Private Intersection-Weighted-Sum

Koji Chida, Koki Hamada, Atsunori Ichikawa, Masanobu Kii, and Junichi Tomida

NTT Corporation
koji.chida.eb@hco.ntt.co.jp
koki.hamada.rb@hco.ntt.co.jp
atsunori.ichikawa.nf@hco.ntt.co.jp
masanobu.kii.gw@hco.ntt.co.jp
junichi.tomida.vw@hco.ntt.co.jp

Abstract. We propose secure two-party computation for a new functionality that we call Private Intersection-Weighted-Sum (PIW-Sum) and present an efficient semi-honestly secure protocol. In this computation, two parties own integer matrices $\mathbf{X}$ and $\mathbf{Y}$, respectively, where each row of the matrices is associated with an identifier. Let $I = (i_1, \ldots, i_n)$ be the intersection of the identifier sets for the two parties. The goal is to compute the matrix $\widehat{\mathbf{Y}}^\top \widehat{\mathbf{X}}$ together with the cardinality $|I|$, where the j-th rows of $\widehat{\mathbf{X}}$ and $\widehat{\mathbf{Y}}$ are the rows of $\mathbf{X}$ and $\mathbf{Y}$ with identifier i_j, respectively. This functionality is a generalization of Private Intersection-Sum (PI-Sum) proposed by Ion *et al.* (EuroS&P'20) and has important real-world applications such as computing a cross tabulation after the equijoin of two tables owned by different parties.
Our protocol is built on a new variant of oblivious pseudorandom function (OPRF), and we construct the new variant of OPRF from the decisional Diffie-Hellman (DDH) assumption. We implement both our PIW-Sum protocol and the the most efficient PI-Sum protocol by Ion *et al.* and compare their performance in the same environment. This shows that both communication cost and computational cost of our protocol are only about 2 times greater than those of the PI-Sum protocol in the case where $\mathbf{X}$ and $\mathbf{Y}$ are column vectors, i.e., the number of columns of $\mathbf{X}$ and $\mathbf{Y}$ is one.

Keywords: private set intersection, private intersection-sum, privateintersection-weighted-sum, two-party computation, secure computation

Table of Contents

1 Introduction

Secure multi-party computation (MPC) is a technique that allows parties to jointly compute a function value from their inputs without revealing anything else about the inputs. Recent intensive studies have drastically improved efficiency of MPC techniques and show that they can be used for many real-world applications. The problem that we address in this paper is as follows. Suppose two parties have their own database where each record has an identifier such as telephone number, e-mail address, etc., and extra information. They want to make some analysis on the database that is generated by the equijoin of their databases with respect to the identifiers, while both parties are unwilling to reveal any other information about their database. Especially, we consider the case where identifiers including those in the intersection are sensitive and need to be hidden from the communication parter.

A simple example for such a situation is as follows. Suppose that company that operates an online store asks an ad supplier to distribute an advertisement campaign. Later, the client company wants to know the effectiveness of the campaign for increasing purchase at the online store. It is natural to assume that only the client knows identifiers who made a purchase at the store and its amount while only the ad supplier knows identifiers to whom it sent an advertisement. In this case, the average purchase amount of users who receive the advertisement (aggregate conversion rate) would help to know the effectiveness of the campaign. The problem is that the company needs a list of users who received the advertisement to compute the conversion rate, while it is sensitive information and cannot be provided from the ad supplier. Hence, some sort of secure computation is necessary in such a situation.

Private Intersection-Sum with Cardinality. Several works consider secure computation on equijoin of databases [3, 6, 21, 26]. Especially, Private Intersection-Sum with Cardinality (PI-Sum) [21, 26] is exactly the two party computation that is introduced and actually deployed by Google [21] to deal with the situation as the above example. In PI-Sum, one party has a set of pairs of an identifier and an integer $\{(v_i, x_i)\}$ while the other party has an identifier set $\{w_i\}$, and the goal is to compute $\sum_{i:v_i=w_j} x_i$ i.e., the sum of integers x_i that correspond to the intersection of the two identifier sets, together with the cardinality $|\{v_i\} \cap \{w_i\}|$ of the intersection. Hence, PI-Sum can be seen as computation after equijoin of two tables owed by two parties where one table consists of only identifiers. We can observe that the above example can be solved by PI-Sum as follows. Let $\{w_i\}$ be the identifier set held by an ad supplier and $\{(v_i, x_i)\}$ be set of pairs of an identifier and its purchase amount held by a client. Then, the aggregate conversion rate can be obtained by $\sum_{i:v_i=w_j} x_i/|\{v_i\} \cap \{w_i\}|$.

PI-Sum is closely related to Private Set Intersection (PSI) [8, 9, 11–15, 17, 19, 20, 23, 30–38] and PSI-cardinality [3, 10, 17, 20, 22, 29, 39]. PSI is multi-party computation for obtaining the intersection of the sets owned by multiple parties, while PSI-cardinality allows parties to compute just the cardinality of the intersection of their sets. Especially, PSI has many applications such as privacy-preserving location sharing [28], testing of fully sequenced human genomes [4], botnet detection [27], social networks [24], and online gaming [7], and has been extensively studied.

Private Intersection-Weighted-Sum. In this work, we put forward two-party computation for new functionality that we call Private Intersection-Weighted-Sum[1] (PIW-Sum) to deal with the more general situation where each record of *both* tables owned by two parties contain extra information as well as an identifier. A case example that we can solve by PIW-Sum is as follows: the ad supplier in the previous example has an additional attribute such as age of each identifier and wants to analyze the effectiveness of the advertisement by age. Such a case cannot be solved by PI-Sum since its functionality requires that an input of one party must consist of only identifiers. We present a more detailed description of concrete situations that motivate us to consider PIW-Sum in Sec. 1.1.

In PIW-Sum, both parties have a set of pairs of an identifier and an integer vector, $\{(v_i, \mathbf{x}_i)\}$ and $\{(w_i, \mathbf{y}_i = (y_{i,1}, \ldots, y_{i,n}))\}$, and the goal is to compute the weighted sum $\{\sum_{(i,j):v_i=w_j} y_{j,\ell}\mathbf{x}_i\}_{\ell\in[n]}$ with

[1] Actually, since this functionality outputs the cardinality of the intersection additionally to weighted sum, Private Intersection-Weighted-Sum with Cardinality would be a more precise terminology, but we omit "with Cardinality" since the term is long.

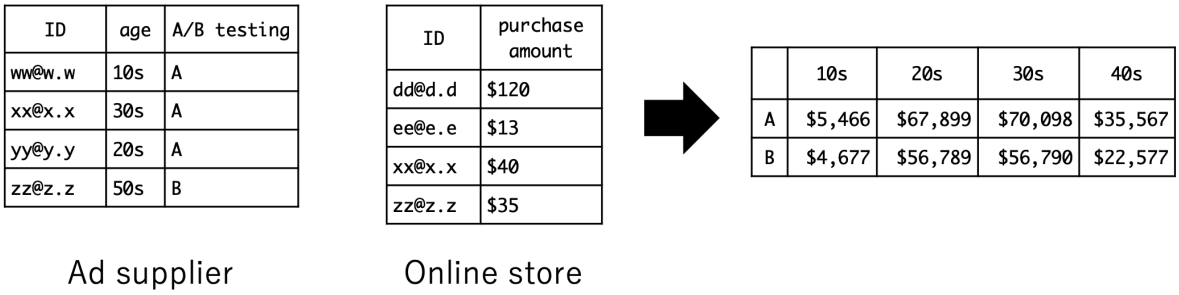

Fig 1: Cross tabulation for A/B testing.

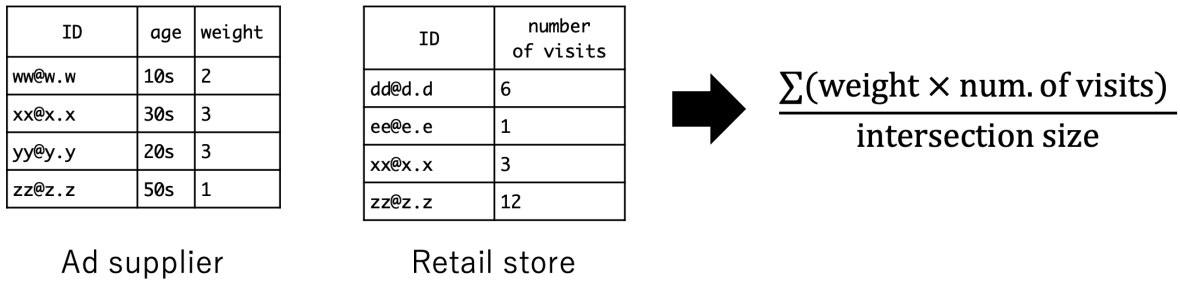

Fig 2: Weighted advertising conversion measurement.

respect to the intersection of identifier sets and the cardinality $|\{v_i\} \cap \{w_i\}|$ of the intersection. We can see $y_{j,\ell}$ as a "weight" and that PI-Sum is the special case of PIW-Sum, where both $\mathbf{x}_i$ and $\mathbf{y}_i$ are one dimensional vectors and $\mathbf{y}_i = 1$ for all i. Alternatively, we can capture PIW-Sum as follows: both parties have integer matrices $\mathbf{X}$ and $\mathbf{Y}$, respectively, where each row of the matrices is associated with an identifier. Let $I = (i_1, \ldots, i_n)$ be the intersection of the identifier sets. The goal is to compute the matrix $\widehat{\mathbf{Y}}^\top \widehat{\mathbf{X}}$ together with the cardinality $|I|$, where the j-th rows of $\widehat{\mathbf{X}}$ and $\widehat{\mathbf{Y}}$ are the rows of $\mathbf{X}$ and $\mathbf{Y}$ with identifier i_j, respectively.

1.1 Applications of PIW-Sum.

PIW-Sum could be applied for various situations where two parties want to perform joint analysis over the database generated by equijoin of their own databases. We give two typical scenarios that we can solve by PIW-Sum.

Case 1. Cross tabulation for A/B testing. The first case is described in Fig 1. An ad supplier delivered two types of advertisement on an online store. The ad supplier wants to know which ad was more effective for the sales and the difference of their effectiveness by age to accumulate knowledge. More precisely, it wants to compute the cross tabulation for the total purchase amounts by age and received ad type (the right table of Fig 1). It knows users' age and which ad was delivered to each user, but only the company that runs the online store knows the purchase amount of each user. When both parties are unwilling to reveal their databases except the cross tabulation, they can compute it using a secure PIW-Sum protocol as follows.

The company sets its input as a set of pairs of an identifier v_i and a purchase amount x_i. On the other hand, the ad supplier sets its input as a set of pairs of an identifier w_i and a one-hot vector $\mathbf{y}_i \in \{0,1\}^{2d}$ where d is the number of choices in age and $y_{i,\ell} = 1$ if the age and the delivered ad of user w_i corresponds to the ℓ-th cell of the cross tabulation. Then, the output $\sum_{(i,j):v_i=w_j} y_{j,\ell} x_i$ for $\ell \in [2d]$ corresponds to the value in the ℓ-th cell of the cross tabulation. Note that in this case, the cardinality of the intersection will become a leakage since it is not used to compute the cross tabulation.

Case 2. Weighted advertising conversion measurement. The second case is described in Fig 2. An ad supplier delivered an advertisement on a campaign in a retail store to encourage its users to visit

the store. Later the company that runs the store wants to know the effectiveness of the advertisement per person. The store is going to strengthen the line up of products for people in their 20s and 30s, and they will multiply the number of visits by some weight according to his or her age for the measurement. In this situation, what the company wants to know is the weighted mean of the number of visits with respect to the people who received the advertisement. Additionally, we assume that only the ad supplier knows the information on age of users. We can deal with this situation via PIW-Sum just by setting the ad supplier's input as pairs of an identifier and a weight according to his or her age while the company's input as pairs of an identifier and a number of visits to the store.

More information on real-world application. We notice that Ion *et al.* gave quite detailed analysis on deploying a PI-Sum protocol for business applications [21]. Since most of their analysis is applicable to our PIW-Sum case, [21] is a great reference for readers who are interested in more practical discussions.

1.2 Our Contributions

Our contributions in this work are two-folds. First, we propose an efficient semi-honestly secure PIW-Sum protocol based on the decisional Diffie-Hellman (DDH) assumption and additively homomorphic encryption (AHE). Our goal is to construct a protocol that has similar efficiency to the DDH-based PI-Sum protocol proposed in [21]. In contrast to PSI where the most efficient protocols in typical environments are based on random oblivious transfer (OT) [8, 38], a traditional DDH-based double masking technique [25] would be the best approach for PI-Sum [21]. This is because the random OT-based PI-Sum protocol needs a shuffle of ciphertexts of (slotted) AHE scheme, which uses expensive fully homomorphic operations of the AHE scheme. Another approach for PI-Sum is circuit-based PSI [19], which is a technique to utilize general two-party computation (garbled circuits [40] or the GMW protocol [18]) to compute PSI-related functions. Garbled-circuit-based PSI protocols could be more computationally efficient than the DDH-based protocol, but the communication cost of the garbled-circuit-based approach is 20 times more than that of the DDH-based protocol [33]. GMW-based PSI protocols with silent OT can be more communication efficient than garbled-circuit-based PSI protocols [5,38], but they need online communication per evaluating an AND gate of circuits and thus are sensitive to network latency.

As discussed in [21], the first priority in business-to-business batch computation would be communication efficiency. In a nutshell, the cost of resources such as CPU, RAM, and especially network are more relevant than the total running time in such batch computation. Increasing the computational resource in the company is much easier than increasing the bandwidth for the communication. Thus the communication efficiency does matter. Another evidence is that when running protocols in a cloud such as Google Cloud Platform, Amazon Web service, and Microsoft Azure, the monetary cost for communication is often much more relevant than that for computation. Since we assume that many applications of PIW-Sum are such business-to-business computation in the WAN setting, where high network latency might exist, we choose the DDH-based approach for our PIW-Sum protocol.

The essential requirement to achieve an efficient PIW-Sum protocol is non-use of expensive fully homomorphic operations including shuffles of AHE ciphertexts, which are not used in the DDH-based PI-Sum protocol. One may think that why not to use fully homomorphism and ciphertext shuffles is a challenge in constructing a PIW-Sum protocol while the DDH-based PI-Sum protocol does not use them. Intuitively, parties would use AHE to compute a weighted sum obliviously to integers to be summed up. However, when a party homomorphically evaluates the weighted sum, the party should be oblivious to even its own integer to multiply as coefficient or weight since otherwise it learns that the corresponding identifier belongs to the intersection. This problem does not occur in PI-Sum since the coefficients are always one. To solve this problem without fully homomorphic encryption is completely nontrivial. We discuss our solution in Sec. 1.3.

The second contribution is the implementation and evaluation of our protocol. We implement our protocol together with the most efficient PI-Sum protocol in [21] and compare their performance in the same environment. We use ideal lattice-based fully homomorphic encryption scheme for an AHE

scheme that allows slot encryption [1]. This is the technique to encrypt many integers to a single ciphertext while we can still perform homomorphic addition over any encrypted integers. This allows us to significantly save the communication cost in the protocol.

Our implementation result shows that the computation cost and communication cost of our protocol are only about 2 times greater than the most efficient PI-Sum protocol when we use tables where each record has one integer element. The estimated monetary cost to run our protocol in a typical cloud environment[2] is 2.16 cents when both identifier sets are equivalent and their size is 2^{20}. We also evaluate the performance of our protocol by changing the vector length of $\mathbf{x}_i$ and $\mathbf{y}_i$ in each record. We find that the computational cost increases by only less than 5% while the communication cost increases by about 100% when we compare the 16×16 case with the 1×1 case (note that $n \times n$ means that the vector length in each record for both parties are n). An important feature of our protocol is that both computation and communication costs are maximized when the two identifier sets are disjoint. The monetary cost in the worst case for 2^{20} sized tables, i.e., two identifier sets are disjoint, is 2.62 cents, which is about 1.2 times greater than that in the most efficient case, i.e., both identifier sets are equivalent. These results show that our protocol is practical even for more than 2^{20}-sized tables.

1.3 Technical Overview

We describe a brief overview of our PIW-Sum protocol. Our starting point is the DDH-based PI-Sum protocol by [21]. Let us briefly recall their protocol. In the setup, two parties, say Alice and Bob, agree on a DDH group $\mathbb{G}$ (a cyclic group where the DDH assumption holds) of order p, its generator g, and a hash function $H : \{0,1\}^* \to \mathbb{G}$ modeled as a random oracle. Alice chooses a random $\mathbb{Z}_p$ element a and a key pair $(\mathsf{pk}, \mathsf{sk})$ of an AHE scheme. Bob chooses a random $\mathbb{Z}_p$ element b. Let m_1 and m_2 be the numbers of records owned by Alice and Bob, respectively. In round 1, on input $\{w_i\}_{i\in[m_2]}$, Bob sends $\{H(w_i)^b\}_i$ to Alice. In round 2, on input $\{(v_i, x_i)\}_{i\in[m_1]}$ and the message from Bob, Alice sends $\{(H(v_i)^a, \mathsf{ct}_i)\}_i$ and shuffled $\{H(w_i)^{ab}\}_i$ to Bob, where ct_i is the AHE ciphertext of x_i. Then, Bob sums up ct_i for all i such that $\exists j, H(v_i)^{ab} = H(w_j)^{ab}$ and sends it to Alice in round 3, and finally Alice obtains the intersection sum by decrypting it.

Let us turn to the PIW-Sum case where Bob's input is $\{(w_i, y_i)\}_i$ instead of $\{w_i\}_i$.[3] Their protocol cannot be applied for PIW-Sum since Bob cannot know the correspondence between the shuffled $\{H(w_i)^{ab}\}_i$ and $\{y_i\}_i$, while Bob has to sum up $y_j\mathsf{ct}_i$ for all $(i,j) : H(v_i)^{ab} = H(w_j)^{ab}$ to compute the intersection weighted sum.

Join of encrypted table and plain table. We solve this problem by joining an encrypted table to a plain table. The goal of this technique is to allow Bob to have $\{\mathsf{ct}_j\}_{j\in[m_2]}$ with the correspondence of w_j and ct_j where ct_j is the ciphertext of x_i if there exists i such that $v_i = w_j$, and the ciphertext of 0 otherwise. If Bob can obtain them, he can homomorphically compute the encryption of the intersection weighted sum $\sum_{(i,j):v_i=w_j} y_j x_i$ by summing up $y_j\mathsf{ct}_j$ for all $j \in [m_2]$. The point is that Bob can perform this computation without knowing whether each w_j is in the intersection or not.

New variant of OPRF. The next challenge is how to compute $\{\mathsf{ct}_j\}_j$ with the correspondence of w_j and ct_j. We solve this by introducing a new variant of oblivious pseudorandom function (OPRF). Recall that OPRF is a two-party protocol of functionality $(\perp, \{z_i\}_i) \to (k, \{f(k, z_i)\}_i)$ where k is a PRF key and f is a PRF [16]. Note that this means that in a OPRF protocol, Alice/Bob's inputs are $\perp/\{z_i\}_i$ and Alice/Bob's outputs are $k/\{f(k, z_i)\}_i$, respectively. Let $\{u_i\}_{i\in[m_2]}$ be public strings out of the identifier space. The functionality of our new variant of OPRF is defined as

$$\phi : (\{v_i\}_{i\in[m_1]}, \{w_i\}_{i\in[m_2]}) \to ((k, S), \{f(k, z_i)\}_{i\in[m_2]}) \tag{1.1}$$

where k is a PRF key, $S = \{j \in [m_2] \mid \forall i, v_i \neq w_j\}$, f is a PRF, $z_i = w_i$ if $i \notin S$, and $z_i = u_i$ if $i \in S$. The differences between OPRF and the variant ϕ are 1) Alice also has input strings $\{v_i\}_i$; 2)

[2] We use the monetary cost for Google Cloud Platform shown in **??** for the estimation.

[3] We consider the one-dimensional vector case here since the n-dimensional vector case is almost the same.

Bob learns the PRF value $f(k, u_i)$ of public string u_i instead of $f(k, w_i)$ if w_i is not included in Alice's input strings; 3) Alice additionally learns the locations S of Bob's input strings that do not match any of Alice's input strings. Note that since S is simulatable from the intersection size, and $f(k, u_i)$ is pseudorandom, they do not leak any additional information on inputs other than PIW-Sum.

If we have a secure protocol for ϕ, we can solve the problem as follows. Alice and Bob run a protocol for ϕ, and then Alice sends $\{(f(k, t_i), \mathsf{ct}_i)\}_{i\in[m_1+|S|]}$ to Bob in a shuffled order, where $t_i = v_i$ for $i \in [m_1]$, $t_{i+m_1} = u_{s_i}$ for $i \in [|S|]$ (s_i is the i-th element of S), ct_i is the AHE ciphertext of x_i for $i \in [m_1]$, and ct_{i+m_1} is the AHE ciphertext of 0 for $i \in [|S|]$. By comparing $f(k, t_i)$ with $f(k, z_i)$, Bob can obtain $\{\mathsf{ct}_j\}_{j\in[m_2]}$ with the correspondence of w_j and ct_j where ct_j is the ciphertext of x_i if there exists i such that $v_i = w_j$, and the ciphertext of 0 otherwise.

How to instantiate the new variant of OPRF. We can construct a three-round protocol for ϕ from a DDH group as follows. In the setup, Alice chooses two random $\mathbb{Z}_p$ elements a_1, a_2 while Bob chooses a random $\mathbb{Z}_p$ element b. First, Alice sends $\{H(v_i)^{a_1}\}_i$ to Bob. Next, Bob computes $\{H(v_i)^{a_1 b}\}_i$, $\{H(w_i)^b\}_i$, $\{H(u_i)^b\}_i$, shuffles $\{H(v_i)^{a_1 b}\}_i$, and sends them to Alice. Then, Alice computes $\{H(v_i)^b\}_i$ by $\{H(v_i)^{a_1 b}\}_i$ to the power of $1/a_1$, sets $r_j = H(w_j)^{a_2 b}$ if $\exists i, H(v_i)^b = H(w_j)^b$ and $r_j = H(u_j)^{a_2 b}$ otherwise for all $j \in [m_2]$, and sends $\{r_i\}_{i\in[m_2]}$ to Bob. Finally, Bob outputs $\{r_j^{1/b}\}_j$. In this protocol, we can observe that $k = a_2$ is the PRF key, and $f(k, z) = H(z)^{a_2}$.

2 Preliminaries

2.1 Notations

For a prime p, $\mathbb{Z}_p$ denotes $\mathbb{Z}/p\mathbb{Z}$. For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \ldots, n\}$. For a set S, $s \leftarrow S$ means that s is uniformly chosen from S. A function $f : \mathbb{N} \to \mathbb{R}$ is called negligible if $f(\lambda) = \lambda^{-\omega(1)}$. For families of distributions $X := \{X_\lambda\}_{\lambda\in\mathbb{N}}$ and $Y := \{Y_\lambda\}_{\lambda\in\mathbb{N}}$, we say X and Y are computationally indistinguishable, denoted by $X \approx_c Y$, if for all probabilistic polynomial-time (PPT) adversaries $\mathcal{A}$, there exists a negligible function negl and we have $|\Pr[1 \leftarrow \mathcal{A}(1^\lambda, X_\lambda)] - \Pr[1 \leftarrow \mathcal{A}(1^\lambda, Y_\lambda)]| \le \mathsf{negl}(\lambda)$.

2.2 Basic Tools and Assumption

Definition 2.1 (DDH assumption). Let $\mathbb{G}_\lambda$ be a family of cyclic groups indexed by $\lambda \in \mathbb{N}$, the order and generator of which are p_λ and g_λ, respectively. We omit λ from the parameters in what follows. We say that the DDH assumption holds in $\mathbb{G}$ if $(g, g^\alpha, g^\beta, g^{\alpha\beta}) \approx_c (g, g^\alpha, g^\beta, g^\gamma)$ where $\alpha, \beta, \gamma \leftarrow \mathbb{Z}_p$. It is well known that if the DDH assumption holds in $\mathbb{G}$, we have $(g, g^\alpha, g^{\boldsymbol{\beta}}, g^{\alpha\boldsymbol{\beta}}) \approx_c (g, g^\alpha, g^{\boldsymbol{\beta}}, g^{\boldsymbol{\gamma}})$ where $\alpha \leftarrow \mathbb{Z}_p$ and $\boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow \mathbb{Z}_p^n$ for all $n \in \mathbb{N}$.

Definition 2.2 (Additively homomorphic encryption). An additively homomorphic (AHE) encryption scheme is a public key encryption scheme that has additive homomorphism. It consists of four algorithms:

$\mathsf{AGen}(1^\lambda)$ **::** It takes a security parameter 1^λ and outputs a public key and a secret key $(\mathsf{pk}, \mathsf{sk})$. The public key specifies a plaintext space $\mathcal{M}$ and a ciphertext space $\mathcal{C}$ that are additive abelian groups.
$\mathsf{AEnc}(\mathsf{pk}, m)$ **::** It takes pk a plaintext $m \in \mathcal{M}$ and outputs a ciphertext $\mathsf{ct} \in \mathcal{C}$ for m.
$\mathsf{ARef}(\mathsf{pk}, \mathsf{ct})$ **::** It takes pk, ct and outputs a ciphertext $\mathsf{ct}' \in \mathcal{C}$.
$\mathsf{ADec}(\mathsf{sk}, \mathsf{ct})$ **::** It takes sk and ct and outputs a decryption value d.

Refreshability. The scheme is refreshable if for all $n, \lambda \in \mathbb{N}, m_1, \ldots, m_n \in \mathcal{M}$ and valid pk, the two distributions are statistically close:

$$\left\{ \{\mathsf{ct}_i\}_i, \mathsf{ARef}(\mathsf{pk}, \sum_{i\in[n]} \mathsf{ct}_i) \right\}, \left\{ \{\mathsf{ct}_i\}_i, \mathsf{AEnc}(\mathsf{pk}, \sum_{i\in[n]} m_i) \right\}$$

where $\mathsf{ct}_i \leftarrow \mathsf{AEnc}(\mathsf{pk}, m_i)$.

Correctness. The scheme is correct if there exists a negligible function negl, and the following holds for all $n, \lambda \in \mathbb{N}, m_1, \ldots, m_n \in \mathcal{M}$:

$$\Pr\left[\mathsf{ADec}(\mathsf{sk}, \mathsf{ct}) = \sum_{i\in[n]} m_i \;\middle|\; \begin{array}{l} \mathsf{pk}, \mathsf{sk} \leftarrow \mathsf{AGen}(1^\lambda) \\ \mathsf{ct}_i \leftarrow \mathsf{AEnc}(\mathsf{pk}, m_i) \\ \mathsf{ct} = \sum_{i\in[n]} \mathsf{ct}_i \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Security. We say the scheme is IND-CPA secure if there exists a negligible function negl, and the following holds for all stateful PPT adversaries $\mathcal{A}$ and $\lambda \in \mathbb{N}$:

$$\left|\Pr\left[\beta \leftarrow \mathcal{A}(\mathsf{ct}) \;\middle|\; \begin{array}{l} \mathsf{pk}, \mathsf{sk} \leftarrow \mathsf{AGen}(1^\lambda) \\ \beta \leftarrow \{0,1\} \\ m_0, m_1 \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk}) \\ \mathsf{ct} \leftarrow \mathsf{AEnc}(\mathsf{pk}, m_\beta) \end{array}\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda).$$

Vector encryption. A vector $\mathbf{m} \in \mathcal{M}^n$ can be encrypted by just element-wise encryption. We denote this by $\mathsf{AEnc}(\mathsf{pk}, \mathbf{m})$. It is not hard to see that additive homomorphism similarly holds as addition of vectors. Note that this is a different notion from slot encryption that we explain in Sec. 4.1.

2.3 Private Intersection-Weighted-Sum

Private Intersection-Weighted-Sum (PIW-Sum) is a special case of two-party computation. Two-party computation between parties P_1 and P_2 is a protocol where they jointly compute some function $f = (f_1, f_2)$ over their inputs x and y. In the input phase, P_1 and P_2 prepare their inputs x and y, respectively. Then, they interactively compute $f(x, y) = (f_1(x, y), f_2(x, y))$ where P_1 wants to obtain $f_1(x, y)$ while P_2 wants to obtain $f_2(x, y)$ as output. The primary requirement for *secure* two-party computation is that P_1 (resp. P_2) leans only $f_1(x, y)$ (resp. $f_2(x, y)$) and nothing else about the input of the communication partner after the computation.

Basically, there are two types of security model for two-party computation, namely, semi-honest security and malicious security. Semi-honest security 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. On the other hand, malicious security requires that a party can never learn any information about the other party's input even if it deviates from the protocol. In this paper, we consider only semi-honest security.

Definition 2.3 (Semi-honest security). Let Π be a two-party protocol computing $f = (f_1, f_2)$. Let $\mathsf{view}_i^\Pi(x, y)$ be the view of P_i (entire distribution that P_i can see) and $\mathsf{out}^\Pi(x, y) = (\mathsf{out}_1^\Pi(x, y), \mathsf{out}_2^\Pi(x, y))$ 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 $\mathcal{S}_1, \mathcal{S}_2$, and the following holds for all inputs x, y:

$$(\mathsf{view}_1^\Pi(x, y), \mathsf{out}^\Pi(x, y)) \approx_c (\mathcal{S}_1(1^\lambda, x, f_1(x, y)), f(x, y))$$
$$(\mathsf{view}_2^\Pi(x, y), \mathsf{out}^\Pi(x, y)) \approx_c (\mathcal{S}_2(1^\lambda, y, f_2(x, y)), f(x, y)).$$

The goal of this paper is to present an efficient semi-honestly secure Private Intersection-Weighted-Sum protocol, which computes the following function.

Definition 2.4 (Private Intersection-Weighted-Sum). This is secure two-party computation where inputs and outputs for two parties are defined as in Fig 3. Initially, P_1 and P_2 have a set of pairs of an identifier and an integer vector $\{v_i, \mathbf{x}_i\}_{i\in[m_1]}$ and $\{w_i, \mathbf{y}_i\}_{i\in[m_2]}$, respectively, where v_i, w_i are identities belonging to an identifier space $\mathcal{I}$ and $\mathbf{x}_i \in \mathbb{Z}^{n_1}, \mathbf{y}_i \in \mathbb{Z}^{n_2}$ are integer vectors. Both parties know the

table size of the communication partner. The goal is to allow P_1 to learn Private Intersection-Weighted-Sum $\{\sum_{(i,j):v_i=w_j} y_{j,\eta}\mathbf{x}_i\}_{\eta\in[n_2]}$ and the cardinality of the intersection of both identifier sets $\{v_i\}_i$ and $\{w_i\}_i$ while P_2 to learn nothing. Our protocol additionally allows P_2 to learn the cardinality of the intersection, which is captured by the output for P_2. Note that this leakage for P_2 also occurs in the PI-Sum protocols in [21].

Revealing the Intersection Size in PIW-Sum. Our protocol leaks the cardinality of the intersection for both parties. In applications such as Case 2 in Sec. 1.1, the cardinality is beneficial to normalize the weighted sum to compare each other. However, it becomes sometimes unnecessary leakage in applications such as Case 1 in Sec. 1.1. Additionally, when P_2 should not learn anything in the computation, the leakage for P_2 is not desirable.

- Input for both parties: m_1, m_2, n_1, n_2.
- Input for P_1 : $\{v_i, \mathbf{x}_i\}_{i\in[m_1]}$ where $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,n_1})$.
- Input for P_2 : $\{w_i, \mathbf{y}_i\}_{i\in[m_2]}$ where $\mathbf{y}_i = (y_{i,1}, \ldots, y_{i,n_2})$.
- Output for P_1 : $\{\sum_{(i,j):v_i=w_j} y_{j,\eta}\mathbf{x}_i\}_{\eta\in[n_2]}$, $|\{v_i\}_i \cap \{w_j\}_j|$.
- Output for P_2 : $|\{v_i\}_i \cap \{w_j\}_j|$.

Fig 3: Private Intersection-Weighted-Sum

3 Our PIW-Sum Protocol

We present our PIW-Sum protocol in Fig 4. As explained in Sec. 1.3, a building block of our protocol can be seen as a variant ϕ of OPRF (Eq. (1.1)). However, we describe our protocol without the OPRF abstraction and explicitly describe the entire protocol. This is mainly because some protocol message not related to the OPRF protocol is sent in parallel with an OPRF message to reduce the number of rounds, and thus the explicit description is more convenient. Note that $\mathbf{c}$ in Round 1, $\mathbf{d}, \mathbf{e}, \mathbf{f}$ in Round 2, $\mathbf{h}$ in Round 3 are the messages related the OPRF variant ϕ (the description of ϕ is also described in Sec. 1.3).

In the setup, P_1 and P_2 chooses random elements (a_1, a_2) and b, respectively, which is used to run the variant ϕ of OPRF. Additionally, P_1 chooses a key pair of an AHE scheme. They also shuffle their inputs since the order of the records might reveal information of their inputs. Let $\{(v_i, \mathbf{x}_i)\}_i$ and $\{(w_i, \mathbf{y}_i)\}_i$ be P_1 and P_2's input after the shuffle, respectively.

Round 1 to 3 are used to run ϕ as described in Sec. 1.3, where P_1's input is $\{v_i\}_i$, and P_2's input is $\{w_i\}_i$. Additionally, P_1 sends a set of pairs of PRF values and AHE ciphertexts of its input or $\mathbf{0}$ under the key generated in the setup ((2) and (3) in round 3). In (1) of round 4, P_2 makes the correspondence between ciphertexts sent by P_1 and its input $\mathbf{y}_i$ by comparing the PRF values sent by P_1 and those obtained via the execution of ϕ. Then, in (2) of round 4, P_2 homomorphically evaluates PIW-Sum over the ciphertexts using the correspondence. That is, since the ciphertexts that correspond to the identifiers out of the intersection are encryption of $\mathbf{0}$, P_2 can compute the wighted sum obliviously to whether each ciphertext belongs to the intersection. In the final round, P_1 decrypts the ciphertexts and obtains PIW-Sum. The intersection size for P_1 can be obtained from the size of S that is computed in round 3. On the other hand, P_2 can obtain the intersection size by $m_2 - ((m_1 + m_3) - m_1)$ where m_1 and m_2 are public, and m_3 is obtained from the number of ciphertexts sent by P_1 in round 3.

3.1 Security

The proposed protocol Π has semi-honest security. This can be stated by the following theorem.

– Inputs:
  - Both parties: A cyclic group $\mathbb{G}$ of prime order p, its generator g, an identifier space $\mathcal{I}$, a set $\{u_i\}_{i\in[m_2]}$ such that $u_i \notin \mathcal{I}$, a hash function $H : \{0,1\}^* \to \mathbb{G}$ modeled as a random oracle, and table-size parameters m_1, m_2, n_1, n_2.
  - P_1: A set of pairs $\{(v'_i, \mathbf{x}'_i)\}_{i\in[m_1]}$ where $v'_i \in \mathcal{I}, \mathbf{x}'_i = (x'_{i,1}, \ldots, x'_{i,n_1}) \in \mathbb{Z}^{n_1}$.
  - P_2: A set of pairs $\{(w'_i, \mathbf{y}'_i)\}_{i\in[m_2]}$ where $w'_i \in \mathcal{I}, \mathbf{y}'_i = (y'_{i,1}, \ldots, y'_{i,n_2}) \in \mathbb{Z}^{n_2}$.
– Setup:
  - P_1 chooses $a_1, a_2 \leftarrow \mathbb{Z}_p$ and a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{AGen}(1^\lambda)$ of an additive homomorphic encryption scheme.
  - P_2 chooses $b \leftarrow \mathbb{Z}_p$.
  - P_1 and P_2 choose random permutations π_1 in $[m_1]$ and π_2 in $[m_2]$, and set $(v_i, \mathbf{x}_i) = (v'_{\pi_1(i)}, \mathbf{x}'_{\pi_1(i)})$ for $i \in [m_1]$ and $(w_i, \mathbf{y}_i) = (w'_{\pi_2(i)}, \mathbf{y}'_{\pi_2(i)})$ for $i \in [m_2]$, respectively.
– Round 1:
  1. P_1 computes $\mathbf{c} = (c_1, \ldots, c_{m_1})$ where $c_i = H(v_i)^{a_1}$ and sends $(\mathsf{pk}, \mathbf{c})$ to P_2.
– Round 2:
  1. P_2 chooses a random permutation π_3 in $[m_1]$ and computes $\mathbf{d} = (d_1, \ldots, d_{m_1}) = (c^b_{\pi_3(1)}, \ldots, c^b_{\pi_3(m_1)})$.
  2. P_2 computes $\mathbf{e} = (e_1, \ldots, e_{m_2}), \mathbf{f} = (f_1, \ldots, f_{m_2})$ where $e_i = H(w_i)^b, f_i = H(u_i)^b$ and sends $(\mathbf{d}, \mathbf{e}, \mathbf{f})$ to P_1.
– Round 3:
  1. P_1 computes d_j^{1/a_1} for all $j \in [m_1]$. Let $S = \{i \in [m_2] \mid \forall j \in [m_1], e_i \neq d_j^{1/a_1}\}$. P_1 computes $\mathbf{h} = (h_1, \ldots, h_{m_2})$ where $h_i = e_i^{a_2}$ if $i \notin S$ and $h_i = f_i^{a_2}$ otherwise.
  2. Let $m_3 = |S|$. P_1 chooses a random permutation π_4 in $[m_1 + m_3]$ and computes $(k'_1, \ldots, k'_{m_1+m_3}) = (H(v_1)^{a_2}, \ldots, H(v_{m_1})^{a_2}, \{H(u_i)^{a_2}\}_{i\in S})$ and $(\ell'_1, \ldots, \ell'_{m_1+m_3}) \leftarrow (\mathsf{AEnc}(\mathsf{pk}, \mathbf{x}_1), \ldots, \mathsf{AEnc}(\mathsf{pk}, \mathbf{x}_{m_1}), \mathsf{AEnc}(\mathsf{pk}, \mathbf{0}), \ldots, \mathsf{AEnc}(\mathsf{pk}, \mathbf{0}))$.
  3. Let $\mathbf{k} = (k_1, \ldots, k_{m_1+m_3}) = (k'_{\pi_4(1)}, \ldots, k'_{\pi_4(m_1+m_3)}), \boldsymbol{\ell} = (\ell_1, \ldots, \ell_{m_1+m_3}) = (\ell'_{\pi_4(1)}, \ldots, \ell'_{\pi_4(m_1+m_3)})$. P_1 sends $(\mathbf{h}, \mathbf{k}, \boldsymbol{\ell})$ to P_2.
– Round 4:
  1. P_2 computes $h_i^{1/b}$ for all $i \in [m_2]$ and sets $r_i = \ell_j$ for all $i \in [m_2]$ where $j \in [m_1 + m_3]$ is the index such that $h_i^{1/b} = k_j$.
  2. P_2 computes $z_j = \mathsf{ARef}(\mathsf{pk}, \sum_{i\in[m_2]} y_{i,j} r_i)$ for all $j \in [n_2]$ and sends $\{z_j\}_{j\in[n_2]}$ to P_1.
– Output:
  - P_1 outputs $\mathsf{ADec}(\mathsf{sk}, z_j)$ for all $j \in [n_2]$ and $|S|$ while P_2 outputs $m_2 - ((m_1 + m_3) - m_1)$.

Fig 4: Our Private Intersection-Weighted-Sum Protocol

Theorem 3.1. *Assume that H is a hash function modeled as a random oracle and the DDH assumption holds in $\mathbb{G}$, then there exist simulators $\mathcal{S}_1, \mathcal{S}_2$ in the random oracle model such that*

$$(\mathsf{view}_1^{\Pi}(x,y), \mathsf{out}^{\Pi}(x,y)) \approx_c (\mathcal{S}_1(1^\lambda, x, f_1(x,y)), f(x,y))$$
$$(\mathsf{view}_2^{\Pi}(x,y), \mathsf{out}^{\Pi}(x,y)) \approx_c (\mathcal{S}_2(1^\lambda, y, f_2(x,y)), f(x,y))$$

where

$$\mathsf{cmn} = (\mathbb{G}, m_1, m_2, n_1, n_2), x = (\mathsf{cmn}, \{(v_i, \mathbf{x}_i)\}_{i\in[m_1]}), y = (\mathsf{cmn}, \{(w_i, \mathbf{y}_i)\}_{i\in[m_2]})$$
$$f_1(x,y) = (\{\sum_{(i,j):v_i=w_j} y_{j,\eta}\mathbf{x}_i\}_{\eta\in[n_2]},\ \ |\{v_i\}_i \cap \{w_j\}_j|),\ f_2(x,y) = |\{v_i\}_i \cap \{w_j\}_j|.$$

Proof. It is not difficult to see that we always have $\mathsf{out}^{\Pi}(x,y) = f(x,y)$ with overwhelming probability due to the correctness of the AHE scheme. Thus, we can prove the theorem directly from Lemmata 3.1 and 3.2. □

Security against corrupt P_1. We construct $\mathcal{S}_1$ as follows.

1. $\mathcal{S}_1$ chooses a random tape rt and simulates honest P_1 on inputs $(x; \mathsf{rt})$ to obtain pk and a_1.

2. Let $m_4 = m_1 - |\{v_i\}_i \cap \{w_j\}_j|$. For round 2, $\mathcal{S}_1$ chooses $g_1, \ldots, g_{m_2+m_4} \leftarrow \mathbb{G}$, $\mathbf{f} \leftarrow \mathbb{G}^{m_2}$ and random permutations ρ_1 in $[m_1]$ and ρ_2 in $\{1+m_4, \ldots, m_2+m_4\}$. Then, it sets $\mathbf{d} = (g^{a_1}_{\rho_1(1)}, \ldots, g^{a_1}_{\rho_1(m_1)})$, $\mathbf{e} = (g_{\rho_2(1+m_4)}, \ldots, g_{\rho_2(m_2+m_4)})$.
3. For round 4, $\mathcal{S}_1$ computes $z_\eta \leftarrow \mathsf{AEnc}(\mathsf{pk}, \sum_{(i,j):v_i=w_j} y_{j,\eta}\mathbf{x}_i)$ for all $\eta \in [n_2]$.
4. Finally, $\mathcal{S}_1$ outputs $(x, \mathsf{rt}, \mathbf{d}, \mathbf{e}, \mathbf{f}, \{z_\eta\}_\eta)$.

Lemma 3.1. $\mathsf{view}^{\Pi}_1(x, y) \approx_c \mathcal{S}_1(1^\lambda, x, f_1(x, y))$.

Proof. Consider the following multi-step hybrid argument.

Hyb_0**:** The view $\mathsf{view}^{\Pi}_1(x, y)$ in a real execution of Π.

Hyb_1**:** The same as Hyb_0 except that, in round 2 of Π, $\mathcal{S}_1$ chooses fresh group elements $g_1, \ldots, g_{2m_2+m_4} \leftarrow \mathbb{G}$ and replaces $\mathbf{d}, \mathbf{e}$ and $\mathbf{f}$ with $(g^{a_1}_{\rho_1(1)}, \ldots, g^{a_1}_{\rho_1(m_1)}), (g_{\rho_2(1+m_4)}, \ldots, g_{\rho_2(m_2+m_4)})$ and $(g_{1+m_2+m_4}, \ldots, g_{2m_2+m_4})$ respectively, where ρ_1 (resp. ρ_2) is a random permutation in $[m_1]$ (resp. $\{1 + m_4, \ldots, m_2 + m_4\}$).

Hyb_2**:** The same as Hyb_1 except that $\mathcal{S}_1$ generates fresh ciphertexts $\mathsf{AEnc}(\mathsf{pk}, \sum_{(i,j):v_i=w_j} y_{j,\eta}\mathbf{x}_i)$ instead of computing the intersection-weighted-sum in round 4.

Hyb_3**:** The view $\mathcal{S}_1(1^\lambda, x, f_1(x, y))$.

Now, we show that each neighboring pair of the hybrid distributions is computationally indistinguishable. We focus on indistinguishability between Hyb_0 and Hyb_1 below since it can be straightforwardly observed that Hyb_1 and Hyb_2 are indistinguishable according to the refreshability of the AHE scheme, and Hyb_2 and Hyb_3 are identical.

We show that $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_1$ under the DDH assumption as follows. The reduction algorithm taking $(g, g^\alpha, g^{\boldsymbol{\beta}}, g^{\boldsymbol{\delta}} = g^{\alpha\boldsymbol{\beta}}/g^{\boldsymbol{\gamma}})$ sets $b = \alpha$ and programs the random oracle as $\big(H(t_1), \ldots, H(t_{m_2+m_4}), H(u_1), \ldots, H(u_{m_2})\big) = g^{\boldsymbol{\beta}}$ where t_i is the i-th element of $\{v_i\}_i \cup \{w_j\}_j$. We additionally defines $U_1, U_2 \subseteq [m_2 + m_4]$ as the sets such that $\{t_i\}_{i\in U_1} = \{v_i\}_{i\in[m_1]}$ and $\{t_i\}_{i\in U_2} = \{w_i\}_{i\in[m_2]}$, respectively. It also sets $\mathbf{d} = (g^{a_1\delta_{\sigma_1(1)}}, \ldots, g^{a_1\delta_{\sigma_1(m_1)}})$, $\mathbf{e} = (g^{\delta_{\sigma_2(1)}}, \ldots, g^{\delta_{\sigma_2(m_2)}})$, $\mathbf{f} = (g^{\delta_{1+m_2+m_4}}, \ldots, g^{\delta_{2m_2+m_4}})$ where δ_i is the i-th element of $\boldsymbol{\delta}$, $\sigma_1 : [m_1] \to U_1, \sigma_2 : [m_2] \to U_2$ are random bijective functions. Then, the view of P_1 in round 3 corresponds to Hyb_0 if $\boldsymbol{\delta} = \alpha\boldsymbol{\beta}$, and Hyb_1 otherwise. If there exists a PPT adversary identifying Hyb_0 and Hyb_1, it can distinguish the tuple $(g, g^\alpha, g^{\boldsymbol{\beta}}, g^{\alpha\boldsymbol{\beta}})$ from $(g, g^\alpha, g^{\boldsymbol{\beta}}, g^{\boldsymbol{\gamma}})$ straightforwardly.

Consequently, by the DDH assumption, Hyb_0 and Hyb_1 are computationally indistinguishable, and hence $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_3$. □

Security against corrupt P_2. We construct $\mathcal{S}_2$ as follows.

1. $\mathcal{S}_2$ chooses $\mathsf{pk}, \mathsf{sk} \leftarrow \mathsf{AGen}(1^\lambda)$ and a random tape rt, and simulates honest P_2 on inputs $(y; \mathsf{rt})$ to obtain b.
2. For round 1, $\mathcal{S}_2$ chooses $\mathbf{c} \leftarrow \mathbb{G}^{m_1}$.
3. Let $m_3 = m_2 - |\{v_i\}_i \cap \{w_j\}_j|$. For round 3, $\mathcal{S}_2$ chooses $g_1, \ldots, g_{m_1+m_3} \leftarrow \mathbb{G}$ and a random permutation ρ_3 in $[m_1 + m_3]$. Then, $\mathcal{S}_2$ computes $\mathbf{h} = (g^b_1, \ldots, g^b_{m_2}), \mathbf{k} = (g_{\rho_3(1)}, \ldots, g_{\rho_3(m_1+m_3)}), \boldsymbol{\ell} = (\ell_1, \ldots, \ell_{m_1+m_3}) \leftarrow \mathsf{AEnc}(\mathsf{pk}, \mathbf{0})^{m_1+m_3}$.
4. Finally, $\mathcal{S}_2$ outputs $(y, \mathsf{rt}, \mathsf{pk}, \mathbf{c}, \mathbf{h}, \mathbf{k}, \boldsymbol{\ell})$.

Lemma 3.2. $\mathsf{view}^{\Pi}_2(x, y) \approx_c \mathcal{S}_2(1^\lambda, y, f_2(x, y))$.

Proof. Consider the following multi-step hybrid argument.

Hyb_0**:** The view $\mathsf{view}^{\Pi}_2(x, y)$ in a real execution of Π.

Hyb_1**:** The same as Hyb_0 except that $\mathcal{S}_2$ sends a fresh element $\mathbf{c} \leftarrow \mathbb{G}^{m_1}$ instead of $\big(H(v_1)^{a_1}, \ldots, H(v_{m_1})^{a_1}\big)$ to P_2 in round 1.

Hyb_2**:** The same as Hyb_1 except that, in round 3, $\mathcal{S}_2$ chooses fresh elements $g_1, \ldots, g_{m_1+m_3} \leftarrow \mathbb{G}$ and replaces $\mathbf{h}$ and $\mathbf{k}$ with $(g^b_{\rho_3(1)}, \ldots, g^b_{\rho_3(m_2)})$ and $(g_{\rho_4(1)}, \ldots, g_{\rho_4(m_1+m_3)})$ respectively, where ρ_3 (resp. ρ_4) is a random permutation in $[m_2]$ (resp. $[m_1 + m_3]$).

Protocol	Computation Cost		Communication Cost	
	#Group exponentiation	#AHE operations	#Group elements	#AHE ciphertexts
IKN$^+$20 [21]	$2m_1 + 2m_2$	$m_1 + m_2 - m_3 + 1$	$m_1 + 2m_2$	$m_1 + 1$
Ours	$4m_1 + 4m_2 + m_3$	$m_1 + m_2 + m_3 + 1$	$3m_1 + 3m_2 + m_3$	$m_1 + m_3 + 1$

Table 1: Theoretical costs of each protocol. We compare them with respect to the computation and communication costs on the DDH-group and the AHE scheme. AHE operations consists of addition of ciphertexts, encryption, and decryption. Note that we do not consider slot-encrypting optimization in this comparison. The natural numbers m_1, m_2, m_3 denote the number of identifiers for P_1, that for P_2, and that for P_2 out of the intersection, respectively. That is, $m_3 = 0$ if P_1's identifier set includes P_2's identifier set, and $m_3 = m_2$ if they are disjoint.

Input Size	Our Protocol			IKN$^+$20 [21]		
$m_1 = m_2$	Time [ms]	Comm. [MiB]	Monetary [US Cents]	Time [ms]	Comm. [MiB]	Monetary [US Cents]
2^{12}	1553.2	1.12	0.00918	804.2	0.576	0.00472
2^{16}	24929.4	16.5	0.136	12873.8	8.73	0.0717
2^{20}	405072.2	262	2.16	205560.8	139	1.14

Table 2: Measurements of computation time, network costs and monetary costs in US cents, in settings where $n_1 = n_2 = 1$, $m_1 = m_2 =$ the intersection size (= the size of $\{v_i\}_i \cap \{w_j\}_j$). In other words, the input size of P_1 and P_2 are equal, and both have same identifiers set (i.e. $\{v_i\}_i = \{w_j\}_j$).

Hyb_3**:** The same as Hyb_2 except that, in round 3, $\mathcal{S}_2$ replaces all elements of $\boldsymbol{\ell}$ with fresh ciphertexts of $\mathbf{0}$, i.e., $\mathsf{AEnc}(\mathsf{pk}, \mathbf{0})$.

Hyb_4**:** The view $\mathcal{S}_2(1^\lambda, y, f_2(x, y))$.

We show that each pair of the above consecutive hybrid distributions is computationally indistinguishable. The indistinguishability between Hyb_0 and Hyb_1 is straightforwardly obtained from the DDH assumption. That is, the reduction algorithm taking $(g, g^\alpha, g^{\boldsymbol{\beta}}, g^{\boldsymbol{\delta}} = g^{\alpha\boldsymbol{\beta}}/g^{\boldsymbol{\gamma}})$ sets $a_1 = \alpha$ and programs the random oracle as $\big(H(v_1), \ldots, H(v_{m_1})\big) = g^{\boldsymbol{\beta}}$. It also sets $\mathbf{c} = g^{\boldsymbol{\delta}}$. Then, the view of P_2 in round 2 corresponds to Hyb_0 if $\boldsymbol{\delta} = \alpha\boldsymbol{\beta}$, and Hyb_1 otherwise. If there is an adversary identifying Hyb_0 and Hyb_1, it can easily break the DDH assumption.

We can also obtain the indistinguishability between Hyb_1 and Hyb_2 from the DDH assumption as follows. The reduction algorithm taking $(g, g^\alpha, g^{\boldsymbol{\beta}}, g^{\boldsymbol{\delta}} = g^{\alpha\boldsymbol{\beta}}/g^{\boldsymbol{\gamma}})$ sets $a_2 = \alpha$ and programs the random oracle as $\big(H(v_1), \ldots, H(v_{m_1}), \{H(u_i)\}_{i\in S}\big) = g^{\boldsymbol{\beta}}$ (recall that $S = \{i \in [m_2] \mid \forall j \in [m_1], e_i \neq d_j^{1/a_1}\}$). It also sets $\mathbf{h} = (g^{b\delta_{\sigma_3(1)}}, \ldots, g^{b\delta_{\sigma_3(m_2)}})$ and $\mathbf{k} = (g^{b\delta_{\rho_3(1)}}, \ldots, g^{b\delta_{\rho_3(m_1+m_3)}})$ where δ_i is the i-th element of $\boldsymbol{\delta}$, $\sigma_3 : [m_2] \to [m_1 + m_3]$ is the injective function such that

$$\sigma_3(i) = \begin{cases} j : w_i = v_j & (i \notin S) \\ m_1 + j : i = s_j & (i \in S) \end{cases},$$

(recall that s_j is the j-th element of S), and ρ_3 is a random permutation in $[m_1 + m_3]$. Then, the view of P_2 in round 4 corresponds to Hyb_1 if $\boldsymbol{\delta} = \alpha\boldsymbol{\beta}$, and Hyb_2 otherwise.

The difference between Hyb_2 and Hyb_3 is computationally indistinguishable by the IND-CPA security of the AHE scheme, and Hyb_3 and Hyb_4 is identical. □

4 Implementation and Evaluation

In this section, we present the measurements for our implementation of our new protocol for Private Intersection-Weighted-Sum.

Input Size (m_1, m_2)	Time [ms]	Comm. [MiB]	Monetary [US Cents]
$(2^{10}, 2^{16})$	16187.0	11.4	0.0935
$(2^{12}, 2^{16})$	16602.2	11.6	0.0952
$(2^{14}, 2^{16})$	18318.2	12.6	0.104
$(2^{16}, 2^{16})$	24929.4	16.5	0.136
$(2^{16}, 2^{10})$	12394.2	6.30	0.0527
$(2^{16}, 2^{12})$	13009.8	9.22	0.0756
$(2^{16}, 2^{14})$	15424.8	10.7	0.0879
$(2^{16}, 2^{16})$	24929.4	16.5	0.136

Table 3: Measurements of computation time, network costs and monetary costs in US cents , in settings where $n_1, n_2 = 1$ but $m_1 \neq m_2$. The one's set of identifiers is included to that of the another, so The intersection size is equals to $\min\{m_1, m_2\}$.

Input Size $m_1 = m_2$	Time [ms]	Comm. [MiB]	Monetary [US Cents]
2^{12}	1751.0	1.30	0.0106
2^{16}	28267.2	20.0	0.164
2^{20}	456435.8	319	2.62

Table 4: Measurements of computation time, network costs and monetary costs in US cents , in settings where $n_1 = n_2 = 1$ and $m_1 = m_2$ but their identifiers sets are disjoint i.e. the intersection size is 0.

4.1 Implementation Details

We implement our protocol together with the DDH-based PI-Sum protocol in [21] (the IKN$^+$20 protocol) for comparison using the following instantiation. For DDH-group $\mathbb{G}$, we use "curve25519" implemented in `libsodium` [2], and for the additively homomorphic encryption scheme, we use the BFV scheme implemented in `SEAL` [1]. Note that the BFV scheme is an ideal lattice-based fully homomorphic encryption scheme, but we use multiplicative homomorphism only in the light slot-shifting operation explained later. We use the BFV scheme with a 54-bit ciphertext modulus (q, called `coeff_modulus` in `SEAL`), a 23-bit plaintext modulus (t, `plain_modulus`), and a polynomial of degree 2048 for the polynomial modulus (N, `poly_modulus`). Note that $(N, \log q) = (2048, 54)$ is one of the default parameters for 128-bit security in `SEAL`. The size of plaintext modulus does not affect the security level.

For a hash function $H : \{0,1\}^* \to \mathbb{G}$ to the elliptic curve, we use the hash function implemented in `libsodium`. For preparing the public strings $\{u_i\}_i$ out of the identifier space, we use suffixes "`_real`"

Input Size $n_1 \times n_2$	Time [ms]	Comm. [MiB]	Monetary [US Cents]
1×1	24929.4	16.5	0.136
4×1	25109.6	19.5	0.160
16×1	25631.8	31.6	0.253
1×1	24929.4	16.5	0.136
1×4	24984.2	16.6	0.137
1×16	24994.8	17.2	0.141
1×1	24929.4	16.5	0.136
4×4	25165.8	19.6	0.160
16×16	25740.8	32.3	0.260

Table 5: Measurements of computation time, network costs and monetary costs in US cents , in settings where $m_1 = m_2 =$ the intersection size $= 2^{16}$ but $n_1, n_2 \neq 1$.

and "`_dummy`". That is, each party can generate the public strings by setting $u_i = i$`_dummy` without interaction while adding suffix `_real` for all identifiers.

We use the slotting optimization for the AHE scheme to reduce the communication cost. In a nutshell, we can encrypt many plaintexts to a singe ciphertext of the BFV scheme. This is possible since a BFV plaintext is a polynomial in the ring $\mathbb{Z}_t[x]/(x^N + 1)$ ($N = 2048$ for our implementation), and $\lfloor N/d \rfloor$ polynomials $\{p_k\}_{k=0,\ldots,\lfloor N/d \rfloor - 1}$ of degree $d - 1$ (consisting of d terms) can be represented by a single BFV plaintext $p = \sum_{k=0}^{\lfloor N/d \rfloor - 1} x^{kd} p_k$. This allows us to encrypt $\lfloor N/d \rfloor$ plaintexts into a single ciphertext. Thanks to the homomorphic property of the BFV scheme, we can shift the slots in ciphertext. That is, we can obtain the ciphertext of $x^{-kd}p$ from the ciphertext of p by homomorphic multiplication. Observe that the coefficients of $x^{-kd}p$ in degree-0 to $d - 1$ terms are the same as those of p_k. Thus, we can perform homomorphic addition over arbitrary plaintexts using the least significant slot. More detailed explanation of the slotting optimization technique can be found in [21, Appendix B].

4.2 Methods of Measurement

We run our implementation on a desktop computer with Intel Core i9-9900K CPU (3.60GHz × 8) and 16 GiB RAM. Each measurement is performed nine times, and we report the average of five measurements excluding two largest and smallest values to exclude outliers.

In our results, computation time includes communication time. However, the parties P_1 and P_2 are implemented as parts of a single program, and their communication is implemented just as copying of memories. Therefore, communication time in computation time is negligible.

We also estimate the monetary costs of the protocols using the cost of Google Cloud Platform (GCP), where the network cost is 0.08 USD/GB, and the computational cost is 0.01 USD/hour. For computational cost, we used the n1-standard cost[4]. For network cost, we use the cost of the cheapest 10+TB category of the Premium Tier pricing[5], which corresponds to the communication cost between the Internet and the cloud. Although computation powers of our computer and GCP are somewhat different, the percentage of the monetary cost on the computation in the total monetary cost is less that 10%, and this estimation would be valid.

4.3 Discussion of Measurements

We present the theoretical costs of our protocol and the IKN$^+$20 protocol with respect to the DDH-group and the AHE scheme Table 1. An important feature of our protocol is that the cost of our protocol is maximized when the identifier sets of P_1 and P_2 are disjoint. This is in contrast to the IKN$^+$20 protocol, the cost of which is maximized when the intersection size is maximum.

Comparison with the IKN$^+$20 protocol. Next, we discuss our measurements. In Table 2 we compare our protocol with the baseline, namely DDH-based intersection-sum protocol [21]. We measure the costs with respect to various input sizes when the identifier sets for both parties are equivalent. Although our protocol provides weighted-intersection-sum, its costs are only about twice as much as those of the baseline.

Benchmark under various input sizes. To see the scalability of our protocol, we run our Private Intersection-Weighted-Sum protocol under various input sizes. Columns 2 to 4 in Table 2 show the costs when m_1 and m_2 are equal. As expected from Table 1, both computation time and network costs are approximately proportional to the input size m_1. Table 3 shows the costs when m_1 and m_2 may not be equal. Comparing the cases when $(m_1, m_2) = (2^{10}, 2^{16})$ and $(m_1, m_2) = (2^{16}, 2^{10})$, the former is about 1.3 times as large as the latter in both time and communication. This is consistent with the relationship between m_1, m_2 and the costs predicted from Table 1. This also shows that when the

[4] https://cloud.google.com/compute/all-pricing
[5] https://cloud.google.com/vpc/network-pricing

input sizes of two parties are different, the party who has a larger input should take the role of P_1 if both parties are allowed to learn the weighted intersection sum.

Next, we measure the costs when the two identifier sets are disjoint, which is the worst case for our protocol. Table 4 shows the costs. Compared to the costs in Table 2, the computation time is at most 1.14 times and the network cost is at most 1.22 times for all input sizes. Even in the worst case, our protocol can run at a practical cost of 2.62 cents when the input sizes of the two parties are both 2^{20}.

Benchmark for cross tabulation computations. As mentioned in Sec. 1.1, one of the promising applications of our Private Intersection-Weighted-Sum protocol is the cross-tabulation computation between two parties. To see how the costs increase as the table size increases, we measure the costs for varying n_1 and n_2. Table 5 shows the costs when the table size is $n_1 \times n_2$. When n_1 is fixed, we can see that the costs do not change much as n_2 increases. Compared to the communication cost for the table size of 1×1, that for 1×16 is about 1.05 times. On the other hand, when n_2 is fixed, the costs significantly increase as n_1 increases. Compared to the communication cost for 1×1, that for 16×1 is about 1.92 times. In both cases, any cost is less than the total cost for computing 16 1×1-sized tables.

5 Conclusion

We proposed the new functionality of two-party computation called PIW-Sum and constructed an efficient PIW-Sum protocol from a DDH group and an additive homomorphic encryption scheme. PIW-Sum would have various real-world applications, and we demonstrated typical scenarios that we can use it.

We implemented our protocol together with the most efficient PI-Sum protocol in [21] and evaluated their performance. We showed that the computation, communication, and monetary costs of our protocol is only 2 times greater than those of the PI-Sum protocol. The benchmark of our protocol shows that it is sufficiently efficient for batch computation in business even when we use the tables consisting of more than one million records.

For future works, the construction of maliciously secure PIW-Sum protocol is an interesting open problem. A maliciously secure PI-Sum protocol was proposed in [26], but it seems that we need further work to extend their technique to PIW-Sum. Another interesting question is how our protocol compares to a PIW-Sum protocol based on general two-party computation techniques.

References

1. Microsoft SEAL. https://github.com/Microsoft/SEAL.
2. The sodium crypto library (libsodium). https://doc.libsodium.org.
3. R. Agrawal, A. V. Evfimievski, and R. Srikant. Information sharing across private databases. In A. Y. Halevy, Z. G. Ives, and A. Doan, editors, *ACM SIGMOD 2003*, pages 86–97. ACM, 2003.
4. P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, and G. Tsudik. Countering GATTACA: efficient and secure testing of fully-sequenced human genomes. In Y. Chen, G. Danezis, and V. Shmatikov, editors, *ACM CCS 2011*, pages 691–702. ACM Press, Oct. 2011.
5. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In L. Cavallaro, J. Kinder, X. Wang, and J. Katz, editors, *ACM CCS 2019*, pages 291–308. ACM Press, Nov. 2019.
6. P. Buddhavarapu, A. Knox, P. Mohassel, S. Sengupta, E. Taubeneck, and V. Vlaskin. Private matching for compute. Cryptology ePrint Archive, Report 2020/599, 2020. https://eprint.iacr.org/2020/599.
7. E. Bursztein, M. Hamburg, J. Lagarenne, and D. Boneh. OpenConflict: Preventing real time map hacks in online games. In *2011 IEEE Symposium on Security and Privacy*, pages 506–520. IEEE Computer Society Press, May 2011.
8. M. Chase and P. Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 34–63. Springer, Heidelberg, Aug. 2020.

9. D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Efficient robust private set intersection. In M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, editors, *ACNS 09*, volume 5536 of *LNCS*, pages 125–142. Springer, Heidelberg, June 2009.
10. E. De Cristofaro, P. Gasti, and G. Tsudik. Fast and private computation of cardinality of set intersection and union. In J. Pieprzyk, A.-R. Sadeghi, and M. Manulis, editors, *CANS 12*, volume 7712 of *LNCS*, pages 218–231. Springer, Heidelberg, Dec. 2012.
11. E. De Cristofaro, J. Kim, and G. Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In M. Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 213–231. Springer, Heidelberg, Dec. 2010.
12. S. K. Debnath and R. Dutta. Secure and efficient private set intersection cardinality using bloom filter. In J. Lopez and C. J. Mitchell, editors, *ISC 2015*, volume 9290 of *LNCS*, pages 209–226. Springer, Heidelberg, Sept. 2015.
13. C. Dong, L. Chen, and Z. Wen. When private set intersection meets big data: an efficient and scalable protocol. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 2013*, pages 789–800. ACM Press, Nov. 2013.
14. R. Egert, M. Fischlin, D. Gens, S. Jacob, M. Senker, and J. Tillmanns. Privately computing set-union and set-intersection cardinality via bloom filters. In E. Foo and D. Stebila, editors, *ACISP 15*, volume 9144 of *LNCS*, pages 413–430. Springer, Heidelberg, June / July 2015.
15. B. H. Falk, D. Noble, and R. Ostrovsky. Private set intersection with linear communication from general assumptions. In L. Cavallaro, J. Kinder, and J. Domingo-Ferrer, editors, *WPES@CCS, 2019*, pages 14–25. ACM, 2019.
16. M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In J. Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 303–324. Springer, Heidelberg, Feb. 2005.
17. M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In C. Cachin and J. Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 1–19. Springer, Heidelberg, May 2004.
18. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In A. Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.
19. Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS 2012*. The Internet Society, Feb. 2012.
20. B. A. Huberman, M. K. Franklin, and T. Hogg. Enhancing privacy and trust in electronic communities. In S. I. Feldman and M. P. Wellman, editors, *ACM Conference on Electronic Commerce, 1999*, pages 78–86. ACM, 1999.
21. M. Ion, B. Kreuter, A. E. Nergiz, S. Patel, S. Saxena, K. Seth, M. Raykova, D. Shanahan, and M. Yung. On deploying secure computing: Private intersection-sum-with-cardinality. In *EuroS&P 2020*, pages 370–389. IEEE, 2020.
22. L. Kissner and D. X. Song. Privacy-preserving set operations. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 241–257. Springer, Heidelberg, Aug. 2005.
23. V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu. Efficient batched oblivious PRF with applications to private set intersection. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 2016*, pages 818–829. ACM Press, Oct. 2016.
24. M. Li, N. Cao, S. Yu, and W. Lou. Findu: Privacy-preserving personal profile matching in mobile social networks. In *INFOCOM,2011*, pages 2435–2443. IEEE, 2011.
25. C. A. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *IEEE Symposium on Security and Privacy, 1986*, pages 134–137. IEEE Computer Society, 1986.
26. P. Miao, S. Patel, M. Raykova, K. Seth, and M. Yung. Two-sided malicious security for private intersection-sum with cardinality. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 3–33. Springer, Heidelberg, Aug. 2020.
27. S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov. BotGrep: Finding P2P bots with structured graph analysis. In *USENIX Security 2010*, pages 95–110. USENIX Association, Aug. 2010.
28. A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh. Location privacy via private proximity testing. In *NDSS 2011*. The Internet Society, Feb. 2011.
29. G. S. Narayanan, T. Aishwarya, A. Agrawal, A. Patra, A. Choudhary, and C. P. Rangan. Multi party distributed private matching, set disjointness and cardinality of set intersection with information theoretic

security. In J. A. Garay, A. Miyaji, and A. Otsuka, editors, *CANS 09*, volume 5888 of *LNCS*, pages 21–40. Springer, Heidelberg, Dec. 2009.
30. B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai. SpOT-light: Lightweight private set intersection from sparse OT extension. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 401–431. Springer, Heidelberg, Aug. 2019.
31. B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai. PSI from PaXoS: Fast, malicious private set intersection. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 739–767. Springer, Heidelberg, May 2020.
32. B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: Private set intersection using permutation-based hashing. In J. Jung and T. Holz, editors, *USENIX Security 2015*, pages 515–530. USENIX Association, Aug. 2015.
33. B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai. Efficient circuit-based PSI with linear communication. In Y. Ishai and V. Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 122–153. Springer, Heidelberg, May 2019.
34. B. Pinkas, T. Schneider, C. Weinert, and U. Wieder. Efficient circuit-based PSI via cuckoo hashing. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 125–157. Springer, Heidelberg, Apr. / May 2018.
35. B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on OT extension. In K. Fu and J. Jung, editors, *USENIX Security 2014*, pages 797–812. USENIX Association, Aug. 2014.
36. P. Rindal and M. Rosulek. Improved private set intersection against malicious adversaries. In J.-S. Coron and J. B. Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 235–259. Springer, Heidelberg, Apr. / May 2017.
37. P. Rindal and M. Rosulek. Malicious-secure private set intersection via dual execution. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *ACM CCS 2017*, pages 1229–1242. ACM Press, Oct. / Nov. 2017.
38. P. Rindal and P. Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In A. Canteaut and F.-X. Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 901–930. Springer, Heidelberg, Oct. 2021.
39. J. Vaidya and C. Clifton. Secure set intersection cardinality with application to association rule mining. *J. Comput. Secur.*, 13(4):593–622, 2005.
40. A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, Oct. 1986.