Privately Compute the Item with Maximal Weight Sum in Set Intersection

Hongyuan Cai^{1,2}[0009-0008-1226-8545]</sup>, Xiaodong Wang^{1,2}[0009-0006-9976-2866]</sup>, Zijie Lu¹[0009-0001-2509-917X]</sup>, and Bei Liang¹[0000-0002-8622-8596]

¹ Beijing Institute of Mathematical Sciences and Applications, Beijing 101408, China lzjluzijie@gmail.com, lbei@bimsa.cn

² Department of Mathematical Sciences, Tsinghua University, Beijing 100084, China {chy22, wangxd22}@mails.tsinghua.edu.cn

Abstract. Private Set Intersection (PSI) is a cryptographic primitive that allows two parties to obtain the intersection of their private input sets while revealing nothing more than the intersection. PSI and its numerous variants, which compute on the intersection of items and their associated weights, have been widely studied. In this paper, we revisit the problem of finding the best item in the intersection according to weight sum introduced by Beauregard et al. (SCN '22), which is a special variant of PSI.

We present two new protocols that achieve the functionality. The first protocol is based on Oblivious Pseudorandom Function (OPRF), additively homomorphic encryption and symmetric-key encryption, while the second one is based on Decisional Diffie-Hellman (DDH) assumption, additively homomorphic encryption and symmetric-key encryption. Both protocols are proven to be secure against semi-honest adversaries. Compared with the original protocol proposed by Beauregard et al. (abbreviated as the FOCI protocol), which requires all weights in the input sets to be polynomial in magnitude, our protocols remove this restriction. We compare the performance of our protocols with the FOCI protocol both theoretically and empirically. We find out that the performance of FOCI protocol is primarily affected by the size of the intersection and the values of elements' weights in intersection when fixing set size, while the

performance of ours is independent of these two factors. In particular, in the LAN setting, when the set sizes are n = 10000, intersection size of $\frac{n}{2}$, the weights of the elements are uniformly distributed as integers from [0, n - 1], our DDH-based protocol has a similar run-time to the FOCI protocol. However, when the weights of the elements belonging to [0, 10n - 1] and [0, 100n - 1], our DDH-based protocol is between a factor 2× and 5× faster than the FOCI protocol.

Keywords: secure multiparty computation \cdot PSI \cdot OPRF.

1 Introduction

Generic secure multiparty computation (MPC) allows multiple parties with private inputs to collaborate with each other to evaluate any function on their inputs, while keeping their inputs unknown to others. Recent studies have advanced secure computation from theoretical foundations to practical applications [3,10,16,22,26,36]. However, general MPC of many functions of interest carries high cost for real-world applications, which prompts the researchers to design practical protocols that are specialized for such functionalities. One of the most widely studied topic is Private Set Intersection (PSI) functionality.

PSI allows two parties, each holding a private set of items, to compute the intersection of sets without revealing any information about non-intersecting items. PSI has many applications such as private contact discovery [8,12], privacypreserving location sharing [25], testing of fully sequenced human genomes [1], and botnet detection [24]. For the past decade, PSI has been extensively studied and has become truly practical with fast implementations [4,7,21,30,31,32,34,35]. However, PSI functionality only involves sets of items and cannot handle the potentially existing weights associated with items.

A notable PSI variant that computes on the intersection of items and its corresponding weights is called Private Intersection-Sum with Cardinality (PI-Sum) [17,19,23]. Specifically, PI-Sum allows one party, called Alice, to input a set of item-weight pairs $\{(x_i, u_i)\}$, while the other party, called Bob, inputs a set of items $\{y_j\}$. The functionality outputs the intersection sum $\sum_{i:x_i=y_j} u_i$ to Alice, which is actually the sum of weights u_i whose corresponding item x_i belongs to the intersection $\{x_i\} \cap \{y_j\}$ of two item sets, and outputs the cardinality $|\{x_i\} \cap \{y_j\}|$ of the intersection to Bob. Therefore, PI-Sum achieves computation on weights of the intersection of two item-sets held by two parties, in the case that only one party has associated weights of items.

Recently, Beauregard et al. [2] have considered an interesting variant of PSI, the functionality of which is called "sampling the best item from the intersection, according to a combined score" (detailed description of ideal functionality is depicted in Fig. 7 in [2]). Let us denote this functionality as $\mathcal{F}_{\mathsf{IMWS}}$ (Item with Maximal Weight Sum). In this new setting, two parties, say Alice and Bob, have a set of item-weight pairs $X = \{(x_i, u_i)\}$ and $Y = \{(y_j, v_j)\}$ as input respectively. For any item $x_i = y_j$ in the intersection, the weight sum of the item is defined as $u_i + v_j$. The functionality of $\mathcal{F}_{\mathsf{IMWS}}$ outputs all the weight sums of the items in the intersection $\{x_i\} \cap \{y_j\}$ to Alice, while Bob receives one item in the intersection with the highest weight sum.

Finding the best item according to a weight sum has been widely applied in many real-world applications. For example, in a meeting scheduling application, two parties Alice and Bob aim to arrange a meeting without revealing their entire schedules to each other. They assign weights to each meeting time (namely each item) based on their preferences, and they want to identify the item in the intersection with the highest weight sum as the final meeting time. In a weighted voting application, Alice and Bob aim to conduct weighted voting on candidates (namely items) based on their preferences and ultimately select the winning candidate who has the votes with the highest weight sum. However, they may not want each other to know the weight of their votes for candidates other than the winner candidate, which can be solved by the functionality mentioned above.

Beauregard et al. [2] have proposed a protocol (abbreviated as FOCI) that achieves the ideal functionality of $\mathcal{F}_{\mathsf{IMWS}}$, based on DDH assumption with a cyclic group $\mathbb{G} = \langle g \rangle$ of order q. However, in the protocol execution, Alice obtains a value of the form D^{w_i} , where w_i is the weight sum of an item in the intersection, and needs to compute a discrete logarithm with base D to retrieve the weight sum w_i . The computation of discrete logarithm imposes a constraint that all the weight sums have to be polynomial in magnitude. Such a limitation might hinder the usability of the protocol in practice, especially when the weights are of substantial magnitude, such as investment amounts or prices. For instance, consider the application scenario where two parties, Alice and Bob, are each evaluating a set of projects they are willing to invest in, with each project having an associated investment amount (weight). They want to identify the project with the highest combined investment between them, without revealing individual investment amounts. In this case, the weight sums (investment amounts) may be very large, making discrete logarithm computations inefficient and impractical. Moreover, as the implementation result of Beauregard et al. [2]'s protocol is missing, we do not know how efficient their protocol is and if it is applicable in practice. These naturally motivate us to explore protocols that are based on more reasonable and standard assumptions, and also do not require the constraint on input weights, so that the protocols can be fit for more application scenarios. It is also valuable to implement and evaluate the efficiency of the protocols via their communication and computation costs, which in turn illustrate that they are truly practical.

1.1 Our Contributions

Our contributions are twofold. First, we revisit the problem of finding the best item from the intersection according to a weight sum introduced by Beauregard et al. [2], which is called "compute the item with maximal weight sum in set intersection" (referred to the description of functionality $\mathcal{F}_{\mathsf{IMWS}}$ in Fig. 2) in this paper. We present two novel protocols that achieve the ideal functionality. The first protocol is based on OPRF, additively homomorphic encryption (AHE) and symmetric-key encryption, while the second one is based on Decisional Diffie-Hellman (DDH) assumption, additively homomorphic encryption and symmetric-key encryption. Both protocols are secure against semi-honest adversaries. Compared with the prior protocol by [2] which has the constraint that all the weights in the input sets have to be polynomial in magnitude, both of our protocols remove such restriction on weights.

Our second contribution is the implementation and evaluation of protocols. We compare our protocols with the FOCI protocol both theoretically and empirically. As Beauregard et al. [2] did not provide performance estimation for their FOCI protocol, we implement it in C++ using the same language and library, and run on the same hardware as ours. We find out that the performance of FOCI protocol is primarily affected by the size of the intersection and the values of elements' weights in intersection when fixing set size n, while the performance of our protocols is basically independent of these two factors. In

particular, in the WAN setting with 1000 Mbps network bandwidth and the set sizes are n = 10000, our best protocol requires an execution time of 10.08 seconds and involves 6.28 MiB of communication. In a LAN setting, when the set sizes are n = 10000, intersection size of $\frac{n}{2}$, the weights of the elements are uniformly distributed as integers from [0, n - 1], our DDH-based protocol has a similar run-time to FOCI protocol. However, when the weights of the elements belonging to [0, 10n - 1], our DDH-based protocol is around a factor 2 faster than FOCI protocol, and when the weights belonging to [0, 100n - 1], our DDH-based protocol.

1.2 Technical Overview

It is noted that the FOCI protocol proposed by [2] achieves the functionality $\mathcal{F}_{\mathsf{IMWS}}$ under the DDH assumption, where Alice has to compute a discrete logarithm to obtain the weight sums. This imposes a constraint that all weight sums must be of polynomial magnitude. To eliminate this restriction on weight values, instead of encoding the associated weights in the exponents under the DH paradigm, we adopt the AHE technique to encrypt the weight values directly.

The high-level idea of our OPRF-based protocol is as follows. Given two input sets of n item-weight pairs, $X = \{(x_i, u_i)\}_{i \in [n]}$ for Alice and $Y = \{(y_j, v_j)\}_{j \in [n]}$ for Bob, the two parties execute a multi-point OPRF protocol. Bob, as the sender, obtains a PRF key k, while Alice, as the receiver, inputs all her items x_i and obtains the corresponding OPRF values $F_k(x_i)$. Then Alice, who holds the public-secret key pair (pk, sk) of additively homomorphic encryption scheme, encrypts both the OPRF values and their associated weights to obtain ciphertexts { $ct_i = (AEnc(pk, F_k(x_i)), AEnc(pk, u_i))$ }, which are sent to Bob in order. On receiving the ciphertexts, Bob with public key pk homomorphically adds random values r_i, s_i to each ciphertext of item and weight respectively, and sends the ciphertexts $\{ct'_i = (\mathsf{AEnc}(pk, F_k(x_i) + r_i), \mathsf{AEnc}(pk, u_i + s_i))\}$ back to Alice in a shuffled order π randomly chosen by Bob. Alice can decrypt the ciphertexts to obtain the set of OPRF value and weight pairs with random masks $U = \{(F_k(x_i) + r_i, u_i + s_i)\},$ and it is clear that for each pair in set U, Alice is incapable of relating it with its original item-weight pair, since Bob has done the shuffle.

Next, for each input item y_j held by Bob, Bob sends symmetric-key encryption ciphertexts $\overrightarrow{a_j} = \{\operatorname{Enc}(F_k(y_j) + r_i, v_j - s_i)\}_{i \in [n]} = \{a_{j,i}\}_{i \in [n]}$ to Alice, which are the encryptions of the corresponding weight v_j (with random masks) under the keys that are OPRF values of item y_j with all possible masks. It can be observed that if y_j is an item in the intersection, then there must exist some $(K, S) \in U$ and $a_{j,i} \in \overrightarrow{a_j}$ such that $\operatorname{Dec}(K, a_{j,i}) \neq \bot$, and the weight sum of y_j is exactly $w_j = \operatorname{Dec}(K, a_{j,i}) + S$, since the random masks of the weights are canceled out. Otherwise, the decryption will fail with overwhelming probability for any $(K, S) \in U$ and $a_{j,i} \in \overrightarrow{a_j}$, and the weight sum of y_j is defined as $w_j = \bot$ since y_j does not belong to the intersection. Therefore, Alice is able to obtain the weight sums $(w_1, ..., w_n)$ and sends the index $j^* = \operatorname{argmax}_j w_j$ to Bob, who finally outputs the item y_{j^*} with the maximal weight sum.

One subtle issue is that the communication overhead for Bob is quadratic in n, and the computation overhead for Alice is cubic in n. This is because, for each of Bob's input items y_j , the vector $\overrightarrow{a_j}$ that Bob sends to Alice consists of n ciphertexts. To decrypt $\overrightarrow{a_j}$, Alice must iterate through n possible keys in set U. The reason why $\overrightarrow{a_j}$ must contain n ciphertexts is that Bob needs to account for all possible random mask pairs to cover every possible intersection between his items and Alice's items, including the case where $y_i = x_i$.

Therefore, to reduce the overhead, we leverage the Cuckoo hashing technique with ℓ hash functions $h_1, ..., h_\ell$ on Alice's input items, such that each bin contains at most one item. Then Alice performs single-point OPRF with Bob for each bin, where Bob as sender obtains the PRF key k_i for *i*-th bin, while Alice as receiver inputs the item x in *i*-th bin and learns the OPRF value $F_{k_i}(x)$. The key point is that each random mask pair (r_i, s_i) is now associated with bin's index *i*, which means that (r_i, s_i) is the mask of $(F_{k_i}(x), u)$, where x is the item in *i*-th bin with weight u. For each of Bob's items y_j , all of its possible mask pairs are indexed by $\{h_1(y_j), ..., h_\ell(y_j)\} = I_j$. Therefore, $\vec{a_j} = \{\text{Enc}(F_{k_i}(y_j) + r_i, v_j - s_i)\}_{i \in I_j} =$ $\{a_{j,i}\}_{i \in I_j}$ only needs to hold ℓ ciphertexts rather than n. (We omit the stash of Cuckoo hashing here for simple description, which will be discussed in detail in Sec. 3.2.)

Considering that the homomorphic operations of the underlying AHE protocols are expensive, we propose the second protocol (i.e., our DDH-based protocol in Sec. 3.3) to further reduce the overall cost. In contrast to our OPRF-based protocol mentioned earlier, which utilizes AHE on items, the main idea of our DDH-based protocol is to apply DH-based OPRF on items instead, while the operations on weights remain the same. Specifically, both protocols apply AHE on the associated weights.

1.3 Related Work

In this section, we discuss some works about computing on weights associated with items in the intersection.

In [2], Beauregard et al. firstly develop protocols for privately finding one common item (FOCI) from the intersection of two sets, which can be applied to meeting scheduling. Their protocols differ in how that item is chosen — e.g., uniformly at random from the intersection; the "best" item in the intersection according to one party's ranking; or the "best" item in the intersection according to the sum of both parties' weights. They construct the corresponding protocols and prove the security in the semi-honest model. Our focus is on the last scenario, which is also the most general case among them. In this scenario, their protocol needs to compute the discrete logarithm, so it only supports weights of polynomial size. Additionally, they did not implement their protocols, so it is difficult to judge the performance of their protocols.

In [19], Ion et al. discuss several Private Intersection-Sum (PI-Sum) with Cardinality protocols, which can be applied to compute advertising conversion. In PI-Sum, one party has a set of pairs of an identifier and an value $\{(x_i, u_i)\}$, while the other party has an identifier set y_i . The goal is to compute the sum

of values within the intersection of the two identifier sets (i.e. $\sum_{i:x_i=y_j} u_i$), together with the cardinality of the intersection (i.e. $|\{x_i\} \cap \{y_j\}|$). They construct protocols that rely on a Diffie-Hellman style double masking, Random Oblivious Transfer and encrypted Bloom filters, respectively. Then they implemented the protocols and conducted detailed comparisons. In the case where both input sizes are 10,000, their best protocol requires an execution time of 7.47 seconds and involves 0.81 MB of communication.

In [9], Chida et al. study a functionality called inner product private join and compute (inner product PJC), which can be applied to compute weighted advertising conversion measurement. In inner product PJC, both parties have a set of pairs of an identifier and an value (i.e. $\{(x_i, u_i)\}$ and $\{(y_i, v_i)\}$). The goal is to compute the inner product of values within the intersection of the two identifier sets (i.e. $\sum_{(i,j):x_i=y_j} u_i v_j$), along with the cardinality of the intersection (i.e. $|\{x_i\} \cap \{y_j\}|$). They construct a communication-efficient 4-round inner product PJC protocol and implement it. In the case where both input sizes are 2^{16} , the protocol requires an execution time of 25.92 seconds and involves 17.2 MB of communication.

Another approach for computing functions on the weights in the intersection is by using circuit PSI [33], which is a technique to use general two-party computation (garbled circuits [37] or the GMW protocol [18]) to compute any functions on the intersection and the associated weights. But this approach significantly increases the communication cost or the round complexity.

2 Preliminaries

In this section, we introduce the notation and primitives which will be used in our protocol including oblivious PRF, the definition of symmetric-key encryption and semi-honest security. The definitions of additively homomorphic encryption, DDH assumption, and Cuckoo hash are provided in App. A.

2.1 Notation

Throughout the paper we use the following notation: We use κ and λ to denote the computational and statistical security parameters, respectively. We use [n] to denote the set $\{1, 2, \ldots, n\}$. For some set S, the notation $s \stackrel{\$}{\leftarrow} S$ means that s is assigned a uniformly random element from S. By negl (κ) we denote a negligible function, i.e., a function f such that $f(\kappa) < 1/p(\kappa)$ holds for any polynomial $p(\cdot)$ and sufficiently large κ .

2.2 Security Model

We consider the semi-honest security model in this paper, which guarantees that a party can never learn any information about the other party's input other than its own output as long as it follows the protocol. **Definition 1.** Let Π be a two-party protocol computing $f = (f_1, f_2)$ and $\mathsf{View}_i^{\Pi}(x, y)$ be the view of P_i (the entire distribution that P_i can see), $\mathsf{Out}^{\Pi}(x, y) = (\mathsf{Out}_1^{\Pi}(x, y),$ $\mathsf{Out}_2^{\Pi}(x, y))$ be the output of the protocol where x and y are inputs of P_1 and P_2 , respectively. We say Π has semi-honest security if there exist PPT simulators S_1, S_2 , and the following holds for all inputs x, y:

$$\left(\mathsf{View}_{1}^{\Pi}(x,y), \mathsf{Out}^{\Pi}(x,y) \right) \stackrel{c}{\approx} \left(\mathsf{S}_{1}\left(1^{\lambda}, x, f_{1}(x,y) \right), f(x,y) \right)$$
$$\left(\mathsf{View}_{2}^{\Pi}(x,y), \mathsf{Out}^{\Pi}(x,y) \right) \stackrel{c}{\approx} \left(\mathsf{S}_{2}\left(1^{\lambda}, y, f_{2}(x,y) \right), f(x,y) \right)$$

2.3 Oblivious Pseudorandom Function (OPRF)

An oblivious pseudorandom function (OPRF) [15] is a protocol in which a sender learns (or chooses) a random PRF seed k while the receiver learns $f_k(x)$, the result of the PRF on a single input x chosen by the receiver. In this protocol, sender learns nothing about the receiver's input x. Furthermore, the evaluation of the PRF f_k on all other inputs remains pseudorandom in the view of receiver. The ideal functionality \mathcal{F}_{OPRF} is defined in Fig. 1.

Inputs: sender inputs nothing; receiver holds an evaluation point x. **Outputs:** sender outputs PRF key k; receiver outputs $f_k(x)$.

Fig. 1: Ideal functionality for OPRF \mathcal{F}_{OPRF} .

2.4 Symmetric-Key Encryption

We need a one-time, symmetric-key encryption scheme in which decryption fails when an incorrect (independently random) key is used. Let \mathcal{K} be the set of keys and let \mathcal{M} be the set of plaintexts. Specifically, we require the following properties:

- Correctness: For all $k \in \mathcal{K}$ and $m \in \mathcal{M}$, it holds that

 $\Pr\left[\mathsf{Dec}(k,\mathsf{Enc}(k,m))=m\right]=1.$

- One-time security: For all $m_0, m_1 \in \mathcal{M}, k_0, k_1 \stackrel{\$}{\leftarrow} \mathcal{K}$ and probabilistic polynomial-time algorithms \mathcal{A} , there is a negligible function negl such that

 $|\Pr[\mathcal{A}(\mathsf{Enc}(k_0, m_0)) = 1] - \Pr[\mathcal{A}(\mathsf{Enc}(k_1, m_1)) = 1]| \le \operatorname{neg}(\kappa).$

- Robust decryption: For all $m \in \mathcal{M}$, and $k, k' \stackrel{\$}{\leftarrow} \mathcal{K}$, it holds that

 $\Pr\left[\mathsf{Dec}(k',\mathsf{Enc}(k,m))\neq\perp\right] \le \operatorname{neg}(\kappa).$

3 Privately Compute the Item with Maximal Weight Sum in Set Intersection: Functionality and Constructions

In this section, we give two new protocols based on new approaches. The functionality that the protocols have realized is defined in Sec. 3.1. The protocol based on OPRF and its security proof are shown in Sec. 3.2. The protocol based on DDH assumption and its security proof are presented in Sec. 3.3.

3.1 The Functionality of Privately Compute the Item with Maximal Weight Sum in Set Intersection

Here we provide the functionality of privately compute the item with maximal weight sum in set intersection, which is to identify the common item with highest weight sum. Here the common item's weight sum refers to the sum of its weights associated with the common item from both parties.

 $\mathcal{F}_{\text{IMWS}}$ 1. receive input $X = \{(x_1, u_1), ..., (x_n, u_n)\}$ from Alice and $Y = \{(y_1, v_1), ..., (y_n, v_n)\}$ from Bob. $(x_i \neq x_j \text{ and } y_i \neq y_j \text{ for } i, j \in [n])$ 2. for $j \in [n]$, if $\exists (x, u) \in X$ with $x = y_j$, then let $w_j = u + v_j$; else let $w_j = \bot$.
3. give $w_1, ..., w_n$ to Alice.
4. set $j^* := \arg \max_j w_j$ and give y_{j^*} to Bob.

Fig. 2: Ideal functionality for privately compute the item with maximal weight sum in set intersection.

The ideal functionality for this variant of PSI is defined in Fig. 2. Specifically, Alice as sender inputs a set X of item-weight pairs (x_i, u_i) in order, while Bob as receiver inputs a set Y of item-weight pairs (y_i, v_i) in order. For each y_j $(j \in [n])$, if y_j is in the intersection of item sets, i.e., there exists some $(x, u) \in X$ such that $x = y_j$, then denote the weight sum of y_j by $w_j = u + v_j$; otherwise, for y_j not in the intersection of item sets, let the corresponding weight sum be $w_j = \bot$. Alice receives weight sums in the ordering of items in Y as $(w_1, ..., w_n)$, and Bob receives the item y_{j^*} with the maximal weight sum, where $j^* := \arg \max_j w_j$. It is important to note that since the order of $(w_1, ..., w_n)$ aligns with that of items in Y, Alice will know which index the item with highest weight sum in Bob's set is located at. However, Alice cannot predict the correspondences between $(w_1, ..., w_n)$ and $(x_1, ..., x_n)$.

In the special case where there are no common items, i.e., $w_j = \bot$ for all $j \in [n]$, we adopt the convention that the index of the maximal weight sum is $\operatorname{argmax}_{i} w_{j} = \bot$. Consequently, if $j^* = \bot$, the corresponding item y_{j^*} is also \bot .

It can be observed that our ideal functionality shown in Fig. 2 is slightly different from that of "finding the best item according to a combined score" (Fig. 7 in [2]). We revise the definition as we find out that their functionality does not align with the FOCI protocol they proposed. Specifically, the functionality in [2]

(step 2 in Fig. 7) indicates that the random ordering of set Y is not revealed to Bob. However, in the FOCI protocol, Bob knows the order of Y as he does the random permutation himself, and the order is crucial for him to finally obtain the output y_{j^*} from the index j^* . To address the issue, we modify the functionality such that the input sets of both Alice and Bob are denoted in order, which means that Alice and Bob know the ordering of their input sets X and Y respectively. In the FOCI protocol, Alice and Bob locally shuffle their input sets at the very beginning, which results in sorted sets. These sorted sets are then used as inputs for the subsequent steps of the protocol, which achieve our functionality $\mathcal{F}_{\text{IMWS}}$.

3.2 **OPRF-based Protocol**

In this section, we present a protocol based on OPRF that achieves the ideal functionality of $\mathcal{F}_{\mathsf{IMWS}}$ described in Fig. 2.

An overview of protocol. Recall that Beauregard et al. [2] have proposed a protocol with the same functionality, the key building block of which is called "2-Blind-Exp subprotocol" (see in Fig. 8 of [2]). In fact, this subprotocol can be abstracted as a shuffled OPRF based on the DDH assumption. The shuffling step is crucial to ensure that the intersection remains hidden from Alice. Alice is only allowed to learn the weight sum $w_j = u_i + v_j$ for common item $x_i = y_j$ without revealing the individual u_i and v_j , which can be achieved by blinding the weights with masks associated to the corresponding items that can be canceled out when the items are the same.

The main drawback of Beauregard et al.'s protocol [2] is that due to the DH paradigm, the weight sums appears on the exponents of group elements which Alice obtains by symmetric-key decryption, therefore, Alice needs to further perform a discrete logarithm process to compute the weight sum.

To avoid the use of discrete logarithm that may cause restrictions on weights or security problems, we replace the DH paradigm applying on both items and weights with additively homomorphic encryption technique, which is inspired by Ion et al. [19] to construct a variant of PSI protocol, i.e., private intersection-sum with cardinality.

Our construction. Our protocol is built upon the following preliminaries: an additively homomorphic encryption scheme (AGen, AEnc, ADec, ASum, ARefresh), a symmetric-key encryption scheme (Enc, Dec), a single-point OPRF protocol and Cuckoo hashing scheme. The detailed protocol is presented in Fig. 3.

Correctness of the protocol follows immediately from inspection, assuming Cuckoo hashing fails with negligible probability and that the OPRF outputs collide with negligible probability.

The security of our protocol follows from the security of the additively homomorphic encryption scheme, the security properties of the single-point OPRF protocol and the security properties of symmetric-key encryption.

We note that the single point OPRF as the main building block of the above protocol can be substitute with a multi-point OPRF, which means after the execution of a multi-point OPRF between Alice and Bob, Alice as a receiver learns N + s PRF values $F_k(x_i)$. The whole protocol is the same except that all the PRF keys for different bins will be the same.

PROTOCOL:

Inputs: Alice has input set $X = \{(x_i, u_i)\}_{i \in [n]}$ and Bob has input set $Y = \{(y_j, v_j)\}_{j \in [n]}$.

Setup: Alice generates an additively homomorphic encryption key-pair (pk, sk) and sends public key pk to Bob. Alice and Bob choose Cuckoo hash table size N, stash size s, and ℓ hash functions $h_1, ..., h_\ell : \{0, 1\}^* \to [N]$.

Protocol Steps:

- 1. Alice hashes her items $\{x_i\}_{i\in[n]}$ using Cuckoo hashing. Alice fills in all empty bins and the empty positions in the stash with a dummy item. Denote the items in the Cuckoo hash table and the stash (including both dummy and real values) by $x_1, ..., x_{N+s}$, where $x_{N+1}, ..., x_{N+s}$ are items in the stash. Denote the corresponding value of x_i by u_i for $1 \le i \le N + s$, where the corresponding value of a dummy item is taken randomly from the field. Alice aborts if the hashing fails.
- 2. For $1 \leq i \leq N + s$, Alice and Bob execute a single-point OPRF, where Bob as a sender inputs nothing and learns an PRF key k_i , while Alice as a receiver inputs item x_i and learns the PRF value $F_{k_i}(x_i)$.
- 3. For $1 \leq i \leq N + s$, Alice does encryption as

$$ct_i = (ct_i^1, ct_i^2)$$

= (AEnc (pk, F_{k_i}(x_i)), AEnc(pk, u_i))

and sends $\{ct_i\}_{i \in [N+s]}$ to Bob.

4. For each $1 \le i \le N + s$, Bob chooses random values r_i and s_i from the input domain of additively homomorphic encryption, and computes

$$\begin{split} ct'_{i} &= \mathsf{ARefresh}\left(ct_{i} + \left(\mathsf{AEnc}(pk, r_{i}), \mathsf{AEnc}(pk, s_{i})\right)\right) \\ &= \left(\mathsf{ARefresh}\left(ct_{i}^{1} + \mathsf{AEnc}(pk, r_{i})\right), \mathsf{ARefresh}\left(ct_{i}^{2} + \mathsf{AEnc}(pk, s_{i})\right)\right). \end{split}$$

Bob sends to Alice the set of ct_i in shuffled order π .

5. Alice receives the set of ct'_i in an unknown order, and uses its private key sk to decrypt each component in all ct'_i . Denote the set of decryption results of all ct'_i by

$$U = \left\{ \left(F_{k_i}(x_i) + r_i, u_i + s_i \right) \right\}_{i \in [N+s]}.$$

Denote the first and second components by

$$Z = \{F_{k_i}(x_i) + r_i\}_{i \in [N+s]}, \qquad W = \{u_i + s_i\}_{i \in [N+s]}.$$

6. For $1 \le j \le n$, Bob computes $I_j = \{h_i(y_j)\}_{i \in [\ell]} \cup \{N+1, ..., N+s\}$ and does the symmetric-key encryption as

$$\vec{a_j} = \{ \mathsf{Enc} \left(F_{k_i}(y_j) + r_i, v_j - s_i \right) \}_{i \in I_j} = \{ a_{j,i} \}_{i \in I_j}.$$

Then Bob sends $\{\overrightarrow{a_j}\}_{j \in [n]}$ with every $\ell + s$ items in $\overrightarrow{a_j}$ shuffled in order π_j for $1 \leq j \leq n$ to Alice.

7. For $1 \leq j \leq n$, Alice tries all items in set Z to decrypt the ciphertexts in $\overrightarrow{a_j}$. That is, if there exists an $a_{j,i} \in \overrightarrow{a_j}$ and an $(K, S) \in U$ such that $\mathsf{Dec}(K, a_{j,i}) \neq \bot$, then let $w_j = \mathsf{Dec}(K, a_{j,i}) + S$, else let $w_j = \bot$. Then Alice computes $j^* = \operatorname{argmax}_j w_j$ and sends it to Bob.

Outputs: Alice outputs $(w_1, ..., w_n)$ and Bob outputs y_{j^*} .

Fig. 3: Π_{OPRF} : the protocol based on single-point OPRF.

Privately Compute the Item with Maximal Weight Sum in Set Intersection

Security Analysis. The protocol Π_{OPRF} in Fig. 3 securely realizes \mathcal{F}_{IMWS} in the semi-honest model. The correctness is detailed beforehand. We then prove the privacy by the following theorem.

Theorem 1. Assume that the additively homomorphic encryption scheme is IND-CPA secure, the underlying OPRF protocol is secure in semi-honest model, and the one-time, symmetric-key encryption scheme has robust decryption property. Then there exist simulators SIM_A , SIM_B such that for security parameter λ and inputs $X = \{(x_i, u_i)\}_{i \in [n]}, Y = \{(y_i, v_i)\}_{i \in [n]},$

 $\textit{View}_{A}^{\Pi_{\textit{OPRF}}}(\{(x_{i}, u_{i})\})_{i \in [n]}, \{(y_{i}, v_{i})\})_{i \in [n]}) \overset{c}{\approx} \textit{SIM}_{A}(1^{\lambda}, \{(x_{i}, u_{i})\})_{i \in [n]}, (w_{1}, ..., w_{n}))$

$$View_{B}^{\Pi_{OPRF}}(\{(x_{i}, u_{i})\})_{i \in [n]}, \{(y_{i}, v_{i})\})_{i \in [n]}) \stackrel{\sim}{\approx} SIM_{B}(1^{\lambda}, \{(y_{i}, v_{i})\})_{i \in [n]}, y_{j^{*}})$$

where $(w_1, ..., w_n)$ is the weight sums for the intersection and y_{j^*} is the item in the intersection which has the maximum weight sum.

Proof. Security against corrupt Alice. We describe the simulator algorithm SIM_A in Algorithm 1.

Algorithm 1 The simulator for Alice in the OPRF-based Protocol

Input: $(1^{\lambda}, \{(x_i, u_i)\})_{i \in [n]}, (w_1, ..., w_n))$ Output: SimView(Alice)SIM_A $(1^{\lambda}, \{(x_i, u_i)\})_{i \in [n]}, (w_1, ..., w_n))$:

- 1. Honestly simulate the Setup phase between Alice and Bob.
- 2. Honestly simulate steps 1-3 between Alice and Bob, receiving the set $\{ct_i = (AEnc(pk, F_k(x_i)), AEnc(pk, u_i))\}_{i \in [N+s]}$ at the end of step 3.
- 3. In step 4, choose a set C^* consisting of N + s random elements c_i from the input domain of additively homomorphic encryption, a set D^* consisting of N + s random elements d_i from the input domain of additively homomorphic encryption. Compute the fresh encryptions to each c_i and d_i under Alice's public key pk, and send $ct'_i = (\mathsf{AEnc}(pk, c_i), \mathsf{AEnc}(pk, d_i))$ to Alice.
- 4. Simulate step 5 for Alice honestly.
- 5. In step 6, choose the set $\{\overrightarrow{a_j}\}_{j\in[n]} = \{a_{j,i}\}_{j\in[n],i\in[\ell+s]}$ such that each $\overrightarrow{a_j}$ contains exactly $\ell + s$ ciphertexts which are the encryptions of 0 under randomly chosen keys $K_{j,i}$, that is $a_{j,i} = \operatorname{Enc}(K_{j,i}, 0)$. For each $1 \leq j \leq n$, if $w_j \neq \bot$, then for a randomly chosen index $t_j \in [\ell + s]$, chose some c_{l_j} randomly from the set C^* (each element in C^* is used at most once) with the corresponding indexed element d_{l_j} in set D^* , and replace $a_{j,t_j} = \operatorname{Enc}(K_{j,t_j}, 0)$ with $\operatorname{Enc}(c_{l_j}, w_j d_{l_j})$. Send the set $\{\overrightarrow{a_j}\}_{j\in[n]} = \{a_{j,i}\}_{j\in[n],i\in[\ell+s]}$ to Alice.
- 6. Simulate step 7 for Alice honestly.
- 7. Output the view of Alice in this interaction.

We argue that

 $\mathsf{View}_{A}^{\Pi_{\mathsf{OPRF}}}(\{(x_{i}, u_{i})\})_{i \in [n]}, \{(y_{i}, v_{i})\})_{i \in [n]}) \stackrel{\mathrm{c}}{\approx} \mathsf{SIM}_{A}(1^{\lambda}, \{(x_{i}, u_{i})\})_{i \in [n]}, (w_{1}, ..., w_{n}))$

using a multi-step hybrid argument, where each neighboring pair of hybrid distributions is computationally indistinguishable.

- 12 H. Cai et al.
- Hyb_0 : The transcript corresponding to the view of Alice in a real execution of the protocol.
- **Hyb**₁: The same as Hyb₀, except in step 6, for all $w_j = \bot$, Bob replaces the ciphertexts $\{a_{j,i}\}_{i \in [\ell+s]}$ with fresh encryptions of 0 under the key $\{F_{k_i}(y_j) + r_i | i \in I_j\}$, that is $\{a_{j,i}\}_i = \{\text{Enc}(F_{k_i}(y_j) + r_i, 0) | i \in I_j\}$.

 Hyb_1 and Hyb_0 are indistinguishable by the one time security of symmetrickey encryption and the pseudorandomness of non-retrieved items in OPRF.

Hyb₂: The same as Hyb₁, except in step 6, for all $w_j = \bot$, Bob replaces the ciphertexts $\{a_{j,i}\}_{i \in [\ell+s]}$ with fresh encryptions of 0 under randomly chosen key $K_{j,i}$, that is $a_{j,i} = \text{Enc}(K_{j,i}, 0)$. Hyb₂ and Hyb₁ are indistinguishable by the pseudorandomness of PRF val-

 Hyb_2 and Hyb_1 are indistinguishable by the pseudorandomness of PRF values, and the one-time-pad property of randomness r_i .

Hyb₃ : The same as Hyb₂, except in step 4, the first component of each ct'_i is replaced with an encryption of a uniformly random value c_i , that is $ct'_i = (\mathsf{AEnc}(pk, c_i), \mathsf{AEnc}(pk, u_{\pi^{-1}(i)} + s_{\pi^{-1}(i)}))$, and in step 6, for each $w_j \neq \bot$, for the corresponding element index t_j , Bob replace the ciphertexts $\{a_{j,i}\}_{i\in[\ell+s]\setminus\{t_j\}}$ with fresh encryptions of $v_j - s^{(i)}$ under randomly chosen key $K_{j,i}$, where $s^{(i)} = s_{h_{\pi_i^{-1}(i)}}(y_j)$ if $i \in [\ell], s^{(i)} = s_{N-\ell+\pi_j^{-1}(i)}$ if

 $i \in [\ell + 1, \ell + s]$, that is $a_{j,i} = \mathsf{Enc}(K_{j,i}, v_j - s^{(i)})$, and replace the ciphertext a_{j,t_j} with fresh encryptions under the corresponding key c_{l_j} , that is $a_{j,t_j} = \mathsf{Enc}(c_{l_j}, v_j - s^{(t_j)})$. (The correspondence of t_j and c_{l_j} means that c_{l_j} can successfully decrypts a_{j,t_j} , which holds that $\pi^{-1}(l_j) = h_{\pi_j^{-1}(t_j)}(y_j)$ if

 $t_j \in [\ell], \ \pi^{-1}(l_j) = N - \ell + \pi_j^{-1}(t_j) \text{ if } t_j \in [\ell + 1, \ell + s].)$

Hyb₃ and Hyb₂ are indistinguishable by the one-time-pad property of adding the random value r_i , the hiding property of additively homomorphic encryption scheme, and the pseudorandomness of PRF values.

 $\begin{aligned} \mathbf{Hyb}_{4} &: \text{The same as Hyb}_{3}, \text{ except in step 4, the second component of each} \\ ct_{i}' \text{ is replaced with an encryption of a uniformly random value } d_{i}, \text{ that} \\ \text{ is } ct_{i}' &= (\mathsf{AEnc}(pk,c_{i}),\mathsf{AEnc}(pk,d_{i})), \text{ and in step 6, for each } w_{j} \neq \perp, \text{ for the} \\ \text{ corresponding element index } t_{j}, \text{ Bob replace the ciphertexts } \{a_{j,i}\}_{i\in[\ell+s]\setminus\{t_{j}\}} \\ \text{ with fresh encryptions of 0 under key } K_{j,i}, \text{ that is } a_{j,i} &= \mathsf{Enc}(K_{j,i},0), \text{ and} \\ \text{ replace the ciphertext } a_{j,t_{j}} \text{ with fresh encryptions of } w_{j} - d_{l_{j}} \text{ under the} \\ \text{ corresponding key } c_{l_{j}}, \text{ that is } a_{j,t_{j}} &= \mathsf{Enc}(c_{l_{j}},w_{j}-d_{l_{j}}). \end{aligned}$

 Hyb_4 and Hyb_3 are indistinguishable by the one-time-pad property of adding the random value s_i , the hiding property of additively homomorphic encryption scheme, and the one time security of symmetric key encryption.

 \mathbf{Hyb}_5 : The view of Alice output by SIM_A .

 Hyb_5 and Hyb_4 are identically distributed.

Security against corrupt Bob. We observe that Bob's view in the protocol consists of the following:

- 1) A public key pk of additively homomorphic encryption (Setup).
- 2) Whether Alice aborts due to Cuckoo hashing failure (step 1).
- 3) The sender's view in single-point OPRF executions (step 2).

Privately Compute the Item with Maximal Weight Sum in Set Intersection

4) The sets $\{ct_i\}_{i \in [N+s]}$ of ciphertexts encrypted under Alice's key pk (step 3).

13

5) An index j^* corresponding to Bob's output (step 7).

We describe the simulator algorithm SIM_B in Algorithm 2.

Algorithm 2 The simulator for Bob in the OPRF-based Protocol

Input: $(1^{\lambda}, \{(y_i, v_i)\})_{i \in [n]}, y_{j^*})$ **Output:** SimView(Bob)SIM_B $(1^{\lambda}, \{(y_i, v_i)\})_{i \in [n]}, y_{j^*})$:

- 1. Randomly choose a pair of homomorphic encryption keys (pk, sk) and sends pk to Bob in setup phase.
- 2. Never abort in step 1.
- 3. In step 2, choose N + s random elements t_i from the domain of input items, and request t_i to be receiver's inputs in the single-point OPRF protocols, with Bob plays the role of sender.
- 4. In step 3, send Bob $\{ct_i\}_{i \in [N+s]}$ to be fresh encryptions of 0, that is $ct_i = (AEnc(pk, 0), AEnc(pk, 0)).$
- 5. Simulate step 4 and step 6 for Bob honestly.
- 6. In step 7, send j^* corresponding to Bob's output y_{j^*} to Bob.
- 7. Output the view of Bob in this interaction.

We argue that

 $\mathsf{View}_{B}^{\Pi_{\mathsf{OPRF}}}(\{(x_{i}, u_{i})\})_{i \in [n]}, \{(y_{i}, v_{i})\})_{i \in [n]}) \stackrel{\mathrm{c}}{\approx} \mathsf{SIM}_{B}(1^{\lambda}, \{(y_{i}, v_{i})\})_{i \in [n]}, y_{j^{*}})$

using a multi-step hybrid argument, where each neighboring pair of hybrid distributions is computationally indistinguishable.

- \mathbf{Hyb}_{0} : The transcript corresponding to the view of Bob in a real execution of the protocol.
- Hyb_1 : The same as Hyb_0 , except in step 1, Alice never abort when executing Cuckoo hashing.

 Hyb_1 and Hyb_0 are indistinguishable by the property that Cuckoo hashing fails with negligible probability.

Hyb₂: The same as Hyb₁, except in step 2, Alice replaces the inputs $\{x_i\}_{i \in [N+s]}$ for single-point OPRF protocols with N + s random elements $\{t_i\}_{i \in [N+s]}$ from the domain of input items.

 Hyb_2 and Hyb_1 are indistinguishable by the security of single-point OPRF protocol.

Hyb₃: The same as Hyb₂, except in step 3, Alice replaces the ciphertexts $\{ct_i\}_{i\in[N+s]}$ with fresh encryptions of 0 under public key pk, that is $ct_i = (\mathsf{AEnc}(pk, 0), \mathsf{AEnc}(pk, 0))$.

 Hyb_3 and Hyb_2 are indistinguishable by the hiding property of additively homomorphic encryption scheme.

 Hyb_4 : The view of Bob output by SIM_B . Hyb₄ and Hyb₃ are identically distributed.

PROTOCOL:

Inputs: Alice has input set $X = \{(x_i, u_i)\}_{i \in [n]}$ and Bob has input set $Y = \{(x_i, u_i)\}_{i \in [n]}$ $\{(y_j, v_j)\}_{j\in[n]}.$

Setup: $\mathbb{G} = \langle g \rangle$ is a cyclic group with prime order q and $H : \{0,1\}^* \to \mathbb{G}$ is a hash function. Alice generates an additively homomorphic encryption key-pair (pk, sk)and sends the public key pk to Bob. Alice and Bob choose Cuckoo hash table size N, stash size s, and ℓ hash functions $h_1, \ldots, h_\ell : \{0, 1\}^* \to [N]$.

Protocol Steps:

- 1. Alice hashes her items $\{x_i\}_{i \in [n]}$ using Cuckoo hashing. Alice fills in all empty bins and the empty positions in the stash with a dummy item. Denote the items in the Cuckoo hash table and the stash (including both dummy and real values) by x_1, \ldots, x_{N+s} , where x_{N+1}, \ldots, x_{N+s} are items in the stash. Denote the corresponding value of x_i by u_i for $1 \le i \le N + s$, where the corresponding value of a dummy item is taken randomly from the field. Alice aborts if the hashing fails.
- 2. Alice randomly chooses $a \in \mathbb{Z}_q$ and performs the encryption as

$$ct_i = (H(x_i)^a, \mathsf{AEnc}(pk, u_i))$$

then sends $\{ct_i\}_{i \in [N+s]}$ to Bob.

3. For each $1 \leq i \leq N + s$, Bob chooses random values r_i from the input domain of the additively homomorphic encryption and b_i from \mathbb{Z}_q , and computes

$$ct'_{i} = \left(H(x_{i})^{a \cdot b_{i}}, \mathsf{ARefresh}\left(\mathsf{AEnc}(pk, u_{i}) + \mathsf{AEnc}(pk, r_{i})\right)\right)$$

Bob sends the set $\{ct'_i\}$ in shuffled order π to Alice.

4. Alice uses her private key sk to decrypt each component in all ct'_i . She computes

$$Z = \{ (H(x_i)^{a \cdot b_i})^{\frac{1}{a}} \}_{i \in [N+s]} = \{ H(x_i)^{b_i} \}_{i \in [N+s]}, \qquad W = \{ u_i + r_i \}_{i \in [N+s]}.$$

Denote $U = \{(H(x_i)^{b_i}, u_i + r_i)\}_{i \in [N+s]}$. 5. For $1 \leq j \leq n$, Bob computes $I_j = \{h_i(y_j)\}_{i \in [\ell]} \cup \{N+1, \dots, N+s\}$ and performs the symmetric-key encryption as

$$\overrightarrow{a_j} = \left\{ \mathsf{Enc} \left(H(y_j)^{b_i}, v_j - r_i \right) \right\}_{i \in I_j} = \{a_{j,i}\}_{i \in I_j}$$

Then Bob sends $\{\overline{a_j}\}_{j \in [n]}$ with every $\ell + s$ items in $\overrightarrow{a_j}$ shuffled in order π_j for $1 \leq j \leq n$ to Alice.

6. For $1 \leq j \leq n$, Alice tries all items in set Z to decrypt the ciphertexts in $\overline{a'_j}$. That is, if there exists an $a_{j,i} \in \overrightarrow{a_j}$ and an $(K, S) \in U$ such that $\mathsf{Dec}(K, a_{j,i}) \neq \bot$, then let $w_j = \text{Dec}(K, a_{j,i}) + S$, else let $w_j = \bot$. Then Alice computes $j^* = \operatorname{argmax}_j w_j$ and sends it to Bob.

Outputs: Alice outputs (w_1, \ldots, w_n) and Bob outputs y_{i^*} .

Fig. 4: Π_{DDH} : the protocol based on DDH.

Privately Compute the Item with Maximal Weight Sum in Set Intersection

3.3 DDH-based Protocol

In this section we present our DDH-based protocol that achieves the ideal functionality of $\mathcal{F}_{\mathsf{IMWS}}$ described in Fig. 2.

An overview of protocol. In the aforementioned protocol, we applied additively homomorphic encryption on both items and weights. It can be observed that the reason why computing the discrete logarithm is required in Beauregard et al. [2]'s protocol is that the the DH paradigm is used for weights.

As the DH paradigm is more efficient than that of additively homomorphic encryption, in our DDH-based protocol we use DH paradigm for items while apply additively homomorphic encryption on weights to save cost as well as avoid the requirement of discrete logarithm.

Our construction. Our protocol is built upon the following primitives: an additively homomorphic encryption scheme (AGen, AEnc, ADec, ASum, ARefresh), a symmetric-key encryption scheme (Enc, Dec) and DH-based key exchange. The detailed protocol is presented in Fig. 4.

Correctness of the protocol follows immediately from inspection, assuming Cuckoo hashing fails with negligible probability.

The security of our protocol follows from the hardness assumption of DDH, the security of the additively homomorphic encryption scheme and the security properties of symmetric-key encryption.

Security Analysis. The protocol Π_{DDH} in Fig. 4 securely realizes $\mathcal{F}_{\text{IMWS}}$ in the semi-honest model. The correctness is detailed beforehand. We state the theorems for security below. The formal security proof appears in App. B.

Theorem 2. Assume that the additively homomorphic encryption scheme is IND-CPA secure, the DDH assumption holds in G, the hash function H is modeled as a random oracle, and the one-time, symmetric-key encryption scheme has robust decryption property. Then there exist simulators SIM_A , SIM_B such that for security parameter λ and inputs $X = \{(x_i, u_i)\}_{i \in [n]}, Y = \{(y_i, v_i)\}_{i \in [n]}, \}$

 $\textit{View}_{A}^{\Pi_{\textit{DDH}}}(\{(x_{i}, u_{i})\})_{i \in [n]}, \{(y_{i}, v_{i})\})_{i \in [n]}) \stackrel{c}{\approx} \textit{SIM}_{A}(1^{\lambda}, \{(x_{i}, u_{i})\})_{i \in [n]}, (w_{1}, ..., w_{n}))$

 $View_{B}^{\Pi_{DDH}}(\{(x_{i}, u_{i})\})_{i \in [n]}, \{(y_{i}, v_{i})\})_{i \in [n]}) \stackrel{c}{\approx} SIM_{B}(1^{\lambda}, \{(y_{i}, v_{i})\})_{i \in [n]}, y_{j^{*}})$

where $(w_1, ..., w_n)$ is the weight sums for the intersection and y_{j^*} is the item in the intersection which has the maximum weight sum.

4 Extensions and Applications

In this section, we discuss some extensions of the functionality $\mathcal{F}_{\mathsf{IMWS}}$ which can be used in many real-world applications. All of these extensions can be easily achieved by modifying our protocols outlined in Sec. 3.

Compute the Items with Large Weight Sums. Under the $\mathcal{F}_{\text{IMWS}}$ functionality, Bob only obtains the item with the maximal weight sum. However, in some applications, Bob may wish to obtain all elements in the intersection whose weight sum exceeds a certain threshold. For example, in some competitions, there may be more than one final winner. A simple modification to the final step of our protocol can achieve this functionality. Specifically, Alice sends to Bob all indices in $(w_1, ..., w_n)$ that exceed the given threshold, then Bob finds the corresponding elements.

Compute the Items with Large Weight Differences. Similarly, in certain applications, Bob may wish to identify all elements in the intersection whose weight difference exceeds a specified threshold. For example, consider a scenario where two reviewers are evaluating a job by scoring various metrics, with the items they score potentially differing. If a metric is scored by both reviewers and there is a significant difference in their scores, it needs to be submitted to a higher-level supervisor for further evaluation. The two parties do not wish to disclose any additional information. Our protocol is well-suited for this scenario. By modifying Bob's score v_j to $-v_j$ in the protocol and running it, Alice will obtain the weight differences for all elements in the intersection. She can then send the indices of those elements whose weight difference exceeds the threshold to Bob, who can identify the relevant elements and submit them to the higher-level supervisor. It is important to note that, at this point, Alice will know the weight differences for all intersecting elements but will not know which specific elements correspond to those differences.

Compute the Items with Large Weight Products. In data analysis, sometimes both parties may wish to obtain the elements in the intersection with weight product exceeds a certain threshold. For example, Alice owns a store and holds information on the expenditure of individuals who shop there. Here, denote x_i as the consumer ID, and u_i represents the amount spent by the consumer. On the other hand, Bob is an advertising company and holds information on users who have viewed the advertisements. Bob assigning weights to users based on the duration of ad views. Here, denote y_j as the ID of the user who viewed the ad, and v_j represents the reciprocal of the viewing time. They want to identify the target audience with the best advertising effectiveness, meaning all users in the intersection whose $u_i v_j$ (for any matching ID $x_i = y_j$) exceeds a certain threshold. To achieve this functionality, we can map the additive group operations of u_i and v_j in our protocol to the multiplicative group operations.

5 Implementation and Evaluation

In the following, we evaluate our protocols and FOCI proposed by [2]. We first discuss their implementational features and compare them theoretically. We then give an empirical performance comparison between the protocols for different settings.

5.1 Theoretical Evaluation

Asymptotic performance comparison with protocol by [2]. In Tab. 1 and Tab. 2, we depict the asymptotic computation and communication complexity for the party with the majority of the workload of our OPRF-based protocol and DDH-based protocol, as well as the protocol by [2], in terms of counts of different types of operations and different types of elements transferred.

Table 1: Computation complexities for protocols. "Exp." is the number of group exponentiations. "AHE" is the number of additively homomorphic encryption operations, including encryptions, decryptions and additions. "Sym." is the number of symmetric-key operations, including encryptions and decryptions. Cuckoo(x) denotes the computation cost of Cuckoo hashing for x items. OPRF(x) refers to the computation cost of invoking OPRFs x times. "Dlog" refers to the computation cost of discrete logarithm. n is the size of input set, N is the size of Cuckoo hash table, s is the size of stash, and ℓ is the number of hash functions.

	Exp.	AHE	Sym.	Misc.	Dlog	
$\begin{array}{c} \Pi_{OPRF} \\ (\mathrm{Sec.} \ 3.2) \end{array}$	-	8(N+s)	$n(\ell+s) + (N+s)n(\ell+s)$	$\operatorname{Cuckoo}(n) + \operatorname{OPRF}(N+s)$	-	
П _{DDH} (Sec. 3.3)	$\begin{array}{c} 3(N\!+\!s)+\\ n(\ell+s) \end{array}$	4(N+s)	$n(\ell + s) + (N + s)n(\ell + s)$	$\operatorname{Cuckoo}(n)$	-	
FOCI [2]	10n	-	n(n+1)	-	$ \{x_i\} \cap \{y_j\} $	

Based on the comparison we observe that Π_{OPRF} requires double additively homomorphic encryption operations than Π_{DDH} . This is due to the use of DHbased shuffled OPRF in Π_{DDH} instead of OT-based OPRF in Π_{OPRF} . We expect that the DDH protocol will have best communication efficiency, while the most computationally efficient protocol relies on the relative costs of exponentiation and additively homomorphic operations, which we will investigate through our experiments.

Comparing to the protocol in [2], our protocols use additively homomorphic operations to compute weight sums instead of computationally expensive discrete logarithms in [2]. However, our protocols require approximately $(\ell + s)$ times more symmetric-key operations than [2]. (The commonly used Cuckoo hash parameters are $\ell = 3$ and s = 0).

We note that there is another approach for computing functions on the weights in the intersection by using circuit PSI [33], which is to use general twoparty computation (garbled circuits [37] or the GMW protocol [18]) to compute any functions on the intersection and the associated weights. Pinkas et al. [33] constructed circuit PSI with associated payloads by leveraging batch (Oblivious Programmable Pseudo-Random Function) OPPRF and circuit with O(n)input wires that computes pairwise comparisons and the target function associated with payloads. Compared with the protocols specialized for functionality $\mathcal{F}_{\mathsf{IMWS}}$, this approach significantly increase the communication cost or the round complexity. Therefore, we omit the comparison with circuit PSI protocol [33].

Table 2: Communication complexities for protocols. "Group Elts." is the number of group elements sent in the protocol. "AHE" is the number of additively homomorphic encryption ciphertexts. "Sym." is the number of symmetric-key ciphertexts. OPRF(x) refers to the communication cost of invoking OPRFs x times. n is the size of input set, N is the size of Cuckoo hash table, s is the size of stash, and ℓ is the number of hash functions. Moreover, both of our protocols have an index information j^* to be transferred from Alice to Bob, so that Bob can output the item with maximal weight sum in the intersection.

	Group Elts.	AHE	Sym.	Misc.
Π_{OPRF} (Sec. 3.2)	-	4(N+s)	$n(\ell + s)$	$OPRF(N+s) + \log n$
П _{DDH} (Sec. 3.3)	2(N+s)	2(N+s)	$n(\ell + s)$	$\log n$
FOCI [2]	4n + 1	-	n	$\log n$

5.2 Implementation Details

We implement³ our two protocols Π_{OPRF} and Π_{DDH} in C++ using [29]. The computational statistical security parameter are $\kappa = 128$ and $\lambda = 40$. The parameters of Cuckoo Hash is selected according to [12], with three hash functions $\ell = 3$ and stash size s = 0. Once ℓ , s and the set size n are determined, the number of bins N is calculated via the formula in [12] (ref. Appendix B). For symmetric encryption, we uses AES and prefix 40 bits of zeros in the plaintext to determine if decryption is successful. For OPRF, we use KKRT protocol from [21]. For DDH, we use elliptic curve group "prime256v" from OpenSSL, which is a widely used NIST elliptic curve with 256-bit group elements. We use Paillier encryption scheme from pailliercryptolib with OpenMP disabled. We use the same parameters as [19]. The implementation is single-threaded, but most computations can be parallelized. Therefore, we expect that server setup time can achieve approximately linear speedup when using multi-threading.

5.3 Benchmark

The experiments were run on a desktop computer with AMD 3950X CPU and 32GiB RAM. We considered localhost environment and simulated WAN network settings with 80ms RTT and different bandwidths using the Linux tc command.

In Tab. 3, we have a detailed benchmark set size $n \in \{100, 1000, 10000, 100000\}$ We can observe that the DDH-based protocol outperforms the OPRF-based protocol in all comparisons. Bandwidth does not have a very big impact on the runtime, since the communication cost is relatively low. We note that the Paillier encryption library we used does not have the Damgard-Jurik optimization [11] as in [19], so our results could be further improved.

We show the communication and computation costs of our two protocols for various input sizes and those with respect to each component operation in Tab. 4.

³ Our implementation is available at https://github.com/lzjluzijie/imws.

Protocol		Comm	Time					
1 1010001			localhost	$1000 \mathrm{Mbps}$	$100 \mathrm{Mbps}$	$10 \mathrm{Mbps}$		
	100	0.09	0.11	0.29	0.30	0.36		
п	1000	0.62	0.76	1.62	1.64	1.99		
TIDDH	10000	6.28	9.64	10.52	10.79	15.04		
	100000	63.46	273.37	274.13	279.41	327.48		
	100	0.17	0.15	0.68	0.69	0.80		
Π_{OPRF}	1000	1.13	1.03	1.63	1.66	2.18		
	10000	11.29	11.83	13.06	13.25	17.20		
	100000	113.90	297.02	301.83	311.31	388.29		

Table 3: Communication cost (in MiB) and running time (in seconds).

Table 4: Detailed communication cost (in MiB) and running time (in seconds).

Protocol	n	Exp.		OPRF		AHE		Sym.		Total	
		Time	Comm	Time	Comm	Time	Comm	Time	Comm	Time	Comm
П _{DDH}	100	0.02	0.01	-	-	0.07	0.07	0.00	0.01	0.11	0.09
	1000	0.12	0.08	-	-	0.50	0.52	0.02	0.04	0.76	0.62
	10000	1.21	0.81	-	-	5.04	5.01	1.76	0.46	9.64	6.28
	100000	12.89	8.20	-	-	51.75	50.68	196.24	4.58	273.37	63.46
П _{орrf}	100	-	-	0.01	0.02	0.14	0.14	0.00	0.01	0.15	0.17
	1000	-	-	0.01	0.09	1.00	0.99	0.02	0.04	1.03	1.13
	10000	-	-	0.01	0.80	10.05	10.13	1.76	0.46	11.83	11.29
	100000	-	-	0.06	7.96	101.15	101.36	195.66	4.58	297.02	113.90

It shows that most of the cost in our protocols is taken by additive homomorphic encryption. Although the time costs of exponentiation is much more than that of OPRF, the additional AHE operations make Π_{OPRF} slower than Π_{DDH} . The total times of our two protocols in the case of input size n = 100000 are significantly higher, due to the quadratic comparisons in the last step.

5.4 Empirical Comparison

We empirically evaluate and compare the performance of our two protocols with that of the FOCI protocol [2]. To provide a fair comparison, we implement the FOCI protocol in C++ using [29], the same language and library as our protocol, and run on the same hardware. We also used the same elliptic curve "prime256v1" and the "baby-step giant-step" algorithm to compute discrete logarithm.

As discussed in Sec. 5.1, the theoretical analysis indicates that the performance of the FOCI protocol varies significantly depending on various factors.

Specifically, for a given set size n, the FOCI protocol's performance is primarily affected by the size of the intersection $|X \cap Y|$ and the values of weights of the elements in intersection, since it needs to compute discrete logarithms on these elements. In contrast, the performance of our protocols is basically independent of these two factors (the size of the intersection and the values of weights). We analyze the effect of these two parameters separately. In the following comparison, we fix the set size n. Let max denote the maximum weight of the elements, with the weights of the elements uniformly distributed as integers between 0 and max - 1.

Effects of Varying Intersection Sizes. In the following, we compare the performance of our protocols with the FOCI protocol under different intersection sizes. To represent a real-world setting, we simulate a WAN setting with an 80ms RTT and 1000Mbps network bandwidth. We fix the sizes of both input sets n = 10000, the maximum weight of elements max = 10n, and evaluate the running time and communication cost of the FOCI and our protocols for different intersection sizes $\{0, \frac{n}{4}, \frac{n}{2}, \frac{3n}{4}, n\}$. The results are illustrated in Tab. 5. Since the performance of our protocols are basically independent of the intersection size, their results are not classified on $|X \cap Y|$ in the table. From Tab. 5, it can

Protocol	$ X \cap Y $	Time [s]	Comm [MiB]		
11000001	121111	T IIIIO [0]			
П _{DDH}	-	10.08	6.28		
Π_{OPRF}	-	12.63	11.29		
	0	5.20	1.72		
	$\frac{n}{4}$	11.90	1.72		
FOCI [2]	$\frac{n}{2}$	18.94	1.72		
	$\frac{3n}{4}$	25.67	1.72		
	n	32.99	1.72		

Table 5: The comparison of running time and communication cost across protocols for different intersection sizes. In all tests, the set size is n = 10000 and the bandwidth is 1000Mbps.

be observed that as the intersection size increases, the running time of the FOCI protocol grows approximately linearly. Due to the high computational cost of calculating discrete logarithms, in the tested environment, when the intersection size is $\frac{n}{4}$, the running time of our protocol is similar to that of the FOCI protocol. However, when the intersection size exceeds $\frac{n}{4}$, our protocols shows a significant speed advantage over the FOCI protocol. It is worth noting that the communication cost of the FOCI protocol is lower than that of our protocol, making it more advantageous when network bandwidth is limited.

Effects of Varying Maximum Weight. We compare the performance of our protocols with the FOCI protocol under different maximum weights max of the set elements. We fix the intersection size to be $\frac{n}{2}$ and evaluate the running time of the FOCI and our protocols as max/n varies from 1 to 100, for $n \in \{100, 1000, 100000\}$, respectively. The results are depicted in Fig. 5.



Fig. 5: FOCI protocol [2] vs Ours for different settings. To clarify, the x and y axis has been scaled logarithmically rather than shown proportionally. All tests were conducted in a LAN network, with the intersection size to be $\frac{n}{2}$. The weights of the elements were uniformly distributed as integers between 0 and max - 1.

From Fig. 5, we can observe that the run-time of FOCI protocol increases significantly as max/n increases, while the run-time of our protocols are unaffected. It is evident that our protocols has a more noticeable advantage when the set size n is large, whereas the FOCI protocol outperforms ours when n is small. Additionally, it can be observed that as n increases, the gap between the running times of our OPRF-based protocol and the DDH-based protocol gradually decreases. The reason is that, in terms of computational cost, the DDH-based protocol requires exponentiation, while the OPRF-based protocol uses OT extension to compute the OPRF, making it more advantageous when n is large.

6 Conclusion

We present two novel protocols that achieve the functionality called "privately compute the item with maximal weight sum in set intersection", which are based on OPRF and DDH assumption respectively. Compared with the prior protocol

proposed by Beauregard et al. [2] which has the constraint that all the weights in the input sets have to be polynomial in magnitude, both of our protocols remove such restriction on weights.

We compare the performance of our protocols with the FOCI protocol both theoretically and empirically. We find out that the performance of FOCI protocol is primarily affected by the size of the intersection and the values of elements' weights in intersection when fixing set size, while the performance of ours is basically independent of these two factors. Therefore, FOCI protocol is more suitable for scenarios with small element weights and intersections, such as in "meeting scheduling" applications. In contrast, our protocols are better suited for scenarios with larger element weights or larger intersections, such as "weighted voting" applications.

Acknowledgment. We would like to thank the anonymous reviewers for their helpful comments. This work is supported by National Key R&D Program of China (No. 2021YFB2701304).

References

- Baldi, P., Baronio, R., De Cristofaro, E., Gasti, P., Tsudik, G.: Countering gattaca: efficient and secure testing of fully-sequenced human genomes. In: Proceedings of the ACM CCS 2018. pp. 691–702 (2011)
- Beauregard, T., Xia, J., Rosulek, M.: Finding one common item, privately. In: International Conference on Security and Cryptography for Networks. pp. 462– 480. Springer (2022)
- Ben-David, A., Nisan, N., Pinkas, B.: Fairplaymp: a system for secure multi-party computation. In: Proceedings of the ACM CCS 2008. pp. 257–266 (2008)
- Benny, P., Thomas, S., Michael, Z.: Faster private set intersection based on ot extension. In: Usenix Security 14. pp. 797–812 (2014)
- Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) 6(3), 1–36 (2014)
- Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) lwe. SIAM Journal on computing 43(2), 831–871 (2014)
- Chase, M., Miao, P.: Private set intersection in the internet setting from lightweight oblivious prf. In: CRYPTO 2020, Proceedings, Part III 40. pp. 34–63. Springer (2020)
- Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: Proceedings of the ACM CCS 2017. pp. 1243–1255 (2017)
- Chida, K., Hamada, K., Ichikawa, A., Kii, M., Tomida, J.: Communication-efficient inner product private join and compute with cardinality. In: Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security. pp. 678– 688 (2023)
- Dahlia, M.: Fairplay-a secure two-party computation system. In: Usenix Security Symposium. pp. 1–17 (2004)
- Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In: PKC 2001. Lecture Notes in Computer Science, vol. 1992, pp. 119–136. Springer (2001)

Privately Compute the Item with Maximal Weight Sum in Set Intersection

- 12. Demmler, D., Rindal, P., Rosulek, M., Trieu, N.: Pir-psi: scaling private contact discovery. Proceedings on Privacy Enhancing Technologies (2018)
- ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE transactions on information theory **31**(4), 469–472 (1985)
- 14. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive (2012)
- Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: TCC 2005, Proceedings 2. pp. 303–324. Springer (2005)
- Frikken, K., Atallah, M., Zhang, C.: Privacy-preserving credit checking. In: Proceedings of the 6th ACM conference on Electronic commerce. pp. 147–154 (2005)
- Garimella, G., Mohassel, P., Rosulek, M., Sadeghian, S., Singh, J.: Private set operations from oblivious switching. In: IACR International Conference on Public-Key Cryptography. pp. 591–617. Springer (2021)
- Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game, or a completeness theorem for protocols with honest majority. In: Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali, pp. 307–328 (2019)
- Ion, M., Kreuter, B., Nergiz, A.E., Patel, S., Saxena, S., Seth, K., Raykova, M., Shanahan, D., Yung, M.: On deploying secure computing: Private intersection-sumwith-cardinality. In: 2020 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 370–389. IEEE (2020)
- Kirsch, A., Mitzenmacher, M., Wieder, U.: More robust hashing: Cuckoo hashing with a stash. SIAM Journal on Computing 39(4), 1543–1561 (2010)
- Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious prf with applications to private set intersection. In: Proceedings of the ACM CCS 2016. pp. 818–829 (2016)
- Malka, L.: Vmcrypt: modular software architecture for scalable secure computation. In: Proceedings of the ACM CCS 2011. pp. 715–724 (2011)
- Miao, P., Patel, S., Raykova, M., Seth, K., Yung, M.: Two-sided malicious security for private intersection-sum with cardinality. In: Annual International Cryptology Conference. pp. 3–33. Springer (2020)
- 24. Nagaraja, S., Mittal, P., Hong, C.Y., Caesar, M., Borisov, N.: {BotGrep}: Finding {P2P} bots with structured graph analysis. In: USENIX Security 10 (2010)
- 25. Narayanan, A., Thiagarajan, N., Lakhani, M., Hamburg, M., Boneh, D., et al.: Location privacy via private proximity testing. In: NDSS. vol. 11 (2011)
- Nikolaenko, V., Weinsberg, U., Ioannidis, S., Joye, M., Boneh, D., Taft, N.: Privacypreserving ridge regression on hundreds of millions of records. In: 2013 IEEE symposium on security and privacy. pp. 334–348. IEEE (2013)
- Pagh, R., Rodler, F.F.: Cuckoo hashing. Journal of Algorithms 51(2), 122–144 (2004)
- Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International conference on the theory and applications of cryptographic techniques. pp. 223–238. Springer (1999)
- 29. Peter Rindal, L.R.: libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe
- Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Spot-light: lightweight private set intersection from sparse of extension. In: CRYPTO 2019, Proceedings, Part III 39. pp. 401–431. Springer (2019)

- 24 H. Cai et al.
- Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Psi from paxos: fast, malicious private set intersection. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 739–767. Springer (2020)
- Pinkas, B., Schneider, T., Segev, G., Zohner, M.: Phasing: Private set intersection using permutation-based hashing. In: USENIX Security 15. pp. 515–530 (2015)
- Pinkas, B., Schneider, T., Tkachenko, O., Yanai, A.: Efficient circuit-based psi with linear communication. In: EUROCRYPT 2019, Proceedings, Part III 38. pp. 122–153. Springer (2019)
- Raghuraman, S., Rindal, P.: Blazing fast psi from improved okvs and subfield vole. In: Proceedings of the ACM CCS 2022. pp. 2505–2517 (2022)
- Rindal, P., Schoppmann, P.: Vole-psi: fast oprf and circuit-psi from vector-ole. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 901–930. Springer (2021)
- Rosulek, M., Roy, L.: Three halves make a whole? beating the half-gates lower bound for garbled circuits. In: Annual International Cryptology Conference. pp. 94–124. Springer (2021)
- 37. Yao, A.C.C.: How to generate and exchange secrets. In: 27th annual symposium on foundations of computer science (Sfcs 1986). pp. 162–167. IEEE (1986)

A Preliminaries

A.1 Additively Homomorphic Encryption

Homomorphic encryption is a form of encryption that allows computations to be performed on encrypted data without first having to decrypt it. An additively homomorphic encryption scheme consists of the following probabilistic polynomial-time algorithms:

- AGen: Given a security parameter κ , AGen(κ) outputs a public-private key pair (pk, sk), and specifies a message space \mathcal{M} .
- AEnc: Given the public key pk and a plaintext message $m \in \mathcal{M}$, one can compute a ciphertext $\mathsf{AEnc}(pk, m)$, an encryption of m under pk.
- ADec: Given the secret key sk and a ciphertext AEnc(pk, m), ADec is to recover a plaintext m.
- ASum: Given the public key pk and a set of ciphertexts {AEnc (pk, m_i) } which are the encryption of messages $\{m_i\}$, one can homomorphically compute a ciphertext which is the encryption of the sum of the underlying messages:

$$\mathsf{AEnc}\left(pk,\sum_{i}m_{i}\right) = \mathsf{ASum}\left(\{\mathsf{AEnc}\left(pk,m_{i}\right)\}_{i}\right)$$

 ARefresh: One can randomize ciphertexts using a randomized procedure denoted as ARefresh.

The commonly used additively homomorphic encryption schemes include Paillier encryption [28], Exponential ElGamal encryption [13] and Ring-LWEbased encryption schemes [6,5,14]. We depend on the standard concept of CPA security in encryption, which essentially implies that, without the private key sk, encrypted messages are computationally indistinguishable from one another.

A.2 Cuckoo Hash

A Cuckoo hash table [27] is a data structure supporting insertion and membership tests. It addresses the issue of hash collisions of values of hash functions in a table, with worst-case constant lookup time. It is parameterized by a number of bins N, a stash size s, and by k randomly chosen hash functions. An empty Cuckoo Hash Table has N empty bins. It works as follows:

- Insertion: When inserting an item x into the table, if any of the bins $\{h_i(x)\}_{i=1}^{k}$ is empty, then x is placed in one of those bins. Otherwise, a bin in $\{h_i(x)\}_{i=1}^{k}$ is randomly chosen, and the item in that bin is replaced with x. The evicted item is then recursively inserted. If this process does not terminate after a fixed set of iterations, then the final evicted element is placed in a special bin called the stash. If the stash already contains s items, the insertion algorithm fails.
- Lookup: To check if an item x is in the Cuckoo hash table, one checks each of the bins in $\{h_i(x)\}_{i=1}^k$ for the item.

It was shown in [20] that Cuckoo hashing of n elements into $N = (1 + \varepsilon)n$ bins with $\varepsilon \in (0, 1)$ for any $k \ge 2(1 + \varepsilon)ln(\frac{1}{\varepsilon})$ and $s \ge 0$ fails with probability $O(n^{1-c(s+1)})$, for a constant c > 0 and $n \mapsto \infty$.

A.3 Decisional Diffie-Hellman Assumption

Definition 2. We say that the DDH problem is hard relative to \mathcal{G} if for all probabilistic polynomial-time algorithms \mathcal{A} there is a negligible function negl such that

$$\left|\Pr\left[\mathcal{A}\left(\mathbb{G}, q, g, g^{x}, g^{y}, g^{z}\right) = 1\right] - \Pr\left[\mathcal{A}\left(\mathbb{G}, q, g, g^{x}, g^{y}, g^{xy}\right) = 1\right]\right| \le \operatorname{neg}(\kappa),$$

where in each case the probabilities are taken over the experiment in which $\mathcal{G}(1^{\kappa})$ outputs (\mathbb{G}, q, g) , and then uniform $x, y, z \in \mathbb{Z}_q$ are chosen.

In other words, the distributions (g, g^x, g^y, g^{xy}) and (g, g^x, g^y, g^z) are computationally indistinguishable.

B Security Analysis for DDH-Based Protocol

We prove the security of our DDH-based protocol Π_{DDH} , presented in Fig. 4.

Theorem 2. Assume that the additively homomorphic encryption scheme is IND-CPA secure, the DDH assumption holds in G, the hash function H is modeled as a random oracle, and the one-time, symmetric-key encryption scheme has robust decryption property. Then there exist simulators SIM_A , SIM_B such that for security parameter λ and inputs $X = \{(x_i, u_i)\}_{i \in [n]}, Y = \{(y_i, v_i)\}_{i \in [n]}, \}$

$$View_{A}^{\Pi_{DDH}}(\{(x_{i}, u_{i})\})_{i \in [n]}, \{(y_{i}, v_{i})\})_{i \in [n]}) \stackrel{\circ}{\approx} SIM_{A}(1^{\lambda}, \{(x_{i}, u_{i})\})_{i \in [n]}, (w_{1}, ..., w_{n}))$$

$$\textit{View}_{B}^{\Pi_{\textit{DDH}}}(\{(x_{i}, u_{i})\})_{i \in [n]}, \{(y_{i}, v_{i})\})_{i \in [n]}) \stackrel{c}{\approx} \textit{SIM}_{B}(1^{\lambda}, \{(y_{i}, v_{i})\})_{i \in [n]}, y_{j^{*}})$$

where $(w_1, ..., w_n)$ is the weight sums for the intersection and y_{j^*} is the item in the intersection which has the maximum weight sum.

Proof. Security against corrupt Alice. We describe the simulator algorithm SIM_A in Algorithm 3.

Algorithm 3 The simulator for Alice in the DDH-based Protocol

Input: $(1^{\lambda}, \{(x_i, u_i)\})_{i \in [n]}, (w_1, ..., w_n))$ Output: SimView(Alice) SIM_A $(1^{\lambda}, \{(x_i, u_i)\})_{i \in [n]}, (w_1, ..., w_n)$):

- 1. Honestly simulate the Setup phase between Alice and Bob.
- 2. Honestly simulate steps 1-2 between Alice and Bob, receiving the set $\{ct_i = (H(x_i)^a, \mathsf{AEnc}(pk, u_i))\}_{i \in [N+s]}$ at the end of step 2.
- 3. In step 3, choose a set C^* consisting of N + s random elements c_i from group G, a set D^* consisting of N + s random elements d_i from the input domain of additively homomorphic encryption. Compute the exponentiations to each c_i with Alice's Diffie-Hellman key a, and the fresh encryptions to each d_i under Alice's public key pk, then send $ct'_i = (c^a_i, \mathsf{AEnc}(pk, d_i))$ to Alice.
- 4. Simulate step 4 for Alice honestly.
- 5. In step 5, choose the set $\{\overrightarrow{a_j}\}_{j\in[n]} = \{a_{j,i}\}_{j\in[n],i\in[\ell+s]}$ such that each $\overrightarrow{a_j}$ contains exactly $\ell+s$ ciphertexts which are the encryptions of 0 under randomly chosen keys $K_{j,i}$ from group \mathbb{G} , that is $a_{j,i} = \operatorname{Enc}(K_{j,i}, 0)$. For each $1 \leq j \leq n$, if $w_j \neq \bot$, then for a randomly chosen index $t_j \in [\ell+s]$, chose some c_{l_j} randomly from the set C^* (each element in C^* is used at most once) with the corresponding indexed element d_{l_j} in set D^* , and replace $a_{j,t_j} = \operatorname{Enc}(K_{j,t_j}, 0)$ with $\operatorname{Enc}(c_{l_j}, w_j d_{l_j})$. Send the set $\{\overrightarrow{a_j}\}_{j\in[n]} = \{a_{j,i}\}_{j\in[n],i\in[\ell+s]}$ to Alice.
- 6. Simulate step 6 for Alice honestly.
- 7. Output the view of Alice in this interaction.

We argue that

 $\mathsf{View}_{A}^{\Pi_{\mathsf{DDH}}}(\{(x_{i}, u_{i})\})_{i \in [n]}, \{(y_{i}, v_{i})\})_{i \in [n]}) \stackrel{\mathrm{c}}{\approx} \mathsf{SIM}_{A}(1^{\lambda}, \{(x_{i}, u_{i})\})_{i \in [n]}, (w_{1}, ..., w_{n}))$

by hybrid.

- \mathbf{Hyb}_0 : The transcript corresponding to the view of Alice in a real execution of the protocol.
- **Hyb**₁ : The same as Hyb₀, except in step 5, for all $w_j = \bot$, Bob replaces the ciphertexts $\{a_{j,i}\}_{i \in [\ell+s]}$ with fresh encryptions of 0 under the key $\{H(y_j)^{b_i} | i \in I_j\}$, that is $\{a_{j,i}\}_i = \{\text{Enc}(H(y_j)^{b_i}, 0) | i \in I_j\}$.

 Hyb_1 and Hyb_0 are indistinguishable by the one time security of symmetric-key encryption.

Privately Compute the Item with Maximal Weight Sum in Set Intersection

 \mathbf{Hyb}_2 : The same as Hyb_1 , except in step 5, for all $w_j = \perp$, Bob replaces the ciphertexts $\{a_{j,i}\}_{i \in [\ell+s]}$ with fresh encryptions of 0 under randomly chosen key $K_{j,i}$ from group \mathbb{G} , that is $a_{j,i} = \mathsf{Enc}(K_{j,i}, 0)$. Hyb₂ and Hyb₁ are indistinguishable by the Decisional Diffie-Hellman (DDH)

assumption of group G, and random oracle assumption on hash function H. \mathbf{Hyb}_3 : The same as Hyb₂, except in step 3, the first component of each ct_i is replaced with an exponentiation of a uniformly random value c_i from group G under Alice's DH key a, that is $ct_i' = (c_i^a, \mathsf{AEnc}(pk, u_{\pi^{-1}(i)} + r_{\pi^{-1}(i)}))$, and in step 5, for each $w_j \neq \bot$, for the corresponding element index t_j , Bob replace the ciphertexts $\{a_{j,i}\}_{i\in[\ell+s]\setminus\{t_j\}}$ with fresh encryptions of $v_j - r^{(i)}$ under randomly chosen key $K_{j,i}$ from group G, where $r^{(i)} = r_{h_{\pi_j^{-1}(i)}(y_j)}$ if $i \in [\ell]$, $r^{(i)} = r_{N-\ell+\pi_j^{-1}(i)}$ if $i \in [\ell+1, \ell+s]$, that is $a_{j,i} = \mathsf{Enc}(K_{j,i}, v_j - r^{(i)})$, and replace the ciphertext a_{j,t_j} with fresh encryptions under the corresponding key c_{l_j} , that is $a_{j,t_j} = \mathsf{Enc}(c_{l_j}, v_j - r^{(t_j)})$. (The correspondence of t_j and c_{l_j} means that c_{l_j} can successfully decrypts a_{j,t_j} , which holds that $\pi^{-1}(l_j) = h_{\pi_j^{-1}(t_j)}(y_j)$ if $t_j \in [\ell], \pi^{-1}(l_j) = N - \ell + \pi_j^{-1}(t_j)$ if $t_j \in [\ell + 1, \ell + s]$.) Hyb₃ and Hyb₂ are indistinguishable by the Decisional Diffie-Hellman (DDH)

Hyb₃ and Hyb₂ are indistinguishable by the Decisional Diffie-Hellman (DDH) assumption of group \mathbb{G} , random oracle assumption on hash function H, and the security of symmetric-key encryption on hiding keys.

Hyb₄ : The same as Hyb₃, except in step 3, the second component of each ct'_i is replaced with an encryption of a uniformly random value d_i , that is $ct'_i = (c^a_i, \mathsf{AEnc}(pk, d_i))$, and in step 5, for each $w_j \neq \bot$, for the corresponding element index t_j , Bob replace the ciphertexts $\{a_{j,i}\}_{i \in [\ell+s] \setminus \{t_j\}}$ with fresh encryptions of 0 under key $K_{j,i}$, that is $a_{j,i} = \mathsf{Enc}(K_{j,i}, 0)$, and replace the ciphertext a_{j,t_j} with fresh encryptions of $w_j - d_{l_j}$ under the corresponding key c_{l_i} , that is $a_{j,t_j} = \mathsf{Enc}(c_{l_i}, w_j - d_{l_j})$.

 Hyb_4 and Hyb_3 are indistinguishable by the one-time-pad property of adding the random value r_i , the hiding property of additively homomorphic encryption scheme, and the one time security of symmetric key encryption.

 \mathbf{Hyb}_5 : The view of Alice output by SIM_A .

 Hyb_5 and Hyb_4 are identically distributed.

Security against corrupt Bob.

We observe that Bob's view in the protocol consists of the following:

- 1) A public key pk of additively homomorphic encryption (Setup).
- 2) Whether Alice aborts due to Cuckoo hashing failure (step 1).

3) The sets $\{ct_i\}_{i \in [N+s]}$ of ciphertexts encrypted under Alice's Diffie-Hellman key a and homomorphic encryption key pk (step 2).

4) An index j^* corresponding to Bob's output (step 6).

we describe the simulator algorithm SIM_B in Algorithm 4. We argue that

$$\mathsf{View}_{B}^{\Pi_{\mathsf{DDH}}}(\{(x_{i}, u_{i})\})_{i \in [n]}, \{(y_{i}, v_{i})\})_{i \in [n]}) \stackrel{\circ}{\approx} \mathsf{SIM}_{B}(1^{\lambda}, \{(y_{i}, v_{i})\})_{i \in [n]}, y_{j^{*}})$$

using a multi-step hybrid argument, where each neighboring pair of hybrid distributions is computationally indistinguishable. Algorithm 4 The simulator for Bob in the DDH-based Protocol

Input: $(1^{\lambda}, \{(y_i, v_i)\})_{i \in [n]}, y_{j^*})$ **Output:** SimView(Bob)SIM_B $(1^{\lambda}, \{(y_i, v_i)\})_{i \in [n]}, y_{j^*})$:

- 1. Randomly choose a pair of homomorphic encryption keys (pk, sk) and sends pk to Bob in setup phase.
- 2. Never abort in step 1.
- 3. In step 2, send Bob $\{ct_i\}_{i \in [N+s]}$ to be pairs of random elements g_i from group G and fresh encryptions of 0 under pk, that is $ct_i = (g_i, \mathsf{AEnc}(pk, 0))$.
- 4. Simulate step 3 and step 5 for Bob honestly.
- 5. In step 6, send j^* corresponding to Bob's output y_{j^*} to Bob.
- 6. Output the view of Bob in this interaction.
- \mathbf{Hyb}_0 : The transcript corresponding to the view of Bob in a real execution of the protocol.
- Hyb_1 : The same as Hyb_0 , except in step 1, Alice never abort when executing Cuckoo hashing.

 Hyb_1 and Hyb_0 are indistinguishable by the property that Cuckoo hashing fails with negligible probability.

Hyb₂: The same as Hyb₂, except in step 2, Alice replaces the messages $\{ct_i\}_{i \in [N+s]}$ with pairs of random elements g_i from group G and fresh encryptions of 0 under public key pk, that is $ct_i = (g_i, \mathsf{AEnc}(pk, 0))$. Hyb₂ and Hyb₁ are indistinguishable by the Diffie-Hellman assumption of

group \mathbb{G} , random oracle assumption on hash function H, and the hiding property of additively homomorphic encryption scheme.

- \mathbf{Hyb}_3 : The view of Bob output by SIM_B .
 - Hyb_3 and Hyb_2 are identically distributed.