O-Ring and K-Star: Efficient Multi-party Private Set Intersection

Mingli Wu, *The University of Hong Kong;* Tsz Hon Yuen, *Monash University;* Kwan Yin Chan, *The University of Hong Kong*

https://www.usenix.org/conference/usenixsecurity24/presentation/wu-mingli

This paper is included in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

O-Ring and K-Star: Efficient Multi-party Private Set Intersection

Mingli Wu
The University of Hong Kong
mingliwu@connect.hku.hk

Tsz Hon Yuen
Monash University
john.tszhonyuen@monash.edu

Kwan Yin Chan
The University of Hong Kong
kychan@cs.hku.hk

Abstract

Multi-party private set intersection (mPSI) securely enables multiple parties to know the intersection of their sets without disclosing anything else. Many mPSI protocols are not efficient in practice. In this paper, we propose two efficient mPSI protocols that are secure against an arbitrary number of colluding parties. In the protocol *O-Ring*, we take advantage of the ring network topology such that the communication costs of the party with the largest workload can be cheaper than other mPSI protocols with a star topology. In the protocol *K-Star*, we take advantage of the star topology to support better concurrency such that the protocol can run fast. K-Star is suitable for applications with a powerful centralized server. Different from KMPRT (CCS'17) and CDGOSS (CCS'21) that rely on *Oblivious Programmable PRF* primitive, we simply utilize the cheaper Oblivious PRF (OPRF) and a data structure Oblivious Key-value Store (OKVS). We further propose two fine-grained optimizations for OKVS and OPRF in multi-party cases to improve runtime performance.

After extensive experiments, we demonstrate that both protocols run the fastest and achieve the lowest total communication costs compared with the state-of-the-art counterparts in most settings. Specifically, O-Ring/K-Star is respectively $1.6\times \sim 48.3\times$ and $4.0\times \sim 39.8\times$ (except one setting) cheaper than KMPRT (CCS'17) and CDGOSS (CCS'21) in the total communication costs. For the total running time, K-Star can be respectively $1.4\times \sim 9.0\times$ and $1.0\times \sim 15.3\times$ as fast as them in the LAN setting.

1 Introduction

Two-party private set intersection (PSI) allows two parties (i.e., a receiver and a sender), each of them has a set, to secretly know the intersection of their sets without disclosing anything else. Specifically, a receiver gets the intersection without disclosing any item of its set to the sender; the sender discloses no more than the intersection to the receiver. Different from two-party PSI, multi-party PSI (mPSI) extends the number of parties to $n \geq 3$, which makes it more challenging. Two-party PSI does not need to consider collusion attacks. However, the attack becomes more complicated when it comes to multiple parties. Given n parties, the number of colluding parties can be any t out of them. This explains that though there are many efficient two-party PSI protocols, there are relatively fewer mPSI protocols (e.g., [2, 4, 17]).

mPSI have many applications, e.g., medical data integration [19, 20], cache sharing in edge computation [22], network intrusion attack detection [13], and highly risky individual identifications in the spread of disease [1]. Among the proposed mPSI protocols, some of them are not concretely efficient because of using expensive cryptographic primitives, e.g., homomorphic encryption [6, 12, 15, 18]. CDGOSS'21 [4] is efficient, but it can only work when the number of colluding parties t is smaller than $n/2$. KMPRT'17 [17] can be secure against an arbitrary number of colluding parties. However, it is not sufficiently efficient in both the running time and the communication costs. In KMPRT'17 [17], though the authors also proposed an optimized 3-party PSI protocol, it is insecure. Therefore, proposing an efficient mPSI protocol that can be secure against an arbitrary number of colluding parties is of great significance.

1.1 The high-level idea of our protocol

In this paper, we first propose an efficient mPSI protocol called *O-Ring*. We show a simple example of O-Ring with three parties. For parties P_1, P_2, and P_3, each one has a set (e.g., $X_1 = \{x_1, x_3, x_4\}$, $X_2 = \{x_1, x_2, x_3\}$, and $X_3 = \{x_1, x_2, x_4\}$). The core idea is to utilize a data structure called *oblivious key-value store* (OKVS) to build a ring to filter out non-common items. An OKVS is a data structure that encodes m key-value pairs into a table T. Because of the *oblivious* property, T hides the keys if the values are random. Now, P_1 can respectively generate three random values $\{v_1, v_3, v_4\}$ for $X_1 = \{x_1, x_3, x_4\}$ and encode $\{(x_1, v_1), (x_3, v_3), (x_4, v_4)\}$ into an OKVS table T_1. Upon receiving T_1 from P_1, P_2 decodes to get the values $\{v_1 = Decode(T_1, x_1), v_2' = Decode(T_1, x_2), v_3 =$

$Decode(T_1, x_3)\}$. Here, since $x_2 \notin X_1$, the decoded value v_2' is random. After the transition, P_2 keeps the common key-value pairs $(x_1, v_1), (x_3, v_3)$ but filter out the non-common value v_4. P_2 encodes $(x_1, v_1), (x_2, v_2'), (x_3, v_3)$ into another OKVS table T_2 and sends T_2 to P_3. Then P_3 decodes and gets $\{v_1 = Decode(T_2, x_1), v_2' = Decode(T_2, x_2), Decode(T_2, x_4)\}$. Now, v_3 will also be filtered out because $x_3 \notin X_3$. Finally, P_3 encodes $(x_1, v_1), (x_2, v_2'), (x_4, Decode(T_2, x_4))$ into a table T_3 and returns it to P_1. After decoding, P_1 will know the intersection $I = \{x_1\}$ by checking $v_1 = Decode(T_3, x_1)$, $v_3 \neq Decode(T_3, x_3)$, and $v_4 \neq Decode(T_3, x_4)$.

The above idea is simple, but not secure. First, it is vulnerable to the brute-force attack when the item domain is small. In the above example, if P_1 enumerates all items including x_2 in the domain, P_1 will easily find that $x_2 \in X_2 \cap X_3$ by checking $Decode(T_1, x_2) = Decode(T_3, x_2)$. To thwart this attack, we utilize a cryptographic primitive *oblivious pseudorandom function* (OPRF). It is commonly used in two-party PSI protocols [16, 26], such that the sender with a private PRF key can send the OPRF values to the receiver without disclosing non-common items.

Second, the design is insecure against the collusion attack, which is the greatest challenge in mPSI. For example, if P_1 colludes with P_3, after receiving T_2 from P_2, P_1 and P_3 can easily know $X_1 \cap X_2 = \{x_1, x_3\}$ by checking $v_1 = Decode(T_2, x_1)$, $v_3 = Decode(T_2, x_3)$, but $v_4 \neq Decode(T_2, x_4)$. Ideally, a secure mPSI protocol only discloses the intersection $I = X_1 \cap X_2 \cap X_3$ to the colluding parties. To be secure against an arbitrary number of colluding parties [17], we further exploit the OPRF. Assuming the maximum number of colluding parties is t, in the OKVS chain (or ring) $P_1 \to P_2 \to \cdots \to P_n \to P_1$, we select t parties $P_{n-t+1}, P_{n-t+2}, \cdots, P_n$, each of whom performs OPRFs with the other $n-1$ parties by acting as a common OPRF sender. Also, we select the protocol receiver P_1 to share PRF keys with the remaining $n-t-1$ parties. The idea is that there must be at least one honest party among the $t+1$ selected parties. Then for each item x_j with a value v_j in a party, v_j can be masked by $F_k(x_j)$ or $F_s(x_j)$ such that the colluding parties cannot identify x_j from an obfuscated value $v_j \oplus F_k(x_j)$ or $v_j \oplus F_s(x_j)$, where k is the common sender's OPRF key, s is P_1's PRF key, and F is a PRF. More details can be found in section 5.

1.2 Ring vs star topology

Most of previous protocols (e.g., [2, 4, 9, 12, 17]) are based on a star topology, in which the central party (denoted as *leader*) takes a much heavier burden in communication and computation costs than other parties (denoted as *clients*). Our ring-based protocol can share a part of the central party's costs among the other parties, thus reducing leader's costs. This is motivated by real-world applications in which the communication and computation resources of the leader are limited. Different topologies may be suitable for different applications. For applications without a powerful leader, the ring topology may be more suitable. In contrast, star topology is suitable for applications in which the leader has powerful resources (e.g., a cloud server). In the star topology, the leader can concurrently execute the protocol with different clients, such that the protocol can be faster.

1.3 Our contributions

In this paper, we propose two mPSI protocols, O-Ring and K-Star, tailored for different application scenarios. O-Ring is designed by utilizing a ring topology, while K-Star has a traditional star topology. Briefly, we make the following contributions.

1. O-Ring has three advantages. First, the communication costs of the bearer in O-Ring are cheaper than other mPSI protocols. The *bearer* is the party with the largest workload. Compared with the state-of-the-art protocols KMPRT'17 [17] and CDGOSS'21 [4], O-Ring can be respectively $1.6\times \sim 48.3\times$ and $1.9\times \sim 27.5\times$ as cheap as them in the bearer's communication costs. Therefore, O-Ring is suitable for applications in which the bearer has limited resources. Second, the total communication of O-Ring is low. O-Ring is respectively $1.6\times \sim 48.3\times$ and $4.0\times \sim 39.8\times$ (except one setting) as cheap as them in the total communication costs. Third, O-Ring runs fastest in most settings. For the total running time, in the LAN setting, O-Ring can be respectively $1.4\times \sim 6.0\times$ and $1.2\times \sim 12.9\times$ as fast as them in most of the settings; in the WAN setting, the ratios are respectively $1.2\times \sim 18.1\times$ and $1.1\times \sim 2.7\times$.

2. By modifying O-Ring into a star topology, we further propose another protocol K-Star. K-Star also has three advantages. First, K-Star runs the fastest in most settings. Compared with O-Ring, it is $13.6\% \sim 19.3\%$ faster when $n = 10$ in LAN. The reason is that O-Ring needs to run sequentially among the parties by using the OKVS ring, while K-Star can runs concurrently by using the star topology. Also, compared with KMPRT'17 [17] and CDGOSS'21 [4], in the LAN setting, K-Star can be respectively $1.4\times \sim 9.0\times$ and $1.0\times \sim 15.3\times$ as fast as them; in the WAN setting, the ratios are respectively $1.2\times \sim 18.9\times$ and $1.0\times \sim 4.4\times$ in most of the settings. Second, K-Star gains the same lowest total communication costs as O-Ring. Third, for the bearer's communication costs, K-Star can be $3.0\% \sim 63.4\%$ more expensive than O-Ring but is still much cheaper than KMPRT'17 [17] and CDGOSS'21 [4]. K-star is suitable for applications in which the central party is powerful.

3. We further make two fine-grained optimizations when we implement the OKVS and OPRF in O-Ring and K-Star. In the first optimization, we divide the encoding

scheme of the OKVS [8] into a peeling phase and an unpeeling phase. The peeling phase is only related to each party's items and can be done offline and parallelly by all parties. By applying this optimization in the OKVS chain, it can be 6.3% $\sim$ 52.5% faster. In the offline phase of OPRF protocol [26], a receiver needs to encode item-value pairs into an OKVS table. For each item x, its value is generated by a public cryptographic hash function $H(x)$ and is only related to the item itself. Having this observation, instead of simply invoking n' OPRFs for a party who acts as a common OPRF receiver, we optimize by only encoding the item-value pairs once for this party. After applying this optimization, it saves 1.0% $\sim$ 24.0% computation costs than naively invoking n' OPRFs. More details can be found in subsection 6.1.

4. We identify insecurity in previous efficient mPSI works KMPRT [17] and NTY [21]. Specifically, the augmented semi-honest protocol and the specially optimized three-party PSI protocol in KMPRT [17] are insecure for the collusion attacks. Also, the semi-honest and malicious protocols ($t > 1$) in NTY [21] are prone to the collusion attacks. More details can be found in section 6.4.1.

2 Related work

In this section, we introduce the related mPSI protocols. The computation and communication complexities of these protocols and our protocols are shown in Table 1. More descriptions of the related protocols with ours are as follows.

KMPRT'17 [17]. Kolesnikov et al. [17] proposed the Oblivious Programmable Pseudorandom Function (OPPRF) for their mPSI protocols. In OPPRF, the sender owns an item-value set $\{(x_j, v_j)\}_{\forall j\in[m]}$ and the receiver owns an item set Y, where m is the set size. After OPPRF, the receiver gets the correct value v_j of an item $y \in Y$ if $y = x_j$; otherwise, the value is random to the receiver. After designing their OPPRF protocol, they presented their mPSI protocols. In their first semi-honest mPSI protocol, each party P_i ($i = 1, 2, \cdots, n$) randomly generates n zero shares for each item x_j^i (i.e., $\beta_j^{i,1} \oplus \beta_j^{i,2} \oplus \cdots \oplus \beta_j^{i,n} = 0$). Then P_i invokes an OPPRF with another party P_u who owns a set $X^u = \{x_j^u\}_{\forall j\in[m]}$, by taking $\{(x_j^i, \beta_j^{i,u})\}_{\forall j\in[m]}$ as P_i's item-value set. This phase is called conditional *zero-sharing*. In the next phase, one party P_1 acts as the 'dealer' of the protocol to receive each item's shares from other $n-1$ parties. It is obvious that if an item is in the intersection, its corresponding share sum is 0; otherwise not.

Their second protocol turns the conditional zero-sharing into an unconditional zero-sharing. Specifically, P_i ($i = 1, 2, \cdots, n$) shares a PRF key $s^{i,j}$ to P_j ($j = i+1, i+2, \cdots, n$). If P_i and P_j have a common item x, both of them will have the same PRF value $F_{s^{i,j}}(x)$; then their value sum will be 0 (i.e., the pairwise property). After the unconditional zero-sharing, the protocol receiver P_1 acts as a common OPPRF receiver and respectively invoke an OPPRF instances with P_i ($\forall i \in [2, n]$) who inputs an item-value set $\{(x_j^i, \bigoplus_{u=1}^{i-1} F_{s^{u,i}}(x_j^i) \oplus \bigoplus_{u=i+1}^{n} F_{s^{i,u}}(x_j^i))\}_{\forall j\in[m]}$. Given the pairwise property, if x is in the intersection, its value sum that P_1 computes will be 0. Though the authors claimed that this protocol was secure in the *augmented semi-honest* model, it is prone to collusion attacks. More details are shown in 6.4.1.

To optimize the performance of their semi-honest protocol, the authors also proposed a three-party PSI protocol. However, this protocol is also insecure for the collusion attacks. The detailed analysis is shown in subsection 6.4.1.

CDGOSS'21 [4]. To design a mPSI protocol, Chandran et al. [4] first introduced a functionality *weak private set membership* (wPSM), which can be directly instantiated by the OPPRF in [17]. In their protocol, first, the dealer P_1 hashes all items into a cuckoo hashing table and the other parties hash their items by using the same hash functions. Then each party does wPSM with the dealer P_1 by acting as a sender, such that all parties can get the corresponding value shares for each item. After getting the value shares, all parties need to use a primitive *ConvertShares* (in which a secret sharing scheme is used) to generate the linear shares for the items. As a result, if the majority of parties are honest, the dealer can get the intersection. An interesting point about their work is that the performance of their protocol is independent of the number of colluding parties t by using the primitives. Their protocol can be more efficient than KMPRT'17 [17]. Concretely, their protocol has $6(t+2)/5\times$ less communication costs and is up to $5\times$ (resp. $6.2\times$) faster in LAN (resp. WAN) settings. However, an obvious drawback of their protocol is that their protocol can only work when $t < n/2$. Ideally, a protocol should support an arbitrary number of colluding parties.

NTY'21 [21]. Nevo et al. [21] proposed their mPSI protocols by using an OKVS and OPPRF. To ensure the security of their mPSI protocol, they exploited a zeroXOR protocol based on the augmented semi-honest protocol in KMPRT'17 [17] by changing the input from an item set to an item-value set. A party in zeroXOR needs to sum each item's input value with the PRF value sum before the OPPRF interaction. When $t = n-1$, the input item's value is set as 0 in zeroXOR and their mPSI is almost the same as the augmented semi-honest protocol in KMPRT'17 [17], which is subjected to the attack in subsection 6.4.1. When $t < n-1$, the n parties in the mPSI protocol can be divided into three parts: non-zeroXOR parties $P_2, \cdots, P_{w-1}$, a pivot P_w, and zero-XOR parties $P_{w+1}, P_{w+2}, \cdots, P_n, P_1$, where $w = n-t+1$. In the first step, P_i ($\forall i \in [2, w-1]$) respectively sends a PRF key $s^{i,u}$ to P_u ($u = w+1, w+2, \cdots, n, 1$). Then P_i ($i \in [w+1, n] \cup \{1\}$) can compute the value $\bigoplus_{u=2}^{w-1} F_{s^{u,i}}(x_j^i)$ of each item $x_j^i \in X_i$. P_i ($i \in [2, w-1]$) can compute an item x_j^i's value $F_{s^{i,1}}(x_j^i) \oplus \bigoplus_{u=w+1}^{n} F_{s^{i,u}}(x_j^i)$ and encode these item-

Protocol	Communication		Computation		Corruption threshold	Security	Concrete efficient
	Leader	Client	Leader	Client			
HV'17 [12]	$O(nm\kappa)$	$O(m\kappa)$	$O(nm\log\log m)$	$O(m)$	$t<n$	semi	No
	$O((n^2+nm\log m)\kappa)$	$O((n+m\log m)\kappa)$	$O(m^2)$			malicious	
GN'19 [9]	$O((n^2+nm)\kappa)$		$O(nm\log m)$	$O(m\log m)$	$t<n$	malicious	No
CDGOSS'21 [4]	$O(nm(\lambda+\kappa+\log m))$	$O(m(\lambda+\kappa+\log m)$	$O(nm\kappa)$	$O(m\kappa)$	$t<n/2$	semi	Yes
ENOC'22 [2]	$O(nm\kappa^2+nm\kappa\log(m\kappa))$	$O(m\kappa^2+m\kappa\log(m\kappa))$	$O(nm\kappa)$	$O(m\kappa)$	$t<n$	malicious	Yes
KMPRT'17 [17]	$O(nm(\lambda+\kappa+\log m))$	$O(m(\lambda+\kappa+\log m))$	$O(nm\kappa)$	$O(m\kappa)$	$t<n$	augmented	Yes
		$O(tm(\lambda+\kappa+\log m))$		$O(tm\kappa)$		semi	Yes
NTY'21 [21]	$O(m(n\lambda+\kappa+n\log m))$	$O(m(\lambda+\log m))$	$O(m\kappa)$	$O(m)$	$t=1$	malicious	Yes
O-Ring ($t=1$)	$O(m(\lambda+\kappa+\log m))$	$O(m(\lambda+\log m))$	$O(m\kappa)$	$O(m)$	$t=1$	semi	Yes
O-Ring ($t>1$)	$O(nm(\lambda+\kappa+\log m))$	$O(tm(\lambda+\kappa+\log m))$	$O(nm\kappa)$	$O(tm\kappa)$	$t<n$	semi	Yes
K-Star ($t=1$)	$O(m(n\lambda+\kappa+n\log m))$	$O(m(\lambda+\log m))$	$O(m\kappa)$	$O(m)$	$t=1$	semi	Yes
K-Star ($t>1$)	$O(nm(\lambda+\kappa+\log m))$	$O(tm(\lambda+\kappa+\log m))$	$O(nm\kappa)$	$O(tm\kappa)$	$t<n$	semi	Yes

Table 1: The communication and computation complexity of different mPSI protocols. 'semi' indicates semi-honest. 'augmented' indicates augmented semi-honest. λ and κ are respectively the statistical and computational security parameters. m is the set size of all parties. 'Concrete efficient' indicates whether the protocol is efficient in practice.

value pairs into an OKVS table T_i, which is sent to the pivot P_w. Then the pivot can also compute each item x_j^w's value as $\bigoplus_{i=2}^{w-1} Decode(T_i, x_j^w)$. Now, all P_w and $P_{w+1}, P_{w+2}, \cdots, P_n, P_1$ have got their items' values and they can input these item-value pairs into the zeroXOR protocol to get the intersection. Their protocol is still insecure when $1<t<n-1$. More discussions are in subsection 6.4.2.

Others. Garimella et al. [8] proposed a malicious mPSI protocol based on the augmented semi-honest protocol in KMPRT'17 [17]. In their work, they proved that their protocol was secure in the malicious model but not in the semi-honest model by introducing a random oracle model. However, their protocol is still prone to collusion attacks as KMPRT'17 [17].

Inbar et al. [13] also designed their protocols in both semi-honest and augmented semi-honest security models. Their building primitives are Oblivious Transfer (OT) and an OKVS called Garbled Bloom filter (GBF). The core idea of their protocols is the GBF-based two-party PSI protocol from Dong et al. [7]. In [7], the sender first builds a GBF from his/her set. For an item x, its λ shares are stored in the bins of the GBF by using λ hash functions. The receiver also constructs a Bloom filter [3] by using the same hash functions. In the Bloom filter, if a bin is hashed, then this bin is set as '1'; otherwise '0'. Then the receiver invokes OTs with the sender. If the bin is '1', the receiver gets the correct share in the sender's GBF; otherwise, the receiver can only get a random share. After all OTs, the receiver can check whether the sum of an item's λ corresponding shares equals this item to get the intersection. Inbar et al. [13] extended the work of Dong et al. [7] to multi-party case. In the mPSI protocol, each party works as an OT receiver to query the value shares of other parties. After getting these shares, each party sums these shares and sends the share sums to one party who computes the intersection by reconstructing the share sum of each item.

Ben-Efraim et al. [2] followed Inbar et al. [13]'s work to propose a malicious mPSI protocol. Similar to Dong et al. [7], they aimed to limit the number of '1's one party uses in the OT interactions. To achieve this, they exploited K-out-of-N OT in [25]. A similar work extending [25] can be found in [27]. Another work combining GBF and OPRF can be found in [14]. In addition, some works used additive homomorphic encryption [6,12,15,18], which is expensive and makes the protocols not efficient. Ghosh and Nilges [9] replaced the homomorphic encryption with oblivious polynomial evaluation, which is also expensive [21].

3 Preliminaries

3.1 Notations

Notations	Comments
n	The number of parties
t	The number of colluding parties
P_i	Party i, $i=1,2,\cdots,n$
m	The set size of each party P_i
X_i	The set of party P_i, i.e., $X_i=\{x_1^i, x_2^i, \cdots, x_m^i\}$
T_i	The OKVS table of P_i. $T_{i,j}$ is the jth bin
ℓ	The output bit length
λ	The statistical security parameter
κ	The computational security parameter

In addition to the above notations, $\oplus$ is the bitwise XOR; $\bigoplus_{i=1}^n x_i$ denotes $x_1 \oplus x_2 \oplus \cdots \oplus x_n$. $\bigcap_{i=1}^n X_i$ denotes $X_1 \cap X_2 \cap \cdots \cap X_n$. $[m]$ denotes the set $\{1,2,\cdots,m\}$. $[a,b]$ denotes the set $\{a,a+1,\cdots,b\}$. For any notation with a superscript, e.g., x^i, the superscript i indicates the party index (not the power in math operation).

3.2 Multi-party PSI

The ideal functionality of mPSI is shown in Figure 1. In this functionality, each party P_i inputs a set X_i; P_1 is the only receiver who gets the intersection.

Parameters: The number of parties n, the number of corrupted parties $t < n$, and the set size of each party m. The bit length of each item σ.
Inputs: Each party P_i inputs a set $X_i = \{x_j^i\}_{\forall j \in [m]}$ for $i \in [n]$, and $x_j^i \in \{0,1\}^\sigma$.
Outputs: P_1 gets the intersection $I = \bigcap_{i=1}^n X_i$. The other parties receive $\perp$.

Figure 1: The ideal functionality of mPSI $\mathcal{F}_{m-psi}$.

A computation protocol in the real world is said to be secure with respect to certain adversarial behavior if the possible real executions with such an adversary can be simulated in the corresponding ideal world [10]. In the ideal world, a trusted third party performs the computation after giving the inputs and returns the outputs. To prove the security, one needs to prove the view of the adversary in the real world is indistinguishable from the view of the simulator in the ideal world. Usually, there are two types of security models: semi-honest model and malicious model. In the semi-honest model, the parties follow the protocol strictly but are curious to know extra information beyond their outputs. In the malicious model, the malicious parties can arbitrarily deviate from the protocol. In this paper, we only focus on the semi-honest security model.

mPSI is a special multi-party secure computation. Different from two-party PSI, the greatest threat for the mPSI is the collusion attack. In two-party PSI, only the receiver is allowed to learn the intersection, while the sender learns nothing. Keeping consistent with the two-party security requirement, in mPSI, with an arbitrary number of colluding parties (i.e., $t < n$), if the receiver is in the colluding parties, the colluding parties can learn only the intersection $I = \bigcap_{i=1}^n X_i$; otherwise, the colluding parties learns nothing. In other words, even $t = n-1$ and the receiver is in the colluding parties, they can only know $I = \bigcap_{i=1}^n X_i$ from the honest party. If the receiver is not colluded, the colluding parties know nothing about the receiver's items. It is noted that the colluding parties can know the set information about their own sets, e.g., the intersection of their own sets.

3.3 OKVS

An Oblivious Key-Value Store (OKVS) is a data structure to store key-value pairs. Specifically, given m key-value pairs $S = \{(x_1, v_1), (x_2, v_2), \cdots, (x_m, v_m)\}$, an OKVS encodes them into a table T, which denotes as $T = Encode(S)$. For decoding, one can query to get each key x's corresponding value $Decode(T, x)$. If $x = x_i$, then $Decode(T, x) = v_i$; otherwise, $Decode(T, x)$ is pseudorandom.

Definition 1. *Obliviousness* [8]. If the values in set $\{v_1, v_2, \cdots, v_m\}$ are uniform, then $T_1 = Encode(S_1)$ and $T_2 = Encode(S_2)$ are computationally indistinguishable, where $S_1 = \{(x_1^1, v_1), (x_2^1, v_2), \cdots, (x_m^1, v_m)\}$ and $S_2 = \{(x_1^2, v_1), (x_2^2, v_2), \cdots, (x_m^2, v_m)\}$.

In OKVS, the expansion rate η shows that one only needs a table with size $m\eta$ for encoding. Pinkas et al. [24] proposed an OKVS *PaXoS* with linear computation costs and low expansion rate of $\eta = 2.4$. Later, Garimella et al. [8] proposed *3H-GCT* based on PaXoS to reduce the expansion rate to $\eta = 1.3$. In this paper, for simplicity, we denote an OKVS as its stored table T. In PaXoS/3H-GCT, there are two phases in the encoding: peeling and unpeeling. In the peeling phase, the encoder exploits the keys to find an ordering sequence. Then in the unpeeling phase, the encoder assigns the values based on the ordering sequence. For more details about PaXoS/3H-GCT, one can refer to [8, 24].

3.4 OPRF

Oblivious Pseudorandom Function (OPRF) allows the receiver to obliviously get the PRF values $\{F_k(x_1), F_k(x_2), \cdots, F_k(x_m)\}$ after inputting a set $\{x_1, x_2, \cdots, x_m\}$, where $F_{(\cdot)}(\cdot)$ is a common pseudorandom function (PRF), and the key k is the sender's output. The ideal functionality of OPRF is depicted in Figure 2.

Parameters: The set size m of the receiver. A PRF $F_{(\cdot)}(\cdot)$.
Inputs: The receiver inputs the sets $\{x_i\}_{\forall i \in [m]}$.
Outputs: The receiver gets the PRF outputs $\{F_k(x_i)\}_{\forall i \in [m]}$. The sender gets a pseudorandom secret key k.

Figure 2: The ideal functionality of OPRF $\mathcal{F}_{oprf}$.

OPRF can be utilized to realize a two-party PSI (e.g., [5, 8, 16, 26]). After running the OPRF, the sender gets the OPRF key k and computes the PRF values $V = \{F_k(y_i) | \forall y_i \in Y\}$. Then the sender directly sends V to the receiver. The receiver gets the intersection $I = \{x | F_k(x_i) \in V, \forall i \in [m]\}$. To securely get the intersection, the core idea is to compare the OPRF values of the sender and the receiver. The workflow of a two-party PSI by utilizing OPRF can be found in Figure 3. The security goes that all $\{F_k(y_i) | y_i \in Y \backslash I\}$ are pseudorandom to the receiver. Hence, the receiver learns only the intersection. Here, for security, it is noted that *only the sender can send the OPRF values to the receiver; the receiver cannot send the ORPF values to the sender*. Therefore, it is unidirectional. Once the receiver sends $\{F_k(x_1), F_k(x_2), \cdots, F_k(x_m)\}$ to the sender, it is possible for the sender to get the items of the receiver since the sender holds the key k, especially when the item domain is small.

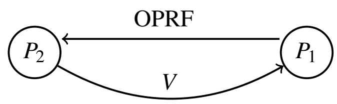


Figure 3: The workflow of two-party PSI by using OPRF. The start and the end of the OPRF arrow line respectively represent the OPRF receiver P_1 and OPRF sender P_2. P_2 returns its OPRF value set V to P_1 after OPRF invocation.

In the above flow, the sender can also return an OKVS table T that encodes $\{(y_i, F_k(y_i))\}_{\forall i \in [m]}$ to the receiver rather than the set V. Then the receiver gets the intersection $I = \{x | F_k(x_i) = Decode(T, x_i), \forall i \in [m]\}$. By returning an OKVS table, the output bit length of $F_k(\cdot)$ can be reduced by $\log_2(n)$ [23] because the receiver no longer needs to compare n OPRF values for each item's OPRF value to know whether it is in the intersection. In this paper, for simplicity, the sender returns an OKVS table encoding the item-value pairs to the receiver.

4 Our protocol without collusion

In this section, we propose a simple and efficient semi-honest mPSI protocol without collusion by only using OKVS and OPRF. The general idea is to reduce the mPSI to the two-party PSI. The receiver P_1 and the last party P_n run as the two-party PSI in Figure 3. All parties run in an OKVS chain.

4.1 A simple but insecure protocol

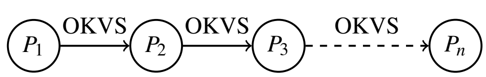


Figure 4: The workflow of an OKVS chain.

We first introduce a protocol called *OKVS chain*. The workflow of an OKVS chain is shown in Figure 4. In the OKVS chain, each party P_i has an item-value pair set $S_i = \{(x_j^i, v_j^i)\}_{\forall j \in [m]}$ and needs to transit the XOR value sum of the common items. Specifically, the first party P_1 encodes his/her item-value pairs $S_i = \{(x_j^i, v_j^i)\}_{\forall j \in [m]}$ into an OKVS table T_1 and transits it to P_2. The intermediate party P_i $(i = 2, 3, \cdots, n)$ receives an OKVS table T_{i-1} from its previous party P_{i-1}. Then P_i decodes each item $x_j^i \in X_i$ to get its corresponding value $Decode(T_{i-1}, x_j^i)$ and encodes to get the next OKVS table $T_i = Encode\{\{(x_j^i, v_j^i \oplus Decode(T_{i-1}, x_j^i))\}_{\forall j \in [m]}\}$. After receiving T_{n-1} from P_{n-1}, P_n computes $v_j^n \oplus Decode(T_{n-1}, x_j^n)$ for each item x_j^n and the transition stops.

In the transition, each party transits the value XOR sum of the intersection items. The obliviousness of the OKVS chain inherits from the OKVS. For P_2, if an item $x_j^2 \notin X_1$, then $Decode(T_1, x_j^2)$ is pseudorandom and P_2's updated value $v_j^2 \oplus Decode(T_1, x_j^2)$ for x_j^2 will be randomized. Then P_2 keeps only the correct value sum for the items in $X_1 \cap X_2$. For P_3, if $x_j^3 \notin X_1 \cap X_2$, its updated value sum $v_j^3 \oplus Decode(T_2, x_j^3)$ will also be randomized. The transition continues until P_n keeps the correct value sum for all items in the intersection $X_1 \cap X_2 \cap \cdots \cap X_n$; otherwise, their value sums are randomized. From another angle, the OKVS chain is a filter to filter out non-common items by randomizing their corresponding values. In the above transition, it is noted that P_i $(i = 2, 3, \cdots, n)$ cannot simply compute $T_i = T_{i-1} \oplus Encode(\{(x_j^i, v_j^i)\}_{j \in [m]})$ because of a possible false positive issue. If doing so, although the common items remain, the non-common items also remain. Then P_i cannot filter out the non-common items. Therefore, P_i has to decode to get the corresponding values from T_{i-1}.

By utilizing the OKVS chain, one can trivially design a mPSI protocol. First, P_1 can generate m random values $\{v_j^1\}_{\forall j \in [m]}$ for his/her m items to input m key-value (item-value) pairs in the chain. The other parties simply input their items with associated values as 0s. After the intersection value transition, P_n will get an updated value set V and send it to P_1. P_1 includes an item into the intersection if its associated value v_j^1 is in V. This protocol is very simple. However, there are two security concerns. First, it is vulnerable to the brute-force attack when the item domain is small. It is noted that P_1 sends out an OKVS table T_1 and receives the value set from P_n. If P_1 simply brute-forces the item domain to get all the associated values by decoding on T_1. Then it is easy to know the intersection of all other parties (i.e., $\bigcap_{i=2}^n X_i$) by intersecting the full domain values with the received value set. Second, it is vulnerable to the collusion attack. For example, if P_3 colludes with P_1, it is easy for them to attack P_2 to know $X_1 \cap X_2$ because P_3 receives the intersection information from P_2 in the chain. It is noted that the input value v_j^i for each item x_j^i are set depending on the specific protocol needs. In the above protocol, $v_j^2, v_j^3, \cdots, v_j^n$ are set as 0. In late sections, they will be set differently.

4.2 O-Ring(t=1): a secure protocol without collusion

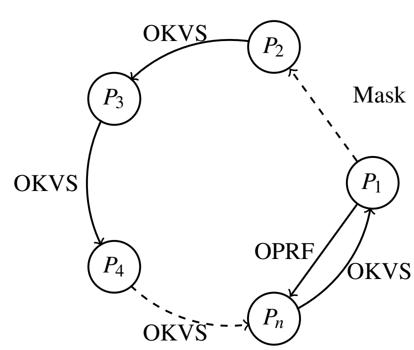


Figure 5: The workflow of our mPSI protocol O-Ring $(t = 1)$ without collusion.

Since we do not consider the collusion attack in this section, we put our focus on handling the brute-force attack by combining with an OPRF primitive. The workflow of our protocol is shown in Figure 5. One can easily find that our design combines the OKVS chain with the two-party PSI in Figure 3. When there are only two parties, it simply becomes the two-party PSI. In the flow, 'Mask' indicates that P_1 shares a PRF key with P_2. The full protocol is shown in Figure 6. The correctness goes that if x is in the intersection, the PRF and OPRF value sum will be 0 because PRF and OPRF values are in pairs. Otherwise, the probability that the value sum is 0 will be negligible. For the security, because of the OPRF between P_1 and P_n, P_1 can no longer has the brute-force attack against the other parties because he/she does not have the OPRF key. Due to the page limit, we put the formal security proof of this protocol in Appendix A.1.

Parameters: The number of parties n and set size m. The item's bit length σ. An OKVS scheme $Encode(\cdot)$, $Decode(\cdot,\cdot)$. A PRF $F_{(\cdot)}(\cdot):\{0,1\}^{\kappa}\times\{0,1\}^{\sigma}\mapsto\{0,1\}^{\ell}$.
Inputs: Each party P_i inputs a set $X_i=\{x^i_j\}_{\forall j\in[m]}$ for $i\in[n]$, and $x^i_j\in\{0,1\}^{\sigma}$.
Protocol:

1. **[Initiation]**. P_1 sends a PRF key s to P_2. Also, P_1 invokes $\mathcal{F}_{oprf}$ with P_n, where P_1 and P_n respectively act as the receiver and sender. Then P_n receives an OPRF key k and computes an item-value set $S_n=\{(x^n_j,F_k(x^n_j))\}_{\forall j\in[m]}$. P_1 and P_2 respectively compute $S_1=\{(x^1_j,F_s(x^1_j)\oplus F_k(x^1_j))\}_{\forall j\in[m]}$ and $S_2=\{(x^2_j,F_s(x^2_j))\}_{\forall j\in[m]}$.
2. **[Transition]**. P_2 starts the OKVS transition by inputting S_2 in the OKVS chain $P_2\rightarrow P_3\rightarrow\cdots\rightarrow P_n\rightarrow P_1$. In the transition, P_i inputs $\{(x^i_j,0)\}_{\forall j\in[m]}$ $(i=3,4,\cdots,n-1)$ and P_n inputs S_n.
3. **[Intersect]**. P_1 computes the intersection $I=\{x^1_j|F_s(x^1_j)\oplus F_k(x^1_j)\oplus Decode(T_n,x^1_j)=0,\forall j\in[m]\}$, where T_n is the P_1's received OKVS table from P_n.

Figure 6: O-Ring ($t=1$): mPSI without collusion.

5 Our protocol with arbitrary collusion

In the above section, we have designed an elegant mPSI protocol without collusion (i.e., $t=1$) by utilizing OKVS and OPRF. In this section, we continue to design a mPSI protocol that can thwart $1<t<n$ colluding parties.

5.1 O-Ring(P_1 honest): a naive protocol

In the mPSI protocol, P_1 is the only one who receives the intersection. In Figure 5, we also observe that P_1 is the end node of the ring, which indicates that P_1 plays a different role from others in the protocol. If P_1 is honest, we utilize its special role to design a mPSI protocol with arbitrary collusion. Specifically, P_1 sends a secret key $s^{1,i}$ to each non-OPRF party P_i respectively $(i=2,3,\cdots,n-1)$. Then each of the non-OPRF parties can use this secret PRF key to mask its items' decoded values, thus making it random to other parties. It is noted that there is no need for P_1 to do masking with P_n because they have already invoked an OPRF instance. The general workflow is shown in Figure 7. In this flow, the masked parties P_i $(i=2,3,\cdots,n-1)$ can utilize these PRF keys to mask their items' values by updating the value v as $v\oplus F_{s^{1,i}}(x)$, thus keeping it pseudorandom to the other parties.

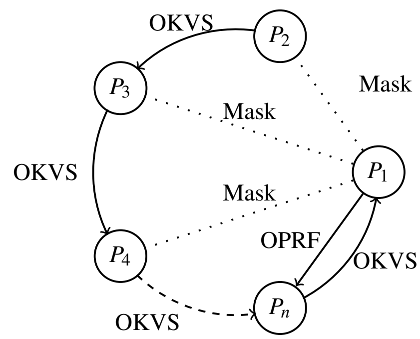


Figure 7: The workflow of our mPSI protocol with arbitrary collusion when P_1 is honest.

5.2 Identifying an honest party

The above protocol assumes that P_1 is honest and does masking between P_1 and the other parties such that the protocol can be secure against an arbitrary number of colluding parties. However, if P_1 joins the collusion, the protocol will be insecure. Therefore, assuming that P_1 is not honest, we need to find an honest party to ensure the security of the protocol.

In addition to having masking between P_1 and the other parties in Figure 7, we follow the masking idea to assume that P_n is honest and try to do the masking between P_n and P_i with PRF keys $s^{n,i}$ $(i=2,3,\cdots,n-1)$ to thwart the colluding attack. Unfortunately, there will be a colluding attacks even after masking. If P_1 joins the collusion with a masked party P_i $(i\in[2,n-1])$, they will know extra information beyond the intersection. Specifically, assuming P_1 has an item x but P_i does not, they can know if the other $n-2$ parties have x. P_1 and P_i can simply check if $Decode(T_n,x)\oplus Decode(T_{i-1},x)=Decode(T_i,x)\oplus F_{s^{n,i}}(x)$[1] if $i>2$. Here P_i has T_i, T_{i-1}, and $s^{n,i}$; P_1 has T_n. Based on the property of OKVS chain, if $P_2,P_3,\cdots,P_{i-1},P_{i+1},\cdots,P_n$ has x, we have $Decode(T_{i-1},x)=\bigoplus_{j=2}^{i-1}F_{s^{n,j}}(x)$ and $Decode(T_n,x)=Decode(T_i,x)\oplus\bigoplus_{j=i+1}^{n-1}F_{s^{n,j}}(x)\oplus$

[1]For simplicity, we do not consider PRF values between P_1 and the other parties. If consider, P_1 and P_i should check $Decode(T_n,x)\oplus Decode(T_{i-1},x)=Decode(T_i,x)\oplus F_{s^{n,i}}(x)\oplus\bigoplus_{j=2}^{n}F_{s^{1,j}}(x)\oplus F_{s^{1,i}}(x)$.

$\bigoplus_{j=2}^{n-1} F_{s^{n,j}}(x) = Decode(T_i,x) \oplus \bigoplus_{j=2}^{i-1} F_{s^{n,j}}(x) \oplus F_{s^{n,i}}(x)$. Then it is easy to know $Decode(T_n,x) \oplus Decode(T_{i-1},x) = \bigoplus_{j=2}^{i-1} F_{s^{n,j}}(x) \oplus Decode(T_i,x) \oplus \bigoplus_{j=2}^{i-1} F_{s^{n,j}}(x) \oplus F_{s^{n,i}}(x) = Decode(T_i,x) \oplus F_{s^{n,i}}(x)$. Now, P_i knows x is in the intersection of the other $n-2$ parties' sets, which should not be allowed because x is not in the intersection of all parties. If $i = 2$, P_2 and P_1 can simply check if $Decode(T_n,x) = Decode(T_2,x) \oplus F_{s^{n,2}}(x)$.

The reason why P_i and P_1 can have the collusion attack is that masking by sharing PRF keys is symmetric. Namely, both P_i and P_n know the PRF key $s^{n,i}$. By using this key, P_i can compute the PRF value $F_{s^{n,i}}(x)$ for x to do the checking even he/she does not have x. To thwart the colluding attack, instead of using masking, the asymmetric OPRF can be used. Specifically, P_n can invoke an OPRF instance with P_i by acting as a sender who holds the OPRF key $k^{n,i}$. Then if P_i does not have x, he/she cannot compute $F_{k^{n,i}}(x)$ to finish the checking. Since $i \in [2,n-1]$, P_n needs to do OPRF with $P_2,P_3,\cdots,P_{n-1}$. Taking P_1 into attack, P_n still needs to do OPRF with P_1 by acting as a sender. As a common OPRF sender to the other $n-1$ parties, if one colluding party P_i who acts as an OPRF receiver does not have x, then its OPRF value $F_{k^{n,i}}(x)$ of P_n will be pseudorandom to P_i and the colluding parties, thus ensuring the security of P_n. For an honest party P_j, $j \in [2,n-1]$ who is an OPRF receiver, his/her OPRF value $F_{k^{n,j}}(x)$ is also pseudorandom to other parties because the honest P_n holds the key $k^{n,j}$. Therefore, it is also secure for P_j.

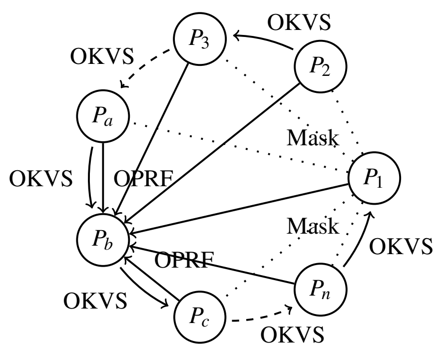


Figure 8: The workflow of a protocol that assumes P_b is honest. P_a and P_c are respectively the previous party and posterior party of P_b in the OKVS chain.

Having the above analysis, we know that if P_n (who acts as a common OPRF sender to the other $n-1$ parties) is honest, the protocol is secure against the collusion attack. In a general discussion, let us assume P_b is honest, the protocol workflow is shown in Figure 8. In this workflow, P_b acts as the common OPRF sender to the other $n-1$ parties and the protocol receiver P_1 shares PRF keys with the other parties except for P_b. It is noted that P_b has already done OPRF with P_1 and there is no need for P_1 to do the masking with him/her. In the OKVS chain, P_b's input key-value pair set in the OKVS chain is $\{(x_j^b, \bigoplus_{i=1}^{b-1} F_{k^{b,i}}(x_j^b) \oplus \bigoplus_{i=b+1}^{n} F_{k^{b,i}}(x_j^b))\}_{\forall j \in [m]}$. From the value sum, one can know that if x_j^b is not in common, at least one of the OPRF values will be pseudorandom to the other parties and the encoded value in T_b is also pseudorandom.

5.3 O-Ring($t > 1$): a protocol against arbitrary collusion

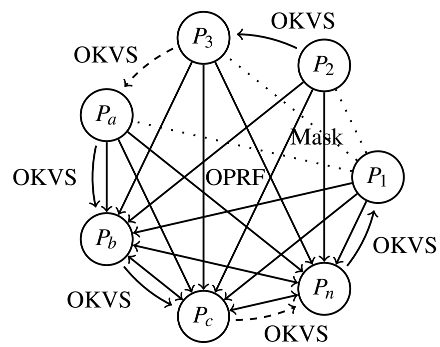


Figure 9: The workflow of O-Ring ($t > 1$). P_i ($\forall i \in [b,n]$) is a common OPRF sender for the other parties $P_1,\cdots,P_{i-1},P_{i+1},\cdots,P_n$. When $t < n-1$, P_1 still needs to share PRF keys with $P_2,P_3,\cdots,P_{b-1}$. The bidirectional arrow line indicates that the two parties invoke two OPRF instances with switched OPRF roles.

After the discussion about the honest party P_b in the above subsection, we design the mPSI protocol *O-Ring* in which there are at most t colluding parties. The workflow of O-Ring ($t > 1$) is shown in Figure 9. In this Figure, we set $b = n-t+1$ and make P_i ($\forall i \in [b,n]$) as a common OPRF sender to the other $n-1$ parties. From subsection 5.1, if P_1 is honest, it is easy to know the protocol is secure. Assuming P_1 is a colluder, there must be at least one honest party among the t parties $P_b,P_{b+1},\cdots,P_n$ because the number of colluders is at most t.

The full protocol is shown in Figure 10. In the protocol, there are three phases: share distribution, share collection, and share transition. In the share distribution phase, P_i ($\forall i \in [b,n]$) has OPRFs with other parties and P_1 shares PRF keys with P_j ($\forall j \in [2,b-1]$). Since the OPRF values and masking values are pairwise, the XOR value sum for a common item will be zero. In other words, each OPRF value or masking value is a share of 0 for the common items. After share distribution, each party P_i collects the shares it distributes and receives for each item from other parties in the OKVS chain by XOR these shares. After preparing the items' corresponding values (i.e., the share sum), all parties input their key-value (i.e., item-value) sets into the OKVS chain and start the intersection value sum transition. Given the property of OKVS chain, the end party P_1 will get the correct value sum of all the parties for each item x_j^1 if $x_j^1 \in X_2 \cap X_3 \cap \cdots \cap X_n \cap X_1$; otherwise not. The correctness of our protocol is obvious because both

Parameters: The number of parties n. The number of colluding parties t. The set size m. The statistical and computational security parameter λ and κ. The item's bit length σ. An OPRF functionality $\mathcal{F}_{oprf}$. A PRF $F_{(\cdot)}(\cdot) : \{0,1\}^{\kappa} \times \{0,1\}^{\sigma} \mapsto \{0,1\}^{\ell}$.
Inputs: Each party P_i ($i \in [n]$) inputs a set $X_i = \{x_j^i\}_{\forall j \in [m]}$.
Protocol:

1. **[Share distribution]**. Denote $b = n - t + 1$.
   (a) **[OPRF]**. P_i ($\forall i \in [b,n]$) respectively invokes an instance of $\mathcal{F}_{oprf}$ with P_j ($\forall j \in [1,n] \backslash \{i\}$) to get $\{Q^i(x_u^i) = \bigoplus_{j=1}^{i-1} F_{k^{i,j}}(x_u^i) \oplus \bigoplus_{j=i+1}^{n} F_{k^{i,j}}(x_u^i)\}_{\forall u \in [m]}$ and $\{F_{k^{i,j}}(x_u^j)\}_{\forall u \in [m]}$, where $k^{i,j}$ is the common OPRF sender P_i's OPRF key.
   (b) **[Mask]**. If $b > 2$, P_1 respectively sends a random PRF key $s^{1,j}$ to P_j ($\forall j \in [2, b-1]$).
2. **[Share collection]**.
   - $b > 2$. P_i ($\forall i \in [2, b-1]$) computes $v_j^i = F_{s^{1,i}}(x_j^i) \oplus \bigoplus_{u=b}^{n} F_{k^{u,i}}(x_j^i)$ ($\forall j \in [m]$). P_i ($\forall i \in [b,n]$) computes $v_j^i = Q^i(x_j^i) \oplus \bigoplus_{u=b}^{i-1} F_{k^{u,i}}(x_j^i) \oplus \bigoplus_{u=i+1}^{n} F_{k^{u,i}}(x_j^i)$ ($\forall j \in [m]$). P_1 computes $v_j^1 = \bigoplus_{u=2}^{b-1} F_{s^{1,u}}(x_j^1) \oplus \bigoplus_{u=b}^{n} F_{k^{u,1}}(x_j^1)$ ($\forall j \in [m]$).
   - $b = 2$. P_i ($\forall i \in [1, b-1]$) computes $v_j^i = \bigoplus_{u=b}^{n} F_{k^{u,i}}(x_j^i)$ ($\forall j \in [m]$). P_i ($\forall i \in [b,n]$) computes $v_j^i = Q^i(x_j^i) \oplus \bigoplus_{u=b}^{i-1} F_{k^{u,i}}(x_j^i) \oplus \bigoplus_{u=i+1}^{n} F_{k^{u,i}}(x_j^i)$ ($\forall j \in [m]$).
3. **[Share transition]**.
   - For O-Ring, in the OKVS chain (i.e., $P_2 \to P_3 \to \cdots \to P_n \to P_1$), each party P_i has $\{(x_j^i, v_j^i)\}_{\forall j \in [m]}$. P_2 starts the transition and P_1 gets the transition value $v_j^1 = v_j^1 \oplus \bigoplus_{i=2}^{n} v_j^i$ for each item x_j^1. P_1 gets the intersection: $I = \{x_j^1 | v_j^1 = 0, \forall j \in [m]\}$.
   - For K-Star, P_i ($i = 2, \cdots, n-1$) builds an OKVS table T_i by encoding $\{(x_j^i, v_j^i)\}_{\forall j \in [m]}$ and sends it to P_n. Upon receiving these tables, P_n also builds an OKVS table T_n by encoding $\{(x_j^n, v_j^n \oplus \bigoplus_{u=2}^{n-1} Decode(T_u, x_j^n))\}_{\forall j \in [m]}$ and sends it to P_1. Then P_1 gets the intersection: $I = \{x_j^1 | v_j^1 \oplus Decode(T_n, x_j^1) = 0, \forall j \in [m]\}$.

Figure 10: O-Ring and K-Star ($t > 1$): our mPSI protocols against $\leq t$ colluding parties.

masking and OPRF enable pairwise sharing. If an item is common, both parties will receive the same mask value and OPRF value. XORing the same value twice will get 0.

Theorem 1. *Our O-Ring ($t > 1$) in Figure 10 realizes the functionality $\mathcal{F}_{m-psi}$ against $\leq t$ colluding semi-honest adversaries in the $\mathcal{F}_{oprf}$ hybrid model.*

Proof. (Sketch). Due to the space limitation, we only sketch the security proof. Dividing the n parties into two groups C and O, which are respectively the coalition of corrupt parties and honest parties. If the protocol receiver P_1 is honest, the protocol is secure (see subsection 5.1 for related analysis). Now, assuming P_1 is a colluder in C, there must be at least one common OPRF sender (say it P_i) among the t common OPRF senders $P_b, P_c, \cdots, P_n$. If $|O| = 1$ (i.e., only P_i in O), it is easy to know the protocol is secure. If $|O| > 1$, denoting P_j as another honest party, then the OPRF values of P_i and P_j are pseudorandom to C because the honest P_i holds the secret OPRF key $k^{i,j}$. Then P_i and P_j's encoded values in step 3 is also pseudorandom. When P_i and P_j both have an item x and they are adjacent, their OPRF value sum by using $k^{i,j}$ will be 0, which is not pseudorandom. However, assuming $x \notin I$, there must be one party in C who does not have x, say P_u. Then $F_{k^{u,j}}(x)$ is pseudorandom to C and it is an XOR share of P_i's value sum in step 2. Then x's encoded value in step 3 is also pseudorandom for C. Since all honest parties' encode values in step 3 are pseurandom to C, the protocol is secure. More details are in Appendix A.3. □

5.4 K-Star: a protocol with star topology

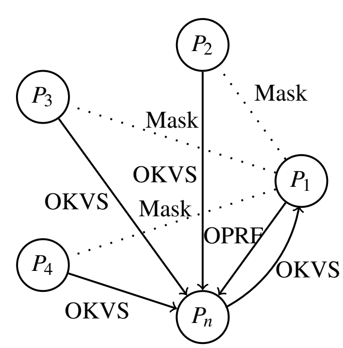


Figure 11: The workflow of K-Star when $t = 1$.

In O-Ring, there is no central party. We notice that many previous works [2, 17, 21] used the star topology to design their protocol. In their works, one party needs to carry the workload much larger than the other parties, especially when n is large and t is small. This cannot be avoided because the star structure requires one party to handle the shares from other parties. In real-world applications, this central party can be a server with strong computation power and is willing to pay more communication costs.

Based on the ring design, we also proposed a star design *K-Star*. The full protocol of K-Star when $t > 1$ is shown in Figure 10. The only difference between O-Ring and K-Star is in the last step. Instead of transiting the shares via the OKVS chain in O-Ring, the parties $P_2, P_3, \cdots, P_{n-1}$ simply send the value shares to the center P_n via the OKVS tables. Then P_n transits the collected shares to the protocol receiver P_1 such that P_1 can compute the intersection. The security of K-Star $(t > 1)$ is the same as O-Ring $(t > 1)$.

Theorem 2. *Our K-Star $(t > 1)$ in Figure 10 realizes the functionality $\mathcal{F}_{m-psi}$ against $\leq t$ colluding semi-honest adversaries in the $\mathcal{F}_{oprf}$ hybrid model.*

Proof. (Sketch). In O-Ring $(t > 1)$, the share distribution and share collection steps by using OPRF and masking ensure the security of the protocol. Specifically, having OPRF enables each item's encoded value in step 3 is pseudorandom to the colluded parties. K-Star and O-Ring only differs in the share transition step. Therefore, if O-Ring $(t > 1)$ is secure, K-Star $(t > 1)$ is also secure. More analysis are in Appendix A.4. $\square$

We also illustrate the working flow of K-Star when there is no collusion (i.e., $t = 1$) in Figure 11. In the flow, P_1 first sends PRF keys to P_i $(\forall i \in [2, n-1])$ and has an OPRF instance with P_n by acting as an OPRF receiver. Then each party collect their shares. Parties $P_2, P_3, \cdots, P_{n-1}$ directly send their values to the center P_n via OKVS tables. Finally, P_n sends the collected shares to P_1 via an OKVS table to compute the intersection. The full protocol is shown in Figure 12. We put the security proof for this protocol in Appendix A.2.

Parameters: The number of parties n and set size m. The statistical security parameter λ and the computational security parameter κ. The item's bit length σ. An OKVS scheme $Encode(\cdot)$, $Decode(\cdot,\cdot)$. A PRF $F_{(\cdot)}(\cdot) : \{0,1\}^\kappa \times \{0,1\}^\sigma \mapsto \{0,1\}^\ell$.
Inputs: Each party P_i inputs a set $X_i = \{x^i_j\}_{\forall j \in [m]}$ for $i \in [n]$, and $x^i_j \in \{0,1\}^\sigma$.
Protocol:

1. **[Initiation].** P_1 sends a PRF key $s^{1,i}$ to P_i $(i = 2, 3, \cdots, n-1)$. Also, P_1 invokes $\mathcal{F}_{oprf}$ with P_n, where P_1 and P_n respectively act as the receiver and sender. Then P_n receives an OPRF key k and computes an item-value set $S_n = \{(x^n_j, F_k(x^n_j))\}_{\forall j \in [m]}$. P_1 and P_i $(i \in [2, n-1])$ respectively compute $S_1 = \{(x^1_j, v^1_j = \bigoplus_{i=2}^{n-1} F_{s^{1,i}}(x^1_j) \oplus F_k(x^1_j))\}_{\forall j \in [m]}$ and $S_i = \{(x^i_j, F_{s^{1,i}}(x^i_j))\}_{\forall j \in [m]}$.
2. **[Transition].** P_i $(i = 2, 3, \cdots, n-1)$ encode S_i into an OKVS table T_i and sends T_i to P_n. Then P_n encodes $\{(x^n_j, F_k(x^n_j) \oplus \bigoplus_{i=2}^{n-1} Decode(T_i, x^n_j))\}_{\forall j \in [m]}$ to get an OKVS table T_n and sends it to P_1.
3. **[Intersect].** P_1 computes the intersection $I = \{x^1_j | v^1_j \oplus Decode(T_n, x^1_j) = 0, \forall j \in [m]\}$.

Figure 12: K-Star $(t = 1)$: mPSI without collusion.

6 Experiments and evaluations

The benchmark machine is with 32-core Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz and 236G RAM. The network is simulated by the localhost network as other mPSI works [4, 17]. The bandwidth and the latency are controlled by the Linux *tc* command. There are two network settings: for the LAN setting, the bandwidth is 10Gbps and the round-trip latency is 0.06ms; for the WAN setting, the bandwidth is 200Mbps and the round-trip latency is 96ms.

To implement our protocols (O-Ring and K-Star), we utilize the OPRF protocol in [26], which is composed of a primitive VOLE and an OKVS. We utilize the OKVS *3H-GCT* [8] with an expansion rate about 1.3. The statistical and computational security parameters are respectively set as $\lambda = 40$ and $\kappa = 128$. The item bit length is $\sigma = 128$. As [17], the item's share bit length is $\ell = \lambda + \log_2(m)$. For each party, we run it by using a single thread as previous works [4, 17]. For each setting, we run 10 times and take the average as [17]. Our protocols are written in C++ as [4, 17]. Our source codes are at https://github.com/private-panda/oring.

6.1 Optimizations

There are two fine-grained optimizations when we implement the OKVS and OPRF. For the OKVS, there are two phases: peeling and unpeeling. The peeling phase is only related to the keys and the unpeeling phase is to assign the values for the keys. Our first optimization is to do the peeling for each party (by taking the items as the keys) before interactions with other parties. Then each party can immediately do the unpeeling (rather than peeling and unpeeling) upon receiving the values from other parties, thus supporting better concurrency. Having this optimization, the running time of the OKVS chain can be 6.3% $\sim$ 52.5% faster, as is shown in Table 2.

$m \backslash n$	3	10
2^{12}	33/46 (27.3%)	169/216 (22.1%)
2^{16}	429/699 (38.6%)	2883/3078 (6.3%)
2^{20}	10412/15001 (30.6%)	29823/62766 (52.5%)

Table 2: The running time (in ms) comparisons for OKVS chain with/without our optimization. In each cell, the left and right data of '/' are respectively for our optimization and the naive one; the data in '()' is the improvement percentage.

In the OPRF invocations, each receiver needs to do the OKVS encoding before interactions with multiple senders. Specifically, an OPRF receiver P_i needs to do

$Encode(\{(x_j^i, H_1(x_j^i))\}_{\forall j\in[m]})$, which is independent with other senders. Here $H_1(\cdot) : \{0,1\}^\sigma \mapsto \{0,1\}^\ell$ is a random oracle. Therefore, there is no need for an OPRF receiver P_i to repeatedly do the encoding for multiple times. In our second optimization, we only do $Encode(\{(x_j^i, H(x_j^i))\}_{\forall j\in[m]})$ once for each OPRF receiver, thus saving computation costs. In Table 3, we compare the running time by utilizing this optimization with that by having n' naive OPRF invocations. From this table, we can find that one can save $1.0\% \sim 24.0\%$ running time for $n' = 2, 9$ and $2^{12} \le m \le 2^{20}$ after optimization. We call our optimization as a multi-sender OPRF. More details are in Appendix B.

$m\backslash n'$	2	9
2^{12}	49/51/113 (3.1%)	95/113 (16.1%)
2^{16}	363/367 (1.0%)	659/867 (24.0%)
2^{20}	9093/9336 (2.6%)	14565/15555 (6.4%)

Table 3: The running time (in ms) comparisons between our optimization (by having an OPRF receiver with n' OPRF senders) with naive n' OPRF invocations. In each cell, the left and right data of '/' are respectively for our optimization and the naive one; data in '()' is the improvement percentage.

6.2 Performance comparisons

We respectively compare the running time and communication costs of O-Ring and K-Star after the above optimizations with other state-of-the-art mPSI protocols [4, 17] in Table 4 and 6. The comparisons are made in the same benchmark environment (i.e., same machine and bandwidth) by running their source codes. In both tables, the client is the party with the lowest workload; the *bearer* is the party with the largest workload[2]. For our K-Star and O-Ring, the *bearer* is P_n and the client is P_2.

6.2.1 K-Star *vs* O-Ring

The difference between O-Ring and K-Star lies in the last step in Figure 10. The ring-based design O-Ring distributes the transition costs to all parties. However, the star-based design K-Star assigns the costs to the bearer P_n. Therefore, P_n gets larger computation costs and communication costs.

In Table 4, we do not show the running time of P_n. The running time of P_n is slightly smaller than the total running time of P_1. For other star-based protocols [4, 25], the running time is the central party's running time. When $n = 3$, O-Ring and K-Star have the same topology, thus having the same performance. Since the star structure gains better concurrency, K-Star runs faster than O-Ring in other settings. For example, when $n = 10$, K-Star runs $13.6\% \sim 19.3\%$ faster in the LAN setting. For the client running time, K-Star and O-Ring are close. O-Ring can share a part of the bearer's cost in K-Star to the other parties, especially the communication costs. From Table 6, we can find that O-Ring has the lowest communication costs for the bearer P_n, which is $3.0\% \sim 63.4\%$ lower than K-Star except for the case when $n = 3$. For the client P_1 in both protocols, the only difference is that P_1 sends an OVKS table to P_2 in O-Ring and sends an OKVS table to the bearer P_n in K-Star, thus having the same communication costs. Though the topology of K-Star is different from O-Ring, the total communication costs do not change. For theoretical communication costs, we put them in Table 5.

From the above analyses, one can find that O-Ring is more suitable than K-Star in applications where the bearer has relatively limited communication and computation resources. In contrast, if the bearer's resources are sufficient, K-Star is preferable because it runs faster.

After comparing O-Ring and K-Star, we continue to make comparisons between them with [4, 17]. Compared with these protocols, O-Ring and K-Star run the fastest and achieve the lowest communication costs in the most of the settings. More details are as follows.

6.2.2 Comparisons with KMPRT'17 [17]

We compare K-Star and O-Ring with the semi-honest protocol in [17] under the same collusion security model, i.e., supporting an arbitrary $t \in [1, n-1]$. Kolesnikov et al. [17] proposed their mPSI protocol by using an OPPRF primitive. Their OPPRF protocols are built upon the OPRF primitives, thus more expensive than OPRF protocol. We build our protocols by using OPRF. Considering $t = n-1$, in their protocol, each party first needs to have $n-1$ OPPRF invocation with other parties and have another invocation with the bearer. Therefore, the total number of invocations is $n \times (n-1) + (n-1) = (n+1)(n-1)$. In comparison, our O-Ring and K-Star only needs $n \times (n-1)$ invocations of the OPRF functionality. When $t < n-1$, they need $(t+1) \times n + (n-1)$ OPPRF invocations. In comparison, we need $t(n-1)$ OPRF invocations. Additionally, we need $n-1$ OKVS encoding and decoding. The cost of OKVS can be very small [8][3]. Therefore, our protocols are theoretically more efficient than [17].

From Table 4, in the LAN setting, O-Ring is respectively $1.5\times \sim 13.2$ and $1.4\times \sim 6.0\times$ as fast as KMPRT'17 [17] in the client running time and the total running time (except for one setting $(n, t, m) = (10, 1, 2^{20})$). The exception is because O-Ring takes a ring design while KMPRT'17 [17] has a star structure. in the WAN setting, we also observe one exceptions in the settings $(n, t, m) = (10, 1, 2^{12})$, in which O-Ring is 25.5% and 11.1% slower in the total running time. In other

[2] Here, we do not use *leader* as the traditional mPSI protocols that are based on a star topology. The reason is that we do not have such a central party in O-Ring that is based on a ring topology.

[3] In [8], one can encode $m = 2^{20}$ key-values pairs for 7.13s and decode $m = 2^{20}$ keys for 1.09s with a table size about $1.3m$.

Network	Protocol	(n,t)	(3,1)			(3,2)			(10,1)			(10,4)			(10,9)		
		m	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
LAN	KMPRT'17 [17]	Client	0.17	1.1	15.7	-	-	-	0.27	1.7	25.2	0.48	4.3	62.9	0.90	11.0	156
		Total	0.20	1.5	21.3	-	-	-	0.33	2.4	35.3	0.53	5.0	73.9	0.98	11.9	171
	CDGOSS'21 [4]	Client	0.82	1.5	17.8	-	-	-	1.67	2.3	22.9	1.66	2.4	22.8	-	-	-
		Total	0.82	1.5	17.9	-	-	-	1.68	2.4	23.1	1.66	2.4	23.1	-	-	-
	O-Ring	Client	**0.02**	**0.4**	**10.7**	**0.08**	**0.5**	**13.0**	**0.03**	**0.5**	**11.2**	**0.11**	**0.7**	**13.3**	**0.18**	**0.8**	**16.3**
		Total	**0.06**	**0.5**	**14.8**	**0.10**	**0.7**	**16.9**	0.14	1.6	39.6	0.22	1.9	41.4	0.29	2.0	44.6
	K-Star	Client	**0.02**	**0.4**	**10.7**	**0.08**	**0.5**	**13.0**	**0.03**	**0.5**	**11.2**	**0.11**	**0.7**	**13.3**	**0.18**	0.9	**16.6**
		Total	**0.06**	**0.5**	**14.8**	**0.10**	**0.7**	**16.9**	**0.12**	**1.0**	**20.1**	**0.19**	**1.1**	**22.3**	**0.23**	**1.3**	**25.3**
WAN	KMPRT'17 [17]	Client	1.86	18.3	284.1	-	-	-	2.65	7.7	147	**2.86**	27.9	570	4.79	60.4	745
		Total	3.21	21.7	324.9	-	-	-	4.43	18.9	290	4.98	43.0	709	6.72	74.9	1261
	CDGOSS'21 [4]	Client	3.14	6.6	46.1	-	-	-	3.62	10.6	136	3.70	10.6	135	-	-	-
		Total	3.24	6.7	46.5	-	-	-	3.72	10.9	137	**3.80**	10.9	136	-	-	-
	O-Ring	Client	**0.36**	**0.7**	**12.8**	**3.24**	**3.8**	**19.3**	**1.3**	1.7	**13.2**	3.67	**4.4**	**35.1**	5.24	**7.4**	**64.8**
		Total	**1.92**	**3.1**	**18.0**	**3.36**	**4.3**	**24.3**	**3.6**	6.7	50.1	5.60	8.7	69.9	5.84	10.4	98.5
	K-Star	Client	**0.36**	**0.7**	**12.8**	**3.24**	**3.8**	**19.3**	**1.3**	**1.6**	15.4	3.70	**4.4**	37.8	**5.23**	7.5	65.4
		Total	**1.92**	**3.1**	**18.0**	**3.36**	**4.3**	**24.3**	**3.6**	**6.6**	**31.4**	5.60	**7.8**	**50.5**	**5.74**	**9.0**	**78.0**

Table 4: The running time comparisons (in seconds) between our protocols with other mPSI protocols. The best results are marked in bold. Cell with '-' indicates that the setting is not supported.

Protocol	Client	Bearer	Total
O-Ring ($t=1$)	ϕ	$2\phi+\omega$	$\phi'+\omega$
O-Ring ($t>1$)	$\phi+t\omega$	$2\phi+(n-1)\omega$	$\phi'+(n-1)t\omega$
K-Star ($t=1$)	ϕ	$\phi'+(n-1)\omega$	$\phi'+\omega$
K-Star ($t>1$)	$\phi+t\omega$	$\phi'+(n-1)\omega$	$\phi'+(n-1)t\omega$

Table 5: The theoretical communication costs (in bits) for O-Ring and K-Star. $\omega = 1.3\kappa m + \ell m + 2^{13}\kappa m^{0.13}$ is the OPRF cost [26]. $\phi = (1.3m+\lambda+0.5\log(m))\ell$ is the OKVS table bit size [8]. $\phi' = (n-1)\phi$.

settings, O-Ring is $1.2\times \sim 18.1\times$ as fast as theirs. Our O-Star has the same star structure as KMPRT'17 [17]. From Table 6, in the LAN setting, K-Star is $1.6\times \sim 12.7\times$ and $1.4\times \sim 9.0\times$ as fast in the client running time and the total running time. in the WAN setting, the ratios for the client running time are $2.0 \sim 26.2\times$ with two exceptions in the settings $(n,t,m) = (10,4,2^{12}),(10,9,2^{12})$. In these two exceptions, K-Star is respectively 22.7% and 8.5% slower. As for the total running time, K-Star is $1.2\times \sim 18.9\times$ as fast with one exception in the setting $(n,t,m) = (10,4,2^{12})$. In this exception, K-Star runs 11.1% slower. In communication, O-Ring and K-Star are respectively $1.4\times \sim 82.6\times$ and $1.6\times \sim 48.3\times$ as cheap as KMPRT'17 [17] in the client's costs and the total costs. For bearer's communication costs, O-Ring and K-Star are respectively $1.4\times \sim 26.9\times$ and $1.4\times \sim 14.4\times$ as cheap.

In KMPRT'17 [17], the authors specially proposed an optimized protocol for $n=3$. However, when $(n,t) = (3,2)$, their protocol is insecure. More details are in subsection 6.4.3. Therefore, we do not report their data in this setting in Table 4 and 6. In [17], the authors also proposed another protocol that gained better performance based on a weaker security model, i.e., the augmented semi-honest security model. In this paper, we only focus on the semi-honest model. Also, their augmented semi-honest protocol is prone to the collusion attack, which will be also elaborated in subsection 6.4.1.

6.2.3 Comparisons with CDGOSS'21 [4]

Chandran et al. [4] proposed their protocol based on their proposed functionality called wPSM, which was instantiated by the OPPRF protocols in [17]. Additionally, after the OPPRF invocations between the center P_1 and the other $n-1$ parties, they need other multi-party functionalities, e.g., the *ConvertShares* that can exploit (n,t) secret sharing to generate shares. One drawback of their protocol is that their protocol is limited in $t < n/2$. Therefore, we only provide the performance data for $t < n/2$ in Table 4 and Table 6.

From Table 4, for the client running time, O-Ring is respectively $1.7\times \sim 57.1$ and $1.0\times \sim 10.0\times$ as fast as CDGOSS'21 [4] in the LAN and WAN settings. For the total running time, except for two setting $(n,t,m) = (10,1,2^{20}),(10,4,2^{20})$, O-Ring is respectively $1.2\times \sim 12.9\times$ as fast as CDGOSS'21 [4] in the LAN setting. In the WAN setting, except for one setting $(n,t,m) = (10,4,2^{12})$, O-Ring is $1.1\times \sim 2.7\times$ as fast. For K-Star, for the client running time, it is respectively $1.7\times \sim 55.5\times$ and $1.0\times \sim 8.7\times$ as fast as CDGOSS'21 [4] in the LAN and WAN settings. In the total running time, K-Star is $1.0\times \sim 15.3\times$ as fast in the LAN setting. Except for one setting $(n,t,m) = (10,4,2^{12})$, K-Star is $1.0\times \sim 4.4\times$ as fast. In this exception, CDGOSS'21 [4] is $1.5\times$ as fast as K-Star. In communication, O-Ring is respectively $1.2\times \sim 69.9\times$ and $2.0\times \sim 67.8\times$ as cheap as CDGOSS'21 [4] in the client costs and the bearer's costs. For the total costs, except for the setting $(n,t,m) = (10,4,2^{12})$, O-Ring is $4.0\times \sim 39.8\times$ as cheap as theirs. In $(n,t,m) = (10,4,2^{12})$, CDGOSS'21 [4] is $1.1\times$ as cheap as O-Ring. In the bearer's costs, K-Star is $1.9\times \sim 27.5\times$ as cheap as CDGOSS'21 [4].

Protocol	(n,t)	(3,1)			(3,2)			(10,1)			(10,4)			(10,9)		
	m	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
KMPRT'17 [17]	Client	2.19	36.9	639	-	-	-	3.14	46.6	743.2	3.14	46.6	743.2	19.37	326.6	5668
	Bearer	2.19	36.9	639	-	-	-	4.04	68.6	1195	10.63	179.2	3112	19.37	326.6	5668
	Total	3.29	55.3	958.5	-	-	-	19.75	331.8	5751	49.39	829.6	14377	96.87	1633	28341
CDGOSS'21 [4]	Client	1.79	27.6	441.1	-	-	-	2.55	39.8	635.7	2.55	39.8	635.7	-	-	-
	Bearer	3.02	46.3	738.9	-	-	-	12.37	189.1	3018	12.37	189.1	3018	-	-	-
	Total	3.29	50.8	810.5	-	-	-	17.65	273.6	4369	**17.65**	273.6	4369	-	-	-
O-Ring	Client	**0.04**	**0.6**	**11.0**	**1.56**	**5.8**	**78.5**	**0.04**	**0.6**	**11.0**	**2.07**	**7.5**	**101.0**	**8.68**	**30.2**	**393.3**
	Bearer	**0.58**	**2.9**	**44.5**	**1.60**	**6.4**	**89.5**	**0.58**	**2.9**	**44.5**	**6.18**	**22.1**	**291.9**	**8.72**	**30.8**	**404.3**
	Total	**0.58**	**2.9**	**44.5**	**2.11**	**8.1**	**112**	**0.85**	**6.9**	**121.6**	18.64	**67.9**	**908.7**	**41.51**	**146.3**	**1921**
K-Star	Client	**0.04**	**0.6**	**11.0**	**1.56**	**5.8**	**78.5**	**0.04**	**0.6**	**11.0**	**2.07**	**7.5**	**101.0**	**8.68**	**30.2**	**393.3**
	Bearer	**0.58**	**2.9**	**44.5**	**1.60**	**6.4**	**89.5**	0.85	6.9	121.6	6.44	26.0	368.9	8.98	34.8	481.4
	Total	**0.58**	**2.9**	**44.5**	**2.11**	**8.1**	**112**	**0.85**	**6.9**	**121.6**	18.64	**67.9**	**908.7**	**41.51**	**146.3**	**1921**

Table 6: The communication cost (in MB) comparisons between our protocols with other mPSI protocols. The best results are marked in bold. Cell with '-' indicates that the setting is not supported. The *bearer* is the party with the largest workload.

In addition to the above experiment results, we also compare the total theoretical communication costs. From CDGOSS'21 [4], the total communication cost of their protocol is $m(n-1)(4.5\kappa+35\ell+140)$ bits (i.e., in $O(mn)$), which is independent with the number of parties t and linear with n. However, O-Ring and K-star need $(n-1)(1.3m+\lambda+0.5\log(m))\ell+(n-1)t(1.3\kappa m+\ell m+2^{13}\kappa m^{0.13})$ bits (i.e., in $O(mnt)$). Therefore, when t is large (e.g., $t=n/2-1$), their protocol is expected to have better performance than O-Ring and K-Star. For example, in Table 4, when $m=2^{12}$ and $(n,t)=(10,4)$, their protocol needs 17.65 MB in total, which is 5.3% cheaper than O-Ring and K-Star. However, for a large set size $m=2^{20}$, even when we increase n to 54[4], their protocol still needs more communication costs than O-Ring and K-Star. Specifically, their protocol needs 50320.7 MB, while O-Ring and K-Star only need 31573.2 MB, which is 37.3% cheaper. As for the total running time, from Table 4, when $(n,t,m)=(10,4,2^{12})$ in the WAN setting, their protocol can already run 32.1% faster. For a large set size $m=2^{20}$ in the LAN setting, when increasing n to 11 and set the largest $t=5$, their protocol can also run faster. Specifically, their protocol runs in 24.1 seconds, which is 9.8% faster than K-Star.

6.3 Application: secure medical data integration

In [11], Gon et al. showed that integrating medical records between different institutions was useful to investigate the correlation between stroke and cancer. In privacy-preserving medical data integration [19, 20], multiple institutions aim to integrate their patient databases without violating the patient or commercial privacy. To achieve this, Miyaji et al. [20] proposed a mPSI protocol by using Bloom filter and exponential ElGamal homomorphic encryption (exElGamalHE) [5].

[4]For a larger set size, their program simply crashes.

[5]In [19], the authors directly used the mPSI protocol in [20]. Therefore, we do not put it into analysis in this paper.

To test the efficiency of our protocols for secure medical data integration, we set the largest number of institutions $n=16$ and the largest database size $m=2^{14}$ as [20]. For the number of colluding parties, we also consider $t=n-1$ as theirs. The bandwidth is also the same as their (i.e., 2 Gbps). As a result, our O-Ring and K-Ring respectively run in 1.5 seconds and 1.2 seconds, which are cheap. In comparison, their protocol needs >1044 seconds [6], which are expensive. In communication costs for the client and the bearer, O-Ring (resp. K-Star) needs to take 22.7 MB (resp. 22.7 MB) and 22.9 MB (resp. 24.9 MB). For the total communication cost, both O-Ring and K-Star need 177.6 MB. The communication costs are very low. In comparison, the protocol in [20] respectively needs to take 1731.2 MB, 22159.8 MB, and 42934.6 MB for the client, the bearer, and the total cost, which are respectively $>76.2\times$, $>890.0\times$, and $>365.1\times$ as expensive as O-Ring and K-Star.

6.4 Insecurity of prior protocols

6.4.1 Augmented semi-honest protocol [17]

In section 2, we have introduced the augmented semi-honest protocol in [17]. Though it gets good performance, it is prone to the collusion attack when the protocol receiver P_1 joins the collusion. One can first check the case for $t=n-1$. Since the PRF keys sharing is symmetric, if $t=n-1$, all the PRF keys are known to the colluding parties, thus bringing no security benefit. Assuming P_i $(i>1)$ is the non-colluding party and its PRF keys are $s^{i,j}$ $(\forall j\in[n]\backslash i)$, then the share of each item $x^i\in X_i$ is $v^i=\bigoplus_{j=1}^{i-1}F_{s^{i,j}}(x^i)\oplus\bigoplus_{j=i+1}^{n}F_{s^{i,j}}(x^i)$. With the collusion between the receiver with the other $n-2$ parties, the receiver P_1 knows all $s^{i,j}$ $(\forall j\in[n]\backslash i)$. After the OPPRF invocation between P_i, it is easy for P_1 to know the intersection

[6]Their protocol is very time-consuming. In our implementation of their protocol, only encrypting the Bloom filter by using exElGamalHE takes 1044 seconds. There are also other expensive operations, e.g., computing the sums of the encrypted Bloom filter.

$X_1 \cap X_i$ by comparing the OPPRF value of each item $x^1 \in X_1$ with v^1. If they are equal, then x^1 is in the intersection; otherwise not. As a consequence, the colluding parties know extra information beyond the intersection $\bigcap_{i=1}^{n} X_i$, which is not allowed. In fact, one can further generalize the above collusion attack for $t < n-1$ such that the colluding parties can know $X_1 \cap \bigcap X_i$, where $\bigcap X_i$ is the intersection of the $n-t$ non-colluding parties. Since there are $t-1$ parties who are colluding with the protocol receiver P_1, P_1 can know all the PRF keys between them with the $n-t$ honest parties. Now, after the OPPRF between these $n-t$ honest parties, for each item $x^1 \in X_1$, P_1 simply checks whether its OPPRF value sum is equal to the PRF value sum by using the PRF keys that are known to colluding parties. If there is a match, it indicates that x^1 is in $X_1 \cap \bigcap X_i$. It is noted that none of colluding parties except for P_1 needs to use their sets in the above attack.

6.4.2 ZeroXOR [21]

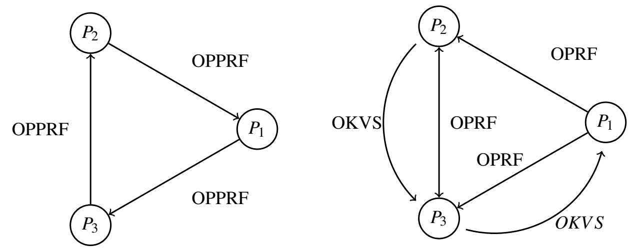
In section 2, we have introduced the zeroXOR protocol and mPSI protocol in [21]. When $t = n-1$, their mPSI protocol becomes the zeroXOR protocol in which all items' values of the n parties are 0s and their mPSI protocol is almost the same as the augmented semi-honest protocol [17], thus it is also prone to the collusion attack.

When $1 < t < n-1$[7], their mPSI protocol is also insecure. Specifically, the pivot P_w can collude with the protocol receiver P_1 to know information beyond the intersection $\bigcap_{i=1}^{n} X_i$. After receiving an OKVS table T_i from a non-zeroXOR party P_i, P_w decodes and computes each item x_j^w's associated value $v_j^w = Decode(T_i, x_j^w) \oplus v_j'$ and performs OPPRF with the protocol receiver P_1 by inputting $\{(x_j^w, v_j^w)\}_{\forall j \in [m]}$, where v_j' is the PRF value sum in the zeroXOR. If the final value sum of $x_j^1 \in X_1$ is 0, then P_1 will include x_j^1 into the intersection. Now, assuming all the other parties have an item x except for the pivot party P_w. Then P_w and P_1 can collude to know if P_i has x. Specifically, P_w can decode T_i by using x and compute $v'' = Decode(T_i, x) \oplus v'$, where v' is the PRF value sum in the zeroXOR. P_w shows v'' to P_1. If v'' is matched, it indicates that P_i has x. It is noted that P_w can compute any PRF value sum including v' because he/she has the shared PRF keys with the other t parties in the zeroXOR. Therefore, their protocol is also insecure when $1 < t < n-1$.

6.4.3 Optimized three-party protocol [17]

In addition to their augmented protocol and semi-honest general protocol, the authors [17] also specially proposed an optimized three-party PSI protocol. In this protocol, the protocol receiver P_1 and P_2 first invoke an OPPRF instance, in which P_1 acts as the sender and P_2 acts as the receiver. Then P_2 interacts with P_3 by acting as the OPPRF sender. Finally, P_3 interacts with P_1 by acting as the OPPRF sender. One can find the working flow in sub-Figure 13a.

[7]In [21], the authors also specially proposed a protocol when there is no collusion (i.e., $t = 1$), which is secure by using a two-party PSI primitive.



(a) KMPRT [17] (b) O-Ring

Figure 13: Three-party PSI protocol workflow comparisons.

However, their proposed three-party PSI protocol is also insecure. As the sender, P_1 knows the OPPRF key and can get the key-value pairs in the full item domain. Then P_3 will also know $X_2 \cap X_3$ by colluding with P_1. Actually, P_1 is not supposed to act as a sender in any OPPRF (or OPRF) interaction with other parties in case that he/she launches the brute-force attack when the input domain is small. In another collusion attack, P_2 and P_1 can collude to know if P_3 has an item x even $x \notin I$. Specifically, assuming P_1 and P_3 both have x, then P_1 can get the correct value v from P_3. For P_3, v is got from P_2. If P_2 does not have x, P_3 will get the incorrect v and then give v to P_1 and P_1's colluding party P_2. Since P_2 has the OPPRF key, he/she can have the brute-force attack for the input domain until he/she finds x. Or a more simple way of P_2 is to compute the OPPRF values of P_1's items until he/she finds v and knows x. Now, P_2 and P_1 knows P_3 has x even $x \notin I$. Intuitively, it is insecure for an OPPRF receiver to disclose its OPPRF values to the OPPRF sender.

In comparison, our three-party design can thwart the above collusion attacks because both P_2 and P_3 are a common OPRF sender to the other two parties. The working flow of our three-party protocol is shown in sub-Figure 13. It is noted that O-Ring and K-Star share the same topology when $n = 3$. In the first step of our protocol, P_1 invokes the multi-sender OPRF instances with P_2 and P_3 by acting as the receiver. As the receiver of the multi-sender OPRF, only the intersection items' values can be correctly distributed. P_1 cannot know the item-value pairs in the item domain because he/she does not have the OPRF key. P_2 and P_3 both act as a common OPRF sender to the other parties. For an item x of P_2 or P_3, if it is not in common, at least one of its two OPRF values will be pseudorandom to the other parties because he/she holds the OPRF keys. Then the sum of the three OPRPF values (i.e., two values as an OPRF sender and one value as an OPRF receiver) will also be pseudorandom to the other parties, thus disclosing nothing to them. In the above second collusion attack, if P_2 does not have x, the OPRF value for x of P_3 will be pesudorandom to P_2 because P_3 is an OPRF sender (to P_2) who holds the key.

7 Conclusion and Future work

In this paper, we propose two protocols to achieve mPSI. Both protocols can be secure against an arbitrary number of colluding parties. Our first protocol O-Ring is ring-based and has the minimum bearer communication costs. Our second protocol is star-based and can run the fastest in the most of the settings. Both O-Ring and K-Star can achieve the lowest communication costs in the most of the settings. Technically, we utilize the OKVS and OPRF. To better support concurrency and save computation costs, we make find-grained optimizations when we implement the OKVS and OPRF in our protocols. We believe these optimizations can be utilized in other applications where a party needs to have multiple OPRF invocations with other parties by acting as a common OPRF receiver.

In this paper, we only focus on the semi-honest security model. Though Garimella et al. [8] has proved that their malicious mPSI protocol is not secure in the semi-honest model, there are also other works that focus on the malicious security model (e.g., [2,9]). By introducing a random oracle, Garimella et al. [8] designed their malicious mPSI protocol on top of the augmented semi-honest protocol in KMPRT'17 [17]. We believe using a random oracle is also helpful to make our protocols maliciously secure. We keep it as a future work.

Acknowledgments

This research is supported by the HKU SCF Fintech Academy under grant No. 260910193 and Hong Kong RGC ECS/GRF grant No. 17207322 and 27208621. We appreciate the constructive reviews from the anonymous reviewers.

References

[1] Aslí Bay, Zekeriya Erkin, Jaap-Henk Hoepman, Simona Samardjiska, and Jelle Vos. Practical multi-party private set intersection protocols. *IEEE Trans. Inf. Forensics Secur.*, 17:1–15, 2022.

[2] Aner Ben-Efraim, Olga Nissenbaum, Eran Omri, and Anat Paskin-Cherniavsky. Psimple: Practical multiparty maliciously-secure private set intersection. In Yuji Suga, Kouichi Sakurai, Xuhua Ding, and Kazue Sako, editors, *ASIA CCS '22*, pages 1098–1112. ACM, 2022.

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

[4] Nishanth Chandran, Nishka Dasgupta, Divya Gupta, Sai Lakshmi Bhavana Obbattu, Sruthi Sekar, and Akash Shah. Efficient linear multiparty PSI and extensions to circuit/quorum PSI. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21*, pages 1182–1204. ACM, 2021.

[5] Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020*, volume 12172 of *Lecture Notes in Computer Science*, pages 34–63. Springer, 2020.

[6] Jung Hee Cheon, Stanislaw Jarecki, and Jae Hong Seo. Multi-party privacy-preserving set intersection with quasi-linear complexity. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 95-A(8):1366–1378, 2012.

[7] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *CCS'13*, pages 789–800. ACM, 2013.

[8] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021*, volume 12826 of *Lecture Notes in Computer Science*, pages 395–425. Springer, 2021.

[9] Satrajit Ghosh and Tobias Nilges. An algebraic approach to maliciously secure private set intersection. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019*, volume 11478 of *Lecture Notes in Computer Science*, pages 154–185. Springer, 2019.

[10] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[11] Yasufumi Gon, Daijiro Kabata, Keiichi Yamamoto, Ayumi Shintani, Kenichi Todo, Hideki Mochizuki, and Manabu Sakaguchi. Validation of an algorithm that determines stroke diagnostic code accuracy in a japanese hospital-based cancer registry using electronic medical records. *BMC Medical Informatics Decis. Mak.*, 17(1):157:1–157:8, 2017.

[12] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. Scalable multi-party private set-intersection. In Serge Fehr, editor, *PKC 2017*, volume 10174 of *Lecture Notes in Computer Science*, pages 175–203. Springer, 2017.

[13] Roi Inbar, Eran Omri, and Benny Pinkas. Efficient scalable multiparty private set-intersection via garbled bloom filters. In Dario Catalano and Roberto De Prisco, editors, *SCN 2018*, volume 11035 of *Lecture Notes in Computer Science*, pages 235–252. Springer, 2018.

[14] Alireza Kavousi, Javad Mohajeri, and Mahmoud Salmasizadeh. Efficient scalable multi-party private set intersection using oblivious PRF. In Rodrigo Roman and

Jianying Zhou, editors, *STM 2021*, volume 13075 of *Lecture Notes in Computer Science*, pages 81–99. Springer, 2021.

[15] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 241–257. Springer, 2005.

[16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *CCS 2016*, pages 818–829. ACM, 2016.

[17] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *CCS 2017*, pages 1257–1272. ACM, 2017.

[18] Seitaro Mishima, Kazuhisa Nakasho, Yuuki Takano, and Atsuko Miyaji. A practical parallel computation in a scalable multiparty private set intersection. In *CANDAR 2021*, pages 332–338. IEEE, 2021.

[19] Seitaro Mishima, Kazuhisa Nakasho, Kousuke Takeuchi, Naohiro Hayaishi, Yuuki Takano, and Atsuko Miyaji. Development and application of privacy-preserving distributed medical data integration system. In *ICCE-TW 2020*, pages 1–2. IEEE, 2020.

[20] Atsuko Miyaji, Kazuhisa Nakasho, and Shohei Nishida. Privacy-preserving integration of medical data - A practical multiparty private set intersection. *J. Medical Syst.*, 41(3):37:1–37:10, 2017.

[21] Ofri Nevo, Ni Trieu, and Avishay Yanai. Simple, fast malicious multiparty private set intersection. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21*, pages 1151–1165. ACM, 2021.

[22] Duong Tung Nguyen and Ni Trieu. Mpccache: Privacy-preserving multi-party cooperative cache sharing at the edge. In Ittay Eyal and Juan A. Garay, editors, *FC 2022*, volume 13411 of *Lecture Notes in Computer Science*, pages 80–99. Springer, 2022.

[23] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: Lightweight private set intersection from sparse OT extension. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019*, volume 11694 of *Lecture Notes in Computer Science*, pages 401–431. Springer, 2019.

[24] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from paxos: Fast, malicious private set intersection. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020*, volume 12106 of *Lecture Notes in Computer Science*, pages 739–767. Springer, 2020.

[25] Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017*, volume 10210 of *Lecture Notes in Computer Science*, pages 235–259, 2017.

[26] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: fast OPRF and circuit-psi from vector-ole. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021*, volume 12697 of *Lecture Notes in Computer Science*, pages 901–930. Springer, 2021.

[27] En Zhang, Feng-Hao Liu, Qiqi Lai, Ganggang Jin, and Yu Li. Efficient multi-party private set intersection against malicious adversaries. In Radu Sion and Charalampos Papamanthou, editors, *CCSW@CCS 2019*, pages 93–104. ACM, 2019.

A Security proof of O-ring and K-Star

A.1 O-Ring (t=1)

Theorem 3. *Our protocol in Figure 6 realizes $\mathcal{F}_{m-psi}$ against a semi-honest adversary in the $\mathcal{F}_{oprf}$-hybrid model.*

Proof. For the security, the receiver can no longer brute-force the full input domain even if it is small because of OPRF. Without OPRF between P_1 and P_n, P_1 can have the brute-force attack to know $\bigcap_{i=2}^{n}$ by checking P_n's OKVS table T_n. To check if an item x is in $\bigcap_{i=2}^{n} X_i$, P_1 can simply check $Decode(T_n, x) = F_s(x)$ even when $x \notin X_1$ because P_1 has the PRF key s. If yes, then x is in $\bigcap_{i=2}^{n}$. Having the OPRF, as an ORPF receiver, P_1 cannot know $F_k(x)$ if $x \notin X_1$ because P_n holds the key k. Therefore, P_1 cannot know $\bigcap_{i=2}^{n} X_i$. Here we also provide formal security proof.

- **Corrupt** P_2. In the protocol, P_2 receives an PRF key from P_1. Therefore, P_2's view can be simulated by returning a random seed. This view is indistinguishable.

- **Corrupt** P_i ($i \in [3, n-1]$). P_i receives an OKVS table T_{i-1} from its previous party in the protocol. P_i's view can be simulated by returning a random table with the same size. Given the oblivious property of OKVS, this view is also indistinguishable.

- **Corrupt** P_n. P_n receives an OKVS table T_{n-1} from P_{n-1} in the protocol and gets an OPRF key k from $\mathcal{F}_{oprf}$. T_{n-1} and k are independent. P_n's view can be simulated by returning a random table with the same size as T_{n-1} in the real protocol and a random key. Given the oblivious property of OKVS and the security of OPRF, this view is also indistinguishable.

- **Corrupt** P_1. P_1 receives an OKVS table T_n from P_n and an OPRF set $\{F_k(x_j^1)\}_{\forall j\in[m]}$ from $\mathcal{F}_{oprf}$. The simulator can observe P_1's output set I. P_1's view can be simulated by returning an OKVS table T'_n. Here T'_n is encoded by $\{(x'_i, v'_i)\}_{i=1,2,\cdots,m-|I|} \cup I'$, where $\{x'_i\}_{m-|I|}$ are non-common items, $\{v'_i\}_{m-|I|}$ are randomly sampled values, and I' are item-value pairs that correspond to the common items and each common item x's value is $Decode(T_n, x)$. Given the obliviousness of OKVS and the security of OPRF, this view is indistinguishable.

□

A.2 K-Star ($t = 1$)

Theorem 4. *Our K-Star ($t = 1$) in Figure 12 realizes $\mathcal{F}_{m-psi}$ against a semi-honest adversary in the $\mathcal{F}_{oprf}$-hybrid model.*

Proof. The correctness is the same as O-Ring ($t = 1$). For security, similar to O-Ring ($t = 1$), because of the OPRF between P_1 and P_n, P_1 cannot have the brute-force attack to know $\bigcap_2^n X_i$ even the input domain is small. The view of a corrupt party can be simulated as follows:

- **Corrupt** P_2. Same as O-Ring ($t = 1$).
- **Corrupt** P_i ($i = 3, 4, \cdots, n-1$). Same as P_2.
- **Corrupt** P_n. Similar as O-Ring ($t = 1$), P_n's view can be simulated by returning $n-2$ random OKVS tables with the same size as T_i in the read protocol and a random key. Given the oblivious property of OKVS and the security of OPRF, this view is also indistinguishable.
- **Corrupt** P_1. Same as O-Ring ($t = 1$).

□

A.3 O-Ring ($t > 1$)

Proof. Dividing the n parties into two groups C and O, which are respectively a corrupt one in which the parties collude and an honest one in which all parties are honest. If the protocol receiver P_1 is honest, the protocol is secure because P_1 discloses nothing about his/her set to the other parties and the values of each other party is obfuscated by P_1's shared values. Now, we assume P_1 is a colluder in C. Then C and O can be taken as two large parties in the protocol. C can be the protocol receiver who gets the intersection and O can be the protocol sender who sends messages to the receiver such that the receiver can compute the intersection. Since the OPRF are the security building block of the protocol. O is supposed to an OPRF sender to C. Since there must be at least one common OPRF sender (say it P_i) among the t common OPRF senders $P_b, P_c, \cdots, P_n$ and all other parties in O are connected with P_i, O can be taken as an OPRF sender to C. Specifically, we consider the following cases:

1. $|O| = 1$. In this case, there is only one party P_i is O. Since P_i is a common OPRF sender to the others, if $x^i \in X_i$ is not in common, its OPRF value sum v^i in step 2 of the protocol is pseudorandom to C. Then C's view can be simulated by returning an OKVS table T'_i (in step 3) in which x^i's value is randomly sampled. This view is indistinguishable. Therefore, the protocol is secure when $|O| = 1$.
2. $|O| > 1$. In this case, denoting P_j as another honest party, for an item x of C, there are two further cases to be considered:
   (a) $x \in X^i \cap X^j$. If x is common in C, C will know x is in the intersection, which is allowed. If not, assuming P_u does not has x, then $F_{k^{i,u}}(x)$ is pseudorandom to P_u and P_i's value sum in step 2 is pseudorandom to C. Since the honest P_i holds the key $k^{i,j}$, P_j's OPRF value $F_{k^{i,j}}(x)$ is also pseudorandom to C and P_j's value sum v^j in step 2 of the protocol is also pseudorandom to C. If P_i and P_j are adjacent, C's view can be simulated by returning an OKVS table $T'_{\max(i,j)}$ in which x's value is randomly sampled. This view is indistinguishable because the pseudorandom $F_{k^{i,u}}(x)$ is an XOR share of $P_{\max(i,j)}$'s value sum $Decode(T_{\min(i,j)}, x) \oplus v^{\max(i,j)}$ in step 3. It is noted that P_i and P_j cannot rely on the security of their key $k^{i,j}$ when they are adjacent because the XOR sum of their OPRF values by using this key is 0. If P_i and P_j are not adjacent, C's view can simulated by returning two OKVS tables T'_i and T'_j in which x's value are randomly sampled. This view is indistinguishable because $F_{k^{i,u}}(x)$ and $F_{k^{i,j}}(x)$ are pseudorandom to C.
   (b) $x \notin X^i \cap X^j$. In this case, $F_{k^{i,j}}(x^i)$ and $F_{k^{i,j}}(x^j)$ are both pseudorandom to C because P_i has the OPRF key $k^{i,j}$, where $x^i \in X_i$, $x^j \in X_j$, and $x^i \neq x^j$. Since $F_{k^{i,j}}(x^i)$ and $F_{k^{i,j}}(x^j)$ are respectively an XOR share of v^i and v^j, the value sums $Decode(T_{i-1}, x) \oplus v^i$ and $Decode(T_{j-1}, x) \oplus v^j$ are also pseudorandom to C. If P_i and P_j are adjacent, C's view can be simulated by returning an OKVS table $T'_{\max(i,j)}$ in which x's value is randomly sampled. This view is indistinguishable because $F_{k^{i,j}}(x^i)$ is pseudorandom and it is an XOR share of $Decode(T_{\min(i,j)}, x) \oplus v^{\max(i,j)}$ in step 3. It is noted that $Decode(T_{\min(i,j)}$ is also pseudorandom based on the property of OKVS when $x^i \neq x^j$. However, we do not rely its security because of the possible brute-force attack (see subsection 4.1 for related analysis). If P_i and P_j are

not adjacent in step 3, C's view can be simulated by returning two OKVS tables T_i' and T_j' in which both x^i's value and x^j' value are randomly sampled. This view is indistinguishable because $F_{k^{i,j}}(x^i)$ and $F_{k^{i,j}}(x^j)$ are pseudorandom to C.

From above analyses, C cannot distinguish the above two cases. Therefore, O-Ring is also secure when $|O| > 1$.

□

A.4 K-Star ($t > 1$)

Proof. In O-Ring ($t > 1$), the share distribution and share collection steps by using OPRF and masking ensure the security of the protocol. Specifically, having OPRF enables each item's value that is encoded in step 3 is pseudorandom to the colluded parties. If P_1 is honest, having masking can also ensure the encoded values are pseudorandom. K-Star ($t > 1$) shares the same OPRF and masking steps (i.e., step 1 and step 2), thus having the same security. K-Star and O-Ring only differ in the share transition step. In K-Sar, there is a center P_n. In the simulation, if the common OPRF sender P_i is P_n, P_i and P_j will be adjacent. The view of C can be simulated by returns an OKVS table T_n' in which an item x's value is randomly sampled. This view is indistinguishable. If P_i is not P_n, P_i and P_j are not adjacent. The view of C can be simulated by returning two OKVS tables T_i' and T_j' in which x's values are randomly sampled. This view is also indistinguishable. □

B Multi-sender OPRF

Parameters: The number of senders n', the set size of each sender m_i and the receiver m_r. The bit length of each item σ. A PRF $F_{(\cdot)}(\cdot)$ and a function $Q_{(\cdot)}(\cdot)$.
Inputs: The receiver has a set $X = \{x_j\}_{\forall j \in [m_r]}$. The n senders respectively have n sets $Y_i = \{y^i_j\}_{\forall i \in [m_i]}$ for $i \in [n]$, and $x_j, y^i_j \in \{0,1\}^\sigma$.
Outputs: The receiver gets an item-value set $\{(x_j, Q_k(x_j)\}_{\forall j \in [m_r]}$ and the n' senders respectively get an item-value set $\{(y^i_j, F_{s^i}(y^i_j))\}_{\forall j \in [m_i]}$ $(i \in [n'])$, where s_i is the i-th sender's PRF key, $k = (s_1, s_2, \cdots, s_{n'})$. If $x_j \in X \cap \bigcap_{i=1}^{n'} Y_i$, then its corresponding value $Q_k(x_j) = \bigoplus_{i=1}^{n'} F_{s^i}(y^i_*))$, where $*$ is the index of an item $y^i_* \in Y_i$ equaling to x_j; otherwise, $Q_k(x_j)$ is pseudorandom to the receiver.

Figure 14: The ideal functionality of multi-sender OPRF $\mathcal{F}_{ms-oprf}$.

We propose the notation of multi-sender OPRF in Figure 14. In multi-sender OPRF, if an item $x \in X \cap \bigcap_{i=1}^{n'} Y_i$, the receiver gets its OPRF value $Q_k(x)$ and each sender gets a share of $Q_k(x)$. Before describing our construction, we first review the Vector Oblivious Linear Evaluation (VOLE) [26], which is a building block for our construction. The ideal functionality of VOLE is shown in Figure 15.

Parameters: The set size m. The computational security parameter κ.
Inputs: Both the receiver and the sender input nothing.
Outputs: The sender receives $\Delta \in \{0,1\}^\kappa$, $B \in m \times \{0,1\}^\kappa$. The receiver receives $A \in m \times \{0,1\}^\kappa$ and $C = \Delta A + B$.

Figure 15: The ideal functionality of VOLE $\mathcal{F}_{vole}$.

We design a multi-sender OPRF protocol in Figure 16. The correctness of this protocol is obvious. If an item x_j is common, we can get $Decode(B_i + \Delta_i A_i' - \Delta_i H_1(y^i_*), y^i_*) = Decode(C_i, y^i_*) = Decode(C_i, x_j)$, where $y^i_* \in Y_i$ equals to x_j. Then the XOR sum of the senders for y^i_* equals the PRF value of the receiver for x_j. More details and security proof for a single OPRF can be found in [26]. In this protocol, the receiver only needs to build the OKVS once when interacting with the n' senders. In the OPRF protocol [26], the OKVS encoding time costs are much larger than VOLE. Therefore, our optimization can save many computation costs compared with simply invoking n' instances of the OPRF protocol, especially when n' is large.

Parameters: The number of senders n'. The statistical and the security parameter are respectively λ and κ. The item's bit length σ. Two random oracles $H_1(\cdot): \{0,1\}^\sigma \mapsto \{0,1\}^\kappa$ and $H_2(\cdot,\cdot): (\{0,1\}^\kappa, \{0,1\}^\sigma) \mapsto \{0,1\}^\ell$. An OKVS scheme $Encode(\cdot), Decode(\cdot,\cdot)$. A VOLE primitive $\mathcal{F}_{vole}$. Two PRFs $Q_\cdot(\cdot), F_\cdot(\cdot): \{0,1\}^\kappa \times \{0,1\}^\sigma \mapsto \{0,1\}^\ell$.
Inputs: The receiver has a set $X = \{x_j\}_{\forall j \in [m_r]}$. Each one of the n' senders has a set $Y_i = \{y^i_j\}_{\forall j \in [m_i]}$ $(i \in [n'])$.
Protocol:

1. **[Initiation]**. The i-th sender samples a random $\omega^s_i \in \{0,1\}^\kappa$ and sends $H_1(\omega^s_i)$ to the receiver.
2. **[OKVS]**. The receiver encodes to get an OKVS table $T = Encode(\{(x_j, H_1(x_j))\}_{\forall j \in [m_r]})$.
3. **[VOLE]**. The i-th sender and the receiver invoke $\mathcal{F}_{vole}$ to respectively get (Δ_i, B_i) and $(A_i, C_i = \Delta_i A_i + B_i)$, where B_i, A_i, C_i are vectors with size m_r, and Δ_i is a constant. The receiver sends $A_i' = A_i + T$ and a random $\omega^r \in \{0,1\}^\kappa$ to the i-th sender.
4. **[PRF valuing]**. For the i-th sender, he/she computes $F_{s^i}(y^i_j) = H_2(Decode(B_i + \Delta_i A_i', y^i_j) - \Delta_i H_1(y^i_j) + \omega^s + \omega^r, y^i_j)$ $(\forall j \in [m_i])$. The receiver computes $Q_k(x_j) = \bigoplus_{i=1}^{n} H_2(Decode(C_i, x_j) + \omega^s + \omega^r), x_j)$ $(\forall j \in [m_r])$, where $k = (s_1, s_2, \cdots, s_{n'})$.

Figure 16: Our multi-sender OPRF protocol.