arbitrary number of colluding parties [27]. They also point out that

scalable MPSI protocols [27], [28] are prone to the collusion attack.

Gao et al. [30] introduced a primitive known as bicentric zero-sharing, which simplifies MPSI to a two-party PSI scenario involving central participants named Pivot and Leader. They subsequently developed an efficient MPSI protocol ensuring semi-honest security. However, their protocol does not protect against simultaneous corruption of both Pivot and Leader, nor does it achieve security against an arbitrary number of colluding parties.

II. Preliminaries

A. Notation

For $n, m \in \mathbb{N}$, let $[n]$ denote the set $\{1, 2, \cdots, n\}$ and $[m, n]$ denote the set $\{m, m+1, \cdots, n\}$. 1^λ denotes the string of λ ones. If S is a set, $s \leftarrow S$ indicates sampling s from S at random. We denote vectors by lowercase bold letters, e.g. $\mathbf{s}$. $\boxplus$ denotes as the homomorphic addition operation and $\boxtimes$ denotes as the scalar multiplication operation. We denote the parties as P_k, $k \in [T]$, and their respective input sets as Y^k, including M small sets and N large sets, and $M + N = T$. Let (N, M) denote the number of parties holding large (resp. small) is N (resp. M). Let (n, m) denote the large (resp. small) set size is n (resp. m), and $m \ll n$.

B. MPSI and MPSI-Card

The ideal functionality of MPSI is shown in Figure 5. In this functionality, each party P_i, $i \in [T]$ inputs a set Y^i; P_{rec} is the receiver who gets the intersection $I = \bigcap_{i=1}^{T} Y^i$.

Parameters: T parties: P_i, $i \in [T]$, where P_{rec} is the receiver, $rec \in [T]$. The bit length of set items is l.
Functionality $\mathcal{F}_{\text{MPSI}}$:
1) On input $Y^i = \{y_1^i, \cdots, y_{m_i}^i\}$ from P_i.
2) Give $I = \bigcap_{i=1}^{T} Y^i$ output to P_{rec}.

Fig. 5. Multi-party private set intersection

The ideal functionality of MPSI-Card is shown in Figure 6, where P_i, $i \in [T]$ inputs a set Y^i, and P_{rec} is the receiver who obtains the intersection cardinality $card = |\bigcap_{i=1}^{T} Y^i|$.

Parameters: T parties: P_i, $i \in [T]$, where P_{rec} is the receiver, $rec \in [T]$. The bit length of set items is l.
Functionality $\mathcal{F}_{\text{MPSI-Card}}$:
1) On input $Y^i = \{y_1^i, \cdots, y_{m_i}^i\}$ from P_i.
2) Give $card = |\bigcap_{i=1}^{T} Y^i|$ output to P_{rec}.

Fig. 6. Multi-party private set intersection cardinality

C. Building Blocks

Batched oblivious pseudorandom function (bOPRF). OPRF [36] is a central primitive in the area of private set operations. A batched OPRF (bOPRF) [35] allows the receiver to input $\{x_i\}_{i\in[m]}$ and obtains all PRF values $\{F(k_i, x_i)\}_{i\in[m]}$, and the keys $\{k_i\}_{i\in[m]}$ is known to the sender. We recall the functionality $\mathcal{F}_{\text{bOPRF}}$ in Figure 7.

Parameters: A PRF F. Two parties: $\mathcal{S}$ and $\mathcal{R}$.
Functionality $\mathcal{F}_{\text{bOPRF}}$:
1) Wait for input $\{x_1, \cdots, x_m\}$ from $\mathcal{R}$.
2) Sample random PRF keys $\{k_1, \cdots, k_m\}$ and compute $\{F(k_1, x_1), \cdots, F(k_m, x_m)\}$.
3) Give the keys $\{k_1, \cdots, k_m\}$ to $\mathcal{S}$. Give $\{F(k_1, x_1), \cdots, F(k_m, x_m)\}$ to $\mathcal{R}$.

Fig. 7. Batched oblivious pseudorandom function

Joint zero secret sharing (JZSS). JZSS [37], [38], [39] generates T random values $s^k \in \mathbb{Z}_q$, $k \in [T]$ for T parties, such that $\sum_{k=1}^{T} s^k = 0 \bmod q$. In Figure 8, we describe the functionality of JZSS, denoted as $\mathcal{F}_{\text{JZSS}}$. In this work, we use JZSS based on Shamir's Secret Sharing [37], which enjoys security against $T - 2$ colluding parties.

Parameters: T parties: P_i, $i \in [T]$.
Functionality $\mathcal{F}_{\text{JZSS}}$:
1) Generate T random values $s^k \in \mathbb{Z}_q$, $k \in [T]$ such that $\sum_{k=1}^{T} s^k = 0 \bmod q$.
2) Give the values s^k to P_k, $k \in [T]$.

Fig. 8. Joint zero secret sharing

Secret-shared private equality test. Secret-shared private equality test (ssPEQT) can be seen as a secret share of private equality test (PEQT). More concretely, the two parties $\mathcal{S}$ and $\mathcal{R}$ hold strings x_0 and x_1, respectively. ssPEQT outputs random bits a to $\mathcal{S}$ and b to $\mathcal{R}$ such that if $x_0 = x_1$, $a \oplus b = 1$, otherwise $a \oplus b = 0$. Existing works [40], [41], [42], [43] design linear ssPEQT protocols. We review the functionality $\mathcal{F}_{\text{ssPEQT}}$ in Figure 9.

Parameters: Two parties: $\mathcal{S}$ and $\mathcal{R}$.
Functionality $\mathcal{F}_{\text{ssPEQT}}$:
1) Wait for the input x_0 from $\mathcal{S}$.
2) Wait for the input x_1 from $\mathcal{R}$.
3) Generate two random bits a and b such that if $x_0 = x_1$, $a \oplus b = 1$, otherwise, $a \oplus b = 0$. Gend a to $\mathcal{S}$, and b to $\mathcal{R}$.

Fig. 9. Secret-shared private equality test

Random oblivious transfer. Oblivious transfer (OT) [44] is a central cryptographic primitive in the area of MPC. In random oblivious transfer (ROT), the sender outputs random messages, rather than selecting them as in standard OT. We recall the

1-out-of-2 random oblivious transfer functionality $\mathcal{F}_{\text{ROT}}$ in Figure 10.

Parameters: Two parties: $\mathcal{S}$ and $\mathcal{R}$. The message length l.
Functionality $\mathcal{F}_{\text{ROT}}$:

1) Wait for input b from $\mathcal{R}$.
2) Sample $r_0, r_1 \leftarrow \{0,1\}^l$. Give (r_0, r_1) to $\mathcal{S}$ and give r_b to $\mathcal{R}$.

Fig. 10. 1-out-of-2 random oblivious transfer

Oblivious key-value stores (OKVS). OKVS [45], [46], [20], [47] is a data structure that compactly represents a desired mapping from a set of keys to corresponding values.

Definition 1. *An OKVS is parameterized by a set $\mathcal{K}$ of keys, a set $\mathcal{V}$ of values, and consists of two algorithms:*

- $\mathsf{Encode}(\{(k_1, v_1), \cdots, (k_n, v_n)\})$*: On input key-value pairs $\{(k_i, v_i)\}_{i\in[n]} \subseteq \mathcal{K} \times \mathcal{V}$, outputs an object D (or, with statistically small probability, an error $\perp$).*
- $\mathsf{Decode}(D, k)$*: On input D and a key k, outputs $v \in \mathcal{V}$.*

Correctness. For all $A \subseteq \mathcal{K} \times \mathcal{V}$ with distinct keys: $(k, v) \in A$ and $\perp \neq D \leftarrow \mathsf{Encode}(A) \Rightarrow \mathsf{Decode}(D, k) = v$.

Obliviousness. For all distinct $\{k_1^0, \cdots, k_n^0\}$ and all distinct $\{k_1^1, \cdots, k_n^1\}$, if Encode does not output $\perp$ for $\{k_1^0, \cdots, k_n^0\}$ and $\{k_1^1, \cdots, k_n^1\}$, then the distribution of $\{D | v_i \leftarrow \mathcal{V}, i \in [n], \mathsf{Encode}((k_1^0, v_1), \cdots, (k_n^0, v_n))\}$ is computationally indistinguishable to $\{D | v_i \leftarrow \mathcal{V}, i \in [n], \mathsf{Encode}((k_1^1, v_1), \cdots, (k_n^1, v_n))\}$.

Randomness. We also require an additional randomness property [48] from OKVS. For any $A = \{(k_1, v_1), \cdots, (k_n, v_n)\}$ and $k^* \notin \{k_1, \cdots, k_n\}$, the output of $\mathsf{Decode}(D, k^*)$ is indistinguishable to that of uniform distribution over V, where $D \leftarrow \mathsf{Encode}(A)$.

Hash-to-bin from cuckoo/simple hash. The hash-to-bin from cuckoo/simple hash technique was introduced by Pinkas et al. [12], [49], [11], which is originally applied to construct PSI [35], [15], [17], [5], [50], [45], [19], [20] and private set union (PSU) [51], [52], [34], [53]. At the high level, the sender uses hash functions $h_1, h_2, h_3 : \{0,1\}^* \to [m_c]$ to assign its items $X = \{x_i\}_{i\in[m]}$ to m_c bins $\{X_c[1], \cdots, X_c[m_c]\}$ via cuckoo hashing [33], such that each bin has at most one item, where for each x_i there is some $\gamma \in \{1,2,3\}$ such that $X_c[h_\gamma(x_i)] = x_i||\gamma$. The receiver uses the same hash functions $h_1, h_2, h_3 : \{0,1\}^* \to [m_c]$ to assign its items $Y = \{y_j\}_{j\in[n]}$ to m_c bins $\{Y_1, \cdots, Y_{m_c}\}$ via simple hashing, where for $j \in [n]$, all items are concatenated with hash function indices $(y_j||1, y_j||2, y_j||3)$ and are inserted to the bins $(Y_{h_1(y_j)}, Y_{h_2(y_j)}, Y_{h_3(y_j)})$, respectively. Therefore, if $x_i \in Y$, $\exists\ j \in [n]$, $x_i = y_j$, and $\exists\ \gamma \in \{1,2,3\}$, such that $h_\gamma(x_i) \in \{h_1(y_j), h_2(y_j), h_3(y_j)\}$ and $x_i||\gamma \in \{y_j||1, y_j||2, y_j||3\}$[6].

[6]Appending the index of the hash function helps deal with edge cases like $h_1(y) = h_2(y)$, which happen with non-negligible probability [51], ensuring that there are no identical items in the hash table and all bins are mutually exclusive.

Following [15], we adjust the number of items and table size to reduce the stash size to 0 while achieving a hashing failure probability of $2^{-\lambda}$. The probability of failure is analyzed by Pinkas et al. [11], who show that choosing $\gamma = 3$ and $m_c = 1.27m$ yields a failure probability of 2^{-40}. Due to space limitations, we refer the reader to [15], [11] for detailed parameters.

Threshold additive homomorphic encryption (TAHE). A TAHE consists of a tuple of probabilistic polynomial-time (PPT) algorithms ($\mathsf{TKeyGen}$, TEnc, TDec, $\mathsf{Combine}$, $\mathsf{AddEval}$) as follows:

- $\mathsf{TKeyGen}(1^\lambda, T, t) \to (pk, [sk_i]_{i\in[T]})$: On input the security parameter 1^λ and threshold parameters (T, t), where T denotes the number of users and t denotes the threshold value, the threshold key generation algorithm outputs a public key pk and T shared secret keys sk_i for each user i.
- $\mathsf{TEnc}(pk, m) \to c$: On input the public key pk and a message m, the threshold encryption algorithm outputs a ciphertext c.
- $\mathsf{TDec}(sk_i, c) \to \delta_i$: On input any shared secret key $sk_i, i \in [T]$ and ciphertext c, the threshold decryption algorithm outputs a decryption share δ_i.
- $\mathsf{Combine}(\delta_1, \cdots, \delta_t) \to m$: On input any t decryption shares $\delta_{i_j}, j = 1, \cdots, t$, the combine algorithm outputs the message m.
- $\mathsf{AddEval}(pk, c_1, c_2) \to c^*$: On input the public key pk and ciphertexts $c_1 \leftarrow \mathsf{TEnc}(pk, m_1)$ and $c_2 \leftarrow \mathsf{TEnc}(pk, m_2)$, the evaluation algorithm outputs a new ciphertext $c^* \leftarrow \mathsf{TEnc}(pk, m_1 + m_2)$.

Security. Let $\mathcal{A}$ be a PPT adversary against the IND-CPA security of TAHE. Its advantage function $\mathbf{Adv}^{\text{ind-cpa}}_{\text{TAHE},\mathcal{A}}(\lambda)$ is defined as

$$\left| \Pr \left[b = b' : \begin{array}{l} (i_1, \cdots, i_{t-1}) \leftarrow \mathcal{A}(1^\lambda, T, t); \\ (pk, [sk_i]_{i\in[T]}) \leftarrow \mathsf{TKeyGen}(1^\lambda); \\ (m_0, m_1) \leftarrow \mathcal{A}(pk, sk_{i_1}, \cdots, sk_{i_{t-1}}); \\ b \leftarrow \{0,1\}, c^* \leftarrow \mathsf{TEnc}(pk, m_b); \\ b' \leftarrow \mathcal{A}(pk, sk_{i_1}, \cdots, sk_{i_{t-1}}, c^*); \end{array} \right] - \frac{1}{2} \right|$$

The (T, t)-TAHE scheme is IND-CPA secure if for all PPT adversaries, the advantage function is negligible. In this work, we use an efficient threshold exponential ElGamal encryption scheme [54], [55] as the instantiation of TAHE, which enjoys security against an arbitrary number of colluding parties.

Fully homomorphic encryption (FHE). FHE is a family of encryption schemes that allow arbitrary operations to be performed on encrypted data without decryption. The leveled fully homomorphic encryption supports circuits of a certain bounded depth. In this work, we use an array of optimization techniques of FHE as [15], [17], [19], [34], such as batching (SIMD), windowing, and partitioning, to significantly reduce the depth of the homomorphic circuit. For a detailed explanation, we refer the reader to [15], [17], [19], [34]. For the implementation, we use the homomorphic encryption library SEAL [56], which implements the BFV scheme [57] following [15], [17].

III. Batched Membership Conditional Randomness Generation

In this section, we formally abstract a new cryptographic protocol named batched membership conditional randomness generation (bMCRG) from [17], [27]. In Figure 11, we define the functionality of bMCRG, denoted as $\mathcal{F}_{\text{bMCRG}}$.

Parameters: Two parties: P_1 inputs a set $Y^1 = \{y_i^1\}_{i\in[m]}$; P_2 inputs m mutually exclusive sets $\{Y_i^2\}_{i\in[m]}$.
Functionality $\mathcal{F}_{\text{bMCRG}}$:

1) Wait for an input of a set $Y^1 = \{y_i^1\}_{i\in[m]}$ from P_1.
2) Wait for an input of m sets $\{Y_i^2\}_{i\in[m]}$ from P_2.
3) Generate two random characteristic vectors $\mathbf{s} = [s_i]_{i\in[m]}$ and $\mathbf{t} = [t_i]_{i\in[m]}$, where for $i \in [m]$, if $y_i^1 \in Y_i^2$, $s_i = t_i$, otherwise $s_i \neq t_i$.
4) Give the vector $\mathbf{s} = [s_i]_{i\in[m]}$ to P_1. Give the vector $\mathbf{t} = [t_i]_{i\in[m]}$ to P_2.

Fig. 11. Batched membership conditional randomness generation

A. bMCRG Construction in the Balanced Setting

We give a construction of bMCRG based on bOPRF and OKVS in the balanced setting, as described in Figure 12, where $|Y^1| \approx |\cup_{i=1}^m \{Y_i^2\}|$.

Input: P_1 inputs a set $Y^1 = \{y_i^1\}_{i\in[m]}$. P_2 inputs m mutually exclusive sets $\{Y_i^2\}_{i\in[m]}$.
Output: P_1 outputs a vector $\mathbf{s} = [s_i]_{i\in[m]}$. P_2 outputs a vector $\mathbf{t} = [t_i]_{i\in[m]}$.

1) P_1 and P_2 invoke the bOPRF functionality $\mathcal{F}_{\text{bOPRF}}$.
   a) P_1 input a set $Y^1 = \{y_i^1\}_{i\in[m]}$.
   b) P_1 obtains all PRF values $F(k_i, y_i^1)$, $i \in [m]$. P_2 obtains PRF keys $\{k_1, \cdots, k_m\}$.
2) For all $i \in [m]$, P_2 computes PRF values $F(k_i, Y_i^2[j])$, where $Y_i^2[j]$ denotes j-th item in Y_i^2.
3) P_2 encodes an OKVS:
   a) P_2 chooses m random values $[t_i]_{i\in[m]}$, and defines $\mathcal{P} = \{(Y_i^2[j], F(k_i, Y_i^2[j]) \oplus t_i)\}_{i\in[m], j\in[|Y_i^2|]}$.
   b) P_2 computes an OKVS: $D = \mathsf{Encode}(\mathcal{P})$, and sends D to P_1.
4) P_1 decodes $s_i = \mathsf{Decode}(D, y_i^1)) \oplus F(k_i, y_i^1)$, $i \in [m]$.
5) P_1 outputs $\mathbf{s} = [s_i]_{i\in[m]}$. P_2 outputs $\mathbf{t} = [t_i]_{i\in[m]}$.

Fig. 12. bMCRG from bOPRF and OKVS

Theorem 1. *The protocol Π_{bMCRG} shown in Figure 12 securely implements the functionality $\mathcal{F}_{\text{bMCRG}}$ (as in Figure 11) in the $\mathcal{F}_{\text{bOPRF}}$-hybrid model, against semi-honest adversaries, provided a secure OKVS scheme.*

Proof. We construct Sim_{P_1} and Sim_{P_2} to simulate the views of corrupt P_1 and corrupt P_2 respectively, and argue the indistinguishability of the produced transcript from the real execution.

- **Corrupt P_1.** $\text{Sim}_{P_1}(Y^1, \mathbf{s})$ simulates the view of corrupt P_1 as follows: Sim_{P_1} randomly chooses $\mathbf{r} = [r_1, \cdots, r_m]$ and invokes $\text{Sim}_{\text{bOPRF}}^{\text{Sender}}(Y^1, \mathbf{r})$ and appends the output to the view. Sim_{P_1} computes a random OKVS D by selecting random key-value pairs, except for the encoding $\{y_i^1, r_i \oplus s_i\}_{i\in[m]}$. We argue that the outputs of Sim_{P_1} are indistinguishable from the real view of P_1 by the following hybrids:
  - Hyb_0: P_1's view in the real protocol.
  - Hyb_1: Same as Hyb_0 except that the output of $\mathcal{F}_{\text{bOPRF}}$ is replaced by $\mathbf{r} = [r_1, \cdots, r_m]$ chosen by Sim_{P_1}, and Sim_{P_1} runs the $\mathcal{F}_{\text{bOPRF}}$ simulator to produce the simulated view for P_1. The security of bOPRF guarantees the view in simulation is computationally indistinguishable from the view in the real protocol.
  - Hyb_2: Same as Hyb_1 except that the object D is generated by Sim_{P_1}, and Sim_{P_1} encodes $\{y_i^1, r_i \oplus s_i\}_{i\in[m]}$ and random key-value pairs into D. Briefly, this simulation is indistinguishable for the following reasons: The pseudorandomness of PRF value is indistinguishable from random, and then by the obliviousness of OKVS, D is distributed uniformly.
- **Corrupt P_2.** $\text{Sim}_{P_2}(Y^2, \mathbf{t})$ simulates the view of corrupt P_2 as follows: Sim_{P_2} randomly chooses $\mathbf{k} = [k_1, \cdots, k_m]$ and invokes $\text{Sim}_{\text{bOPRF}}^{\text{Receiver}}(\perp, \mathbf{k})$ and appends the output to the view. Sim_{P_2} computes the OKVS D like the real protocol. We argue that the outputs of Sim_{P_2} are indistinguishable from the real view of P_2 by the following hybrids:
  - Hyb_0: P_2's view in the real protocol.
  - Hyb_1: Same as Hyb_0 except that the output of $\mathcal{F}_{\text{bOPRF}}$ is replaced by $\mathbf{k} = [k_1, \cdots, k_m]$ chosen by Sim_{P_2}, and Sim_{P_2} runs the $\mathcal{F}_{\text{bOPRF}}$ simulator to produce the simulated view for P_2.

The security of bOPRF guarantees the view in simulation is computationally indistinguishable from the view in the real protocol. □

B. bMCRG Construction in the Unbalanced Setting

Here, We give the construction of bMCRG from bOPRF and FHE in the unbalanced case following [17], as described in Figure 13, where $|Y^1| \ll |\cup_{i=1}^m \{Y_i^2\}|$.

Correctness. Following the constructions of [15], [17], [19], we decouple the hash-to-bin technique to abstract the bMCRG functionality, which is then instantiated as the unbalanced bMCRG constructions. P_2 encodes PRF values $\bar{Y}_i^2$, $i \in [m]$, as a polynomial of degree $B_i = |Y_i^2|$ and masks it with a random value t_i. Consequently, when P_1 decrypts the ciphertext corresponding to PRF value $\bar{y}_i^1$, it obtains a random value s_i. For each $i \in [m]$, if $y_i^1 \in Y_i^2$, there exists $j \in [B_i]$ such that $\bar{y}_i^1 = \bar{Y}_i^2[j]$. We have $s_i = F_i(\bar{y}_i^1) = \prod_{j=1}^{B_i}(\bar{y}_i^1 - \bar{Y}_i^2[j]) + t_i = t_i$. Otherwise, for all $j \in [B_i]$, $\bar{y}_i^1 \neq \bar{Y}_i^2[j]$, and hence $s_i = F_i(\bar{y}_i^1) = \prod_{j=1}^{B_i}(\bar{y}_i^1 - \bar{Y}_i^2[j]) + t_i \neq t_i$.

Input: P_1 inputs a set $Y^1 = \{y_i^1\}_{i\in[m]}$. P_2 inputs m mutually exclusive sets $\{Y_i^2\}_{i\in[m]}$.
Output: P_1 outputs a vector $\mathbf{s} = [s_i]_{i\in[m]}$. P_2 outputs a vector $\mathbf{t} = [t_i]_{i\in[m]}$.

1) Both parties agree on parameters of bOPRF and FHE.
2) P_1 and P_2 invoke the bOPRF functionality $\mathcal{F}_{\text{bOPRF}}$.
   a) P_1 input a set $Y^1 = \{y_i^1\}_{i\in[m]}$.
   b) P_1 obtains all PRF values $\bar{y}_i^1 = F(k_i, y_i^1)$, $i \in [m]$. P_2 obtains PRF keys $\{k_1, \cdots, k_m\}$.
3) For $i \in [m]$, P_2 computes PRF values $\bar{Y}_i^2[j] = F(k_i, Y_i^2[j])$, where $Y_i^2[j]$ denotes j-th item, $B_i = |Y_i^2|$.
4) P_2 chooses a random vector $\mathbf{t} = [t_i]_{i\in[m]}$. For all $i \in [m]$, P_2 computes polynomials $F_i(x) = f_i(x) + t_i$, where for all $j \in [B_i]$, $f_i(\bar{Y}_i^2[j]) = 0$. Thus, P_2 obtains coefficient matrix $\mathbf{A}$, where i-th column of $\mathbf{A}$ are the coefficients of F_i.
5) P_1 uses its FHE public key to encrypt $\bar{\mathbf{y}}^1 = [\bar{y}_i^1]_{i\in[m]}$ and sends ciphertexts $[\![\bar{y}_i^1]\!]$, $i \in [m]$ to P_2.
6) For each $[\![\bar{y}_i^1]\!]$, P_2 homomorphically computes encryptions of all powers $\mathbf{C}_i = [[\![0]\!], [\![(\bar{y}_i^1)^1]\!], \cdots, [\![(\bar{y}_i^1)^{B_i}]\!]]$. Then, P_2 homomorphically evaluates $\mathbf{C}_i' = \mathbf{A}_i \boxtimes \mathbf{C}_i$, and sends all ciphertexts to P_1.
7) P_1 decrypts the ciphertexts into $\mathbf{s} = [s_i]_{i\in[m]}$.
8) P_1 outputs $\mathbf{s} = [s_i]_{i\in[m]}$. P_2 outputs $\mathbf{t} = [t_i]_{i\in[m]}$.

Fig. 13. bMCRG from bOPRF and FHE

Theorem 2. *The protocol Π_{bMCRG} shown in Figure 13 securely implements the functionality $\mathcal{F}_{\text{bMCRG}}$ (as in Figure 11) in the $\mathcal{F}_{\text{bOPRF}}$-hybrid model, against semi-honest adversaries, provided that the fully homomorphic encryption scheme is IND-CPA secure.*

Proof. We construct Sim_{P_1} and Sim_{P_2} to simulate the views of corrupt P_1 and corrupt P_2 respectively, and argue the indistinguishability of the produced transcript from the real execution.

- **Corrupt P_1.** $\text{Sim}_{P_1}(Y^1, \mathbf{s})$ simulates the view of corrupt P_1 as follows: Sim_{P_1} randomly chooses $\mathbf{r} = [r_1, \cdots, r_m]$ and invokes $\text{Sim}_{\text{bOPRF}}^{\text{Sender}}(Y^1, \mathbf{r})$ and appends the output to the view. Sim_{P_1} encrypts $\mathbf{s}$ in place of the ciphertexts in step 6.
  We argue that the outputs of Sim_{P_1} are indistinguishable from the real view of P_1 by the following hybrids:
  – Hyb_0: P_1's view in the real protocol.
  – Hyb_1: Same as Hyb_0 except that the output of $\mathcal{F}_{\text{bOPRF}}$ is replaced by $\mathbf{r} = [r_1, \cdots, r_m]$ chosen by Sim_{P_1}, and Sim_{P_1} runs the $\mathcal{F}_{\text{bOPRF}}$ simulator to produce the simulated view for P_1. The security of bOPRF guarantees the view in simulation is computationally indistinguishable from the view in the real protocol.
  – Hyb_2: Same as Hyb_1 except that the ciphertexts are generated by Sim_{P_1} by encrypting $\mathbf{s}$. The simulation is indistinguishable from the real view.
- **Corrupt P_2.** $\text{Sim}_{P_2}(Y^2, \mathbf{t})$ simulates the view of corrupt P_2 as follows: Sim_{P_2} randomly chooses $\mathbf{k} = [k_1, \cdots, k_m]$ and invokes $\text{Sim}_{\text{bOPRF}}^{\text{Receiver}}(\perp, \mathbf{k})$ and appends the output to the view. Sim_{P_2} encrypts random values in place of the ciphertexts in step 5.
  We argue that the outputs of Sim_{P_2} are indistinguishable from the real view of P_2 by the following hybrids:
  – Hyb_0: P_2's view in the real protocol.
  – Hyb_1: Same as Hyb_0 except that the output of $\mathcal{F}_{\text{bOPRF}}$ is replaced by $\mathbf{k} = [k_1, \cdots, k_m]$ chosen by Sim_{P_2}, and Sim_{P_2} runs the $\mathcal{F}_{\text{bOPRF}}$ simulator to produce the simulated view for P_2. The security of bOPRF guarantees the view in simulation is computationally indistinguishable from the view in the real protocol.
  – Hyb_2: Same as Hyb_1 except that the ciphertexts are generated by Sim_{P_2} by encrypting random values. The simulation is indistinguishable from the real view.

The IND-CPA security of the fully homomorphic encryption scheme guarantees that the view in simulation is computationally indistinguishable from the view in the real protocol. □

Optimizations. An array of optimization techniques including batching, windowing, and partitioning, can be used to reduce the depth of the homomorphic circuit significantly, consistent with the optimizations in [15], [17], [19].

When applying the partitioning optimization technique, P_2 aims to reduce the depth of the homomorphic circuit (the polynomial degree) by dividing each set Y_i^2 into α partitions. Each partition is then encoded as a separate polynomial of degree B_i/α and masked with an independent random value $t_{i1}, \ldots, t_{i\alpha}$. Accordingly, P_1 decrypts the ciphertext corresponding to y_i^1 and obtains α random values $s_{i1}, \ldots, s_{i\alpha}$. As pointed out in [34], if $y_i^1 \in Y_i^2$, the item y_i^1 can only be encoded in one of the α polynomials; that is, there exists a unique $j \in [\alpha]$ such that $s_{ij} = t_{ij}$, while $s_{ij} \neq t_{ij}$ for all other indices. In [17], this case is handled using generic MPC protocols. In our work, we employ the ss-PEQT functionality (depicted in Figure 9) together with the ROT functionality (depicted in Figure 10) to aggregate multiple pairs of characteristic values (s_{ij}, t_{ij}) into a single pair (s_i, t_i), thereby ensuring compatibility with the subsequent J-PEQT/JP-PEQT protocols, which operate on a single pair of characteristic values. The construction proceeds as follows:

1) P_1 and P_2 respectively input $[s_{ij}]_{j\in[\alpha]}$ and $[t_{ij}]_{j\in[\alpha]}$ into the ss-PEQT functionality. As a result, P_1 obtains $[e_{ij}]_{j\in[\alpha]}$ and P_2 obtains $[d_{ij}]_{j\in[\alpha]}$, where $e_{ij} \oplus d_{ij} = 1$, if $s_{ij} = t_{ij}$, otherwise $e_{ij} \oplus d_{ij} = 0$. Both parties then locally compute $e_i = \oplus_{j=1}^{\alpha} e_{ij}$, $d_i = \oplus_{j=1}^{\alpha} d_{ij}$.
2) P_1 and P_2 invoke the ROT functionality: P_2 inputs d_i. The result is that P_1 obtains $r_{i,0}$ and $r_{i,1}$, P_2 obtains r_{i,d_i}.
3) P_1 outputs $s_i = r_{i,e_i\oplus 1}$ and P_1 outputs $t_i = r_{i,d_i}$.

Consequently, if $\bigwedge_{j=1}^{\alpha}(s_{ij} \neq t_{ij})$, then $\bigwedge_{j=1}^{\alpha}(e_{ij} \oplus d_{ij} = 0)$, which implies $e_i \oplus d_i = 0$ and thus $s_i = r_{i,e_i\oplus 1} \neq t_i = r_{i,d_i}$. If there exists a unique $j \in [\alpha]$ such that $s_{ij} = t_{ij}$, then $e_{ij} \oplus d_{ij} = 1$ holds for exactly one index j, implying $e_i \oplus d_i = 1$, and therefore $s_i = r_{i,e_i\oplus 1} = r_{i,d_i} = t_i$.

Comparison with the balanced bMCRG. In Figure 10, the communication cost of our balanced bMCRG equals the sum of the costs incurred by bOPRF and OKVS. Specifically, the communication complexity of bOPRF scales linearly with the size of Y^1, while that of OKVS scales linearly with the sizes of $\{Y_i^2\}_{i\in[m]}$. Hence, this construction is well-suited for balanced scenarios, where the input sizes of P_1 and P_2 are comparable. In Figure 11, the unbalanced bMCRG inherits the advantages of the constructions of [15], [17], [19], achieving communication that scales linearly only with the size of Y^1. By applying the windowing optimization technique, the design trades communication for computation, reducing the polynomial degree and resulting in communication that scales logarithmically with the sizes of $\{Y_i^2\}_{i\in[m]}$. Therefore, it is particularly suitable for unbalanced scenarios in which the input set of P_1 is significantly smaller than that of P_2.

IV. Joint Private Equality Test

In this section, we introduce a new cryptographic protocol named the joint private equality test (J-PEQT). In Figure 14, we define the functionality of J-PEQT, denoted as $\mathcal{F}_{\text{J-PEQT}}$.

Parameters: Parties: P_1 inputs $T-1$ vectors $\mathbf{s}^k = [s_i^k]_{i\in[m]}$, $k \in [2,T]$. $P_k, k \in [2,T]$ inputs a vector $\mathbf{t}^k = [t_i^k]_{i\in[m]}$.
Functionality $\mathcal{F}_{\text{J-PEQT}}$:

1) Wait for an input of $T-1$ vectors $\mathbf{s}^k = [s_i^k]_{i\in[m]}$, $k \in [2,T]$ from P_1.
2) Wait for each input $\mathbf{t}^k = [t_i^k]_{i\in[m]}, k \in [2,T]$ from $T-1$ parties $P_k, k \in [2,T]$.
3) Generate a random bit string $\mathbf{b} = [b_i]_{i\in[m]}$ such that for $i \in [m]$, if $\bigwedge_{k=2}^{T}(s_i^k = t_i^k)$, $b_i = 1$, otherwise, $b_i = 0$.
4) Give the bit vector $\mathbf{b} = [b_i]_{i\in[m]}$ to P_1.

Fig. 14. Joint private equality test

A. J-PEQT from JZSS

We give a construction of J-PEQT from the joint zero secret sharing (JZSS), as described in Figure 15. We generate secret shares of zero for all parties P_k, $k \in [T]$, using JZSS. Then, P_k, $k \in [2,T]$ publishes its input, hiding it with the corresponding zero share, similar to a one-time pad. Finally, P_1 checks whether the sums are equal or not.

Lemma 1. *For two random vectors* $\mathbf{s} = [s_k]_{k\in[2,T]}$ *and* $\mathbf{t} = [t_k]_{k\in[2,T]}$, *where* $s_k, t_k \in \mathbb{Z}_q$ *with* q *being a large prime number of* λ*-bit length, let* E *denote the event* $\bigwedge_{k=2}^{T}(s_k = t_k)$. *There exists a negligible function* $\mathsf{negl}(\cdot)$ *such that* $\Pr\left[\bar{E} \wedge \left(\sum_{k=2}^{T} s_k = \sum_{k=2}^{T} t_k\right)\right] \le \mathsf{negl}(\lambda)$.

Proof. Since $\mathbf{s} = [s_k]_{k\in[2,T]}$ and $\mathbf{t} = [t_k]_{k\in[2,T]}$ are two random vectors, where $s_k, t_k \in \mathbb{Z}_q$, we have $\Pr\left[\sum_{k=2}^{T} s_k = \sum_{k=2}^{T} t_k\right] = \frac{q^{2T-3}}{q^{2T-2}} = \frac{1}{q}$. Due to $\Pr[E] = \frac{q^{T-1}}{q^{2T-2}} = \frac{1}{q^{T-1}}$, we have $\Pr[\bar{E}] = 1 - \frac{1}{q^{T-1}}$. Therefore,

$$\Pr\left[\bar{E} \wedge \left(\sum_{k=2}^{T} s_k = \sum_{k=2}^{T} t_k\right)\right] = \Pr[\bar{E}] \times \Pr\left[\sum_{k=2}^{T} s_k = \sum_{k=2}^{T} t_k\right] = \left(1 - \frac{1}{q^{T-1}}\right) \cdot \frac{1}{q} \le \mathsf{negl}(\lambda).$$

Correctness. Due to $\mathcal{F}_{\text{JZSS}}$: $e_i + \sum_{k=2}^{T} d_i^k \bmod q = 0$, we have $p_i = \sum_{k=2}^{T} c_i^k + e_i - \left(\sum_{k=2}^{T} s_i^k\right) \bmod q = \sum_{k=2}^{T} t_i^k - \left(\sum_{k=2}^{T} s_i^k\right) \bmod q$. If $p_i = 0$, it follows that $\sum_{k=2}^{T} s_k = \sum_{k=2}^{T} t_k$. Otherwise, $\sum_{k=2}^{T} s_k \neq \sum_{k=2}^{T} t_k$. According to Lemma 1, $\Pr\left[\bar{E} \wedge \left(\sum_{k=2}^{T} s_k = \sum_{k=2}^{T} t_k\right)\right] \le \mathsf{negl}(\lambda)$, which means the event $\bar{E} \wedge \left(\sum_{k=2}^{T} s_k = \sum_{k=2}^{T} t_k\right)$ occurs

Input: Parties: P_1 inputs $T-1$ vector $\mathbf{s}^k = [s_i^k]_{i\in[m]}$, $k \in [2,T]$. $P_k, k \in [2,T]$ inputs a vector $\mathbf{t}^k = [t_i^k]_{i\in[m]}$.
Output: P_1 outputs the bit vector $\mathbf{b} = [b_i]_{i\in[m]}$.

1) $P_k, k \in [T]$ encode all strings $\mathbf{s}^k, \mathbf{t}^k, k \in [2,T]$ into $\mathbb{Z}_q$.
2) P_1 and $P_k, k \in [2,T]$ invoke JZSS, such that P_1 obtains $\mathbf{e} = [e_i]_{i\in[m]}$ and $P_k, k \in [2,T]$ obtains $\mathbf{d}^k = [d_i^k]_{i\in[m]}$.
3) $P_k, k \in [2,T]$ computes and sends $c_i^k = d_i^k + t_i^k \bmod q$ to P_1.
4) P_1 computes $p_i = \sum_{k=2}^{T} c_i^k + e_i - (\sum_{k=2}^{T} s_i^k) \bmod q$. From $i = 1$ to $i = m$, P_1 sets $b_i = 1$, if $p_i = 0$, otherwise, $b_i = 0$. Finally, P_1 outputs vector $\mathbf{b} = [b_i]_{i\in[m]}$.

Fig. 15. JZSS-based J-PEQT

Input: Parties: P_1 inputs $T-1$ vectors $\mathbf{s}^k = [s_i^k]_{i\in[m]}$, $k \in [2,T]$. $P_k, k \in [2,T]$ inputs a vector $\mathbf{t}^k = [t_i^k]_{i\in[m]}$.
Output: P_1 outputs the bit vector $\mathbf{b} = [b_i]_{i\in[m]}$.

1) $P_k, k \in [T]$ run $\mathsf{TKeyGen}\ (1^\lambda) \to (pk, [sk_k]_{k\in[T]})$. P_k, $k \in [T]$ obtain sk_k, respectively.
2) P_k, $k \in [2,T]$ encrypts $\mathbf{t}^k = [t_i^k]_{i\in[m]}$, $k \in [2,T]$: $c_i^k = \mathsf{TEnc}\ (pk, t_i^k)$, and send all ciphertexts to P_1.
3) P_1 computes $\bar{c}_i^1 = (\boxplus_{k=2}^{T} c_i^k) \boxplus \mathsf{TEnc}(pk, \sum_{k=2}^{T}(-s_i^k))$, and sends $\bar{c}_i^1$, $i \in [m]$ to P_2.
4) From $k = 2$ to $k = T$, $P_k, k \in [2,T]$ chooses m random values α_i^k, $i \in [m]$ and computes $\bar{c}_i^k = \alpha_i^k \boxtimes \bar{c}_i^{k-1}$, and sends $\bar{c}_i^k$ to P_{k+1}. P_T sends $\bar{c}_i^T$ to P_1 and $P_k, k \in [2,T-1]$.
5) P_k, $k \in [2,T]$ decrypts the plaintext share $p_i^k = \mathsf{TDec}\ (sk_k, \bar{c}_i^T)$, and sends the plaintext share to P_1.
6) P_1 decrypts the plaintext share $p_i^1 = \mathsf{TDec}(sk_1, \bar{c}_i^T)$ and combine the plaintext $p_i = \mathsf{Combine}(p_i^1, p_i^2, \cdots, p_i^T)$. From $i = 1$ to $i = m$, P_1 sets $b_i = 1$, if $p_i = 0$, otherwise, $b_i = 0$. Finally, P_1 outputs vector $\mathbf{b} = [b_i]_{i\in[m]}$.

Fig. 16. TAHE-based J-PEQT

Input: P_1 inputs a set $Y^1 = \{y_i^1\}_{i\in[m]}$. $P_k, k \in [2,T]$ input sets $Y^k = \{y_j^k\}_{j\in[m_k]}, k \in [2,T]$.
Output: The receiver P_1 outputs the intersection $I = \bigcap_{k=1}^{T} Y^k$.

1) P_1 and P_k, $k \in [2,T]$ invoke cuckoo/simple hashing:
   a) P_1 inserts set Y^1 into the cuckoo hash table and fills empty bins with the dummy item $\perp$, where the cuckoo hash table $Y_c^1 = \{Y_c^1[i]\}_{i\in[m_c]}$ consists of m_c bins and each bin $Y_c^1[i]$ has only one item, where for each y_i^1 there is $\gamma \in \{1,2,3\}$ such that $Y_c^1[h_\gamma(y_i^1)] = y_i^1||\gamma$.
   b) P_k, $k \in [2,T]$ uses the same hash functions to insert Y^k into the simple hash table, where all item y_j^k are concatenated with hash function indices ($y_j^k||1, y_j^k||2$, $y_j^k||3$) and are inserted to the bins ($Y^k_{h_1(y_j)}, Y^k_{h_2(y_j)}$, $Y^k_{h_3(y_j)}$), respectively, the table consists of m_c bins $\{Y_1^k, Y_2^k, \cdots, Y_{m_c}^k\}$.
2) P_1 and P_k, $k \in [2,T]$ invoke the functionality $\mathcal{F}_{\text{bMCRG}}$ (balanced and unbalanced bMCRG can be selected according to the difference in the set size of P_1 and P_k):
   a) P_1 inputs Y_c^1 and $P_k, k \in [2,T]$ inputs $\{Y_i^k\}_{i\in[m_c]}$.
   b) P_1 obtains $\mathbf{s}^k = [s_i^k]_{i\in[m_c]}$, $k \in [2,T]$ and P_k obtains $\mathbf{t}^k = [t_i^k]_{i\in[m_c]}$, $k \in [2,T]$.
3) P_1 and P_k, $k \in [2,T]$ invoke the functionality $\mathcal{F}_{\text{J-PEQT}}$:
   a) P_1 inputs $\mathbf{s}^k = [s_i^k]_{i\in[m_c]}$, $k \in [2,T]$ and P_k inputs $\mathbf{t}^k = [t_i^k]_{i\in[m_c]}$, $k \in [2,T]$.
   b) P_1 obtains $\mathbf{b} = [b_i]_{i\in[m_c]}$.
4) P_1 outputs $I = \{y_{i^*}^1\}$, where for $i^* \in [m_c]$, $b_{i^*} = 1$.

Fig. 17. MPSI from bMCRG and J-PEQT

with negligible probability. Therefore, $\mathcal{F}_{\text{J-PEQT}}$ satisfies correctness.

Theorem 3. *The construction of Figure 15 securely implements functionality $\mathcal{F}_{\text{J-PEQT}}$ (as in Figure 14) in the $\mathcal{F}_{JZSS}$ hybrid model, against a semi-honest adversary colluding with up to $t < (T-1)$ parties.*

Proof. Let $\mathcal{C}$ denote the set of all corrupted parties, where $|\mathcal{C}| = t < T-1$. Intuitively, the protocol is secure because all outputs generated by each party are hidden by the outputs of $\mathcal{F}_{\text{JSSZ}}$, which is secure against semi-honest adversaries colluding with up to $t < T-1$ parties. Specifically, if at least two parties remain uncorrupted, the simulator can easily simulate their outputs by generating random values that are independent of the inputs of honest parties.

- $P_1 \notin \mathcal{C}$. $\text{Sim}_{\mathcal{C}}(\mathbf{t}^{\mathcal{C}})$ simulates the view of corrupt $\mathcal{C}$ as follows: $\text{Sim}_{\mathcal{C}}$ randomly chooses $\mathbf{r}^k = [r_i^k]_{i\in[m]}$ for $P_k \in \mathcal{C}$ and invokes $\text{Sim}^{\mathcal{C}}_{\text{JZSS}}$ $([r_i^k]_{i\in[m]})$ and appends the output to the view. The security of JZSS guarantees that the view in simulation is indistinguishable from the view in the real protocol.
- $P_\alpha \notin \mathcal{C}$. This can be divided into the following two cases.
  - $P_1 \notin \mathcal{C}$. This case has already been proven above.
  - $P_1 \in \mathcal{C}$. $\text{Sim}_{\mathcal{C}}(\mathbf{s}^k_{k\in[T]}, \mathbf{t}^k_{P_k\in\mathcal{C}\setminus P_1}, \mathbf{b})$ simulates the view of corrupt $\mathcal{C}$ as follows: $\text{Sim}_{\mathcal{C}}$ randomly chooses $\mathbf{r}^k = [r_i^k]_{i\in[m]}$ for all $P_k \in \mathcal{C}$ and invokes $\text{Sim}^{\mathcal{C}}_{\text{JZSS}}([r_i^k]_{i\in[m]})$ and appends the output to the view. For $b_i = 0, i \in [m]$, $\text{Sim}_{\mathcal{C}}$ chooses $T - |\mathcal{C}|$ random values and appends them to the view. For $b_i = 1, i \in [m]$, $\text{Sim}_{\mathcal{C}}$ chooses $T - |\mathcal{C}|$ random values $r_i^{T-|\mathcal{C}|}$ such that $\sum_{k=2}^{T} s_i{}^k + \sum t_i^{\mathcal{C}\setminus P_1} + \sum r_i^{\mathcal{C}\setminus P_1} + \sum r_i^{T-|\mathcal{C}|} = 0 \bmod q$ and appends them to the view.

The security of JZSS guarantees that the view in simulation is indistinguishable from the view in the real protocol. □

Instantiation. In this work, we use Shamir's Secret Sharing [37] to instantiate JZSS. Specifically, T parties choose a random polynomial of degree $T-1$ with the first coefficient as 0, and use the unique identifier of P_k, $k \in [T]$ as input to evaluate its zero share, which is secretly sent to P_k. After receiving the shares from all other parties, anyone can sum the shares to obtain the joint zero share. This JZSS construction enjoys security against any $T-2$ colluding parties.

Remark 1. *The construction in Figure 15 does not entirely restrict the output result to P_1, since any participant, after receiving all the shares from the others, can verify $p_i = \sum_{k=2}^{T} c_i^k + e_i - \left(\sum_{k=2}^{T} s_i^k\right) \bmod q = 0$. Therefore, any party can act as the receiver denoted as P_{rec}.*

Parameters: Parties: P_1 inputs $T-1$ vectors $\mathbf{s}^k = [s_i^k]_{i\in[m]}$, $k \in [2,T]$. $P_k, k \in [2,T]$ inputs a vector $\mathbf{t}^k = [t_i^k]_{i\in[m]}$, and a random permutation π_k over $[m]$.
Functionality $\mathcal{F}_{\text{JP-PEQT}}$:

1) Wait for an input of $T-1$ vectors $\mathbf{s}^k = [s_i^k]_{i\in[m]}$, $k \in [2,T]$ from P_1.
2) Wait for each input $\mathbf{t}^k = [t_i^k]_{i\in[m]}, k \in [2,T]$, and a permutation π_k, $k \in [2,T]$ over $[m]$ from $T-1$ parties $P_k, k \in [2,T]$.
3) Generate a random bit string $\mathbf{b} = [b_i]_{i\in[m]}$ such that for $i \in [m]$, if $\bigwedge_{k=2}^{T}(s^k_{\pi(i)} = t^k_{\pi(i)})$, $b_i = 1$, otherwise, $b_i = 0$, where $\pi = \pi_2 \circ \pi_3 \circ \cdots \circ \pi_T$.
4) Give the bit vector $\mathbf{b} = [b_i]_{i\in[m]}$ to P_1.

Fig. 18. Joint permuted private equality test

B. J-PEQT from TAHE

We give a construction of J-PEQT from the threshold additive homomorphic encryption (TAHE), as described in Figure 16. In this work, ciphertexts must be re-randomized before being published by any party. We omit the details for convenience.

Theorem 4. *The construction of Figure 16 securely implements functionality $\mathcal{F}_{\text{J-PEQT}}$ (as in Figure 14) against a semi-honest adversary colluding with up to any $t < T$ parties, provided that the TAHE scheme is IND-CPA secure.*

Input: P_1 inputs $T-1$ vectors $\mathbf{s}^k = [s_i^k]_{i\in[m]}$, $k \in [2,T]$. $P_k, k \in [2,T]$ inputs a vector $\mathbf{t}^k = [t_i^k]_{i\in[m]}$, and a random permutation π_k over $[m]$.

Output: P_1 outputs the bit vector $\mathbf{b} = [b_i]_{i\in[m]}$.

1) P_1 and $P_k, k \in [2,T]$ run $\mathsf{TKeyGen}(1^\lambda) \to (pk, sk_1, \cdots, sk_T)$.
2) $P_k, k \in [2,T]$ encrypts $\mathbf{t}^k = [t_i^k]_{i\in[m]}, k \in [2,T]$: $c_i^k = \mathsf{TEnc}(pk, t_i^k)$, and send all ciphertexts to P_1.
3) P_1 computes $\bar{c}_i^1 = (\boxplus_{k=2}^T c_i^k) \boxplus \mathsf{TEnc}(pk, \sum_{k=2}^T (-s_i^k))$, and sends $\bar{c}_i^1$, $i \in [m]$ to P_2.
4) From $k=2$ to $k=T$, $P_k, k \in [2,T]$ chooses m random values α_i^k, $i \in [m]$ and computes $\bar{c}_i^k = \pi_k(\alpha_i^k \boxtimes \bar{c}_i^{k-1})$, and sends $\bar{c}_i^k$ to P_{k+1}. P_T sends $\bar{c}_i^T$ to P_1 and $P_k, k \in [2, T-1]$.
5) $P_k, k \in [2,T]$ decrypts the plaintext share $p_i^k = \mathsf{TDec}(sk_k, \bar{c}_i^T)$, and sends the plaintext share to P_1.
6) P_1 decrypts the plaintext share $p_i^1 = \mathsf{TDec}(sk_1, \bar{c}_i^T)$ and combine the plaintext $p_i = \mathsf{Combine}(p_i^1, p_i^2, \cdots, p_i^T)$. From $i=1$ to $i=m$, P_1 sets $b_i = 1$, if $p_i = 0$, otherwise, $b_i = 0$. Finally, P_1 outputs the bit vector $\mathbf{b} = [b_i]_{i\in[m]}$.

Fig. 19. TAHE-based JP-PEQT

Proof. Let $\mathcal{C}$ denote the set of all corrupted parties, where $|\mathcal{C}| = t < T$. Intuitively, the protocol is secure because all outputs generated by each party are hidden by the IND-CPA secure TAHE. Thereby, the simulator can easily simulate their outputs by encrypting random values that are independent of the inputs of honest parties.

- $P_1 \notin \mathcal{C}$**.** $\mathsf{Sim}_\mathcal{C}(\mathbf{t}^\mathcal{C})$ simulates the view of corrupt $\mathcal{C}$ as follows: $\mathsf{Sim}_\mathcal{C}$ invokes TAHE.$\mathsf{TKeyGen}$ algorithm with other parties as the real protocol, and obtains (pk, sk_1). Upon receiving $T-1$ ciphertexts, it encrypts random values as the ciphertexts $\bar{c}^1$. The IND-CPA security of TAHE guarantees that the view in simulation is indistinguishable from the view in the real protocol.
- $P_\alpha \notin \mathcal{C}$**,** $\alpha \neq T$**.** This can be divided into the following two cases.
  - $P_1 \notin \mathcal{C}$. This case has already been proven above.
  - $P_1 \in \mathcal{C}$. $\mathsf{Sim}_\mathcal{C}(\mathbf{s}^k_{k\in[2,T]}, \mathbf{t}^k_{\mathcal{C}\setminus P_1}, \mathbf{b})$ simulates the view of corrupt $\mathcal{C}$ as follows: $\mathsf{Sim}_\mathcal{C}$ simulates the view of corrupt $\mathcal{C}$ as follows: $\mathsf{Sim}_\mathcal{C}$ invokes TAHE.$\mathsf{TKeyGen}$ algorithm with other parties as the real protocol, and obtains (pk, sk_α). Upon receiving a ciphertext $\bar{c}^{\alpha-1}$, it simulates the view as follows: If $b_i = 0$, $i \in [m]$, it encrypts random values as the ciphertext $\bar{c}^\alpha$. If $b_i = 1$, $i \in [m]$, it encrypts zero as the ciphertext $\bar{c}^\alpha$. Upon receiving a ciphertext $\bar{c}^T$, it decrypts the ciphertext $\bar{c}^T$ as the real protocol. The IND-CPA security of TAHE guarantees that the view in simulation is indistinguishable from the view in the real protocol.
- $P_T \notin \mathcal{C}$**.** This can be divided into the following two cases.
  - $P_1 \notin \mathcal{C}$. This case has already been proven above.
  - $P_1 \in \mathcal{C}$. $\mathsf{Sim}_\mathcal{C}(\mathbf{s}^k_{k\in[2,T]}, \mathbf{t}^k_{\mathcal{C}\setminus P_1}, \mathbf{b})$ simulates the view of corrupt $\mathcal{C}$ as follows: $\mathsf{Sim}_\mathcal{C}$ simulates the view of corrupt $\mathcal{C}$ as follows: $\mathsf{Sim}_\mathcal{C}$ invokes TAHE.$\mathsf{TKeyGen}$ algorithm with other parties as the real protocol, and obtains (pk, sk_T). Upon receiving a ciphertext $\bar{c}^{T-1}$, it simulates the view as follows: If $b_i = 0$, $i \in [m]$, it encrypts random values as the ciphertext $\bar{c}^T$, and decrypts the ciphertext $\bar{c}^T$ as the real protocol. If $b_i = 1$,$i \in [m]$, it encrypts zero as the ciphertext $\bar{c}^T$, and decrypts the ciphertext $\bar{c}^T$ as the real protocol.

The IND-CPA security of TAHE guarantees that the view in simulation is indistinguishable from the view in the real protocol. □

Input: P_1 inputs a set $Y^1 = \{y_i^1\}_{i\in[m]}$. $P_k, k \in [2,T]$ input set $Y^k = \{y_j^k\}_{j\in[m_k]}, k \in [2,T]$.

Output: The receiver P_1 outputs $|card = \bigcap_{k=1}^T Y^k|$.

1) P_1 and P_k, $k \in [2,T]$ invoke cuckoo/simple hashing:
   a) P_1 inserts set Y^1 into the cuckoo hash table and fills empty bins with the dummy item $\perp$, where the cuckoo hash table $Y_c^1 = \{Y_c^1[i]\}_{i\in[m_c]}$ consists of m_c bins and each bin $Y_c^1[i]$ has only one item, where for each y_i^1 there is some $\gamma \in \{1,2,3\}$ such that $Y_c^1[h_\gamma(y_i^1)] = y_i^1||\gamma$.
   b) P_k, $k \in [2,T]$ uses the same hash functions to insert Y^k into the simple hash table, where all item y_j^k are concatenated with hash function indices $(y_j^k||1, y_j^k||2, y_j^k||3)$ and are inserted to the bins $(Y^k_{h_1(y_j)}, Y^k_{h_2(y_j)}, Y^k_{h_3(y_j)})$, respectively, the table consists of m_c bins $\{Y_1^k, Y_2^k, \cdots, Y_{m_c}^k\}$.
2) P_1 and P_k, $k \in [2,T]$ invoke the functionality $\mathcal{F}_{\text{bMCRG}}$ (balanced and unbalanced bMCRG can be selected according to the difference in the set size of P_1 and $P_k, k \in [2,T]$):
   a) P_1 inputs Y_c^1 and P_k, $k \in [2,T]$ inputs $\{Y_i^k\}_{i\in[m_c]}$.
   b) P_1 obtains $\mathbf{s}^k = [s_i^k]_{i\in[m_c]}$, $k \in [2,T]$ and P_k obtains $\mathbf{t}^k = [t_i^k]_{i\in[m_c]}$, $k \in [2,T]$.
3) P_1 and P_k, $k \in [2,T]$ invoke the functionality $\mathcal{F}_{\text{JP-PEQT}}$:
   a) P_1 inputs $\mathbf{s}^k = [s_i^k]_{i\in[m_c]}$, $k \in [2,T]$ and P_k inputs $\mathbf{t}^k = [t_i^k]_{i\in[m_c]}$, $k \in [2,T]$ and a permutation π_k, $k \in [2,T]$ over $[m_c]$.
   b) P_1 obtains $\mathbf{b} = [b_i]_{i\in[m_c]}$.
4) P_1 outputs $\sum_{i=1}^{m_c} b_i$.

Fig. 20. MPSI-Card from bMCRG and JP-PEQT

Instantiation. Here, we use threshold exponential ElGamal encryption scheme as the instantiation of TAHE [54], [55]. This scheme enjoys security against an arbitrary number of colluding parties. More importantly, our use of this scheme is not constrained by the inefficiency of exponential ElGamal decryption, as we only need to verify whether decryption result is zero, without requiring decryption for non-zero results.

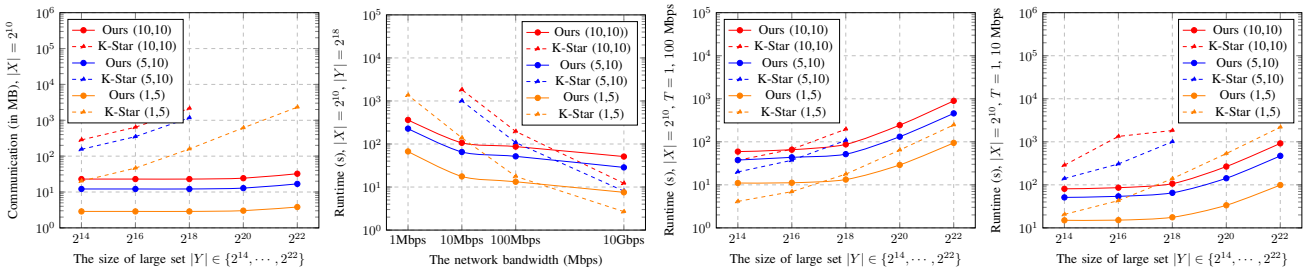

Fig. 21. Comparisons of communication and runtime between our MPSI and K-Star [29]. Both x and y-axis are in log scale. The first figure shows the communication cost increases as the large set size increases. The second figure shows the runtime decreases as the bandwidth increases. The last two figures show the runtime increases as the large set size increases.

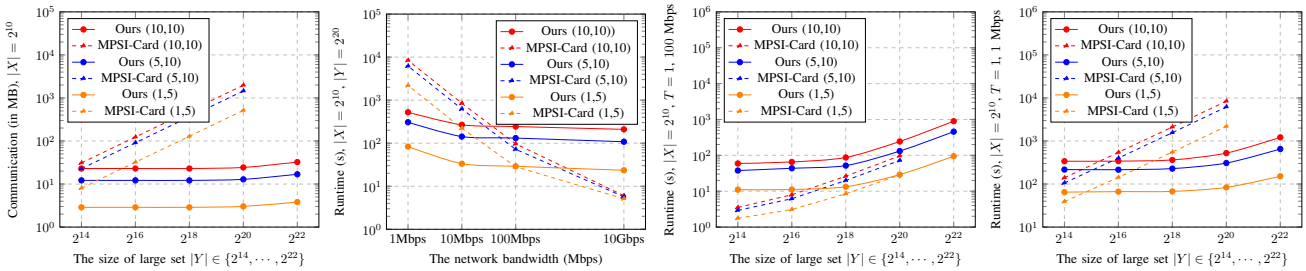

Fig. 22. Comparisons of communication and runtime between our MPSI-Card and MPSI-Card [31] in the unbalanced setting. Both x and y-axis are in log scale. (N, M) denotes the number of parties holding large (resp. small) is N (resp. M). The first figure shows the communication cost increases as the large set size increases. The second figure shows that the runtime decreases as the bandwidth increases. The last two figures show the runtime increases as the large set size increases.

Remark 2. *The construction in Figure 16 does not restrict the output result to P_1, because any participant, after receiving all decryption shares from the others, can execute the combine algorithm to verify whether the plaintext is zero or not. Hence, any participant can serve as the receiver, denoted as P_{rec}.*

V. A New Framework of MPSI

In this section, we present a new MPSI framework from bMCRG and J-PEQT as described in Figure 17.

Theorem 5. *The protocol Π_{MPSI} shown in Figure 17 realizes the functionality $\mathcal{F}_{\text{MPSI}}$ (as in Figure 5) in the ($\mathcal{F}_{\text{bMCRG}}$, $\mathcal{F}_{\text{J-PEQT}}$)-hybrid model, against semi-honest adversaries.*

Proof. Let $\mathcal{C}$ denote the set of all corrupted parties, where $|\mathcal{C}| = t < T$. Intuitively, the protocol is secure because all outputs of bMCRG are hidden by J-PEQT. Thereby, the simulator can easily simulate their outputs by encrypting random values that are independent of the inputs of honest parties.

- $P_1 \notin \mathcal{C}$. $\text{Sim}_{\mathcal{C}}(Y^{\mathcal{C}})$ simulates the view of corrupt $\mathcal{C}$ as follows: For all $P_k \in \mathcal{C}$, $\text{Sim}_{\mathcal{C}}$ runs simple hash to insert Y^k into hash table $\{Y_1^k, \cdots, Y_{m_c}^k\}$, and then chooses random vectors $\mathbf{t}^k$ and invokes $\text{Sim}_{\text{bMCRG}}^{P_k}(\{Y_1^k, \cdots, Y_{m_c}^k\}, \mathbf{t}^k)$ and appends the output to the view. After that, $\text{Sim}_{\mathcal{C}}$ invokes $\text{Sim}_{\text{J-PEQT}}^{\mathcal{C}}(\mathbf{t}_{P_k \in \mathcal{C}}^k)$ and appends the output to the view. The security of bMCRG and J-PEQT guarantees that the view in simulation is indistinguishable from the view in the real protocol.
- $P_\alpha \notin \mathcal{C}$. This can be divided into the following two cases.
  - $P_1 \notin \mathcal{C}$. This case has already been proven above.
  - $P_1 \in \mathcal{C}$. $\text{Sim}_{\mathcal{C}}(Y^{\mathcal{C}}, I)$ simulates the view of corrupt $\mathcal{C}$ as follows: $\text{Sim}_{\mathcal{C}}$ runs cuckoo hash to insert Y^1 into hash table $\{Y_1^1, \cdots, Y_{m_c}^1\}$, and then chooses random vectors $\mathbf{s}^\alpha$ and invokes $\text{Sim}_{\text{bMCRG}}^{P_\alpha}(\{Y_1^1, \cdots, Y_{m_c}^1\}, \mathbf{s}^\alpha)$ and appends the output to the view. $\text{Sim}_{\mathcal{C}}$ simulates other bMCRG protocols as the real protocols appends the output $\mathbf{s}_{k\in[2,T]}^k, \mathbf{t}_{\mathcal{C}\setminus P_1}^k$ to the view. After that, for $i \in [m_c]$, $\text{Sim}_{\mathcal{C}}$ sets $b_i = 1$, if $Y_i^1 \in I$, and obtains a bit vector $\mathbf{b} = [b_i]_{i\in[m_c]}$. Finally, $\text{Sim}_{\mathcal{C}}$ invokes $\text{Sim}_{\text{J-PEQT}}^{\mathcal{C}}$ $(\mathbf{s}_{k\in[2,T]}^k, \mathbf{t}_{\mathcal{C}\setminus P_1}^k, \mathbf{b})$ and appends the output to the view.

The security of bMCRG and J-PEQT guarantees that the view in simulation is indistinguishable from the view in the real protocol. □

Security against collusion. Since pairwise interactions in the bMCRG phase are immune to collusion, our MPSI inherits the security of J-PEQT. TAHE-based MPSI enjoys security against an arbitrary number of colluding parties. JZSS-based MPSI enjoys security against $T - 2$ colluding parties.

VI. Joint Permuted Private Equality Test

In this section, we introduce a new cryptographic protocol named the joint permuted private equality test (JP-PEQT). In Figure 18, we define the functionality of JP-PEQT, denoted as $\mathcal{F}_{\text{JP-PEQT}}$.

Theorem 6. *The construction of Figure 19 securely implements functionality $\mathcal{F}_{\text{JP-PEQT}}$ against a semi-honest adversary*

(N, M)	Size $(\lvert Y\rvert, \lvert X\rvert)$	Protocols	Comm. (MB)	Total running time (s) with single thread			
				10Gbps	100Mbps	10Mbps	1Mbps
$(1, 2)$	$(2^{22}, 2^{10})$	O-Ring	421.37	41.459	82.262	398.066	–
		K-Star	421.37	42.428	79.661	398.697	–
		ours	3.432	86.612	89.735	92.313	121.868
	$(2^{20}, 2^{10})$	O-Ring	111.97	9.485	21.996	104.977	976.904
		K-Star	111.97	9.667	22.318	106.409	962.161
		ours	2.635	21.281	24.264	26.738	54.141
	$(2^{18}, 2^{10})$	O-Ring	28.538	2.013	7.175	28.129	248.433
		K-Star	28.538	1.944	7.206	28.041	253.022
		ours	2.494	5.253	8.511	11.179	37.312
	$(2^{16}, 2^{10})$	O-Ring	8.11	0.407	3.883	9.858	73.365
		K-Star	8.11	0.388	3.848	9.727	73.371
		ours	2.494	3.485	6.436	8.709	35.076
	$(2^{14}, 2^{10})$	O-Ring	3.418	0.145	3.004	5.434	34.059
		K-Star	3.418	0.125	3.02	5.492	33.815
		ours	2.494	2.864	5.977	8.341	34.867
$(1, 5)$	$(2^{22}, 2^{10})$	O-Ring	2318.981	76.01	268.567	2017.932	–
		K-Star	2318.981	54.433	246.747	2184.695	–
		ours	3.816	88.526	94.423	98.964	151.542
	$(2^{20}, 2^{10})$	O-Ring	617.26	17.136	68.074	536.716	5343.444
		K-Star	617.26	12.196	64.291	533.313	5341.29
		ours	3.019	23.493	29.036	33.508	83.323
	$(2^{18}, 2^{10})$	O-Ring	158.324	3.551	18.604	139.344	1374.508
		K-Star	158.324	2.685	17.719	139.257	1363.314
		ours	2.878	7.542	13.272	17.624	67.125
	$(2^{16}, 2^{10})$	O-Ring	46.418	0.733	7.096	43.106	406.14
		K-Star	46.418	0.58	6.939	42.976	406.446
		ours	2.878	5.353	11.176	15.138	66.626
	$(2^{14}, 2^{10})$	O-Ring	20.231	0.189	4.079	20.441	187.3
		K-Star	20.231	0.169	4.127	20.582	184.507
		ours	2.878	5.151	11.023	14.998	64.65
$(5, 5)$	$(2^{22}, 2^{10})$	O-Ring/K-Star	–	–	–	–	–
		ours	16.224	429.676	445.273	456.808	598.564
	$(2^{20}, 2^{10})$	O-Ring	1920.668	30.892	188.019	1645.629	–
		K-Star	1920.668	19.183	191.438	1776.662	–
		ours	12.2	104.625	118.519	129.861	255.627
	$(2^{18}, 2^{10})$	O-Ring	493.729	6.48	49.71	426.785	4335.62
		K-Star	493.729	4.187	46.211	424.033	4355.783
		ours	11.514	24.603	39.569	51.776	175.642
	$(2^{16}, 2^{10})$	O-Ring	146.294	1.324	15.938	129.625	1275.621
		K-Star	146.294	1.035	15.741	128.305	1272.85
		ours	11.514	14.709	29.075	39.993	163.752
	$(2^{14}, 2^{10})$	O-Ring	64.462	0.366	8.407	58.996	565.677
		K-Star	64.462	0.289	8.29	59.918	571.568
		ours	11.514	12.806	27.618	38.335	163.319
$(5, 10)$	$(2^{22}, 2^{10})$	O-Ring/K-Star	–	–	–	–	–
		ours	16.865	433.655	457.668	470.921	650.899
	$(2^{20}, 2^{10})$	O-Ring/K-Star	–	–	–	–	–
		ours	12.841	108.818	131.508	142.277	307.192
	$(2^{18}, 2^{10})$	O-Ring	1173.918	11.072	113.82	1008.144	–
		K-Star	1173.918	8.055	107.918	1002.75	–
		ours	12.155	28.593	51.663	65.273	227.849
	$(2^{16}, 2^{10})$	O-Ring	349.568	2.418	38.466	306.232	3067.336
		K-Star	349.568	2.258	37.209	305.533	3050.333
		ours	12.155	19.64	43.568	54.063	215.691
	$(2^{14}, 2^{10})$	O-Ring	154.81	0.698	20.305	140.363	1352.134
		K-Star	154.81	0.647	19.93	139.975	1346.446
		ours	12.155	16.597	37.681	51.088	216.075
$(10, 10)$	$(2^{22}, 2^{10})$	O-Ring/K-Star	–	–	–	–	–
		ours	32.361	860.655	894.835	918.534	1214.008
	$(2^{20}, 2^{10})$	O-Ring/K-Star	–	–	–	–	–
		ours	24.28	209.621	243.665	266.244	523.134
	$(2^{18}, 2^{10})$	O-Ring	2144.032	16.303	203.446	1833.485	–
		K-Star	2144.032	12.228	196.274	1830.635	–
		ours	22.951	51.101	87.146	106.583	362.633
	$(2^{16}, 2^{10})$	O-Ring	639.984	3.784	69.437	582.465	–
		K-Star	639.984	3.08	67.055	1334.815	–
		ours	22.951	30.918	65.449	85.841	340.255
	$(2^{14}, 2^{10})$	O-Ring	284.111	7.14	38.874	357.459	–
		K-Star	284.111	12.228	37.114	285.834	–
		ours	22.951	26.346	59.247	80.87	338.316

TABLE II
COMPARISON OF COMMUNICATION AND RUNTIME BETWEEN OUR MPSI AND THE SOTA MPSI (O-RING/K-STAR) [29]. – INDICATES AN EXECUTION ERROR. THE BEST RESULTS OF OUR MPSI (RESP. [29]) ARE MARKED IN MAGENTA (RESP. BLUE).

colluding with up to any $t < T$ parties, provided that the TAHE scheme is IND-CPA secure.

Proof. The proof follows Theorem 4, requiring simulator to output a randomly shuffled ciphertexts. □

VII. A NEW FRAMEWORK OF MPSI-CARD

In this section, we present a new MPSI-Card framework from bMCRG and JP-PEQT as described in Figure 20.

Theorem 7. *The protocol $\Pi_{\text{MPSI-Card}}$ shown in Figure 20 realizes the functionality $\mathcal{F}_{\text{MPSI-Card}}$ (as in Figure 6) in the ($\mathcal{F}_{\text{bMCRG}}$, $\mathcal{F}_{\text{JP-PEQT}}$)-hybrid model, against a semi-honest adversary colluding with up to any $t < T$ parties.*

Proof. The proof is following Theorem 5. □

Security against collusion. Our MPSI-Card is similar to MPSI from bMCRG and J-PEQT and can be divided into two steps: bMCRG and JP-PEQT. Therefore, our TAHE-based MPSI-Card enjoys security against an arbitrary number of colluding parties.

Remark 3. *The construction in Figure 20 does not fully restrict the output result to P_1, because any participant, after receiving all decryption shares from the others, can execute the combine algorithm to verify whether the plaintext is zero or not and obtain the random indicated bit vector. Hence, any participant can serve as the receiver, denoted as P_{rec}.*

VIII. IMPLEMENTATION AND PERFORMANCE

Here, we evaluate our MPSI/MPSI-Card and then compare with the SOTA MPSI (O-Ring/K-Star [29]) and MPSI-Card [31], in terms of communication and runtime in different network environments.

A. Experimental Setup

We implement our protocols in C++ and the source code is available at https://doi.org/10.5281/zenodo.17927023 and https://github.com/real-world-cryptography/MinBucket-MPSI. The experiments are conducted on a single Intel Core i7-13700 CPU @ 5.20GHz with 16 threads and 32GB of RAM, running Ubuntu. We evaluate our protocols in two network settings: LAN (10Gbps bandwidth, 0.04ms RTT) and WAN (100Mbps, 10Mbps, and 1Mbps bandwidth, 80ms RTT), emulated using the Linux tc command. We leverage the constructions in [35], [58], [20], [57] to implement bOPRF, OKVS, ss-PEQT, ROT, FHE, and use the code from the Vole-PSI library [59], the SEAL library [56], and the APSI library [60]. As for DDH-based TAHE, we follow the OpenSSL library [61]. Our code supports multithreading parallelism following the Vole-PSI library and the OpenMP library [62]. We set the computational security parameter $\kappa = 128$, the statistical security parameter $\lambda = 40$, and use $\gamma = 3$ hash functions in cuckoo/simple hashing. The item length is 64 bits following [42], [43].

We configure the unbalanced multi-party setting as follows: in evaluating the impact of input set sizes, the size of the small set is fixed at 2^{10}, while the size of the large set varies from 2^{14} to 2^{22}, allowing us to evaluate protocol performance under increasing input asymmetry. To analyze scalability with

respect to the number of participants, we consider configurations ranging from 3 parties $(1, 2)$ to 20 parties $(10, 10)$. Since MPSI [29] and MPSI-Card [31] do not support unbalanced input scenarios, all datasets are padded to balanced set sizes for evaluation. Moreover, as the implementations of [29], [31] do not support multi-threading, we report only single-threaded results in Table II and Table III.

B. Performance Comparisons of MPSI

We compare our MPSI with SOTA works (O-Ring/K-Star) [29], and show the results in Table II and Figure 21.

Communication comparison. As shown in Figure 21, regardless of the number of participants, our MPSI consistently demonstrates significantly lower communication overhead[7] than [29][8]. As indicated in Table II, our protocol reduces communication costs by a factor of 1.37 to 607.7. In particular, for set size of $(2^{22}, 2^{10})$ with participants $(1, 5)$, our MPSI protocol requires 3.816 MB, which is about 607× lower than MPSI [29], which requires 2318.981 MB. Since the communication of MPSI [29] is linear in the size of the large set, while the communication of our MPSI is logarithmic in the size of the large set and linear in the size of the small set, the greater the difference in set sizes, or the more participants holding the small set, the more significant the advantage of our MPSI becomes.

Runtime comparison. As shown in Figure 21, our protocol outperforms MPSI [29] in terms of runtime in low-bandwidth environments. Our protocol achieves better performance when there are more participants holding small sets, such as in the $(5, 10)$ and $(10, 10)$ scenarios. As indicated in Table II, for large set sizes from 2^{14} to 2^{16}, the runtime of our protocol surpasses MPSI [29] by a factor of 3.5 to 15.5 depending on the network environment. For large set sizes $\geq 2^{18}$, the runtime of our protocol outperforms MPSI [29] by a factor of 2.5 to 64.1, regardless of the number of participants. Specifically, for set size $(2^{20}, 2^{10})$ with $(1, 5)$ participants under 1 Mbps bandwidth, our MPSI requires 83.323 seconds, 64× faster than MPSI [29], which takes 5343.444 seconds.

(N, M)	Size $(\|Y\|, \|X\|)$	Protocols	Comm. (MB)	Total running time (s) with single thread			
				10Gbps	100Mbps	10Mbps	1Mbps
$(1, 2)$	$(2^{22}, 2^{10})$	MPSI-Card	–	–	–	–	–
		ours	3.432	86.618	89.745	92.267	121.919
	$(2^{20}, 2^{10})$	MPSI-Card	200.644	4.956	14.054	89.313	855.439
		ours	2.635	21.286	23.363	26.646	54.2
	$(2^{18}, 2^{10})$	MPSI-Card	50.164	0.221	4.362	22.592	215.332
		ours	2.494	5.259	8.506	11.101	37.39
	$(2^{16}, 2^{10})$	MPSI-Card	12.544	0.189	2.127	6.587	54.75
		ours	2.494	3.379	6.426	8.625	35.119
	$(2^{14}, 2^{10})$	MPSI-Card	3.139	0.026	1.26	2.106	14.692
		ours	2.494	2.861	6.178	8.342	34.895
$(1, 5)$	$(2^{22}, 2^{10})$	MPSI-Card	–	–	–	–	–
		ours	3.816	88.535	94.437	99.033	151.652
	$(2^{20}, 2^{10})$	MPSI-Card	514.566	5.116	28.424	222.393	2190.1
		ours	3.019	23.496	29.015	33.358	83.446
	$(2^{18}, 2^{10})$	MPSI-Card	128.646	0.232	8.471	56.464	553.204
		ours	2.878	7.546	13.294	17.727	67.46
	$(2^{16}, 2^{10})$	MPSI-Card	32.166	0.194	3.083	15.133	142.632
		ours	2.878	5.407	11.14	15.261	66.598
	$(2^{14}, 2^{10})$	MPSI-Card	8.046	0.032	1.755	4.517	39.192
		ours	2.878	5.151	11.069	15.101	64.808
$(5, 5)$	$(2^{22}, 2^{10})$	MPSI-Card	–	–	–	–	–
		ours	16.224	429.703	445.268	457.023	598.786
	$(2^{20}, 2^{10})$	MPSI-Card	933.131	5.299	48.14	400.062	3968.689
		ours	12.2	104.002	118.471	130.048	255.729
	$(2^{18}, 2^{10})$	MPSI-Card	233.29	0.317	13.557	101.217	1000.285
		ours	11.514	24.634	39.571	51.621	175.955
	$(2^{16}, 2^{10})$	MPSI-Card	58.33	0.199	4.502	26.942	257.212
		ours	11.514	14.681	29.048	39.755	163.929
	$(2^{14}, 2^{10})$	MPSI-Card	14.59	0.035	2.279	7.802	69.96
		ours	11.514	12.787	27.657	38.05	163.523
$(5, 10)$	$(2^{22}, 2^{10})$	MPSI-Card	–	–	–	–	–
		ours	16.865	433.665	457.653	470.556	651.238
	$(2^{20}, 2^{10})$	MPSI-Card	1456.339	5.694	72.112	622.85	6195.436
		ours	12.841	108.061	131.404	142.579	307.034
	$(2^{18}, 2^{10})$	MPSI-Card	364.099	0.417	19.871	156.983	1559.042
		ours	12.155	28.626	51.777	65.301	228.438
	$(2^{16}, 2^{10})$	MPSI-Card	91.039	0.212	6.177	41.218	400.575
		ours	12.155	19.329	43.583	53.754	215.366
	$(2^{14}, 2^{10})$	MPSI-Card	22.774	0.036	2.924	11.637	106.019
		ours	12.155	16.643	37.72	51.254	216.594
$(10, 10)$	$(2^{22}, 2^{10})$	MPSI-Card	–	–	–	–	–
		ours	32.361	860.647	894.863	918.044	1214.168
	$(2^{20}, 2^{10})$	MPSI-Card	1979.55	6.057	96.654	844.843	8419.381
		ours	24.28	209.621	243.722	267.577	522.972
	$(2^{18}, 2^{10})$	MPSI-Card	494.91	0.552	26.189	213.509	2118.025
		ours	22.951	51.198	87.175	106.226	363.087
	$(2^{16}, 2^{10})$	MPSI-Card	123.75	0.259	7.95	55.902	544.023
		ours	22.951	30.074	65.458	86.036	340.548
	$(2^{14}, 2^{10})$	MPSI-Card	30.96	0.039	3.522	15.624	139.693
		ours	22.951	26.319	59.274	81.099	338.859

TABLE III
COMPARISONS OF COMMUNICATION AND RUNTIME BETWEEN OUR MPSI-CARD AND THE SOTA MPSI-CARD [31]. – INDICATES AN EXECUTION ERROR. THE BEST RESULTS OF OUR MPSI-CARD (RESP. [31]) ARE MARKED IN MAGENTA (RESP. BLUE).

C. Performance Comparisons of MPSI-Card

We compare our MPSI-Card with the SOTA work [31], and the results are reported in Table III and Figure 22.

Communication comparison. As shown in Figure 22, our protocol consistently achieves better communication efficiency than MPSI-Card [31]. As indicated in Table III, our protocol achieves a 1.3× to 170.4× reduction in communication costs across all tested settings. In particular, for set size $(2^{20}, 2^{10})$ with participants $(1, 5)$, our MPSI-Card requires 3.019 MB, which is 170× lower than MPSI-Card [31], which requires 514.566 MB. Since the communication of MPSI-Card [31] is linear in the size of the large set, while the communication of our MPSI-Card is logarithmic in the size of the large set and linear in the size of the small set, the greater the difference in set sizes, or the more participants holding the small set, the more significant the advantage of our MPSI-Card becomes.

Runtime comparison. As shown in Figure 22, our protocol outperforms MPSI-Card [31] in terms of runtime in low-bandwidth environments. Our protocol achieves better performance when there are more participants holding small sets, such as in the (5,10) and (10,10) settings. As indicated in Table III, for the large set sizes from 2^{14} to 2^{18}, the runtime of our protocol surpasses MPSI-Card [31] by a factor of 1.5 to 8.2 under the low-bandwidth environments with many

[7]Since both the x- and y-axes are log-scaled, and our protocol exhibits logarithmic growth with the large set size, the resulting performance curve appears approximately flat, reflecting the logarithmic communication complexity with the large set size.

[8]Note that K-Star exhibits marginally better runtime than O-Ring, so we selected K-Star for comparison in Figure 21.

participants. For the large set sizes $\geq 2^{20}$, the runtime of our protocol outperforms MPSI-Card [31] by a factor of 1.03 to 26.24, regardless of the number of participants. Specifically, for set size $(2^{20}, 2^{10})$ with $(1, 5)$ participants under 1 Mbps bandwidth, our MPSI-Card requires 83.446 seconds, 26× faster than MPSI-Card [31], which takes 2190.1 seconds.

D. Performance of Our Sub-Protocols

We evaluate the performance of our J-PEQT and JP-PEQT, along with their communication and computation complexity, in Table IV.

Subprotocols	(N, M)	Comm. (MB)	Total running time (s) with single thread			
			10Gbps	100Mbps	10Mbps	1Mbps
J-PEQT	$(1, 2)$	0.413	1.253	1.899	2.286	7.148
JP-PEQT			1.256	1.919	2.323	7.195
J-PEQT	$(1, 5)$	0.722	2.883	4.314	5.157	17.096
JP-PEQT			2.888	4.337	5.292	17.286
J-PEQT	$(5, 5)$	1.135	5.063	7.574	8.89	31.286
JP-PEQT			5.089	7.643	9.152	31.426
J-PEQT	$(5, 10)$	1.651	7.789	11.577	13.662	48.555
JP-PEQT			7.809	11.608	13.966	48.922
J-PEQT	$(10, 10)$	2.168	10.551	15.685	19.166	65.604
JP-PEQT			10.593	15.713	19.395	66.147

TABLE IV
COMMUNICATION AND RUNTIME OF OUR SUB-PROTOCOLS J-PEQT AND JP-PEQT. 10GBPS BANDWIDTH, 0.04MS RTT; 100MBPS, 10MBPS AND 1MBPS BANDWIDTH, 80MS RTT. (N, M) DENOTES THE NUMBER OF PARTIES HOLDING LARGE (RESP. SMALL) IS N (RESP. M).

E. Discussions of Our Performance Comparisons

In this work, we focus on unbalanced multi-party scenarios, where M parties hold small input sets and N parties hold large input sets. Our protocols achieve communication complexity that scales linearly with the size of the small sets and logarithmically with the size of the large sets. In contrast, in balanced multi-party scenarios where all parties hold comparable-sized sets, this advantage diminishes, and our protocols are therefore not well-suited for such settings. Experimental results, presented in Table II and Table III, demonstrate that our MPSI and MPSI-Card protocols achieve significant improvements in communication efficiency compared to the SOTA protocols [29], [31], and they exhibit notably shorter execution times in bandwidth-constrained environments. However, under bandwidth-unconstrained settings, their execution efficiency becomes relatively lower. This is due to the underlying design differences: our sub-protocols, J-PEQT/JP-PEQT and the unbalanced bMCRG, are constructed from public-key cryptographic components (e.g., TAHE and FHE), whereas the protocols in [29], [31] primarily rely on lightweight symmetric-key components.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful comments. This work is supported by the National Natural Science Foundation of China (No. 62502278, 62272269), Scientific Research Innovation Capability Support Project for Young Faculty (No. ZYGXQNJSKYCXNLZCXMI21), Taishan Scholar Program of Shandong Province, and Shandong Postdoctoral Science Foundation (No. SDBX2024024).

ETHICS CONSIDERATIONS AND COMPLIANCE WITH THE OPEN SCIENCE POLICY

Ethics Considerations. We strictly follow the ethical guidelines set forth by NDSS Symposium. Our research does not involve any ethical issues. All experiments conducted in this paper are based on publicly available datasets and do not involve personal or sensitive data, ensuring full compliance with privacy and data protection standards.

Compliance with Open Science Policy. We fully support the principles of the Open Science Policy. We have incorporated our research artifacts into an open-source repository https://doi.org/10.5281/zenodo.17927023, ensuring transparency and reproducibility. By adhering to the open science principles, we support the broad dissemination of scientific knowledge and facilitate further research in our field. Our approach aligns to enhance the reproducibility and reliability of scientific findings, contributing to a more open and collaborative research environment.

REFERENCES

[1] M. Ion, B. Kreuter, E. Nergiz, S. Patel, S. Saxena, K. Seth, D. Shanahan, and M. Yung, "Private intersection-sum protocol with applications to attributing aggregate ad conversions," *IACR Cryptol. ePrint Arch.*, p. 738, 2017.

[2] M. Ion, B. Kreuter, A. E. Nergiz, S. Patel, S. Saxena, K. Seth, M. Raykova, D. Shanahan, and M. Yung, "On deploying secure computing: Private intersection-sum-with-cardinality," in *IEEE European Symposium on Security and Privacy*. IEEE, 2020, pp. 370–389.

[3] D. Kales, C. Rechberger, T. Schneider, M. Senker, and C. Weinert, "Mobile private contact discovery at scale," in *USENIX Security Symposium*, 2019, pp. 1447–1464.

[4] D. Demmler, P. Rindal, M. Rosulek, and N. Trieu, "PIR-PSI: scaling private contact discovery," *Proc. Priv. Enhancing Technol.*, vol. 2018, no. 4, pp. 159–178, 2018.

[5] 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*. Springer, 2019, pp. 401–431.

[6] M. Ciampi and C. Orlandi, "Combining private set-intersection with secure two-party computation," in *Security and Cryptography for Networks*. Springer, 2018, pp. 464–482.

[7] B. H. Falk, D. Noble, and R. Ostrovsky, "Private set intersection with linear communication from general assumptions," in *ACM Workshop on Privacy in the Electronic Society*. ACM, 2019, pp. 14–25.

[8] A. Cerulli, E. D. Cristofaro, and C. Soriente, "Nothing refreshes like a repsi: Reactive private set intersection," in *Applied Cryptography and Network Security*. Springer, 2018, pp. 280–300.

[9] C. Hazay and M. Venkitasubramaniam, "Scalable multi-party private set-intersection," in *Public-Key Cryptography*. Springer, 2017, pp. 175–203.

[10] P. Rindal and M. Rosulek, "Improved private set intersection against malicious adversaries," in *Advances in Cryptology - EUROCRYPT*, 2017, pp. 235–259.

[11] B. Pinkas, T. Schneider, and M. Zohner, "Scalable private set intersection based on OT extension," *ACM Trans. Priv. Secur.*, 2018.

[12] ——, "Faster private set intersection based on OT extension," in *USENIX Security Symposium*, 2014, pp. 797–812.

[13] C. Dong, L. Chen, and Z. Wen, "When private set intersection meets big data: an efficient and scalable protocol," in *ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2013, pp. 789–800.

[14] Y. Huang, D. Evans, and J. Katz, "Private set intersection: Are garbled circuits better than custom protocols?" in *Network and Distributed System Security Symposium*. The Internet Society, 2012.

[15] H. Chen, K. Laine, and P. Rindal, "Fast private set intersection from homomorphic encryption," in *ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1243–1255.

[16] Á. Kiss, J. Liu, T. Schneider, N. Asokan, and B. Pinkas, "Private set intersection for unequal set sizes with mobile applications," *Proc. Priv. Enhancing Technol.*, vol. 2017, no. 4, pp. 177–197, 2017.

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

[18] A. C. D. Resende and D. F. Aranha, "Faster unbalanced private set intersection," pp. 203–221, 2018.

[19] 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 *ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2021, pp. 1135–1150.

[20] S. Raghuraman and P. Rindal, "Blazing fast PSI from improved OKVS and subfield VOLE," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2022, pp. 2505–2517.

[21] S. Mishima, K. Nakasho, K. Takeuchi, N. Hayaishi, Y. Takano, and A. Miyaji, "Development and application of privacy-preserving distributed medical data integration system," in *IEEE International Conference on Consumer Electronics*. IEEE, 2020, pp. 1–2.

[22] A. Miyaji, K. Nakasho, and S. Nishida, "Privacy-preserving integration of medical data - A practical multiparty private set intersection," *J. Medical Syst.*, vol. 41, no. 3, pp. 37:1–37:10, 2017.

[23] D. T. Nguyen and N. Trieu, "Mpccache: Privacy-preserving multi-party cooperative cache sharing at the edge," in *Financial Cryptography and Data Security*, 2022.

[24] R. Inbar, E. Omri, and B. Pinkas, "Efficient scalable multiparty private set-intersection via garbled bloom filters," in *Security and Cryptography for Networks*, 2018.

[25] A. Bay, Z. Erkin, J. Hoepman, S. Samardjiska, and J. Vos, "Practical multi-party private set intersection protocols," *IEEE Trans. Inf. Forensics Secur.*, vol. 17, pp. 1–15, 2022.

[26] S. Ramanathan, J. Mirkovic, and M. Yu, "BLAG: improving the accuracy of blacklists," in *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.

[27] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu, "Practical multi-party private set intersection from symmetric-key techniques," in *ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1257–1272.

[28] O. Nevo, N. Trieu, and A. Yanai, "Simple, fast malicious multiparty private set intersection," in *ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1151–1165.

[29] M. Wu, T. H. Yuen, and K. Y. Chan, "O-ring and k-star: Efficient multi-party private set intersection," in *33rd USENIX Security Symposium, USENIX Security 2024*. USENIX Association, 2024.

[30] Y. Gao, Y. Luo, L. Wang, X. Liu, L. Qi, W. Wang, and M. Zhou, "Efficient scalable multi-party private set intersection(-variants) from bicentric zero-sharing," in *ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2024, pp. 4137–4151.

[31] J. Gao, N. Trieu, and A. Yanai, "Multiparty private set intersection cardinality and its applications," *Proc. Priv. Enhancing Technol.*, vol. 2024, no. 2, pp. 73–90, 2024.

[32] X. Yang, L. Cai, Y. Wang, K. Yin, L. Sun, and J. Hu, "Efficient unbalanced quorum PSI from homomorphic encryption," in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*. ACM, 2024.

[33] R. Pagh and F. F. Rodler, "Cuckoo hashing," in *Algorithms - ESA 2001, 9th Annual European Symposium*, 2001, pp. 121–133.

[34] 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*. ACM, 2023, pp. 2959–2973.

[35] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient batched oblivious PRF with applications to private set intersection," in *ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 818–829.

[36] 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*, 2005.

[37] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[38] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract)," in *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*. ACM, 1988, pp. 1–10.

[39] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Robust threshold DSS signatures," in *Advances in Cryptology - EUROCRYPT*, 1996.

[40] N. Chandran, D. Gupta, and A. Shah, "Circuit-psi with linear complexity via relaxed batch OPPRF," *Proc. Priv. Enhancing Technol.*, vol. 2022, no. 1, pp. 353–372, 2022.

[41] G. Couteau, "New protocols for secure equality test and comparison," in *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, 2018.

[42] X. Liu and Y. Gao, "Scalable multi-party private set union from multi-query secret-shared private membership test," in *Advances in Cryptology - ASIACRYPT*, 2023.

[43] M. Dong, Y. Chen, C. Zhang, and Y. Bai, "Breaking free: Efficient multi-party private set union without non-collusion assumptions," *IACR Cryptol. ePrint Arch.*, p. 1146, 2024.

[44] M. O. Rabin, "How to exchange secrets with oblivious transfer," *IACR Cryptol. ePrint Arch.*, p. 187, 2005.

[45] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "PSI from paxos: Fast, malicious private set intersection," in *Advances in Cryptology - EUROCRYPT*, 2020.

[46] 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.

[47] 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*. USENIX Association, 2023, pp. 301–318.

[48] 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 2023*. USENIX Association, 2023.

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

[50] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai, "Efficient circuit-based PSI with linear communication," in *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2019, pp. 122–153.

[51] G. Garimella, P. Mohassel, M. Rosulek, S. Sadeghian, and J. Singh, "Private set operations from oblivious switching," in *Public-Key Cryptography*. Springer, 2021, pp. 591–617.

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

[53] Y. Jia, S. Sun, H. Zhou, and D. Gu, "Scalable private set union, with stronger security," in *USENIX Security Symposium*. USENIX Association, 2024.

[54] J. Katz and Y. Lindell, *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

[55] T. C. I. JTC, *ISO/IEC 18033-6:2019(en) IT Security techniques — Encryption algorithms — Part 6: Homomorphic encryption*. Netherlands Standards, 2019.

[56] "https://github.com/microsoft/SEAL."

[57] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, p. 144, 2012.

[58] P. Rindal and P. Schoppmann, "VOLE-PSI: fast OPRF and circuit-psi from vector-ole," in *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, ser. Lecture Notes in Computer Science, vol. 12697. Springer, 2021, pp. 901–930.

[59] "https://github.com/Visa-Research/volepsi."

[60] "https://github.com/microsoft/APSI."

[61] "https://github.com/openssl/openssl."

[62] "https://www.openmp.org/."

APPENDIX A
ARTIFACT APPENDIX

A. Description & Requirements

1) How to access: We have open-sourced the implementation at https://github.com/real-world-cryptography/MinBucket-MPSI. In addition, our artifact is publicly available at https://doi.org/10.5281/zenodo.17927023.

2) Hardware dependencies: The implementation can be executed on a commodity computer. All experiments in this paper were performed on a single Intel Core i7-13700 CPU @ 5.20GHz with 16 threads and 32GB of RAM.

3) Software dependencies: Our implementation requires G++ 11.4.0 and Python 3.10.12, and is currently supported on Ubuntu 22.04. Our implementation relies on the following libraries: the Vole-PSI [59], SEAL [56], OpenSSL [61], and OpenMP [62].

4) Benchmarks: We set the computational security parameter $\kappa = 128$, the statistical security parameter $\lambda = 40$. The item length is 64-bit following [43].

B. Artifact Installation & Configuration

1) Installation: Instructions for installing the required dependencies are provided in https://github.com/real-world-cryptography/MinBucket-MPSI. Evaluators can complete the installation by following the steps described in the accompanying README file. In particular, a **ready-to-use Docker image** is included to streamline the experimental setup.

2) Code Execution Instructions: Our experiments support automated testing via scripts. For details on how to run tests, please refer to *E.Evaluation*.

C. Experiment Workflow

Our implementation supports both the MPSI and MPSI-Card protocols, which consist of three core components: balanced bMCRG, unbalanced bMCRG, and J-PEQT/JP-PEQT. The overall source code structure is summarized as follows:

- MPSI: Our MPSI protcocl is built upon the balanced/unbalanced bMCRG and J-PEQT. The implementation is organized within the "MinBucket-MPSI/" directory as follows:
  - The "MCRG/" subdirectory implements the balanced bMCRG protocol (Fig. 8. bMCRG from bOPRF and OKVS).
  - The "uMCRG/" subdirectory implements the unbalanced bMCRG protocol (Fig. 9. bMCRG from bOPRF and FHE). For processing large sets (size $\geq 2^{20}$), the relevant components are located in "uMCRG/MCRG_diff_large" folder.
  - The J-PEQT protocol (Fig. 12. TAHE-based J-PEQT) is located in "uMCRG/JPEQT/" folder.
- MPSI-Card: Our MPSI-Card protocol is built upon the balanced/unbalanced bMCRG and JP-PEQT. It inherits the bMCRG directory structure from the MPSI implementation. The code for JP-PEQT (Fig. 15. TAHE-based JP-PEQT) resides in the "uMCRG/JPEQT/" folder.

For specific details, please refer to Section *E.Evaluation.*

D. Major Claims

- (C1): Our MPSI protocol reduces communication costs by 1.37× - 607.7× and improves runtime by 3.5× - 15.5× compared with the state-of-the-art MPSI [29]. This is demonstrated by experiment (E1), whose results are reported in Table II.
- (C2): Our MPSI-Card protocol reduces communication costs by 1.3× - 170.4× and improves runtime by 1.03× - 26.24× compared with the state-of-the-art MPSI-Card [31]. This is demonstrated by experiment (E2), whose results are reported in Table III.

E. Evaluation

1) Experiment (E1): [MPSI performance][15 human-minutes + 15 compute-minutes] This experiment evaluates our MPSI protocol, which includes the balanced/unbalanced bMCRG and J-PEQT, and reports both runtime and communication costs. As shown in Table II, we instantiate a six-party scenario in which one large-set participant holds a set of size 2^{14}, while each of the five small-set participants holds a set of size 2^{10}.

[How to] First, execute the shell script in the root directory to set up the test environment. Then, run the generated module test scripts sequentially to measure runtime and communication cost. Detailed procedures are provided in *[Execution]*.

[Preparation] We evaluate our protocols in two network settings: LAN (10Gbps bandwidth, 0.04ms RTT) and WAN (100Mbps, 10Mbps, and 1Mbps bandwidth, 80ms RTT), emulated using the Linux tc command. For example, open a terminal, and execute the following command: `tc qdisc add dev lo root netem delay 0.02ms rate 10Gbit` to configure the local network as 10Gbit bandwidth with 0.04ms RTT. Evaluators can adjust the network settings using different parameters as needed, such as 100Mbps, 10Mbps, and 1Mbps bandwidth, 80ms RTT.

Our MPSI protocol supports multiple participants with heterogeneous set sizes, where the size of each small set is fixed at 2^{10}, and the size of the large sets ranges from 2^{14} to 2^{20}. Evaluators can configure the number of participants by modifying `big_receiver_num` (number of large-set participants) and `small_receiver_num` (number of small-set participants minus one) in the `auto_test.sh` script. To adjust the size of the large sets, modify the parameter `pow(2,14)` in `auto_prepare.py` located in the "MinBucket-MPSI/uMCRG/MCRG" directory. The supported large set sizes are listed in Table II, while the small set size remains fixed in this experiment.

[Execution] Open a terminal and execute the following commands sequentially. First, configure the local network with 10Gbit bandwidth and 0.04ms RTT using: `tc qdisc add dev lo root netem delay 0.02ms rate 10Gbit`.

- Navigate to the "MinBucket-MPSI/" directory and run `./auto_test.sh` to generate the module test scripts. Next, execute the test scripts in the following or-

der: `./run_uMCRG.sh`, `./run_MCRG.sh`, and finally `./run_J-PEQT.sh`.
- We additionally provide automated Python scripts to facilitate experimental evaluation. For example, executing `python3 MPSI_auto_script.py -nn 14 -big 1 -small 5` runs an experiment with a large input set of size 2^{14}, one large-set participant, and five small-set participants.

[Results] The total running time of MPSI is the sum of the output by the script, and the total communication cost of the protocol is the same. After executing the above commands, the experiment prints information as follows:

- `./run_uMCRG.sh`: Sum the following communication cost and runtime to obtain the total cost of unbalanced MCRG.
  - `Communication total: 1896 KB` This indicates that the communication cost of the first part of unbalanced MCRG is 1896 KB. When there are α large-set participants, the corresponding output information will be repeated α times; add them together.
  - `Total Comm cost = 0.204 MB` This indicates that the communication cost of the second part of unbalanced MCRG is 0.204 MB. Regardless of the number of participants in the large set, this information appears only once and can be read directly.
  - `receiver all time1492.90` This indicates that the runtime of the first part of unbalanced MCRG is 1492.90 ms. When there are α large-set participants, the corresponding output information will be repeated α times; add them together.
  - `MCRG 86.0 85.913 **********` This indicates that the runtime of the second part of unbalanced MCRG is 85.913 ms. Regardless of the number of participants in the large set, this information appears only once and can be read directly.
- `./run_MCRG.sh`:
  - `Total Comm cost = 0.100 MB` This indicates that the communication cost of balanced MCRG is 0.100 MB.
  - `pMCRG 688.0 687.957 **********` This indicates that the runtime of balanced MCRG is 796.207 ms.
- `./run_J-PEQT.sh`:
  - `Total Comm cost = 0.722 MB` This indicates that the communication cost of J-PEQT is 0.722 MB.
  - `end000 0002883.0 02883.002 **********` This indicates that the runtime of J-PEQT is 2883.002 ms.

2) Experiment (E2): [MPSI-Card performance][15 human-minutes + 15 compute-minutes] This experiment can test our MPSI-Card protocol, including balanced bMCRG, unbalanced bMCRG, and JP-PEQT, and output the runtime and communication cost. Our MPSI-Card retains all settings from MPSI (E1). The sole modification is the substitution of J-PEQT with JP-PEQT to enable the computation of the intersection cardinality.

[How to] Run the master script and the scripts it generates, like experiment E1.

[Preparation] Consistent with experiment E1.

[Execution] If experiment E1 has already been executed, run `./run_JP-PEQT.sh` to obtain the cardinality results using the original sets from E1. Otherwise, to perform a complete test, follow these steps: First, set up the network environment by running `tc qdisc add dev lo root netem delay 0.02ms rate 10Gbit` to configure the local network with 10Gbit bandwidth and 0.04ms RTT.

- Navigate to the "MinBucket-MPSI/" directory and run `./auto_test.sh` to generate the module test scripts. Then, execute the generated scripts in the following sequence: `./run_uMCRG.sh`, `./run_MCRG.sh`, and `./run_JP-PEQT.sh`.
- We additionally provide automated Python scripts to facilitate experimental evaluation. For example, executing `python3 MPSICA_auto_script.py -nn 14 -big 1 -small 5` runs an experiment with a large input set of size 2^{14}, one large-set participant, and five small-set participants.

[Results] The runtime and communication overhead of MPSI-Card comprises the costs from the balanced MCRG, unbalanced MCRG, and JP-PEQT components. Since the outputs of the first two scripts (balanced MCRG and unbalanced MCRG) are detailed in the *[Result]* section of E1, we focus here on explaining only the output of the third script.

- `./run_JP-PEQT.sh`:
  - `Total Comm cost = 0.722 MB` This indicates that the communication cost of JP-PEQT is 0.722 MB.
  - `end000 0002888.0 02888.015 **********` This indicates that the runtime of JP-PEQT is 2888.015 ms.
  - `The size of the intersection of 6 parties is:2` This indicates that the intersection cardinality of the six-party is 2.

F. Acknowledgement

We would like to express our sincere gratitude to the AE reviewers for their thorough evaluation of our work.