Communication-Efficient Inner Product Private Join and Compute with Cardinality

Koji Chida
Gunma University
Gunma, Japan
chida@gunma-u.ac.jp

Koki Hamada
NTT Social Informatics Laboratories
Tokyo, Japan
koki.hamada.rb@hco.ntt.co.jp

Atsunori Ichikawa
NTT Social Informatics Laboratories
Tokyo, Japan
atsunori.ichikawa.nf@hco.ntt.co.jp

Masanobu Kii
NTT Social Informatics Laboratories
Tokyo, Japan
masanobu.kii.gw@hco.ntt.co.jp

Junichi Tomida
NTT Social Informatics Laboratories
Tokyo, Japan
junichi.tomida.vw@hco.ntt.co.jp

ABSTRACT

Private join and compute (PJC) is a paradigm where two parties owing their private database securely join their databases and compute a function over the combined database. Inner product PJC, introduced by Lepoint *et al.* (Asiacrypt'21), is a class of PJC that has a wide range of applications such as secure analysis of advertising campaigns. In this computation, two parties, each of which has a set of identifier-value pairs, compute the inner product of the values after the (inner) join of their databases with respect to the identifiers. They proposed inner product PJC protocols that are specialized for the unbalanced setting where the input sizes of both parties are significantly different and not suitable for the balanced setting where the sizes of two inputs are relatively close.

We propose an inner product PJC protocol that is much more efficient than that by Lepoint *et al.* for balanced inputs in the setting where both parties are allowed to learn the intersection size additionally. Our protocol can be seen as an extension of the private intersection-sum protocol based on the decisional Diffie-Hellman assumption by Ion *et al.* (EuroS&P'20) and is especially communication-efficient as the private intersection-sum protocol. In the case where both input sizes are 2^{16}, the communication cost of our inner-product PJC protocol is 46× less than that of the inner product PJC protocol by Lepoint *et al.*

CCS CONCEPTS

• **Security and privacy → Cryptography**.

KEYWORDS

private set intersection, private intersection-sum, inner product, private join and compute, two-party computation, secure computation

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 the 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 ("payload"). They want to make some analysis on the database that is generated by the inner join 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 in the intersection are sensitive and need to be hidden from the communication partner.

Private Set Intersection. The above problem is closely related to Private Set Intersection (PSI) [8, 9, 11–15, 17, 20, 21, 24, 32–40] and PSI-cardinality [3, 10, 17, 21, 23, 31, 41]. PSI is a 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 [30], testing of fully sequenced human genomes [4], botnet detection [29], social networks [26], and online gaming [7], and has been extensively studied. However, both functionalities involve only a set of identifiers and cannot handle the payload associated with the identifiers.

Computing on payloads in the intersection. Several works consider secure two-party computation on the intersection of identifier sets and its payloads, e.g., [3, 6, 18, 22, 25, 28, 35]. Although we can compute any functions on the payloads in the intersection by using circuit PSI [35], this significantly increases the communication cost or the round complexity due to the use of generic two-party computation. Another promising approach is to construct protocols for a specific but useful functionality, which often allows us to construct more efficient protocols than those for generic functionalities.

A notable functionality in the latter approach is Private Intersection-Sum with Cardinality (PI-Sum) [18, 22, 28], which is introduced and deployed by Google [22]. In PI-Sum, one party has a set of pairs of an identifier and an value $\{(v_i, x_i)\}$ while the other party has an identifier set $\{w_j\}$, and the goal is to compute $\sum_{i:v_i=w_j} x_i$ i.e., the sum of values x_i within the intersection $\{v_i\} \cap \{w_j\}$ of the two identifier sets, together with the cardinality $|\{v_i\} \cap \{w_j\}|$ of the intersection. Hence, PI-Sum can be seen as computation after the inner join of two databases owed by two parties where only one party has a database with payloads.

Recently, a more generalized functionality called inner product private join and compute (inner product PJC) was introduced by

[25]. In this computation, both parties have a set of pairs of an identifier and a value: $(V, X) = \{(v_i, x_i)\}$ and $(W, Y) = \{(w_j, y_j)\}$. Inner product PJC allows one party (receiver) to learn $\sum_{(i,j):v_i=w_j} x_i y_j$, i.e., the inner product of X and Y within the intersection of V and W[1]. Inner product PJC is a quite useful functionality in secure analysis of advertising campaigns as discussed later in Section 1.2. An exposure notification protocol for COVID-19 is also suggested for another application of inner product PJC [25]. We can easily observe that PI-Sum is basically the special case of inner product PJC where $y_j = 1$ for all i.

The inner product PJC protocols by [25] are suitable for the unbalanced setting where the database size of the receiver is much smaller than that of the sender. This is because the communication and computation costs of the receiver are either independent of or logarithmic to the size of the sender's input. On the other hand, their protocols do not perform well in the balanced setting where the input sizes of the sender and the receiver are relatively close. In fact, both communication and computation costs of the inner product PJC protocol for the online setting by [25] are larger than the circuit PSI protocol by [35] that computes inner product with a circuit. This naturally motivates us to explore a more communication and round efficient inner product PJC protocol for balanced inputs.

1.1 Our Contributions

Our contributions in this work are two-fold. First, we construct a communication-efficient 4-round inner product PJC protocol that allows both parties to learn the cardinality of the intersection additionally. Although this additional leakage is a disadvantage if both parties are unwilling to reveal it, we consider that it is rather beneficial in typical cases. We discuss this in Section 1.2. Our protocol is based on the decisional Diffie-Hellman (DDH) assumption and additively homomorphic encryption (AHE) and is secure against semi-honest adversaries.

While the most efficient PSI protocols are based on random oblivious transfer (OT) [8, 40], it seems difficult to apply the technique to PI-Sum and inner product PJC. In fact, the PI-Sum protocol based on the classical DDH-based double masking technique in [22] is more efficient than that based on random OT in [22]. This is because the random OT-based PI-Sum protocol needs a shuffle of ciphertexts of the AHE scheme, which uses expensive fully homomorphic operations of the underlying AHE scheme. Later, the more computation-efficient PI-Sum protocol based on oblivious switching was proposed by [18], but their technique cannot be straightforwardly extended to inner product PJC.

Another approach for PI-Sum is circuit PSI [20], which is a technique to use general two-party computation (garbled circuits [42] or the GMW protocol [19]) to compute any functions on the intersection and the related payloads. 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 are much higher than that of the DDH-based protocol [22]. GMW-based PSI protocols can be more communication efficient than garbled-circuit-based PSI protocols [5, 40], but this approach significantly increases the round complexity.

[1]In contrast to PI-Sum by [22], the inner product PJC protocol by [25] does not reveal the intersection size.

As discussed in [22], communication efficiency is the first priority in business-to-business batch computation for huge data. In a nutshell, increasing the computational resource such as CPU, RAM is much easier than increasing the bandwidth of the Internet connection of data centers. Another piece of 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. For instance, 1 GB of communication is the same price as 8 hours of computation in Google Cloud Platform. Hence, our protocol is suitable for such business-to-business computation in the WAN setting where a high network latency exists.

The second contribution is the implementation and evaluation of our protocol. We use an ideal lattice-based fully homomorphic encryption scheme for an AHE scheme that allows slot encryption [1]. This technique allows us to encrypt many values to a single ciphertext while we can still perform homomorphic addition over any encrypted values and significantly save the communication cost in the protocol. In the case where both input sizes are 2^{16}, the communication cost of our inner-product PJC protocol is 28× less than that of the circuit-PSI protocol with garbled circuit [35] and 46× less than that of the inner product PJC protocol by [25]. The estimated monetary cost to run our protocol in a typical cloud environment[2] is about 0.15 cents when both identifier sets are equivalent and their size is 2^{16}. It is 26× less than the monetary cost of the circuit-PSI protocol with garbled circuit [35] and 44× less than that of the inner product PJC protocol by [25].

1.2 Applications of Inner Product PJC

Inner product PJC can be applied for various situations where two parties would like to perform joint analysis over the database generated by the inner join of their databases. We give two typical scenarios that we can solve by inner product PJC where the input sizes of both parties can be naturally close. Note that inner product PJC can be easily extended to computing the weighted sum of vectors, that is, two parties compute $\sum_{(i,j):v_i=w_j} y_j \mathbf{x}_i$ from their inputs $(V, X) = \{(v_i, \mathbf{x}_i)\}$ and $(W, Y) = \{(w_j, y_j)\}$.

Case 1. Weighted advertising conversion measurement. The first case is depicted as Fig. 1. An ad supplier delivers an advertisement on a campaign of a retail chain to encourage its users to visit retail stores. Later the retail chain may analyze the effectiveness of the advertisement. The retail chain is going to strengthen the line up of products for people in their 20s and 30s and would like to multiply the number of visits by some weight according to their age for the measurement for analysis. In this situation, what the retail chain needs 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 the age of users. We can deal with this situation via inner product PJC just by setting the retail chain's input as a set of pairs of an identifier v_i and a number x_i of visits to the stores while the ad supplier's input as a set of pairs of an identifier w_j and a weight y_j according to their age. Then, the output is $z_1 = \sum_{(i,j):v_i=w_j} x_i y_j$ and $z_2 = |\{v_i\} \cap \{w_j\}|$, and one can compute z_1/z_2 as desired.

[2]We use the monetary cost of Google Cloud Platform for the estimation.

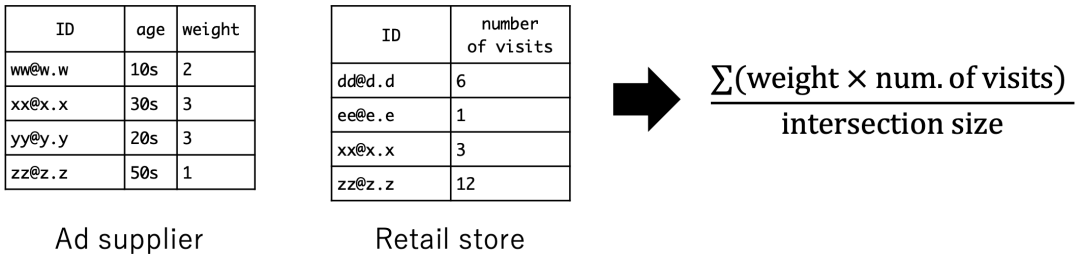

Figure 1: Weighted advertising conversion measurement.

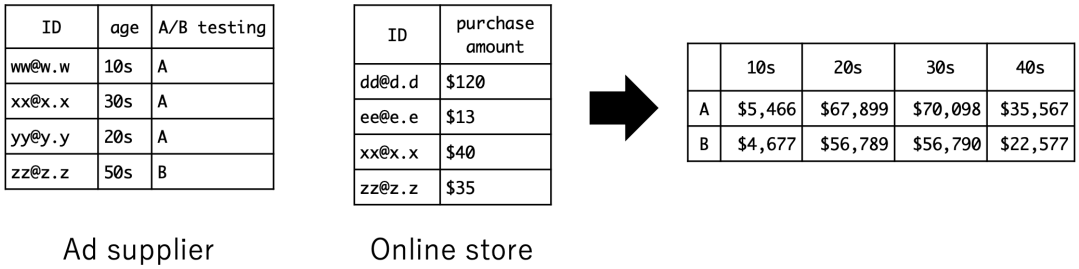

Figure 2: Cross-tabulation for A/B testing.

In this application, it is crucial to compute the cardinality in addition to the inner product within the intersection since the inner product itself has no relation to the effectiveness of the advertisement unless one divides it by the intersection size. Hence, the property of our protocol that it reveals the intersection size is a benefit for this situation.

Case 2. Cross tabulation for A/B testing. The second case is depicted as Fig. 2. An ad supplier delivers two types of advertisements on an online store. The ad supplier wants to know which ad was more effective for the sales and the difference in 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. 2). 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 inner product PJC protocol as follows.

The ad supplier sets its input as a set of pairs of an identifier v_i and a one-hot vector $\mathbf{x}_i \in \{0,1\}^{2d}$ where d is the number of choices in age, and $x_{i,k} = 1$ if and only if the age and the delivered ad of user v_i corresponds to the k-th cell of the cross-tabulation. On the other hand, the company sets its input as a set of pairs of an identifier w_j and a purchase amount y_j. Then, the output $\sum_{(i,j):v_i=w_j} x_{i,k}y_j$ for $k \in [2d]$ corresponds to the value in the k-th cell of the cross tabulation.

Note that in this case, the cardinality of the intersection might be an unnecessary leakage since it is not used to compute the cross-tabulation. In such an application, however, we consider that the intersection size should be rather revealed to validate the correctness of the analysis.

More information on real-world application. We notice that Ion *et al.* gave quite a detailed analysis on deploying a PI-Sum protocol for business applications [22]. Since most of their analysis applies to our inner product PJC case, [22] is an excellent reference for readers interested in more practical discussions.

2 TECHNICAL OVERVIEW

DDH-based PI-Sum protocol. Let us briefly recall the DDH-based PI-Sum protocol by [22], which is our starting point. Their protocol can be seen as construction from a shuffled oblivious pseudorandom function (shuffled OPRF) protocol and an AHE scheme. Shuffled OPRF is a two-party protocol for the following functionality:

$$\text{Shuffled OPRF} : ((\pi, k), \{z_i\}_{i\in[m]}) \to (\perp, \{f(k, z_{\pi(i)})\}_{i\in[m]})$$

where π is a permutation, k is a PRF key, and f is a PRF. That is, the sender's input is a permutation π in $[m]$ and a PRF key k while the receiver's input is m strings $\{z_i\}$, and the sender learns nothing while the receiver obtains the PRF values $\{f(k, z_{\pi(i)})\}$ in the permuted order by π as outputs.

We can construct a PI-Sum protocol from shuffled OPRF as follows. On inputs $\{(v_i, x_i)\}_{i\in[m_1]}$ for the receiver of the PI-Sum protocol, say Alice, and $\{w_j\}_{j\in[m_2]}$ for the sender, say Bob, they run a shuffled OPRF protocol with Alice/Bob's input being $(\pi, k)/\{w_j\}_j$ where π and k is randomly chosen by Alice. At the same time, Alice generates an AHE instance to encrypt $\{x_i\}_{i\in[m_1]}$ and sends $\{\mathsf{Enc}(x_i), f(k, v_i)\}$ to Bob. Since Bob obtains $\{f(k, w_{\pi(j)})\}_{j\in[m_2]}$ from the shuffled OPRF protocol, Bob can homomorphically compute $\mathsf{Enc}(\sum_{i:v_i=w_j} x_i)$ by adding $\mathsf{Enc}(x_i)$ for all i such that $f(k, v_i) \in \{f(k, w_{\pi(j)})\}$. Finally, Alice obtains $\sum_{i:v_i=w_j} x_i$ by decrypting the AHE ciphertext.

The shuffled OPRF protocol is obtained as follows. Let $\mathbb{G}$ be a DDH group of order p, g be its generator, and $H : \{0,1\}^* \to \mathbb{G}$ be a hash function modeled as a random oracle. The receiver chooses a

random $\mathbb{Z}_p$ element b. In round 1, on input $\{z_i\}_{i\in[m]}$, the receiver sends $\{H(z_i)^b\}$ to the sender. In round 2, on input (π, k) and the message from the receiver, the sender sends $\{H(z_{\pi(i)})^{bk}\}$ to the receiver. Then, the receiver can learn the permuted PRF values $\{H(z_{\pi(i)})^k\}$ where the PRF is defined as $f(k,z) = H(z)^k$.

Difficulty of efficient DDH-based inner product PJC. Our protocol uses the same ingredient as the DDH-based PI-Sum protocol by [22], i.e., a DDH group and an AHE scheme. The essential requirement to achieve an efficient inner product PJC protocol is the non-use of expensive shuffles of AHE ciphertexts, which significantly increases the computation cost [22, Table 5 and Figure 7]. Our main technical contribution is the construction of a DDH-based inner product PJC protocol that does not use shuffles of AHE ciphertexts.

One may wonder why the non-use of ciphertext shuffles is a challenge in constructing an inner product PJC protocol based on DDH-group and AHE while the DDH-based PI-Sum protocol does not use them. In inner product PJC, a straightforward approach is that one party, say Alice, encrypts its input $\{x_i\}$ to $\{\mathsf{Enc}(x_i)\}$ by AHE and the other party, say Bob, homomorphically computes the inner product $\{\mathsf{Enc}(x_i y_j)\}_{(i,j):v_i=w_j}$ in the intersection. However, Bob should be oblivious to its values $\{y_j\}$ in the homomorphic computation as otherwise it learns whether the corresponding identifier belongs to the intersection or not. Note that this problem does not occur in PI-Sum since the latter party does not need to know the correspondence between x_i and y_j such that $v_i = w_j$.

This problem can be naively solved as follows. First, the two parties jointly generate a fully homomorphic encryption (FHE) instance so that ciphertexts can be decrypted in only a co-operated manner. Then, both parties encrypt their values $\{x_i\}$ and $\{y_j\}$ to $\{\mathsf{Enc}(x_i)\}$ and $\{\mathsf{Enc}(y_j)\}$, respectively, by the FHE scheme and join them to obtain $\{\mathsf{Enc}(x_i), \mathsf{Enc}(y_j)\}_{(i,j):v_i=w_j}$ in a similar manner to the DDH-based shuffled OPRF protocol [3]. Finally, one party homomorphically computes $z = \mathsf{Enc}(\sum_{(i,j):v_i=w_j} x_i y_j)$ and jointly decrypts it to obtain the desired output. The point is that when the parties join the encrypted values, i.e., when they compute $\{\mathsf{Enc}(x_i), \mathsf{Enc}(y_j)\}_{(i,j):v_i=w_j}$, this pairs of ciphertexts must be *shuffled* as otherwise the party that obtains $\{\mathsf{Enc}(x_i), \mathsf{Enc}(y_j)\}_{(i,j):v_i=w_j}$ learns which identifiers of its input is in the intersection. Thus, we need to contrive a new protocol to remove the use of expensive shuffle and homomorphic multiplication over the ciphertext.

Outer join instead of inner join. Conceptually, the approach discussed so far can be seen as the inner join (or equijoin) of two databases where one or both databases are encrypted. However, this approach would require shuffles of ciphertexts and an FHE scheme, which we would like to avoid using. Our solution to this problem is to use the (right) outer join of two databases instead of the inner join. The goal of this approach is to allow Bob to have the outer joined database where all Bob's plain values are on the right side, each of which is accompanied by the corresponding encrypted value from Alice or the encryption of 0 on the left side. More formally, on inputs $\{v_i, x_i\}_{i\in[m_1]}$ for Alice and $\{w_j, y_j\}_{j\in[m_2]}$ for Bob, Bob obtains the outer joined database $\{\mathsf{Enc}(q_j), y_j\}_{j\in[m_2]}$ where $q_j = x_i$ if there exists i such that $v_i = w_j$ and $q_j = 0$ otherwise. Then, he can homomorphically compute the inner product in the intersection $\mathsf{Enc}(\sum_{(i,j):v_i=w_j} y_j x_i)$ by summing up $y_j\mathsf{Enc}(q_j)$ 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 the outer joined database. We solve this by introducing a new variant of oblivious pseudorandom function (OPRF). 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

$$\phi : ((k, \{v_i\}_{i\in[m_1]}), \{w_j\}_{j\in[m_2]}) \to (S, \{f(k, z_j)\}_{j\in[m_2]}) \quad (1)$$

where k is a PRF key, $S = \{j \in [m_2] \mid \forall i, v_i \neq w_j\}$, f is a PRF, $z_j = w_j$ if $j \notin S$, and $z_j = u_j$ if $j \in S$. The differences between the normal (multi-point) OPRF[3] and the variant ϕ are 1) the sender has input strings $\{v_i\}_i$ in addition to the PRF key k; 2) the receiver 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 the sender's input strings; 3) the sender additionally learns the locations S of the receiver's input strings that do not match any of the sender's input strings.

If we have a secure protocol for ϕ, the two parties can compute the outer joined database as follows. Alice first generates a random PRF key k and an AHE instance. Alice and Bob run the protocol for ϕ on inputs $((k, \{v_i\}_{i\in[m_1]}), \{w_j\}_{j\in[m_2]})$ where Alice learns S while Bob learns $\{f(k, z_j)\}_{j\in[m_2]}$. Then, Alice sends $\{f(k, t_i), \mathsf{Enc}(q_i)\}_{i\in[m_1+|S|]}$ 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), $q_i = x_i$ for $i \in [m_1]$, and $q_{i+m_1} = 0$ for $i \in [|S|]$. Finally, Bob combines $\mathsf{Enc}(q_i)$ and y_j such that $f(k, t_i) = f(k, z_j)$ for all $(i, j) \in [m_1 + |S|] \times [m_2]$. Observe that for $j \in [m_2]\backslash S$ such that $v_i = w_j$ for some $i \in [m_1]$, we have $t_i = v_i$, $z_j = w_j$, $q_i = x_i$. On the other hand, for $j \in S$, we have $z_j = u_j$, $u_j \in \{t_{i+m_1}\}_{i\in[|S|]}$, and $q_{i+m_1} = 0$ for all $i \in [|S|]$. Hence, Bob successfully obtains the outer joined database.

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, the sender chooses two random $\mathbb{Z}_p$ elements a while the receiver chooses a random $\mathbb{Z}_p$ element b. On input $(k, \{v_i\}_{i\in[m_1]})$, the sender sends $\{H(v_i)^a\}_{i\in[m_1]}$ to the receiver. On input $\{w_j\}_{j\in[m_2]}$ and the message from the sender, the receiver computes $\{H(v_i)^{ab}\}_{i\in[m_1]}$, $\{H(w_j)^b\}_{j\in[m_2]}$, $\{H(u_j)^b\}_{j\in[m_2]}$, shuffles $\{H(v_i)^{ab}\}$, and sends them back to the sender. Then, the sender computes $\{H(v_i)^b\}_{i\in[m_1]}$ by $\{H(v_i)^{ab}\}_i$ to the power of $1/a$, sets $r_j = H(w_j)^{bk}$ if $\exists i, H(v_i)^b = H(w_j)^b$ and $r_j = H(u_j)^{bk}$ otherwise for all $j \in [m_2]$, and sends $\{r_j\}_{j\in[m_2]}$ to the receiver. Finally, the receiver outputs $\{r_j^{1/b}\}$. The sender learns $S = \{j \in [m_2] \mid \forall i, H(v_i)^b \neq H(w_j)^b\}$. In this protocol, the PRF is defined as $f(k, z) = H(z)^k$.

3 PRELIMINARIES

3.1 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

[3] We assume the multi-point OPRF functionality as $(k, \{z_i\}_i) \to (\perp, \{f(k, z_i)\}_i)$ here, where f is a PRF and k is a PRF key.

$\mathcal{A}$, there exists a negligible function negl and we have $|\Pr[1 \leftarrow \mathcal{A}(1^\lambda, X_\lambda)] - \Pr[1 \leftarrow \mathcal{A}(1^\lambda, Y_\lambda)]| \leq \mathsf{negl}(\lambda)$.

3.2 Basic Tools and Assumption

Definition 3.1 (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 3.2 (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 C that are additive abelian groups.
- $\mathsf{AEnc}(\mathsf{pk}, m)$ It takes pk a plaintext $m \in \mathcal{M}$ and outputs a ciphertext $\mathsf{ct} \in C$ for m.
- $\mathsf{ARef}(\mathsf{pk}, \mathsf{ct})$ It takes pk, ct and outputs a ciphertext $\mathsf{ct}' \in C$.
- $\mathsf{ADec}(\mathsf{sk}, \mathsf{ct})$ 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 Section 5.1.

3.3 Inner Product Private Join and Compute with Cardinality

Inner product private join and compute with cardinality (inner product PJC) 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 3.3 (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:

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

Definition 3.4 (Inner Product PJC). Inner Product PJC[4] 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 $\{v_i, x_i\}_{i\in[m_1]}$ and $\{w_j, y_j\}_{j\in[m_2]}$, respectively, where v_i, w_j are identities belonging to an identifier space $\mathcal{I}$ and $x_i, y_j \in \mathbb{Z}$ are integers. Both parties know the table size of the communication partner. The goal is to allow P_1 to learn the inner product $\sum_{(i,j):v_i=w_j} x_i y_j$ with in the intersection and the cardinality of the intersection $|\{v_i\}_i \cap \{w_j\}_j|$ while P_2 learns only the cardinality of the intersection $|\{v_i\}_i \cap \{w_j\}_j|$.

- Input for both parties: m_1, m_2.
- Input for P_1 : $\{v_i, x_i\}_{i\in[m_1]}$.
- Input for P_2 : $\{w_j, y_j\}_{j\in[m_2]}$.
- Output for P_1 : $\sum_{(i,j):v_i=w_j} x_i y_j$, $|\{v_i\}_i \cap \{w_j\}_j|$.
- Output for P_2 : $|\{v_i\}_i \cap \{w_j\}_j|$.

Figure 3: The functionality of inner product PJC

[4] In what follows we use the term "inner product PJC" for the functionality that reveals the intersection size unless otherwise stated, while the term is used for the functionality that computes only inner product in [25].

Inner product PJC can be easily extended to the more general case where the values x_i and y_j are changed to integer vectors, and the output is the matrix product with in the intersection. We call the functionality matrix product PJC. The case 2 in Section 1.2 uses the matrix product PJC functionality where each payload of P_1 is an integer vector $\mathbf{x}_i$ while each payload of P_2 is an integer y_j.

Definition 3.5 (Matrix Product PJC). Matrix Product PJC is a generalization of inner product PJC where inputs and outputs for two parties are defined as in Fig. 4. This functionality is the same as inner product PJC except that the payloads in the inputs of P_1 and P_2 are vectors $\mathbf{x}_i$ and $\mathbf{y}_j$, respectively, and the first output of P_1 is $\{\sum_{(i,j):v_i=w_j} x_{i,k}y_{j,\ell}\}_{k\in[n_1],\ell\in[n_2]}$. In other words, let $(I_1,\ldots,I_d)$ be the intersection of $\{v_i\}$ and $\{w_j\}$, and the first output of P_1 can be equivalently defined as $\mathbf{Y}^\top\mathbf{X}$, where the k-th rows of $\mathbf{X}$ and $\mathbf{Y}$ are $\mathbf{x}_i$ and $\mathbf{y}_j$ such that $v_i = w_j = I_k$, respectively.

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

Figure 4: The functionality of matrix product PJC

4 OUR MATRIX PRODUCT PJC PROTOCOL

In this section, we present our matrix product PJC protocol. Note that our inner product PJC protocol is easily obtained from our matrix product PJC protocol by setting $n_1 = n_2 = 1$. Our matrix product PJC protocol is shown in Fig. 5.

As explained in Section 2, a building block of our protocol can be seen as a variant ϕ of OPRF. 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 for the OPRF variant ϕ:

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

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, where a_2 corresponds to the PRF key k. 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 Section 2, where P_1's input is $(a_2, \{v_i\}_i)$, and P_2's input is $\{w_j\}_j$. In steps (2) and (3) of Round 3, P_1 additionally sends a set of pairs of PRF values and AHE ciphertexts of its payload or $\mathbf{0}$. The pairs of the PRF values and the AHE ciphertexts are used to make the (right) outer join of the two databases in round 4.

In step (1) of round 4, P_2 makes the correspondence between ciphertexts sent by P_1 and its input $\mathbf{y}_i$ for the outer join by comparing the PRF values sent by P_1 and those obtained via the execution of ϕ. Then, in step (2) of round 4, P_2 homomorphically evaluates the matrix product over the ciphertexts. In the final round, P_1 decrypts the ciphertexts and obtains the matrix product. 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.

4.1 Security

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

Theorem 4.1. *Assume that H is a hash function modeled as a random oracle, the DDH assumption holds in $\mathbb{G}$, and the AHE scheme is IND-CPA secure, 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

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

Due to the limit of space, we present the proof of the theorem in **??**.

5 IMPLEMENTATION AND EVALUATION

In this section, we present the measurements for our implementation of our matrix product PJC protocol and the comparison to previous works.

5.1 Implementation Details

We implement our protocol together with the DDH-based PI-Sum protocol in [22] 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 the 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`" 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.

- 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 $m_2 - m_3$ while P_2 outputs $m_2 - ((m_1 + m_3) - m_1)$.

Figure 5: Our matrix product PJC Protocol

We use the slotting optimization for the AHE scheme to reduce communication cost. In a nutshell, we can encrypt many plaintexts to a single 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 the 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. A more detailed explanation of the slotting optimization technique can be found in [22, Appendix B].

5.2 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 the 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 executed 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[5]. For network cost, we use the cost of the cheapest 10+TB category of the Premium Tier pricing[6], which corresponds to the communication cost between the Internet and the cloud. Although the 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 than 10%, and this estimation would be valid.

5.3 Discussion of Measurements

Theoretical cost and comparison with the PI-Sum protocol. First, we present the theoretical costs of our protocol and the DDH-based PI-Sum protocol by [22] with respect to the DDH-group and the AHE scheme in Table 1. The comparison shows that both communication and computation costs of our inner product PJC protocol are about 2× more than those of the PI-Sum protocol, while inner product PJC computes a more general functionality than PI-Sum. 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 PI-Sum protocol, the cost of which is maximized when the intersection size is maximum.

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

Protocol	Computation Cost		Communication Cost	
	#Group exponentiation	#AHE operations	#Group elements	#AHE ciphertexts
PI-Sum [22]	$2m_1 + 2m_2$	$m_1 + m_2 - m_3 + 1$	$m_1 + 2m_2$	$m_1 + 1$
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 our protocol and the DDH-based PI-Sum protocol by [22]. We compare them with respect to the computation and communication costs of the DDH-group and the AHE scheme. AHE operations consist 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	Group Operations		AHE Operations		Others	Total	
$m_1 = m_2$	Comm. [MB]	Time [sec.]	Comm. [MB]	Time [sec.]	Time [sec.]	Comm. [MB]	Time [sec.]
2^{12}	1.0	1.52	0.1	0.049	0.051	1.1	1.62
2^{16}	16.1	24.32	1.1	0.82	0.78	17.2	25.92
2^{20}	258	396.0	17	13.6	14.8	275	424.4

Table 2: Measurements of computation time and network costs of our protocol for various input sizes and those with respect to component operations, namely, DDH-group and AHE operations and others. Others include operations such as shuffling elements and comparison of group elements. In these measurements, we use the setting where the inputs of both parties are the same, $n_1 = n_2 = 1$, and all the records of P_1 match all the records of P_2.

Input Size		Our Protocol			Inner Product PJC [25]†			Circuit PSI [35]†			PI-Sum [22]‡		
m_1	m_2	Comm. [MB]	Time [sec.]	Cost [US cent]	Comm. [MB]	Time [sec.]	Cost [US cent]	Comm. [MB]	Time [sec.]	Cost [US cent]	Comm. [MB]	Time [sec.]	Cost [US cent]
2^{16}	2^8	9.2	11.80	0.08	27	172.3	0.26	5	3.02	0.04	5.4	6.05	0.04
	2^{12}	9.7	12.63	0.08	120	139.3	1.00	30	3.81	0.24	5.6	6.23	0.05
	2^{16}	17.2	25.92	0.14	801	131.1	6.44	472	8.52	3.78	9.2	12.4	0.08
2^{20}	2^8	146	191.6	1.22	29	3026	1.07	51	45.6	0.42	86.1	93.6	0.71
	2^{12}	146	192.4	1.22	213	2134	2.30	76	46.3	0.62	86.2	93.7	0.72
	2^{16}	154	205.4	1.29	1821	2228	15.2	522	51.1	4.19	89.8	99.8	0.75

Table 3: Measurements of computation time and network costs, estimated monetary costs in US cent of our protocol, and comparison with other works that can compute the inner product PJC functionality. In this comparison we use the setting where $n_1 = n_2 = 1$ (i.e. the inner product PJC case) and the set for P_2 is included in the set for P_1.
† The measurements for inner product PJC and circuit PSI are taken from [25], which are obtained by a machine with a single core of Intel(R) Xeon(R) CPU E5-2696 v3 @ 2.30GHz.
‡ PI-Sum is a special case of inner product PJC and cannot compute the inner product PJC functionality. Since our protocol is an extension of the PI-Sum protocol by [22], we report it for reference.

Measurement of our protocol. We show the communication and computation costs of our protocol for various input sizes and those with respect to each component operation in Table 2. It shows that most of the cost in our protocol is taken by group operations. Specifically, 93% of the total cost is used for group operations in both communication and computation costs.

Comparison with other inner product PJC protocols. In Table 3, we compare our protocol with other protocols that can compute the inner product functionality with constant rounds [25, 35]. The inner product PJC protocol by [25] is the only direct construction of the inner product PJC functionality, which does not use a generic two-party protocol inside. They also provide the measurements of the circuit-based PSI protocol by [35] that computes the inner product functionality via circuit evaluation, and we include them for our comparison. Note that recent papers on improved circuit PSI protocols such as [40] provide only the measurements with the GMW protocol, the round complexity of which depends on the input sizes, and thus we exclude them from comparison.

The inner product PJC protocol by [25] is specialized for the unbalanced setting and has an excellent performance when the input size of one party is significantly larger than that of the other party. On the other hand, their protocol is not suitable if the input sizes of both parties are relatively close. As shown in Table 3, our protocol outperforms the protocol by [25] in most cases. In

Input Size Parameters					Our Protocol		
m_1	m_2	m_3	n_1	n_2	Comm. [MB]	Time [sec.]	Cost [US cent]
2^{16}	2^{16}	0	1	1	17.2	24.92†	0.14
			1	16	18.0	24.99	0.15
			16	1	33.1	25.63	0.27
			16	16	33.9	25.74	0.28
		2^{16}	1	1	20.9	28.27	0.17

Table 4: Measurements of computation time, network costs, and monetary costs in US cent of our protocol for various parameters, where both input sizes are 2^{16}. Recall that m_3 is the number of the records for P_2 out of the intersection, and n_1 and n_2 are the vector lengths of payloads for P_1 and P_2 in matrix product PJC, respectively. Thus, the first four rows are the measurements in the setting where all the identifiers for P_1 match all the identifiers for P_2, while the last row shows the measurements in the case where the identifier sets of both parties are disjoint.
† The measurements in this table is performed separately from those in Tables 2 and 3 and the time is a bit different from the time for the same condition in Tables 2 and 3.

the case where both input sizes are 2^{16}, our protocol is 46× more communication-efficient and 5× more computation-efficient than the protocol by [25].

The circuit PSI protocol by [35] is more efficient than inner product PJC protocol by [25] in most cases, even though it uses generic two-party computation for label comparison and computing inner product. While the circuit PSI protocol outperforms our inner product PJC protocol with respect to computation cost in all the cases, our protocol is much more communication-efficient than their protocol in the balanced setting. For the case where both input sizes are 2^{16}, the communication cost of ours is 28× less than that of the circuit PSI protocol.

We also estimate the monetary costs of each protocol using Google Cloud Platform pricing. As well as communication cost, our protocol outperforms the other protocols with respect to monetary cost in the balanced setting. Concretely, the cost of our protocol is 26× less than that of the circuit-PSI protocol, and 44× less than that of the inner product PJC protocol by [25] in the case where both input sizes are 2^{16}.

Measurement for various parameters. Table 4 shows measurements of our protocol for various parameters. As mentioned in Section 1.2, one of the promising applications of inner product (matrix product) PJC 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 and report them in the second to the fourth rows. 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. Importantly, all the costs in all the cases are much less than the total cost for running the inner product PJC protocol 16 times.

We also measure the costs when the two identifier sets are disjoint, which is the worst case for our protocol. We show the costs in the last row. Compared to the costs in the case where both identifier sets are equivalent (the first row), the computation time is at most 1.14 times and the network cost is at most 1.22 times for all input sizes.

6 CONCLUSION

We proposed the new inner product (and matrix product) private join and compute protocol from a DDH group and an additive homomorphic encryption scheme. Our protocol is especially communication-efficient and outperforms previous protocols in the balanced setting if both parties are allowed to learn the intersection size additionally.

We implemented our protocol and evaluated its performance. In the case where both input sizes are 2^{16}, the communication cost of our inner-product PJC protocol is 28× less than that of the circuit-PSI protocol with garbled circuit [35] and 46× less than that of the inner product PJC protocol by [35]. The estimated monetary cost to run our protocol in Google Cloud Platform is about 0.15 cents when both identifier sets are equivalent and their size is 2^{16}. It is 26× less than the monetary cost of the circuit-PSI protocol with garbled circuit [35] and 44× less than that of the inner product PJC protocol by [35].

For future works, the construction of maliciously secure inner product PJC protocol is an interesting open problem. A maliciously secure PI-Sum protocol was proposed in [28], but it seems that we need further work to extend their technique to inner product PJC. Another interesting question is the construction of more computation efficient inner product PJC, because there is a significant gap of computation efficiency between our inner product PJC protocol and the state-of-the-art PSI protocols such as [40].

REFERENCES

[1] Microsoft SEAL, https://github.com/Microsoft/SEAL
[2] The sodium crypto library (libsodium), https://doc.libsodium.org
[3] Agrawal, R., Evfimievski, A.V., Srikant, R.: Information sharing across private databases. In: Halevy, A.Y., Ives, Z.G., Doan, A. (eds.) ACM SIGMOD 2003. pp. 86–97. ACM (2003), https://doi.org/10.1145/872757.872771
[4] Baldi, P., Baronio, R., De Cristofaro, E., Gasti, P., Tsudik, G.: Countering GATTACA: efficient and secure testing of fully-sequenced human genomes. In: Chen, Y., Danezis, G., Shmatikov, V. (eds.) ACM CCS 2011. pp. 691–702. ACM Press (Oct 2011)
[5] Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: Efficient two-round OT extension and silent non-interactive secure computation. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 291–308. ACM Press (Nov 2019)
[6] Buddhavarapu, P., Knox, A., Mohassel, P., Sengupta, S., Taubeneck, E., Vlaskin, V.: Private matching for compute. Cryptology ePrint Archive, Report 2020/599 (2020), https://eprint.iacr.org/2020/599

[7] Bursztein, E., Hamburg, M., Lagarenne, J., Boneh, D.: OpenConflict: Preventing real time map hacks in online games. In: 2011 IEEE Symposium on Security and Privacy. pp. 506–520. IEEE Computer Society Press (May 2011)

[8] Chase, M., Miao, P.: Private set intersection in the internet setting from lightweight oblivious PRF. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 34–63. Springer, Heidelberg (Aug 2020)

[9] Dachman-Soled, D., Malkin, T., Raykova, M., Yung, M.: Efficient robust private set intersection. In: Abdalla, M., Pointcheval, D., Fouque, P.A., Vergnaud, D. (eds.) ACNS 09. LNCS, vol. 5536, pp. 125–142. Springer, Heidelberg (Jun 2009)

[10] De Cristofaro, E., Gasti, P., Tsudik, G.: Fast and private computation of cardinality of set intersection and union. In: Pieprzyk, J., Sadeghi, A.R., Manulis, M. (eds.) CANS 12. LNCS, vol. 7712, pp. 218–231. Springer, Heidelberg (Dec 2012)

[11] De Cristofaro, E., Kim, J., Tsudik, G.: Linear-complexity private set intersection protocols secure in malicious model. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 213–231. Springer, Heidelberg (Dec 2010)

[12] Debnath, S.K., Dutta, R.: Secure and efficient private set intersection cardinality using bloom filter. In: Lopez, J., Mitchell, C.J. (eds.) ISC 2015. LNCS, vol. 9290, pp. 209–226. Springer, Heidelberg (Sep 2015)

[13] Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: an efficient and scalable protocol. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 789–800. ACM Press (Nov 2013)

[14] Egert, R., Fischlin, M., Gens, D., Jacob, S., Senker, M., Tillmanns, J.: Privately computing set-union and set-intersection cardinality via bloom filters. In: Foo, E., Stebila, D. (eds.) ACISP 15. LNCS, vol. 9144, pp. 413–430. Springer, Heidelberg (Jun / Jul 2015)

[15] Falk, B.H., Noble, D., Ostrovsky, R.: Private set intersection with linear communication from general assumptions. In: Cavallaro, L., Kinder, J., Domingo-Ferrer, J. (eds.) WPES@CCS, 2019. pp. 14–25. ACM (2019), https://doi.org/10.1145/3338498.3358645

[16] Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 303–324. Springer, Heidelberg (Feb 2005)

[17] Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (May 2004)

[18] Garimella, G., Mohassel, P., Rosulek, M., Sadeghian, S., Singh, J.: Private set operations from oblivious switching. In: Garay, J. (ed.) PKC 2021, Part II. LNCS, vol. 12711, pp. 591–617. Springer, Heidelberg (May 2021)

[19] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press (May 1987)

[20] Huang, Y., Evans, D., Katz, J.: Private set intersection: Are garbled circuits better than custom protocols? In: NDSS 2012. The Internet Society (Feb 2012)

[21] Huberman, B.A., Franklin, M.K., Hogg, T.: Enhancing privacy and trust in electronic communities. In: Feldman, S.I., Wellman, M.P. (eds.) ACM Conference on Electronic Commerce, 1999. pp. 78–86. ACM (1999), https://doi.org/10.1145/336992.337012

[22] 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-sum-with-cardinality. In: EuroS&P 2020. pp. 370–389. IEEE (2020), https://doi.org/10.1109/EuroSP48549.2020.00031

[23] Kissner, L., Song, D.X.: Privacy-preserving set operations. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 241–257. Springer, Heidelberg (Aug 2005)

[24] Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 818–829. ACM Press (Oct 2016)

[25] Lepoint, T., Patel, S., Raykova, M., Seth, K., Trieu, N.: Private join and compute from PIR with default. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part II. LNCS, vol. 13091, pp. 605–634. Springer, Heidelberg (Dec 2021)

[26] Li, M., Cao, N., Yu, S., Lou, W.: Findu: Privacy-preserving personal profile matching in mobile social networks. In: INFOCOM,2011. pp. 2435–2443. IEEE (2011), https://doi.org/10.1109/INFCOM.2011.5935065

[27] Meadows, C.A.: 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. pp. 134–137. IEEE Computer Society (1986), https://doi.org/10.1109/SP.1986.10022

[28] Miao, P., Patel, S., Raykova, M., Seth, K., Yung, M.: Two-sided malicious security for private intersection-sum with cardinality. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 3–33. Springer, Heidelberg (Aug 2020)

[29] Nagaraja, S., Mittal, P., Hong, C.Y., Caesar, M., Borisov, N.: BotGrep: Finding P2P bots with structured graph analysis. In: USENIX Security 2010. pp. 95–110. USENIX Association (Aug 2010)

[30] Narayanan, A., Thiagarajan, N., Lakhani, M., Hamburg, M., Boneh, D.: Location privacy via private proximity testing. In: NDSS 2011. The Internet Society (Feb 2011)

[31] Narayanan, G.S., Aishwarya, T., Agrawal, A., Patra, A., Choudhary, A., Rangan, C.P.: Multi party distributed private matching, set disjointness and cardinality of set intersection with information theoretic security. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 09. LNCS, vol. 5888, pp. 21–40. Springer, Heidelberg (Dec 2009)

[32] Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: SpOT-light: Lightweight private set intersection from sparse OT extension. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 401–431. Springer, Heidelberg (Aug 2019)

[33] Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: PSI from PaXoS: Fast, malicious private set intersection. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part II. LNCS, vol. 12106, pp. 739–767. Springer, Heidelberg (May 2020)

[34] Pinkas, B., Schneider, T., Segev, G., Zohner, M.: Phasing: Private set intersection using permutation-based hashing. In: Jung, J., Holz, T. (eds.) USENIX Security 2015. pp. 515–530. USENIX Association (Aug 2015)

[35] Pinkas, B., Schneider, T., Tkachenko, O., Yanai, A.: Efficient circuit-based PSI with linear communication. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part III. LNCS, vol. 11478, pp. 122–153. Springer, Heidelberg (May 2019)

[36] Pinkas, B., Schneider, T., Weinert, C., Wieder, U.: Efficient circuit-based PSI via cuckoo hashing. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 125–157. Springer, Heidelberg (Apr / May 2018)

[37] Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: Fu, K., Jung, J. (eds.) USENIX Security 2014. pp. 797–812. USENIX Association (Aug 2014)

[38] Rindal, P., Rosulek, M.: Improved private set intersection against malicious adversaries. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part I. LNCS, vol. 10210, pp. 235–259. Springer, Heidelberg (Apr / May 2017)

[39] Rindal, P., Rosulek, M.: Malicious-secure private set intersection via dual execution. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1229–1242. ACM Press (Oct / Nov 2017)

[40] Rindal, P., Schoppmann, P.: VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part II. LNCS, vol. 12697, pp. 901–930. Springer, Heidelberg (Oct 2021)

[41] Vaidya, J., Clifton, C.: Secure set intersection cardinality with application to association rule mining. J. Comput. Secur. 13(4), 593–622 (2005), http://content.iospress.com/articles/journal-of-computer-security/jcs223

[42] Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS. pp. 162–167. IEEE Computer Society Press (Oct 1986)