Suda: An Efficient and Secure Unbalanced Data Alignment Framework for Vertical Privacy-Preserving Machine Learning

Lushan Song
Fudan University & ByteDance

Qizhi Zhang * †
ByteDance

Yu Lin
ByteDance

Haoyu Niu
Fudan University

Daode Zhang
ByteDance

Zheng Qu
Fudan University

Weili Han
Fudan University

Jue Hong
ByteDance

Quanwei Cai
ByteDance

Ye Wu
ByteDance

Abstract

Secure data alignment, which securely aligns the data between parties, is the first and crucial step in vertical privacy-preserving machine learning (VPPML). Practical applications, e.g. advertising, require VPPML for personalized services. Meanwhile, the data held by parties in these applications are usually unbalanced. Existing secure unbalanced data alignment approaches typically rely on Cuckoo Hashing, which introduces redundant data outside the intersection, leading to significantly increasing communication size during secure training in VPPML. Though secure shuffle operations can trim these redundant data, these operations would incur huge communication overhead. As a result, these secure approaches should be optimized for efficiency in VPPML scenarios.

In this paper, we propose Suda, an efficient and secure unbalanced data alignment framework for VPPML. By leveraging polynomial-based operations rather than Cuckoo Hashing, Suda efficiently, directly, and exclusively outputs data shares in the intersection without expensive secure shuffle operations. Consequently, Suda efficiently and seamlessly aligns with secure training in VPPML. Specifically, we first design a novel and efficient batch private information retrieval (PIR) protocol based on the oblivious polynomial reduction and evaluation protocols. Second, we design a batch PIR-to-share protocol extended from the batch PIR protocol with the oblivious polynomial interpolation protocol. Note that the batch PIR-to-share protocol securely obtains feature shares rather than the plaintext features which are the outputs of the batch PIR protocol. Comprehensive experiment results demonstrate that: (1) Suda outperforms the state-of-the-art secure data alignment framework by 31.14× ∼ 210.78× in communication size and up to 8.21× in running time; and (2) Suda outperforms the state-of-the-art batch PIR framework by up to 11.53× in server time.

*Equal contribution.
†Corresponding author. Email: zhangqizhi.zqz@bytedance.com

1 Introduction

Vertical privacy-preserving machine learning (VPPML) [7, 9, 21, 33, 36, 39] enables multiple parties with vertically distributed data, i.e. parties share the same sample IDs for common samples but hold different features or labels, to collaboratively perform machine learning with privacy preservation. E.g., in the advertising scenario [37], an Internet company, which holds user IDs and all user features, collaborates with an advertiser, which holds user IDs and user labels, can leverage VPPML to train a click-through rate predicting model.

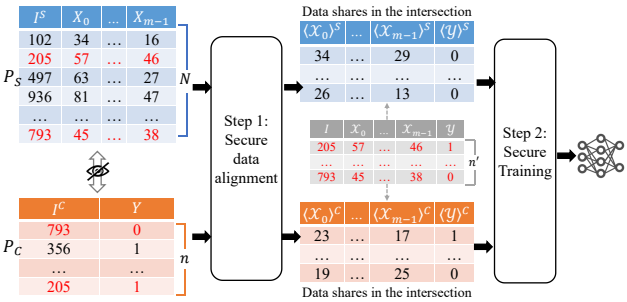

Figure 1: A brief overview of unbalanced two-party VPPML, where the server P_S holds larger data with size N, and the client P_C holds smaller data with size n.

As is shown in Figure 1, secure data alignment is the first and crucial step of VPPML. The parties in VPPML first employ secure data alignment to securely align their data to form a training set, which consists of data shares in the intersection of their data. Then, the parties collaboratively perform secure training over the training set. In the above scenario, designers usually face an unbalanced setting, where one party holds larger data with size N, while another party holds smaller data with size n and $n \ll N$ [36]. For instance, the Internet company typically holds billions of samples, while the advertiser only holds millions of samples.

However, existing secure unbalanced data alignment approaches, e.g. unbalanced circuit-based private set intersec-

tion (circuit-PSI) [12, 34], are inefficient for VPPML. These approaches typically use Cuckoo Hashing to enhance efficiency, which introduces redundant data outside the intersection, leading to significantly increasing communication size during secure training in VPPML. Specifically, the circuit-PSI outputs data shares with size εn, where ε (> 1) is a constant. If a sample is in the intersection, parties obtain boolean shares of $b = 1$ and data shares in the intersection; otherwise, they obtain boolean shares of $b = 0$ and zero shares (redundant data shares). These redundant data shares, as part of the training set, are input into the secure training, which increases the communication size of secure training in VPPML.

Though these redundant data can be trimmed by secure shuffle operations, these secure shuffle operations would incur huge communication overhead. For example, Liu *et al.* [22] proposed `iPrivJoin` to securely trim redundant data outside the intersection. They improve circuit-PSI and shuffle the outputs of the circuit-PSI, then reveal b and finally, trim samples outside the intersection. However, the additional secure shuffle operations introduce huge communication overhead to `iPrivJoin`, making `iPrivJoin` inefficient.

As a result, a significant technical challenge arises: *How to efficiently achieve secure unbalanced data alignment, which directly and exclusively outputs data shares in the intersection for VPPML without secure shuffle operations?*

1.1 Our Contributions

In this paper, we propose `Suda`, an efficient and secure unbalanced data alignment framework for VPPML, to overcome the above challenge. Our contributions are summarized as follows:

- We propose an efficient and secure unbalanced data alignment framework `Suda` based on polynomial operations. `Suda` directly and exclusively outputs data shares in the intersection, thus efficiently and seamlessly aligning with the secure training in VPPML.
- We propose two key protocols based on polynomial operations to enhance the efficiency and security: (1) the bath PIR protocol based on the oblivious polynomial reduction protocol and the oblivious polynomial evaluation protocol; (2) the batch PIR-to-share protocol extended upon the batch PIR protocol with the oblivious polynomial interpolation protocol.
- Comprehensive experiment results demonstrate that `Suda` achieves significant enhancement over the state-of-the-art frameworks. Specifically, compared to the secure data alignment framework `iPrivJoin` [22] and the circuit-PSI framework [30], `Suda` exhibits superior performance during the secure data alignment phase. Meanwhile, during the training phase, [30] causes about $2\times$ more communication size compared to `Suda` and `iPrivJoin`. Besides, `Suda` outperforms `iPrivJoin` by $31.14\times \sim 210.78\times$ in communication size, and by up to $8.21\times$ in running time. Notably, `Suda` achieves close performance in both WAN and LAN settings. Additionally, compared to the batch PIR framework `PIRANA` [20], `Suda` achieves $11.53\times$ faster in server time.

1.2 Our Approaches

The key insight driving our proposed `Suda` is that, by leveraging polynomial-based operations rather than Cuckoo Hashing, `Suda` efficiently, directly, and exclusively outputs data shares in the intersection, without expensive secure shuffle operations. Therefore, `Suda`'s output efficiently and seamlessly aligns with secure training in VPPML. Specifically, two parties (a server and a client) align their IDs in the intersection without leaking any sensitive information about the raw data. Then the client utilizes its IDs in the intersection to directly retrieve the corresponding feature shares.

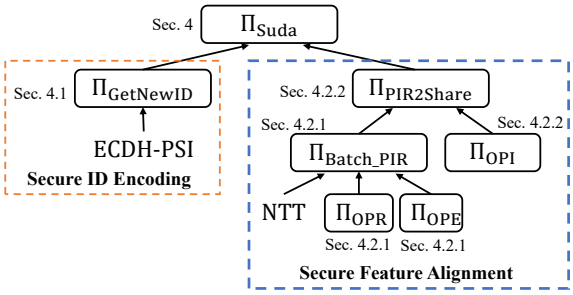

Figure 2: Technical roadmap of `Suda`. The protocols in the rounded rectangles are proposed in `Suda`.

As is shown in Figure 2, we are motivated to undertake two steps, i.e. secure ID encoding and secure feature alignment.

- (Step 1) We design a get new ID protocol $\prod_{\text{GetNewID}}$ based on ECDH-PSI. This protocol enables the server to generate new IDs and securely synchronize them in the intersection with the client, thereby achieving ID alignment.
- (Step 2) This step consists of two key protocols, i.e. the batch PIR protocol and the batch PIR-to-share protocol, based on polynomial operations. (1) We design a novel and efficient batch PIR protocol $\prod_{\text{Batch\_PIR}}$. Specifically, the server encodes its features into polynomials using the Number Theoretic Transform (NTT) based on its new IDs. The client then retrieves the plaintext features by evaluating these polynomials at its new IDs in the intersection. Considering the unbalanced setting, large server-side data size results in high-degree encoded polynomials, significantly increasing computational and communication costs during evaluation. To resolve this technical issue, we design an efficient oblivious polynomial reduction protocol $\prod_{\text{OPR}}$ based on the polynomial module rather than Cuckoo Hashing, significantly reducing the polynomial degree. This makes our batch PIR protocol up to $11.53\times$ faster than the

state-of-the-art batch PIR framework [20], as mentioned in Section 1.1. Additionally, to avoid the client obtaining the information outside the intersection during the evaluation, we design a novel oblivious polynomial evaluation protocol $\prod_{\text{OPE}}$ based on random polynomial masking. (2) To securely obtain feature shares rather than the plaintext features, we design a batch PIR-to-share protocol $\prod_{\text{PIR2Share}}$ extended from the batch PIR protocol. Specifically, the client retrieves the feature shares rather than plaintext features by evaluating the polynomials encoded by feature shares at its new IDs in the intersection. To achieve this, we design a novel oblivious polynomial interpolation $\prod_{\text{OPI}}$ protocol to secure interpolate feature shares into polynomials. During the process of the secure feature alignment step, two parties efficiently and directly obtain feature shares in the intersection, thus eliminating the need to perform secure shuffle operations. Consequently, `Suda` outperforms `iPrivJoin` by up to 210.78× in communication size as mentioned in Section 1.1.

2 Preliminaries

2.1 Homomorphic Encryption

Homomorphic encryption (HE) [32] enables direct computation on encrypted data. It ensures that the encrypted result, which is computed from the ciphertexts, matches the result of the same operation performed on plaintext after decryption. In `Suda`, we utilize two types of HEs, i.e. additive homomorphic encryption (AHE) and leveled fully homomorphic encryption (LFHE) by leveraging their strengths. Specifically, we use AHE to encrypt labels, taking advantage of its shorter ciphertexts, and use LFHE to encrypt polynomials, taking advantage of its efficiency in polynomials encryption.

Additive Homomorphic Encryption. AHE, for example, Paillier HE [28], supports unlimited add operations between ciphertexts as well as multiplication operations between a ciphertext and a constant. The implementation of AHE includes the following algorithm:

- **Key Generation.** Generate the public and private key pair (apk, ask).
- **Encryption.** Given a plaintext pt, outputs a ciphertext ct encrypted by the public key apk, i.e. $ct = \text{A.Enc}_{apk}(pt)$.
- **Homomorphic Operations.** AHE mainly supports three types of operations, i.e. homomorphic addition $\boxplus$, homomorphic subtraction $\boxminus$, and homomorphic constant multiplication $\boxtimes$.
- **Decryption.** Given a ciphertext ct, outputs a plaintext pt decrypted by the private key ask, i.e. $pt = \text{A.Dec}_{ask}(ct)$.

Leveled Fully Homomorphic Encryption. LFHE, such as Brakerski-Gentry-Vaikuntanathan (BGV) [6] and Brakerski-Fan-Vercauteren (BFV) [11] allows a limited number of arbitrary operations, including addition and multiplication, to be performed on ciphertexts.

LFHE is typically defined over a ring $R = \mathbb{Z}/(x^d+1)$ of polynomial modulo x^d+1 with d being a power of 2. In BGV, the plaintext space is $R_p = \mathbb{Z}_p[x]/(x^d+1)$, and the ciphertext space is $(R_q)^2 = (\mathbb{Z}_q[x]/(x^d+1))^2$, where p and q are plaintext module and ciphertext module, respectively, and $p \ll q$. To correctly compute the L depth circuit over ciphertext, the ciphertext module q is required to be $logq \approx (L+1)\log p$. Besides, the size of plaintext and ciphertext is $d\log p$ and $2d\log q$, respectively. The implementation of LFHE includes the following algorithm:

- **Key Generation.** Generate the public and private key pair (fpk, fsk).
- **Encryption.** Given a plaintext pt, outputs a ciphertext ct encrypted by the public key fpk, i.e. $ct = \text{F.Enc}_{fpk}(pt)$.
- **Homomorphic Operations.** In addition to the three types of operations supported by AHE, LFHE also supports homomorphic multiplication, i.e. $\text{F.Enc}_{fpk}(pt_1) \boxtimes \text{F.Enc}_{fpk}(pt_2) = \text{F.Enc}_{fpk}(pt_1 \cdot pt_2)$.
- **Decryption.** Given a ciphertext ct, outputs a plaintext message pt decrypted by the private key fsk, i.e. $pt = \text{F.Dec}_{fsk}(ct)$.

2.2 Elliptic Curve Cryptography

Elliptic curve cryptography (ECC) is proposed by Koblitz [18] and Miller [25] and widely adopted public-key cryptographic algorithm that offers the same functionality as RSA. The notable advantage of ECC is that it uses much shorter key sizes than RSA to achieve the same security level [5, 38]. Therefore ECC saves bandwidth and has faster implementations. In `Suda`, we use ECC for key exchange to perform secure ID encoding. Besides, we also use ECC to encrypt binary labels, taking advantage of its shorter ciphertexts compared to homomorphic encryption.

The elliptic curve E typically defined over the prime finite field $\mathbb{F}_t$ takes the short Weierstrass form:

$$y^2 = x^3 + ax + b \bmod t, \tag{1}$$

where t is a prime number, and $a, b \in \mathbb{F}_t$ satisfies $4a^3 + 27b^2 \not\equiv 0 \pmod t$. A point (x, y) on the elliptic curve satisfies Equation (1), and the point at infinity o is said to be on the elliptic curve ($P + o = P$ for any $P \in E$). ECC can encrypt a plaintext message M into a ciphertext $C \in E$ using a public key Q, i.e. $C = \text{E.Enc}_Q(M)$, and decrypt the ciphertext back into plaintext message using private key κ, i.e. $M = \text{E.Dec}_\kappa(C)$.

ECDH-PSI: ECC can be used to implement DH-PSI [38], which is defined as ECDH-PSI. We define P_S and P_C as two parties executing the ECDH-PSI protocol, holding the data $s = \{s_0, \ldots, s_{N-1}\}$ and $c = \{c_0, \ldots, c_{n-1}\}$ respectively, and

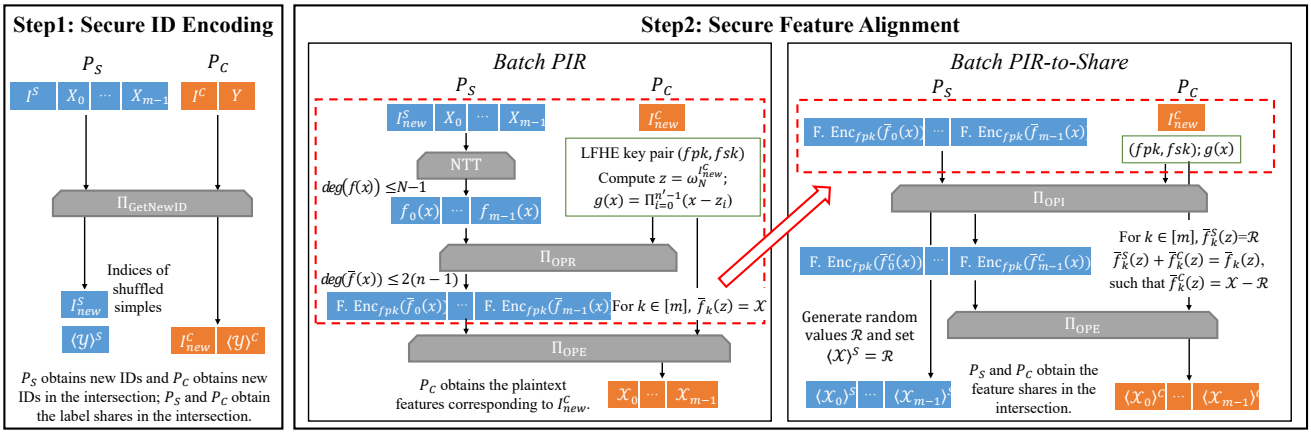

Figure 3: Overview of our batch PIR-based secure unbalanced data alignment framework Suda.

$H(\cdot)$ as a hash function that hashes an element to a point on the elliptic curve. The ECDH-PSI protocol executes the following steps:

(1) P_S and P_C generate their private key α and β respectively.

(2) P_S computes $\alpha H(s_0),\ldots,\alpha H(s_{N-1})$ and sends them to P_C, which computes $\beta H(c_0),\ldots,\beta H(c_{n-1})$.

(3) P_S and P_C exchange their computation results in (2).

(4) P_S computes $\alpha\beta H(c_0),\ldots,\alpha\beta H(c_{n-1})$ and P_C computes $\beta\alpha H(s_0),\ldots,\beta\alpha H(s_{N-1})$.

(5) The intersection is $\{\beta\alpha H(s_0),\ldots,\beta\alpha H(s_{N-1})\} \cap \{\alpha\beta H(c_0),\ldots,\alpha\beta H(c_{n-1})\}$.

2.3 Secret Sharing

Secret Sharing is a basic technology in secure multi-party computation. The main idea of secret sharing is to break a secret value into multiple shares, each of which is held by a party. In this paper, we adopt additive secret sharing as our secret sharing semantic, i.e. the sum of the shares is the secret value. For example, P_S, who holds the secret value $x \in \mathbb{F}_p$, wants to secret share this secret value with another party P_C. To do this, P_S first generates a random value $r \in \mathbb{F}_p$ as its share $\langle x\rangle^S = r$, and then sends $\langle x\rangle^C = x - r \bmod \mathbb{F}_p$ to another party P_C. In particular, if $x \in \{0,1\}$ is an binary bit, then $\langle x\rangle^S \oplus \langle x\rangle^C = x$.

2.4 Number Theoretic Transform

NTT [1] is defined over a finite field $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$, where p is typically a prime. It is a mathematical transform similar to the Fast Fourier Transform (FFT) but based on number theory and transforms a sequence of numbers into another sequence of numbers. NTT is primarily used to accelerate polynomial multiplication operations.

An N-point NTT is typically denoted as NTT_N. Given an input sequence $\boldsymbol{a}$ of length N, i.e. $\boldsymbol{a} = \{a_0,\ldots,a_{N-1}\}$, where each element $a_i \in \mathbb{Z}_p$ for $i = \{0,\ldots,N-1\}$, the discrete form of N-point NTT can be represented as:

$$A_i = \mathrm{NTT}_N(\boldsymbol{a})_i = \sum_{j=0}^{N-1} a_j \cdot \omega_N^{ij} \bmod p, \tag{2}$$

where $\boldsymbol{A}$ is the transformed output sequence of length N, i.e. $\boldsymbol{A} = \{A_0,\ldots,A_{N-1}\}$, each element $A_i \in \mathbb{Z}_p$ for $i = \{0,\ldots,N-1\}$ and ω_N is the $N-th$ primitive root of unity in $\mathbb{Z}_p$. ω_N satisfies the conditions $\omega_N^N \equiv 1 \bmod p$, and $\forall i < N, \omega_N^i \neq 1 \bmod p$, where $p \equiv 1 \bmod N$. The inverse NTT (INTT) is similar to NTT. An N-point INFF is typically denoted as INTT_N and can be represented as:

$$a_i = \mathrm{INTT}_N(\boldsymbol{A})_i = N^{-1} \sum_{j=0}^{N-1} A_j \cdot \omega_N^{-ij} \bmod p, \tag{3}$$

where $i = \{0,\ldots,N-1\}$, N^{-1} and ω_N^{-1} are the inverse of N and ω_N over the finite filed $\mathbb{Z}_p$, respectively, i.e. $N \cdot N^{-1} \equiv 1 \bmod p$ and $\omega_N \cdot \omega_N^{-1} \equiv 1 \bmod p$.

By multiplying two N-degree polynomials, FFT algorithms can reduce the complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N\log N)$. Moreover, FFT algorithms can be applied to NTT by simply replacing complex arithmetic with modular arithmetic. As a result, NTT can also achieve a complexity reduction from $\mathcal{O}(N^2)$ to $\mathcal{O}(N\log N)$.

3 Overview

We summary the notations used in this paper in Appendix A.

3.1 Scenarios

We consider an unbalanced two-party scenario. The server P_S holds larger data $D^S = I^S||X$ with size N, consisting of sample IDs I^S and m-dimensional features X, where $||$ refers to concatenation of two elements, $I^S = \{id_0^S,\ldots,id_{N-1}^S\}$ and

$X = X_0||\ldots||X_{m-1}$. The client P_C holds smaller data $D^C = I^C||Y$ with size n ($n \ll N$), consisting of sample IDs I^C and labels Y, where $I^C = \{id_0^C, \ldots, id_{n-1}^C\}$ and $Y = \{y_0, \ldots, y_{n-1}\}$. After secure unbalanced data alignment, P_S and P_C obtain the shares $\langle\mathcal{X}\rangle^S||\langle\mathcal{Y}\rangle^S$ and $\langle\mathcal{X}\rangle^C||\langle\mathcal{Y}\rangle^C$ in the intersection with size n', respectively. Then, they could use the data shares as input to train a machine learning model securely.

3.2 Overview of Suda

As is shown in Figure 3, we divide Suda into two steps: secure ID encoding and secure feature alignment.

(1) The secure ID encoding step is to align parties' IDs and obtain label shares $\langle\mathcal{Y}\rangle$ in the intersection. To achieve this, we design a get new ID protocol $\prod_{\text{GetNewID}}$ (Protocol 2). Firstly, P_S sets the shuffled sample indices as the new IDs I_{new}^S (i.e. $I_{new}^S = \{0, \ldots, N-1\}$), which are used to reduce the computational complexity of feature encoding. Then we synchronize the new IDs of P_S in the intersection with P_C based on ECDH-PSI, thereby achieving ID alignment. At the same time, $\prod_{\text{GetNewID}}$ achieves the label sharing between two parties.

(2) The secure feature alignment step is to get the feature shares $\langle\mathcal{X}\rangle = \langle\mathcal{X}\rangle_0||\ldots||\langle\mathcal{X}\rangle_{m-1}$ in the intersection, which is the core of Suda. Here, we leverage polynomial operations to enhance our efficiency and security. We first design a novel and efficient batch PIR protocol $\prod_{\text{Batch\_PIR}}$ (Protocol 3), which encodes m-dimension features into m polynomials and evaluates these polynomials at the new IDs in the intersection. The process of batch PIR protocol includes the following three steps.

- **Feature Encoding.** P_S uses NTT to encode its features into polynomials $f_k(x)$ for $k \in [m]$ with degree $deg(f_k(x)) \leq N-1$ based on its new IDs, thereby reducing the encoding computational complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N\log N)$.
- **Oblivious Polynomial Reduction.** To further reduce the computational and communication costs, we design an oblivious polynomial reduction protocol $\prod_{\text{OPR}}$ (Protocol 4), which reduces the degree of $f_k(x)$ from $\mathcal{O}(N)$ to $\mathcal{O}(n)$ and obtains reduced polynomials $\bar{f}_k(x)$, based on the polynomial module.
- **Oblivious Polynomial Evaluation.** We design an efficient oblivious polynomial evaluation protocol $\prod_{\text{OPE}}$ (Protocol 5), which enables P_C to securely evaluate the reduced polynomials $\bar{f}_k(x)$ to obtain the features corresponding to its queries without obtaining any information irrelevant to its queries, based on random polynomial masking.

Second, to obtain feature shares rather than plaintext features which are outputs of the batch PIR protocol, we further design an efficient and secure batch PIR-to-share protocol $\prod_{\text{PIR2Share}}$ (Protocol 6) extended by batch PIR protocol with oblivious polynomial interpolation protocol $\prod_{\text{OPI}}$ (Protocol 7). The main idea of $\prod_{\text{PIR2Share}}$ is to enable P_C to evaluate the polynomials encoded by feature shares $\mathcal{X} - \mathcal{R}$, where $\mathcal{R} = \{r_{i,k}\}_{i\in[n'],k\in[m]}$ is a random value set. Thereby, P_C retrieves the feature shares $\mathcal{X} - \mathcal{R}$ corresponding to its new IDs in the intersection rather than the raw features $\mathcal{X}$. To realize this idea, we design a novel oblivious polynomials interpolation protocol to securely interpolate polynomials $\bar{f}_k^S(x)$, such that $\bar{f}_k^S(z_i) = r_{i,k}$ for $i \in [n'], k \in [m]$. Then polynomials encoding $\mathcal{X} - \mathcal{R}$ can be computed as $\bar{f}_k^C(x) = \bar{f}_k(x) - \bar{f}_k^S(x)$.

3.3 Security Model

In Suda, we consider security against static probabilistic polynomial time (PPT) semi-honest adversaries, the same as the prior works [12, 22]. A static PPT semi-honest adversary $\mathcal{A}$ corrupts one of the parties P_S or P_C at the beginning of the protocol and follows the protocol specification honestly. However, during the processing, $\mathcal{A}$ attempts to infer additional information about the input of the other party by analyzing the corrupted party's view. Here, we use the standard simulation-based security definition as follows.

Definition 1. *Let Π be a two-party protocol that computes an ideal functionality $\mathcal{F}(i_S, i_C) = (\mathcal{F}_S, \mathcal{F}_C)$, where i_S and i_C are the inputs of P_S and P_C, respectively. Let $\text{view}_S^\Pi(i_S, i_C)$ and $\text{view}_C^\Pi(i_S, i_C)$ be the views (consist of input, messages that are sent or received, random tape, and the output) of P_S and P_C during the execution of Π on inputs i_S, i_C. Besides, let $\text{out}(i_S, i_C)$ be the output of Π. We say that the two-party protocol Π securely computes $\mathcal{F}$ on (i_S, i_C) against a semi-honest adversary $\mathcal{A}$, if there exist probabilistic polynomial-time simulators $\mathcal{S}_S$ and $\mathcal{S}_C$ such that,*

$$\{\mathcal{S}_S(i_S, \mathcal{F}_S(i_S, i_C)), \mathcal{F}(i_S, i_C)\} \approx_c \{\text{view}_S^\Pi(i_S, i_C), \text{out}(i_S, i_C)\}$$
$$\{\mathcal{S}_C(i_C, \mathcal{F}_S(i_S, i_C)), \mathcal{F}(i_S, i_C)\} \approx_c \{\text{view}_C^\Pi(i_S, i_C), \text{out}(i_S, i_C)\},$$

where $\approx_c$ represents computational indistinguishability.

The security proof of our proposed protocols is shown in Appendix E.

3.4 Setup and Online Phase

In Suda, all the computations involve parties' input. Therefore, there is no offline phase in Suda. Similar to prior works [12, 20], we divide our protocol into two phases. (1) Setup phase: the server preprocess its input locally to enhance online efficiency. Specifically, the setup phase in Suda encompasses tasks such as shuffling the server's data locally and encoding the feature into polynomials. (2) Online phase: two parties perform computations on the preprocessed data and exchange necessary data.

4 Design of Suda

We formalize Suda as the functionality $\mathcal{F}_{\text{Suda}}$, which is shown in Figure 4, where $\mathcal{F}_{\text{Suda}}$ receives inputs $D^S = I^S||X$ and $D^C =$

$I^C||Y$ from the server P_S and the client P_C, respectively. $\mathcal{F}_{\text{Suda}}$ outputs feature and label shares $\langle\mathcal{X}\rangle^S||\langle\mathcal{Y}\rangle^S$ in the intersection to P_S and $\langle\mathcal{X}\rangle^C||\langle\mathcal{Y}\rangle^C$ to P_C, where $\langle\mathcal{X}\rangle^S+\langle\mathcal{X}\rangle^C=\mathcal{X}$ and $\langle\mathcal{Y}\rangle^S+\langle\mathcal{Y}\rangle^C=\mathcal{Y}$. Protocol 1 securely and efficiently achieves the functionality $\mathcal{F}_{\text{Suda}}$.

Functionality $\mathcal{F}_{\text{Suda}}$

Parameters: Two parties P_S and P_C, where P_S holds larger data $D^S=I^S||X$ with size N, consisting of sample IDs I^S and $m-$dimensional features X, P_C holds smaller data $D^C=I^C||Y$ with size n ($n<<N$), consisting of sample IDs I^C and labels Y. The intersection size is n'.

Functionality:

- Wait for input $D^S=I^S||X$ from P_S.
- Wait for input $D^C=I^C||Y$ from P_C.
- Find the intersection $\mathcal{I}=I^S\cap I^C$ and the corresponding features and labels $\mathcal{X}||\mathcal{Y}$.
- Sample $\langle\mathcal{X}\rangle^S$, $\langle\mathcal{X}\rangle^C$ and $\langle\mathcal{Y}\rangle^S$, $\langle\mathcal{Y}\rangle^C$ with size n' uniformly, such that $\langle\mathcal{X}\rangle^S+\langle\mathcal{X}\rangle^C=\mathcal{X}$ and $\langle\mathcal{Y}\rangle^S+\langle\mathcal{Y}\rangle^C=\mathcal{Y}$.
- Output feature and label shares $\langle\mathcal{X}\rangle^S||\langle\mathcal{Y}\rangle^S$ to P_S and $\langle\mathcal{X}\rangle^C||\langle\mathcal{Y}\rangle^C$ to P_C .

Figure 4: Ideal functionality of $\mathcal{F}_{\text{Suda}}$

Protocol 1: $\prod_{\text{Suda}}$

Input: P_S inputs its data $D^S=I^S||X$ with size N and P_C inputs its data $D^C=I^C||Y$ with size n.

Output: P_S and P_C output feature and label shares $\langle\mathcal{X}\rangle^S||\langle\mathcal{Y}\rangle^S$ and $\langle\mathcal{X}\rangle^C||\langle\mathcal{Y}\rangle^C$ in the intersection, respectively.

1: **Secure ID Encoding:**

- P_S and P_C invoke the get new ID protocol $\prod_{\text{GetNewID}}$ (Protocol 2), where P_S inputs $D^S=I^S||X$, and P_C inputs $D^C=I^C||Y$. After the execution, P_S obtains new IDs I^S_{new} and P_C obtains new IDs I^C_{new} in the intersection. Besides, P_S and P_C obtain label shares $\langle\mathcal{Y}\rangle^S$ and $\langle\mathcal{Y}\rangle^C$ in the intersection, respectively.

2: **Secure Feature alignment:**
P_S and P_C invoke the batch PIR-to-Share protocol $\prod_{\text{PIR2Share}}$ (Protocol 6), where P_S inputs $I^S_{new}||X$, and P_C inputs I^C_{new}. After the execution, P_S and P_C obtain feature shares $\langle\mathcal{X}\rangle^S$ and $\langle\mathcal{X}\rangle^C$ in the intersection, respectively.

4.1 Secure ID Encoding

In the secure ID encoding step, we design a get new ID protocol $\prod_{\text{GetNewID}}$ (Protocol 2) to securely align two parties' IDs and obtain the label shares in the intersection, based on ECDH-PSI. We elaborate on this protocol as follows.

In the setup phase, P_S first locally shuffles its data to prevent P_C from learning the raw order of P_S's samples in the intersection. Then P_S sets the sample indices as new IDs, i.e. $I^S_{new}=\{0,\ldots,N-1\}$. Besides, P_S generates the ECC private key to encrypt its IDs.

Protocol 2: $\prod_{\text{GetNewID}}$

Input: P_S inputs its data $D^S=I^S||X$, P_C inputs its data $D^C=I^C||Y$.

Output: P_S outputs its new IDs I^S_{new} and P_C outputs its new IDs I^C_{new} in the intersection. Besides, P_S and P_C output label shares $\langle\mathcal{Y}\rangle^S$ and $\langle\mathcal{Y}\rangle^C$ in the intersection, respectively.

Parameters: Hash function $H(\cdot)$ used for hashing an element to a point on the elliptic curve.

Setup:

1: P_S shuffles its data locally, and then sets its new ID as $I^S_{new}=\{0,\ldots,N-1\}$.
2: P_S generates an ECC private key α and computes $\alpha H(I^S)\leftarrow\{\alpha H(id^S_i)\}_{i\in[N]}$.

Online:

1: P_C shuffles its data locally.
2: P_C generates an ECC private key β and computes $\beta H(I^C)\leftarrow\{\beta H(id^C_j)\}_{j\in[n]}$. Besides, P_C generates AHE key pair (apk,ask) and encrypts the label, i.e. $\text{A.Enc}_{apk}(Y)\leftarrow\{\text{A.Enc}_{apk}(y_j)\}_{j\in[n]}$. P_C sends $\beta H(I^C)$ and $\text{A.Enc}_{apk}(Y)$ to P_S.
3: P_S computes $\alpha\beta H(I^C)$ and $\text{A.Enc}_{apk}(Y-R)$ (R is the random value set) and shuffles them to $\alpha\beta H(I^C)_\pi\leftarrow\{\alpha\beta H(id^C_{\pi(j)})\}_{j\in[n]}$ and $\text{A.Enc}_{apk}(Y-R)_\pi\leftarrow\{\text{A.Enc}_{apk}(y_{\pi(j)}-r_{\pi(j)})\}_{j\in[n]}$ respectively, where π is a permutation function. Then P_S sends $\alpha\beta H(I^C)_\pi$, $\text{A.Enc}_{apk}(Y-R)_\pi$ and $\alpha H(I^S)$ to P_C.
4: P_C computes $\alpha H(I^C)_\pi\leftarrow\beta^{-1}\alpha\beta H(I^C)_\pi$ and $Y-R\leftarrow\text{A.Dec}_{ask}(\text{A.Enc}_{apk}(Y-R)_\pi)$. Then P_C sends $\Delta\leftarrow\{\pi(j)|\alpha H(id^C_{\pi(j)})=\alpha H(id^S_i)\}_{i\in[N],j\in[n]}$ to P_S.
5: P_C returns $I^C_{new}\leftarrow\{i|\alpha H(id^C_{\pi(j)})=\alpha H(id^S_i)\}_{i\in[N],j\in[n]}$ and label shares $\langle\mathcal{Y}\rangle^C\leftarrow\{y_j-r_j\}_{j\in I^C_{new}}$. P_S returns label shares $\langle\mathcal{Y}\rangle^S\leftarrow\{r_{\pi(j)}\}_{\pi(j)\in\Delta}$.

In the online phase, P_C first locally shuffles its data to prevent P_S from inferring the indices of its samples in the intersection. Then P_C generates the ECC private key and AHE key pairs to encrypt its IDs and labels respectively. After that, P_C sends the encrypted IDs $\beta H(I^C)$ and labels $\text{A.Enc}_{apk}(Y)$ to P_S. Then P_S computes $\alpha\beta H(I^C)$, $\text{A.Enc}_{apk}(Y)\boxminus\text{A.Enc}_{apk}(R)=\text{A.Enc}_{apk}(Y-R)$, where $R=\{r_j\}_{j\in[n]}$ is random value set. To avoid P_C obtaining the raw IDs in the intersection, P_S shuffles $\alpha\beta H(I^C)$ and $\text{A.Enc}_{apk}(Y-R)$ using a permutation function π . Subsequently, P_S sends $\alpha\beta H(I^C)_\pi$ and $\text{A.Enc}_{apk}(Y-R)_\pi$ together with $\alpha H(I^S)$ to P_C. After that, P_C computes $\beta^{-1}\alpha\beta H(I^C)_\pi$ and decrypts $\text{A.Enc}_{apk}(Y-R)_\pi$ to obtain $\alpha H(I^C)_\pi$ and $(Y-R)_\pi$, respectively. Thus, P_C can obtain the new IDs $I^C_{new}=\{id'_0,\ldots,id'_{n'-1}\}=\{i|\alpha H(id^C_{\pi(j)})=\alpha H(id^S_i)\}_{j\in[n],i\in[N]}$ in the intersection by comparing $\alpha H(I^C)_\pi$

and $\alpha H(I^S)$, where n' is the intersection size. Essentially, I^C_{new} is the index set of the intersection elements in the $\alpha H(I^S)$ set. Additionally, P_C obtains the corresponding label shares $\langle \mathcal{Y} \rangle^C = \{y_j - r_j\}_{j \in I^C_{new}}$. Furthermore, to avoid P_S obtaining the raw IDs in the intersection, P_C sends the index set $\Delta = \{\pi(j) | \alpha H(id^C_{\pi(j)}) = \alpha H(id^S_i)\}_{j \in [n], i \in [N]}$ of the intersection elements in the shuffled set $\alpha H(I)^C_\pi$ instead of I^C_{new} to P_S. Finally, P_S obtains the label shares $\langle \mathcal{Y} \rangle^S = \{r_{\pi(j)}\}_{\pi(j) \in \Delta}$. Note that, $|I^C_{new}| = |\Delta| = n'$.

Regarding to the binary classification problem, i.e., the label is a binary value, we propose an optimized get new ID protocol $\prod_{\text{GetNewID2}}$ (Protocol 8) to further reduce the communication overhead in Appendix B.

4.2 Secure Feature Alignment

In the secure feature alignment step, parties P_S and P_C respectively get the feature shares $\langle \mathcal{X} \rangle^S$ and $\langle \mathcal{X} \rangle^C$ in the intersection, such that $\langle \mathcal{X} \rangle^S + \langle \mathcal{X} \rangle^C = \langle \mathcal{X} \rangle$.

4.2.1 Batch PIR

We formalize batch PIR as the functionality $\mathcal{F}_{\text{Batch\_PIR}}$, which is shown in Figure 5. $\mathcal{F}_{\text{batch\_PIR}}$ receives inputs $D^S = I^S_{new} || X$ from P_S, where $X = X_1 || \ldots || X_m = \{x_{i,k}\}_{i \in [N], k \in [m]}$, and queries $I^C_{new} = \{id'_0, \ldots, id'_{n'-1}\}$ from P_C. $\mathcal{F}_{\text{Batch\_PIR}}$ outputs set $\mathcal{X} = \mathcal{X}_0 || \ldots || \mathcal{X}_{m-1} = \{x_{id'_i,k}\}_{i \in [n'], k \in [m]}$ corresponding to P_C's queries. Because queries I^C_{new} are the new IDs in the intersection, the outputs $\mathcal{X}$ are the features in the intersection. The batch PIR protocol $\prod_{\text{Batch\_PIR}}$ (Protocol 3) securely and efficiently achieves the functionality $\mathcal{F}_{\text{Batch\_PIR}}$.

Functionality $\mathcal{F}_{\text{Batch\_PIR}}$

Parameters: Two parties P_S and P_C.

Functionality:

- Wait for input $D^S = I^S_{new} || X$ from P_S, where $X = X_0 || \ldots || X_{m-1} = \{x_{i,k}\}_{i \in [N], k \in [m]}$.
- Wait for input $I^C_{new} = \{id'_0, \ldots, id'_{n'-1}\}$ from P_C.
- Output $\mathcal{X} = \mathcal{X}_0 || \ldots || \mathcal{X}_{m-1} = \{x_{id'_i,k}\}_{i \in [n'], k \in [m]}$ to P_C.

Figure 5: Ideal functionality of $\mathcal{F}_{\text{Batch\_PIR}}$

As is shown in Section 3.2, the batch PIR protocol $\prod_{\text{Batch\_PIR}}$ consists of three steps. Below, we give more technical details to illustrate these steps. Note that the setup phase of the batch PIR protocol $\prod_{\text{Batch\_PIR}}$ (Protocol 3) is feature encoding and the online phase consists of the oblivious polynomial reduction step and oblivious polynomial evaluation step.

Feature Encoding. P_S encodes each column feature $X_k = \{x_{0,k}, \ldots, x_{N-1,k}\}$ of X into a polynomial $f_k(x) = \sum_{i=0}^{N-1} a_{i,k} x^i$ for $k \in [m]$. Here, we use NTT to reduce the encoding computation complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$.

Protocol 3: $\prod_{\text{Batch\_PIR}}$

Input: P_S inputs its data $D^S = I^S_{new} || X$, where $X = X_0 || \ldots || X_{m-1} = \{x_{i,k}\}_{i \in [N], k \in [m]}$, P_C inputs its queries $I^C_{new} = \{id'_0, \ldots, id'_{n'-1}\}$.

Output: P_C outputs set $\mathcal{X} = \mathcal{X}_0 || \ldots || \mathcal{X}_{m-1} = \{x_{id'_i,k}\}_{i \in [n'], k \in [m]}$ corresponding to its queries.

Setup:

1: **Feature Encoding:**
P_S encodes X into m polynomials $f_k(x)$ with $deg(f_k(x)) \leq N-1$ for $k \in [m]$ based on NTT.

Online:

1: **Oblivious Polynomial Reduction:**

- P_C generates LFHE key pair (fpk, fsk) and constructs $g(x) \leftarrow \prod_{i=0}^{n'-1}(x - z_i)$ where $z_i = \omega_N^{id'_i}$ for $i \in [n']$.
- P_S and P_C invoke the oblivious polynomial reduction protocol $\prod_{\text{OPR}}$ (Protocol 4), where P_S inputs $f_k(x)$ for $k \in [m]$, and P_C inputs (fpk, fsk) and $g(x)$. After the execution, P_S obtains the reduced polynomials after encrypting $\text{F.Enc}_{fpk}(\bar{f}_k(x))$ with $deg(\bar{f}_k(x)) \leq 2(n-1)$ for $k \in [m]$.

2: **Oblivious Polynomial Evaluation:**
P_S and P_C invoke the oblivious polynomial evaluation protocol $\prod_{\text{OPE}}$ (Protocol 5), where P_S inputs $\text{F.Enc}_{fpk}(\bar{f}_k(x))$ for $k \in [m]$, and P_C inputs $(fpk, fsk), g(x)$ and z_i for $i \in [n']$. After the execution, P_C obtains set $\mathcal{X} \leftarrow \{x_{id'_i,k} | x_{id'_i,k} = \bar{f}_k(z_i)\}_{i \in [n'], k \in [m]}$.

The encoding process can be represented as Equation (4):

$$x_{i,k} = \sum_{j=0}^{N-1} a_{j,k} \cdot (id^S_i)^j, \tag{4}$$

for $i \in [N]$. The computational complexity of solving Equation (4) is $\mathcal{O}(N^2)$.

During the secure ID encoding step, we assign N new IDs $I^S_{new} = \{0, \ldots, N-1\}$ to P_S, then P_S computes ω_N^i for $i \in [N]$, i.e. $\omega_N^0, \ldots, \omega_N^{N-1}$, where ω_N is the N-th primitive root of unity in $\mathbb{Z}_p$. After that, we assume N is a power of 2, and replace $id^S_0, \ldots, id^S_{N-1}$ with $\omega_N^0, \ldots, \omega_N^{N-1}$, then Equation (4) can be written as:

$$x_{i,k} = \sum_{j=0}^{N-1} a_{j,k} \cdot \omega_N^{ij} = NTT_N(\boldsymbol{a}_k)_i, \tag{5}$$

for $i \in [N]$, where $\boldsymbol{a}_k = \{a_{0,k}, \ldots, a_{N-1,k}\}$. Therefore, $f_k(x) = NTT_N(\boldsymbol{a}_k)$ and the computational complexity of feature encoding is reduced from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$.

If N is not a power of 2, we pad the length of X_k with zeros to a power of 2. Besides, P_S can perform feature encoding locally in the setup phase.

Oblivious Polynomial Reduction. The key rationale behind

the oblivious polynomial reduction is to reduce the degree of the polynomials based on the polynomial module, while guaranteeing that the reduced polynomials have the same results at the new IDs in the intersection. We elaborate on this rationale below.

We first propose Theorem 1, which is the core of our oblivious polynomial reduction, based on the polynomial module.

Theorem 1. *If $\mathbb{F}$ is a field, $f(x) \in \mathbb{F}[x]$, $g(x) = \prod_{i=0}^{n-1}(x - z_i) \in \mathbb{F}[x]$, $\bar{f}(x) \equiv f(x)$ mod $g(x)$, then $\bar{f}(z_i) = f(z_i)$ for $i \in [n]$.*

Proof. Due to $\bar{f}(x) \equiv f(x) \bmod g(x)$, there exists a polynomial $\varphi(x) \in \mathbb{F}[x]$ such that $\bar{f}(x) = f(x) + g(x)\varphi(x)$. Besides, $g(z_i) = 0$. Therefore, $\bar{f}(z_i) = f(z_i) + g(z_i)\varphi(z_i) = f(z_i)$. □

Besides, we observe that the polynomial $f_k(x)$ with $deg(f_k(x)) \leq N-1$ for $k \in [m]$ can be written as:

$$\begin{aligned} f_k(x) &= \sum_{i=0}^{N-1} a_{i,k} x^i \\ &= (a_{0,k} + a_{1,k}x + \cdots + a_{n-1,k}x^{n-1})x^{0\cdot n} \\ &+ (a_{n,k} + a_{n+1,k}x + \cdots + a_{2n-1,k}x^{n-1})x^{1\cdot n} \\ &+ \cdots \\ &+ (a_{\tau n,k} + a_{\tau n+1,k}x + \cdots + a_{N-1,k}x^{N-1-\tau n})x^{\tau\cdot n}, \end{aligned} \tag{6}$$

where $\tau = \lceil N/n \rceil - 1$. For simplicity, we represent Equation (6) in the form of Equation (7).

$$f_k(x) = \sum_{j=0}^{\tau} f_{k,j}(x) \cdot x^{jn}, \tag{7}$$

where $deg(f_{k,j}(x)) \leq n-1$. According to Theorem 1, let $g(x) = \prod_{i=0}^{n'-1}(x - z_i)$, $h_j(x) = x^{jn} \bmod g(x)$ and $\bar{f}_k(x) = \sum_{j=0}^{\tau} f_{k,j}(x) \cdot h_j(x)$, we have $\bar{f}_k(x) \equiv f_k(x) \bmod g(x)$ for $k \in [m]$. Therefore, $f_k(z_i) = \bar{f}_k(z_i)$ for $k \in [m]$. Here, the degree of $\bar{f}_k(x)$ is $deg\bar{f}_k(x) \leq 2(n-1) \ll N$, i.e. we reduce the degree of $f_k(x)$ from $N-1$ to $2(n-1)$ for $k \in [m]$.

Firstly, P_C generates LFHE key pair (fpk, fsk) and constructs $g(x) = \prod_{i=0}^{n'-1}(x - z_i)$ where $z_i = \omega_N^{id_i'}$ for $i \in [n']$. Here, we use $\omega_N^{id_i'}$ rather than the new IDs id_i' for $i \in [n']$ to construct polynomial $g(x)$, this is because the polynomials $f_k(x)$ is encoded by ω_N^i for $i \in [N]$ in the feature encoding step. Then P_S and P_C invoke the oblivious polynomial reduction protocol $\prod_{\text{OPR}}$ (Protocol 4).

As is shown in Protocol 4, P_S first constructs the polynomials $f_{k,j}(x)$ following Equation (6) and (7). Then P_C computes $h_j(x) = x^{jn} \bmod g(x)$ for $j \in [\lceil N/n \rceil]$. To prevent P_S from inferring the IDs of P_C in the intersection through $h_j(x)$, we encrypt $h_j(x)$ and then send them to P_S. Subsequently, P_S computes

$$\text{F.Enc}_{fpk}(\bar{f}_k(x)) = \sum_{j=0}^{\lceil N/n \rceil - 1} \text{F.Enc}_{fpk}(f_{k,j}(x)) \boxtimes \text{F.Enc}_{fpk}(h_\tau(x)), \tag{8}$$

Protocol 4: $\prod_{\text{OPR}}$

Input: P_S inputs polynomials $f_k(x)$ with $deg(f_k(x)) \leq N-1$ for $k \in [m]$, P_C inputs LFHE key pair (fpk, fsk) and $g(x) = \prod_{i=0}^{n'-1}(x - z_i)$ for $i \in [n']$.

Output: P_S outputs $\text{F.Enc}_{fpk}(\bar{f}_k(x))$ for $k \in [m]$ with $deg(\bar{f}_k(x)) \leq 2(n-1)$, such that $\bar{f}_k(x) \equiv f_k(x) \bmod g(x)$.

1: P_S constructs $f_{k,j}(x)$ such that $f_k(x) = \sum_{j=0}^{\lceil N/n \rceil - 1} f_{k,j}(x) \cdot x^{jn}$ for $k \in [m]$.
2: P_C computes $h_j(x) \leftarrow x^{jn} \bmod g(x)$ for $j \in [\lceil N/n \rceil]$. Then P_C encrypts $h_j(x)$ to $\text{F.Enc}_{fpk}(h_j(x))$ for $j \in [\lceil N/n \rceil]$ and sends $\text{F.Enc}_{fpk}(h_j(x))$ to P_S.
3: P_S computes $\text{F.Enc}_{fpk}(\bar{f}_k(x))$ according to Equation (8) for $k \in [m]$.

for $k \in [m]$ to get the reduced polynomials after encrypting.

Oblivious Polynomial Evaluation. After executing the oblivious polynomial reduction protocol $\prod_{\text{OPR}}$ (Protocol 4), a naive method for obtaining the set $\mathcal{X} = \{x_{id_i',k}\}_{i \in [n'], k \in [m]}$ corresponding to P_C's queries is represented as follows: P_S sends $\text{F.Enc}_{fpk}(\bar{f}_k(x))$ for $k \in [m]$ to P_C. Subsequently, P_C decrypts them to $\bar{f}_k(x)$ for $k \in [m]$ and then computes $\bar{f}_k(\omega^{id_i'})$ for $i \in [n'], k \in [m]$ to obtain $\mathcal{X}$.

However, this method would leak sensitive information outside the intersection to P_C. Therefore, we design a secure polynomial evaluation protocol, denoted as oblivious polynomial evaluation protocol $\prod_{\text{OPE}}$ as shown in Protocol 5. The key rationale behind this protocol is to use random polynomials to mask the reduced polynomials, thereby preventing P_C from inferring P_S's feature information outside the intersection through $\bar{f}_k(x)$.

As shown in Protocol 5, P_C first encrypts $g(x) = \prod_{i=0}^{n'-1}(x - z_i)$ and sends $\text{F.Enc}_{fpk}(g(x))$ to P_S . Then P_C generates random polynomials to mask $\bar{f}_k(x)$ for $k \in [m]$ while ensuring that the masked polynomials have the same results at the queries of P_C. Finally, P_C decrypts the masked polynomials and computes the evaluation results corresponding to its queries.

4.2.2 Batch PIR-to-Share

We formalize batch PIR-to-share as the functionality $\mathcal{F}_{\text{PIR2Share}}$, which is shown in Figure 6. $\mathcal{F}_{\text{PIR2Share}}$ receives the same input as $\mathcal{F}_{\text{Batch\_PIR}}$, but outputs shares of $\mathcal{X}$ corresponding to P_C's queries. The batch PIR-to-share protocol $\prod_{\text{PIR2Share}}$ (Protocol 6) securely and efficiently achieves the functionality $\mathcal{F}_{\text{PIR2Share}}$.

Below, we give more technical details to illustrate the oblivious polynomial interpolation step.

Oblivious Polynomial Interpolation. P_S first generates random values $r_{i,k}$ for $i \in [n'], k \in [m]$. Then P_S and P_C invoke the oblivious polynomial interpolation protocol $\prod_{\text{OPI}}$ (Protocol 7)

Protocol 5: $\prod_{\text{OPE}}$

Input: P_S inputs $\text{F.Enc}_{fpk}(\bar{f}_k(x))$ for $k \in [m]$. P_C inputs LFHE key pair $(fpk, fsk), g(x)$ and z_i for $i \in [n']$.

Output: P_C outputs $\mathcal{X} = \{x_{id'_i,k} | x_{id'_i,k} = \bar{f}_k(z_i)\}_{i\in[n'],k\in[m]}$.

1: P_C encrypts $g(x)$ to $\text{F.Enc}_{fpk}(g(x))$ and sends it to P_S.

2: P_S constructs random polynomials $\gamma_k(x) \in \mathbb{F}[x]$ for $k \in [m]$ with $deg(\gamma_k(x)) < d - n'$, where d is the degree of LFHE polynomial modulo. Then P_S computes $\text{F.Enc}_{fpk}(f'_k(x)) \leftarrow \text{F.Enc}_{fpk}(\bar{f}_k(x)) \boxplus (\text{F.Enc}_{fpk}(g(x)) \boxtimes \text{F.Enc}_{fpk}(\gamma_k(x)))$ for $k \in [m]$ and sends them to P_C.

3: P_C decrypts $\text{F.Enc}_{fpk}(f'_k(x))$ to $f'_k(x) \leftarrow \bar{f}_k(x) + g(x) \cdot \gamma_k(x)$ for $k \in [m]$. P_C returns $x_{id'_i,k} \leftarrow f'_k(z_i)$ for $i \in [n'], k \in [m]$.

Functionality $\mathcal{F}_{\text{PIR2Share}}$

Parameters: Two parties P_S and P_C.

Functionality:

- Wait for input $D^S = I^S_{new}||X$ from P_S, where $X = X_0||\ldots||X_{m-1} = \{x_{i,k}\}_{i\in[N],k\in[m]}$.
- Wait for input $I^C_{new} = \{id'_0, \ldots, id'_{n'-1}\}$ from P_C.
- Sample $\langle\mathcal{X}\rangle^S$ and $\langle\mathcal{X}\rangle^C$ with size n' uniformly, such that $\langle\mathcal{X}\rangle^S + \langle\mathcal{X}\rangle^C = \mathcal{X} = \{x_{id'_i,k}\}_{i\in[n'],k\in[m]}$.
- Output the shares $\langle\mathcal{X}\rangle^S$ to P_S and $\langle\mathcal{X}\rangle^C$ to P_C.

Figure 6: Ideal functionality of $\mathcal{F}_{\text{PIR2Share}}$

to enable P_S obtain $\text{F.Enc}_{fpk}(\bar{f}^S_k(x))$, where $f^S_k(z_i) = r_{i,k}$ for $i \in [n'], k \in [m]$. Finally, P_S locally sets $\text{F.Enc}_{fpk}(\bar{f}^C_k(x)) \leftarrow \text{F.Enc}_{fpk}(\bar{f}_k(x)) = \text{F.Enc}_{fpk}(\bar{f}^S_k(x))$.

The key rationale behind the oblivious polynomial interpolation protocol $\prod_{\text{OPI}}$ (Protocol 7) is that P_C generates n' polynomials to assist P_S securely interpolate polynomials $\bar{f}^S_k(x)$, such that $f^S_k(z_i) = r_{i,k}$, while guaranteeing P_S cannot obtain any information about z_i. Furthermore, in order to reduce the communication size of sending n' polynomials, we propose Theorem 2 to obtain n' polynomials by combining $2\lceil\sqrt{n'}\rceil$ basis polynomials. We elaborate on this rationale below.

P_C constructs n' polynomials $BF_0(x), \ldots, BF_{n'-1}(x)$ with $deg(BF_j(x)) < d$, where d is the degree of LFHE polynomial modulo, such that $BF_j(z_i) = 1$ if and only if $i = j$, and 0 otherwise for $i, j \in [n']$. Therefore, $\sum_{j=0}^{n'-1} r_{j,k} BF_j(z_i) = r_{i,k}$ for $i, j \in [n'], k \in [m]$. Then P_C encrypts these n' polynomials and sends these encrypted polynomials to P_S. Subsequently, P_S computes $\text{F.Enc}_{fpk}(\bar{f}^S_k(x)) = \sum_{j=0}^{n'-1} r_{j,k} \boxtimes \text{F.Enc}_{fpk}(BF_j(x))$, such that $\bar{f}^S_k(z_i) = \sum_{j=0}^{n'-1} r_{j,k} BF_j(z_i) = r_{i,k}$.

However, the communication size of n' polynomials ciphertexts is too huge. Therefore, we design an efficient oblivious polynomial interpolation protocol $\prod_{\text{OPI}}$ (Protocol 7) based on

Protocol 6: $\prod_{\text{PIR2Share}}$

Input: P_S inputs its data $D^S = I^S_{new}||X$, where $X = X_0||\ldots||X_{m-1} = \{x_{i,k}\}_{i\in[N],k\in[m]}$, P_C inputs its queries $I^C_{new} = \{id'_0, \ldots, id'_{n'-1}\}$.

Output: P_S and P_C outputs the shares $\langle\mathcal{X}\rangle^S$ and $\langle\mathcal{X}\rangle^C$ respectively, such that $\langle\mathcal{X}\rangle^S + \langle\mathcal{X}\rangle^C = \mathcal{X} = \{x_{id'_i,k}\}_{i\in[n'],k\in[m]}$.

Setup:

1: **Feature Encoding:**
The same as the protocol $\prod_{\text{Batch\_PIR}}$ (Protocol 3).

Online:

1: **Oblivious Polynomial Reduction:**
The same as the protocol $\prod_{\text{Batch\_PIR}}$(Protocol 3).

2: **Oblivious Polynomial Interpolation:**

- P_S generates random values $\mathcal{R} = \{r_{i,k}\}_{i\in[n'],k\in[m]}$ and sets $\langle\mathcal{X}\rangle^S \leftarrow \mathcal{R}$.
- P_S and P_C invoke the oblivious polynomial interpolation protocol $\prod_{\text{OPI}}$ (Protocol 7), where P_S inputs $\mathcal{R} = \{r_{i,k}\}_{i\in[n'],k\in[m]}$, and P_C inputs (fpk, fsk) and z_i for $i \in [n']$. After the execution, P_S obtains $\text{F.Enc}_{fpk}(\bar{f}^S_k(x))$, such that $\bar{f}^S_k(z_i) = r_{i,k}$ for $i \in [n'], k \in [m]$.
- P_S computes $\text{F.Enc}_{fpk}(\bar{f}^C_k(x)) \leftarrow \text{F.Enc}_{fpk}(\bar{f}_k(x)) \boxminus \text{F.Enc}_{fpk}(\bar{f}^S_k(x))$.

3: **Oblivious Polynomial Evaluation:**
P_S and P_C invoke the oblivious polynomial evaluation protocol $\prod_{\text{OPE}}$ (Protocol 5), where P_S inputs $\text{F.Enc}_{fpk}(\bar{f}^C_k(x))$ for $k \in [m]$, and P_C inputs $(fpk, fsk), g(x)$ and z_i for $i \in [n']$. After the execution, P_C obtains set $\langle\mathcal{X}\rangle^C \leftarrow \{\bar{f}^C_k(z_i)\}_{i\in[n'],k\in[m]}$.

the Theorem 2 to reduce the communication size from $\mathcal{O}(n')$ to $\mathcal{O}(\sqrt{n'})$.

As shown in Protocol 7, P_C first generates ι basis polynomials $RF_\mu(x)$ and ι polynomials $CF_\nu(x)$ for $\mu, \nu \in [\iota]$ based on Theorem 2. Then, P_C encrypts $RF_\mu(x)$ and $CB_\nu(x)$ to $\text{F.Enc}_{fpk}(RF_\mu(x))$ and $\text{F.Enc}_{fpk}(CF_\nu(x))$ for $\mu, \nu \in [\iota]$ respectively and sends them to P_S. Finally, P_S computes

$$\text{F.Enc}_{fpk}(\bar{f}^S_k(x)) = \sum_{\mu,\nu=0}^{\iota-1} x^S_{\mu\iota+\nu,k} \text{F.Enc}_{fpk}(RF_\mu(x)) \boxtimes \text{F.Enc}_{fpk}(CF_\nu(x)), \tag{9}$$

for $k \in [m]$. Therefore, $\bar{f}^S_k(z_i) = r_{i,k}$ for $i \in [n'], k \in [m]$.

Theorem 2. *Let $\mathbb{F}_p$ be a prime finite field, $\{z_i\}_{i\in[\iota^2]}$ be a set of distinct elements on $\mathbb{F}_p$, and there exist ι polynomials $\{RF_\mu(x) \in \mathbb{F}[x]\}_{\mu\in[\iota]}$ and ι polynomials $\{CF_\nu(x) \in \mathbb{F}[x]\}_{\nu\in[\iota]}$ which satisfy:*

1. *$deg(RF_\mu(x)), deg(CF_\nu(x)) < \iota^2$;*
2. *$RF_\mu(z_i) = 1$ if and only if $\mu = i/\iota$, and $RF_\mu(z_i) = 0$ otherwise;*
3. *$CF_\nu(z_i) = 1$ if and only if $\nu = i\%\iota$, and $CF_\nu(z_i) = 0$ otherwise.*

Then $RF_\mu(z_i) \cdot CF_\nu(z_i) = 1$ *if and only if* $i = \mu\mathfrak{u} + \nu$, *and* $RF_\mu(z_i) \cdot CF_\nu(z_i) = 0$ *otherwise.*

Proof. The detailed proof is shown in Appendix D. □

Furthermore, for the case of $n \leq d/4$, where n is the smaller data size and d is the degree of LFHE polynomial modulo, we proposed an improved oblivious polynomial reduction protocol $\prod_{\text{OPI2}}$ (Protocol 10), an improved oblivious polynomial evaluation protocol $\prod_{\text{OPR2}}$ (Protocol 9) and an improved oblivious polynomial interpolation protocol $\prod_{\text{OPE2}}$ (Protocol 11) in Appendix B.

Protocol 7: $\prod_{\text{OPI}}$

Input: P_S inputs $\{r_{i,k}\}_{i\in[n'],k\in[m]}$. P_C inputs LFHE key pair (pk, sk) and z_i for $i \in [n']$.

Output: P_S outputs $\text{F.Enc}_{fpk}(\bar{f}_k^S(x))$, such that $\bar{f}_k^S(z_i) = r_{i,k}$ for $i \in [n'], k \in [m]$.

1: P_C generates $\mathfrak{u}$ basis polynomials $RF_\mu(x)$ and $\mathfrak{u}$ polynomials $CF_\nu(x)$ with $deg(RF_\mu(x)), deg(CF_\nu(x)) < \mathfrak{u}^2$ for $\mu, \nu \in [\mathfrak{u}]$, where $\mathfrak{u} = \lceil\sqrt{n'}\rceil$. $RF_\mu(x)$ and $CF_\nu(x)$ satisfy $RF_\mu(z_i) \cdot CF_\nu(z_i) = 1$ if and only if $i = \mu\mathfrak{u} + \nu$ for $\mu, \nu \in [\mathfrak{u}], i \in [n']$. P_C encrypts $RF_\mu(x)$ and $CB_\nu(x)$ to $\text{F.Enc}_{fpk}(RF_\mu(x))$ and $\text{F.Enc}_{fpk}(CF_\nu(x))$ for $\mu, \nu \in [\mathfrak{u}]$ respectively and sends them to P_S.

2: P_S computes $\text{F.Enc}_{fpk}(\bar{f}_k^S(x))$ following Equation (9).

5 Performance Evaluation

5.1 Implementation and Experiment Setting

Implementation. We implement `Suda` in C++ based on the HE library Microsoft SEAL (version 4.1.2)[1], the ECC library libsodium (version 1.0.18)[2]. Besides, we utilize BFV as our LFHE scheme with the degree of LFHE polynomial modulo $d = 8192$, plaintext module $p = 19 \cdot 2^{46} + 1$, ciphertext module $q \approx 2^{168}$ for batch PIR and $q \approx 2^{218}$ for batch PIR-to-share and security parameter $\lambda = 128$. At the same time, we utilize the Ed25519 algorithm [14], defined on the twisted Edwards curve, to fast execute 128-bit security ECC. The curve equation of Ed25519 is $-x^2 + y^2 = 1 + \frac{121665}{121666}x^2y^2 \bmod (2^{255} - 19)$, which is birationally equivalent to Equation (1).

Experiment Setting. We conduct all experiments on a Linux server equipped with an Intel(R) Xeon(R) Platinum 8260 CPU @ 2.40GHz and 720GB RAM. Each party in `Suda` is simulated by a separate process with one thread. We also consider two network settings: the LAN setting with a bandwidth of 1GBps and sub-millisecond round-trip time (RTT) latency. The other is the WAN setting with 75Mbps bandwidth and 40ms RTT latency. We apply the tc tool[3] to simulate these two network settings.

5.2 Evaluation of Secure Data Alignment

Baseline. We consider the following two baselines to evaluate the efficiency of `Suda`.

- `iPrivJoin` [22]: To the best of our knowledge, `iPrivJoin` is the only framework with functionality similar to `Suda`. However, unlike `Suda`, which directly and exclusively outputs only the data shares in the intersection, `iPrivJoin` employs a secure shuffle protocol to trim redundant data outside the intersection. Since `iPrivJoin` is not open-sourced, we re-implement it. Both `Suda` and `iPrivJoin` are evaluated under identical experimental settings, e.g. thread counts. Here, we adopt the configuration where the server P_S (holding the larger dataset) performs simple hashing and the client P_C (holding the smaller dataset) performs Cuckoo Hashing, as this setting is more efficient than the reverse configuration (see Appendix F for details).
- `CPSI` [30]:We also include a circuit-PSI approach [30], referred to as `CPSI`, as a baseline. Notably, `CPSI` differs significantly in functionality from `Suda`. Specifically, `CPSI`, based on Cuckoo Hashing, while `Suda` leveraging polynomial operations. We re-run `CPSI` with their open-sourced code[4] under our experiment setting.

Besides, we consider the total running time, i.e. the sum of setup time and online time during the evaluation of secure data alignment.

5.2.1 Evaluation over Public Datasets

We perform efficiency evaluation of `Suda` over two public datasets, SVHN [27] and Character Font Images [23] under the WAN setting. The SVHN dataset contains 73257 training samples and 26032 test samples, each represented as a 32 × 32 RGB image. The Character Font Images dataset contains 832670 samples, for each sample we select a 20 × 20 grayscale image and 8 additional features. We set the intersection rate as 80% and the smaller data size $n = 1024$. Besides, all data in the intersection are selected randomly.

We compare the efficiency of `Suda` with `iPrivJoin` [22] and `CPSI` [30] in terms of two steps of VPPML: secure data alignment and secure training using the outputs of secure data alignment. For the second step, we employ a state-of-the-art secure multi-party learning framework `Crypten` [17] to train a logistic regression model in 100 epochs.

As is shown in Table 1, the experimental results demonstrate that `Suda` outperforms `iPrivJoin` and `CPSI` during

[1] https://github.com/Microsoft/SEAL
[2] https://github.com/jedisct1/libsodium
[3] https://man7.org/linux/man-pages/man8/tc.8.html
[4] https://github.com/Visa-Research/volepsi.git

Table 1: Comparison of Suda with iPrivJoin and CPSI over public datasets.

Dataset			SVHN			Character Font Images		
Framework			Suda	iPrivJoin [22]	CPSI [30]	Suda	iPrivJoin [22]	CPSI [30]
Secure Data Alignment	Time (s)		2545.99	12203.20	9286.06	433.30	1802.68	2527.80
	Communication (MB)		455.44	63259.04	60988.85	139.59	8439.01	10698.95
	Memory	Server (MB)	55263.50	77105.80	223810.00	9831.62	15980.70	39528.00
		Client (MB)	1163.72	3565.07	61029.60	310.02	2916.68	10713
Secure Training	Time (s)		7182.55		11795.57	6358.21		10048.89
	Communication (MB)		8939.01		18320.48	1439.93		2832.04

secure data alignment in terms of running time and communication size. Furthermore, CPSI incurs a 1.58× to 1.66× increase in running time and a 1.98× to 2.19× increase in communication cost during secure training. This inefficiency stems from CPSI outputting data shares with size εn ($\varepsilon > 1$), whereas both Suda and iPrivJoin exclusively output data shares in the intersection with size n', where $n' < n$. As a result, CPSI is inefficient for VPPML and is excluded from subsequent evaluations of secure data alignment.

5.2.2 Evaluation over Varied Data Sizes

We fix the feature dimension at $m = 100$, and vary the larger data size across $N \in \{2^{20}, 2^{22}, 2^{24}\}$ and the smaller size across $n \in \{1024, 2048, 4096\}$.

Table 2: Comparison of Suda with iPrivJoin over varied larger data size N and smaller data size n, while maintaining a fixed feature dimension $m = 100$.

N	n	Framework	Comm.(MB)	Time (s)
2^{20}	1024	Suda	89.80	187.55
		iPrivJoin [22]	2655.32	487.92
	2048	Suda	108.16	236.46
		iPrivJoin [22]	3012.62	545.65
	4096	Suda	138.30	442.72
		iPrivJoin [22]	3389.94	603.55
2^{22}	1024	Suda	284.26	703.34
		iPrivJoin [22]	9833.19	1840.37
	2048	Suda	302.63	740.43
		iPrivJoin [22]	9989.60	1821.26
	4096	Suda	332.76	959.73
		iPrivJoin [22]	10682.27	1940.06
2^{24}	1024	Suda	1062.11	2783.65
		iPrivJoin [22]	37913.63	7331.64
	2048	Suda	1080.48	2778.59
		iPrivJoin [22]	39389.21	7237.67
	4096	Suda	1110.62	2964.39
		iPrivJoin [22]	39464.18	7215.45

The experimental results demonstrate that:

- Suda outperforms iPrivJoin by $24.51\times \sim 36.45\times$ in communication size across all data sizes N and n. Besides, Suda exhibits greater advantages when the larger data size N is significant. Concretely, when $N = 2^{20}$, Suda outperforms iPrivJoin by up to $29.57\times$, while when $N = 2^{24}$, Suda outperforms iPrivJoin by up to $36.45\times$.
- Suda outperforms iPrivJoin by up to $1.36\times \sim 2.63\times$ in terms of running time across all data sizes N and n. As N increases, the reduced communication size of Suda will increase. Similar to communication size, Suda exhibits greater advantages when the larger data size N is significant. Specifically, when $N = 2^{20}$, Suda outperforms iPrivJoin by up to $2.60\times$, while when $N = 2^{24}$, Suda outperforms iPrivJoin by $2.63\times$.

These improvements of Suda over iPrivJoin all stem from our reduction in communication size from $\mathcal{O}(m(N+n))$ to $\mathcal{O}(N+n+mn')$. As N increases, the reduced communication size further enhances efficiency, resulting in shorter running times.

Furthermore, we conduct the experiment to demonstrate the efficiency of Suda with the fixed feature dimension $m = 1000$, and the experimental results are shown in Appendix F.

5.2.3 Evaluation over Varied Feature Dimensions

We fix the larger data size at $N = 2^{20}$ and the smaller data size at $n = 1024$. Subsequently, we conduct experiments to compare the communication size and running time between Suda and iPrivJoin across varying feature dimensions $m = \{100, 200, 500, 1000, 2000\}$ under both LAN and WAN settings.

The experimental results shown in Table 3 demonstrate that Suda outperforms iPrivJoin by $31.14\times \sim 210.78\times$ in communication size. Besides, Suda outperforms iPrivJoin by $1.07\times \sim 2.83\times$ and $2.67\times \sim 8.21\times$ in the running time under the LAN and WAN settings, respectively. Moreover, we observe that Suda exhibits greater advantages when the feature dimension m is significant. These improvements of Suda over iPrivJoin also stem from our reduction in the communication size from $\mathcal{O}(m(N+n))$ to $\mathcal{O}(N+n+mn')$. As m increases, the reduced communication size further efficiency, resulting in shorter running time (especially in the WAN setting). Additionally, we observe that the running time of Suda in the WAN setting is comparable to the LAN setting. This is attributed to Suda's significantly smaller communication size, which reduces the impact of network conditions on performance.

Table 3: Comparison of Suda with iPrivJoin over varied feature dimensions m, while maintaining fixed larger data size $N = 2^{20}$ and smaller data size $n = 1024$.

m	Framework	Comm.(MB)	LAN (s)	WAN (s)
100	Suda	89.80	171.73	187.30
	iPrivJoin [22]	2796.40	183.21	495.82
200	Suda	101.37	233.37	245.61
	iPrivJoin [22]	5589.09	364.40	981.18
500	Suda	136.11	407.18	420.55
	iPrivJoin [22]	13832.11	832.57	2372.42
1000	Suda	194.00	693.15	718.54
	iPrivJoin [22]	29246.99	1642.92	4907.89
2000	Suda	309.78	1278.14	1304.00
	iPrivJoin [22]	65297.77	3618.93	10711.26

5.2.4 Evaluation over Varied Intersection Size

We fix the larger data size at $N = 2^{20}$, the smaller data size at $n = 4096$, and the feature dimension at $m = 1000$. Subsequently, we conduct experiments to compare the communication size and running time between Suda and iPrivJoin across varying intersection size $n' = \{100\%n, 50\%n, 25\%n\}$.

Table 4: Comparison of Suda with iPrivJoin over varied intersection size n', while maintaining fixed larger data size $N = 2^{20}$, smaller data size $n = 4096$ and feature dimension $m = 1000$.

n'	Framework	Comm.(MB)	LAN (s)	WAN (s)
$100\%n$	Suda	565.85	3072.43	3119.86
	iPrivJoin [22]	33155.39	1813.22	5502.78
$50\%n$	Suda	327.77	1178.93	1195.76
	iPrivJoin [22]	33155.39	1813.22	5502.78
$25\%n$	Suda	205.42	660.492	679.639
	iPrivJoin [22]	33155.39	1813.22	5502.78

The experimental results shown in Table 4 demonstrate that as the intersection size decreases, the efficiency of Suda improves, while the efficiency of iPrivJoin remains constant. This is because, in the secure feature alignment phase of Suda, P_C utilizes n' IDs in the intersection to retrieve corresponding feature shares. Therefore, the overhead of Suda scales proportionally with the intersection size n'. However, the overhead of iPrivJoin depends on the number of hash bins, which is independent of n'.

5.3 Evaluation of Batch PIR

Baseline. We compare Suda with the state-of-the-art batch PIR framework PIRANA [20] to evaluate the efficiency of the batch PIR protocol $\prod_{\text{Batch\_PIR}}$ (Protocol 3). PIRANA uses SIMD to achieve the homomorphic equality operator in Cw-PIR to improve efficiency. In contrast, Suda uses polynomial operations to improve efficiency. We re-run PIRANA with their open-sourced code[5] under our experiment setting.

We consider three data sizes $\{2^{20}, 2^{22}, 2^{24}\}$, three batch sizes $\{1024, 2048, 4096\}$ and two types of payload sizes {256 bytes, 1KB}. The experimental results are shown in Table 5.

Here, the data size refers to the size of the data held by P_S, the batch size refers to the number of queries requested by P_C, and the payload size refers to the feature size. Besides the request size refers to the communication size sent by P_C during the batch PIR protocol, i.e. the sum of communication size in Step 2 of the oblivious polynomial reduction protocol $\prod_{\text{OPR}}$ (Protocol 4) and Step 1 of the oblivious polynomial evaluation protocol $\prod_{\text{OPE}}$ (Protocol 5). These steps could be executed in parallel within one communication round. The response size refers to the communication size sent by P_S during Step 2 of the oblivious polynomial evaluation protocol $\prod_{\text{OPE}}$ (Protocol 5). Additionally, the server time refers to the sum of the setup time and the local computation time of the server in the online phase, while the client time refers to the local computation time of the client in the online phase.

The experimental results demonstrate that Suda outperforms PIRANA by up to 11.53× and 3.79× in server time and client time, respectively. Besides, Suda outperforms PIRANA by up to 3.68× and 5.97× in the memory cost of server and client, respectively. Furthermore, PIRANA claims to be friendly for scenarios with dynamic data due to its efficient setup phase. However, Suda outperforms PIRANA by up to 12.58× in setup time. Therefore, Suda is more friendly for scenarios with dynamic data.

6 Discussion

Practical Applications of Suda. Suda has been applied in practical scenarios to achieve secure large-scale data (billion-level) alignment. Suda only costs around 24 hours to securely align data between one billion samples with thousand-dimensional features held by the server P_S and one million samples held by the client P_C. Firstly, P_C maps its data to T bins $\{t_0^C, \ldots, t_T^C\}$ by calculating $H(I^C)\%T$, where $T = \lceil \frac{n}{d/2} \rceil$, d is the degree of the LFHE polynomial module and $H(\cdot)$ is a hash function. This ensures that the sample number of P_C in each bin is $nb < d/2$. Similarly, P_S maps its data to T bins $\{t_0^S, \ldots, t_T^S\}$ by calculating $H(I^S)\%T$. Then, P_S and P_C perform secure data alignment on the data in each pair bin t_i^S and t_i^C for $i \in [T]$. Finally, P_S and P_C respectively merge their feature and label shares obtained from each bin. In addition, we utilize Spark for parallel optimization to accelerate computations.

Exposing the Intersection Size. Suda exposes the intersection size n', but this should be acceptable in VPPML. That is because the parties usually need the intersection size to decide whether to perform the proceeding training process. For example, in advertising scenarios, if the intersection rate is

[5] https://github.com/zju-abclab/PIRANA

Table 5: Comparison of Suda with PIRANA.

Data Size	Batch Size	Payload Size	Framework	Request Size (MB)	Response Size (MB)	Time (s)			Memory (MB)	
						Setup	Server	Client	Server	Client
2^{20}	1024	256 bytes	Suda	36.55	2.40	13.19	18.06	5.46	3238.68	71.55
			PIRANA [20]	1.94	4.07	136.69	148.38	20.70	8371.47	291.08
		1KB	Suda	36.55	8.00	43.12	51.71	5.74	9520.38	71.68
			PIRANA [20]	1.94	15.91	510.44	548.62	20.79	31732.14	291.38
	2048	256 bytes	Suda	36.55	4.80	13.16	16.21	5.71	3018.41	99.86
			PIRANA [20]	1.94	4.07	136.68	148.37	20.69	8371.52	291.16
		1KB	Suda	36.55	16.00	42.92	49.61	6.34	9303.89	100.63
			PIRANA [20]	1.93	15.91	497.58	531.03	20.85	31732.43	291.12
	4096	256 bytes	Suda	36.55	9.60	12.33	14.68	6.48	2824.55	131.04
			PIRANA [20]	2.87	8.14	142.37	154.72	21.06	8861.11	292.17
		1KB	Suda	36.55	32.00	41.04	47.83	8.23	9164.92	138.27
			PIRANA [20]	2.87	31.82	516.30	551.49	21.15	33717.08	292.14
2^{22}	1024	256 bytes	Suda	146.19	2.40	64.04	86.45	24.79	12883.72	138.61
			PIRANA [20]	3.67	4.07	551.49	595.01	87.36	31548.84	1046.81
		1KB	Suda	146.19	8.00	210.21	253.59	25.03	37992.49	138.16
			PIRANA [20]	3.68	15.91	2102.11	2229.87	89.26	119691.21	1046.66
	2048	256 bytes	Suda	146.20	4.80	64.02	75.49	25.25	11993.43	266.50
			PIRANA [20]	3.70	4.07	582.00	626.19	88.37	31549.80	1046.73
		1KB	Suda	146.19	16.00	211.39	236.51	25.79	37107.08	266.61
			PIRANA [20]	3.69	15.91	2093.34	2222.58	88.36	119691.35	1046.73
	4096	256 bytes	Suda	146.19	9.60	60.95	69.37	26.34	11150.46	330.97
			PIRANA [20]	5.31	8.14	600.33	645.46	87.57	31901.92	1048.31
		1KB	Suda	146.20	32.00	198.62	219.73	27.83	36302.23	330.77
			PIRANA [20]	5.31	31.82	2184.50	2312.15	88.08	121563.27	1048.67
2^{24}	1024	256 bytes	Suda	584.78	2.40	294.27	367.42	111.31	51461.09	425.88
			PIRANA [20]	7.18	4.07	2526.28	2702.53	378.06	122828.41	4074.64
		1KB	Suda	584.77	8.00	970.32	1088.01	112.12	151880.52	425.77
			PIRANA [20]	720.17	159.09	9136.15	9843.09	375.51	466250.22	4074.62
	2048	256 bytes	Suda	584.77	4.80	313.09	369.09	114.78	47881.51	938.11
			PIRANA [20]	7.20	4.07	2537.15	2710.68	377.64	122829.20	4074.38
		1KB	Suda	584.77	16.00	967.53	1058.53	113.65	148305.40	938.64
			PIRANA [20]	719.58	159.08	9539.32	10538.30	379.59	466249.47	4074.60
	4096	256 bytes	Suda	584.77	9.60	280.95	310.99	115.68	44443.95	1194.63
			PIRANA [20]	10.35	8.14	2786.19	2967.39	378.38	123749.79	4089.31
		1KB	Suda	584.77	32.00	926.80	998.61	117.19	144923.12	1194.92
			PIRANA [20]	103.16	318.11	10184.50	10886.30	376.08	471926.59	4089.57

too small (e.g. < 30%) to train a high-performance model, advertisers may choose not to continue with the secure training to avoid unnecessary overhead.

Compatibility with Multi-client Settings. In multi-client settings, a server with larger data executes the secure data alignment protocol with multiple clients, each with its smaller data. Suda should be compatible with such settings. Specifically, the server P_S encrypts its IDs and encodes its features once during the setup phase, and then repeatedly executes the online phase of the secure data alignment protocol $\prod_{\text{Suda}}$ with multiple clients. During this phase, P_S's initial encryption and encoding are reusable, so only need to be computed once.

The efficiency of Suda in multi-client settings still remains acceptable, though Suda is less efficient than the state-of-the-art framework [35], which is designed specifically for multi-client settings. Suda achieves client complexity linear with the large data size, while [35] achieves linear with each client's data size. Besides, as is shown in Table 5, the setup phase in Suda also accounts for a significant proportion of the overall time, especially with the large payload sizes. Therefore, the running time of each client executed the online phase in the secure data alignment protocol is relatively small.

Scenarios where P_C Holds Features. Suda can be simply extended to support the scenario where P_C also holds features, that is, P_S holds data $D^S = I^S||X^S$, and P_C holds data $D^C = I^C||X^C||Y$. Specifically, we only need to perform the same computations on the features held by P_C as on the label in the get new ID protocol $\prod_{\text{GetNewID}}$ (Protocol 2).

Future Work. We aim to efficiently extend Suda to support more parties securely aligning their data.

7 Related Work

Unbalanced Circuit-PSI. There are some efficient unbalanced PSI methods [8, 10, 15, 16, 31] but they directly reveal the plaintext IDs in the intersection, potentially exposing sensitive information about the involved IDs. This poses a significant risk in practical applications where the IDs themselves are sensitive (e.g. customer or patient lists). To protect all the sample information in the intersection, several works have proposed unbalanced circuit-PSI methods. Lepoint et al. [19] introduced unbalanced private join and compute (PJC) functionality that enables secure computation over the data in the intersection. They provided two constructions. The first one has a highly efficient online phase but requires an expensive offline overhead and large storage in the smaller data holder. The second one does not require offline preprocessing but introduces expensive computation overhead. Son and Jeong [34] optimized the constructions of [19]. They removed the large storage requirement of the first construction in [19] and achieved similar online performance. Very recently, Hao et al. [12] designed a more efficient unbalanced circuit-PSI method, exhibiting communication size that scales sublinearly with the size of the larger data. They presented a new functionality named Oblivious Key-Value Retrieval to retrieve values corresponding to keys from a key-value store obliviously. Besides, they conducted the OKVR protocol based on a new notion termed sparse Oblivious Key-Value Store. However, these unbalanced circuit-PSI methods typically use Cuckoo Hashing to enhance efficiency, which introduces redundant data outside the intersection, resulting in more communication size for further secure training.

Balanced Secure Data Alignment Method. Liu *et al.* [22] proposed `iPrivJoin`. They introduced a new private encoding technique to avoid the expensive circuit evaluation needed in circuit-PSI. Besides they also designed an efficient secure shuffle protocol to trim the redundant data outside the intersection. Though `iPrivJoin` optimizes circuit-PSI and only outputs data shares in the intersection, it introduces additional communication overhead by secure shuffle operations.

Batch Private Information Retrieval. Beimel *et al.* [3] proposed a multi-server batch PIR method with preprocessing. The servers preprocess their data and move most of the computation to the offline phase to improve the efficiency of online queries. Ishai *et al.* [13] proposed and applied the batch code approach to batch PIR, which transforms any single-query PIR into a batch PIR, thereby reducing computational costs. Angel *et al.* [2] proposed a batch PIR protocol based on probabilistic batch code. They introduced a small probability of failure (about one in a trillion) so that the client gets only part of its queries. Although [2] has weaker properties, it exhibits significantly higher efficiency compared to [13]. However, these batch PIR methods [2, 3, 13] all suffer from high communication size. Mughees and Ren [26] introduced vectorized HE to improve communication efficiency. Instead of issuing one query to each batch bucket in (probabilistic) batch code-based batch PIR methods, they merge the query and response ciphertexts across buckets. Because they encode multiple entries into a single ciphertext, their method is not suitable for large entry sizes. Patel *et al.* [29] presented keyword PIR schemes where clients issue queries based on keywords rather than indices. The key technique is an encoding algorithm that encodes key-value pairs as a function of multiple entries. Bienstock *et al.* [4] proposed a batch PIR method that reduces response size regardless of entry sizes. Specifically, they design oblivious ciphertext compression techniques to avoid encoding non-essential values, thereby reducing the communication overhead. Liu *et al.* [20] proposed a constant-weight codes-based [24] batch PIR and designed `PIRANA`. `PIRANA` replaces the holomorphic equality operator in the original constant-weight PIR with a SIMD-based one to enhance efficiency. However, `Suda` uses polynomial operations to enhance efficiency and does not require the expensive homomorphic equality operator.

8 Conclusion

In this paper, we propose an efficient and secure unbalanced data alignment framework, referred to as `Suda`, for VPPML. By leveraging polynomial-based operations rather than Cuckoo Hashing, `Suda` efficiently, directly, and exclusively outputs data shares in the intersection without expensive secure shuffle operations. Consequently, `Suda` efficiently and seamlessly aligns with secure training in VPPML. Specifically, we first design a novel and efficient batch PIR protocol based on the oblivious polynomial reduction and evaluation protocols. To securely obtain feature shares in the intersection, we further design a batch PIR-to-share protocol extended by the batch PIR protocol with the oblivious polynomial interpolation protocol. Compared with the state-of-the-art secure data alignment framework `iPrivJoin` [22], `Suda` outperforms it by up to 210.78× and 8.21× in communication size and running time, respectively. Besides, `Suda` achieves comparable efficiency in the WAN setting as it does in the LAN. Additionally, compared with the state-of-the-art batch PIR work `PIRANA` [20], `Suda` outperforms `PIRANA` by up to 11.53× in server time. These results show that `Suda` is much more practical for secure data alignment in VPPML.

9 Ethics Considerations

We analyze the ethics of our paper from the following three principles: beneficence, respect for persons, and justice. Regarding beneficence, because we do not provide any live services or APIs that give access to otherwise non-public algorithms or models, our results would not cause any financial loss. Regarding respect for persons, our study aims to improve the efficiency of secure data alignment and does not cause

disrespect to persons. Regarding justice, the results of our study would expand the practical applications of secure data alignment and not impact any specific stakeholder group.

10 Open Science

We open-source our artifacts in https://doi.org/10.5281/zenodo.14738503, and they are accepted in the artifact evaluation.

Acknowledgments

This paper is supported by Natural Science Foundation of China (92370120, 62172100) and the National Key R&D Program of China (2023YFC3304400). We thank all anonymous reviewers, Guopeng Lin, Shuyu Chen, Wenyan Li and Wenqiang Ruan for their insightful comments.

References

[1] Ramesh C Agarwal and C Sidney Burrus. Number theoretic transforms to implement fast digital convolution. Proceedings of the IEEE, 63(4):550–560, 1975.

[2] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. Pir with compressed queries and amortized query processing. In 2018 IEEE Symposium on Security and Privacy (S&P), pages 962–979, 2018.

[3] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: Pir with preprocessing. In Advances in Cryptology—CRYPTO 2000: 20th Annual International Cryptology Conference Santa Barbara, California, USA, August 20–24, 2000 Proceedings 20, pages 55–73. Springer, 2000.

[4] Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. Batch pir and labeled psi with oblivious ciphertext compression. In 33th USENIX Security Symposium (USENIX Security 24), 2024.

[5] Joppe W Bos, J Alex Halderman, Nadia Heninger, Jonathan Moore, Michael Naehrig, and Eric Wustrow. Elliptic curve cryptography in practice. In Financial Cryptography and Data Security: 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers 18, pages 157–175. Springer, 2014.

[6] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. ACM Trans. Comput. Theory, 6(3), jul 2014.

[7] Chaochao Chen, Jun Zhou, Li Wang, Xibin Wu, Wenjing Fang, Jin Tan, Lei Wang, Alex X Liu, Hao Wang, and Cheng Hong. When homomorphic encryption marries secret sharing: Secure large-scale sparse logistic regression and applications in risk control. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pages 2652–2662, 2021.

[8] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In 2017 ACM SIGSAC Conference on Computer and Communications Security, pages 1243–1255, 2017.

[9] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, Dimitrios Papadopoulos, and Qiang Yang. Secureboost: A lossless federated learning framework. IEEE intelligent systems, 36(6):87–98, 2021.

[10] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled psi from homomorphic encryption with reduced computation and communication. In 2021 ACM SIGSAC Conference on Computer and Communications Security, pages 1135–1150, 2021.

[11] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, 2012.

[12] Meng Hao, Weiran Liu, Liqiang Peng, Hongwei Li, Cong Zhang, Hanxiao Chen, and Tianwei Zhang. Unbalanced circuit-psi from oblivious key-value retrieval. In 33th USENIX Security Symposium (USENIX Security 24), 2024.

[13] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In 36th annual ACM symposium on Theory of computing, pages 262–271, 2004.

[14] Simon Josefsson and Ilari Liusvaara. Edwards-curve digital signature algorithm (eddsa). RFC, 8032:1–60, 2017.

[15] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In 28th USENIX Security Symposium (USENIX Security 19), pages 1447–1464, 2019.

[16] Ágnes Kiss, Jian Liu, Thomas Schneider, N Asokan, and Benny Pinkas. Private set intersection for unequal set sizes with mobile applications. In Privacy Enhancing Technologies Symposium, pages 177–197. De Gruyter, 2017.

[17] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. Advances in Neural Information Processing Systems, 34:4961–4973, 2021.

[18] Neal Koblitz. Elliptic curve cryptosystems. Mathematics of computation, 48(177):203–209, 1987.

[19] Tancrede Lepoint, Sarvar Patel, Mariana Raykova, Karn Seth, and Ni Trieu. Private join and compute from pir with default. In International Conference on the Theory and Application of Cryptology and Information Security, pages 605–634. Springer, 2021.

[20] Jian Liu, Jingyu Li, Di Wu, and Kui Ren. PIRANA: Faster multi-query PIR via constant-weight codes. In 2024 IEEE Symposium on Security and Privacy (S&P), 2024.

[21] Yang Liu, Yan Kang, Tianyuan Zou, Yanhong Pu, Yuanqin He, Xiaozhou Ye, Ye Ouyang, Ya-Qin Zhang, and Qiang Yang. Vertical federated learning: Concepts, advances, and challenges. IEEE Transactions on Knowledge and Data Engineering, 2024.

[22] Yang Liu, Bingsheng Zhang, Yuxiang Ma, Zhuo Ma, and Zecheng Wu. iprivjoin: An id-private data join framework for privacy-preserving machine learning. IEEE Transactions on Information Forensics and Security, 18:4300–4312, 2023.

[23] Richard Lyman. Character Font Images. UCI Machine Learning Repository, 2016. DOI: https://doi.org/10.24432/C5X61Q.

[24] Rasoul Akhavan Mahdavi and Florian Kerschbaum. Constant-weight PIR: Single-round keyword PIR via constant-weight equality operators. In 31st USENIX Security Symposium (USENIX Security 22), pages 1723–1740, Boston, MA, August 2022. USENIX Association.

[25] Victor S Miller. Use of elliptic curves in cryptography. In Conference on the theory and application of cryptographic techniques, pages 417–426. Springer, 1985.

[26] Muhammad Haris Mughees and Ling Ren. Vectorized batch private information retrieval. In 2023 IEEE Symposium on Security and Privacy (S&P), pages 437–452, 2023.

[27] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[28] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In International conference on the theory and applications of cryptographic techniques, pages 223–238. Springer, 1999.

[29] Sarvar Patel, Joon Young Seo, and Kevin Yeo. Don't be dense: Efficient keyword PIR for sparse databases. In 32nd USENIX Security Symposium (USENIX Security 23), pages 3853–3870, Anaheim, CA, August 2023. USENIX Association.

[30] Srinivasan Raghuraman and Peter Rindal. Blazing fast psi from improved okvs and subfield vole. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS '22, page 2505–2517, New York, NY, USA, 2022. Association for Computing Machinery.

[31] Amanda C Davi Resende and Diego F Aranha. Faster unbalanced private set intersection. In Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26–March 2, 2018, Revised Selected Papers 22, pages 203–221. Springer, 2018.

[32] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. Foundations of secure computation, 4(11):169–180, 1978.

[33] Daniele Romanini, Adam James Hall, Pavlos Papadopoulos, Tom Titcombe, Abbas Ismail, Tudor Cebere, Robert Sandmann, Robin Roehm, and Michael A Hoeh. Pyvertical: A vertical federated learning framework for multi-headed splitnn. arXiv preprint arXiv:2104.00489, 2021.

[34] Yongha Son and Jinhyuck Jeong. Psi with computation or circuit-psi for unbalanced sets from homomorphic encryption. In 2023 ACM Asia Conference on Computer and Communications Security, ASIA CCS '23, page 342–356, New York, NY, USA, 2023. Association for Computing Machinery.

[35] Yunqing Sun, Jonathan Katz, Mariana Raykova, Phillipp Schoppmann, and Xiao Wang. Actively secure private set intersection in the client-server setting. In Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, pages 1478–1492, 2024.

[36] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. arXiv preprint arXiv:1812.00564, 2018.

[37] Chuhan Wu, Fangzhao Wu, Lingjuan Lyu, Yongfeng Huang, and Xing Xie. Fedctr: Federated native ad ctr prediction with cross-platform user behavior data. ACM Transactions on Intelligent Systems and Technology (TIST), 13(4):1–19, 2022.

[38] Guiming Wu, Qianwen He, Jiali Jiang, Zhenxiang Zhang, Yuan Zhao, Yinchao Zou, Jie Zhang, Changzheng Wei, Ying Yan, and Hui Zhang. Topgun: An ecc accelerator for private set intersection. ACM Transactions on Reconfigurable Technology and Systems, 16(4):1–30, 2023.

[39] Jie Zhang, Song Guo, Zhihao Qu, Deze Zeng, Haozhao Wang, Qifeng Liu, and Albert Y Zomaya. Adaptive vertical federated learning on unbalanced features. IEEE Transactions on Parallel and Distributed Systems, 33(12):4006–4018, 2022.

A Notations

We summarize the notations used in this paper in Table 6.

Table 6: Notations used in this paper.

Symbol	Description
P_S, P_C	Two parties involved in Suda
N	The size of large data held by P_S.
n	The size of smaller data held by P_C ($N >> n$).
m	The feature dimension.
n'	The intersection size.
$[N]$	The set $\{0,\ldots,N-1\}$.
I^S	The IDs held by P_S, $I^S = \{id_0^S,\ldots,id_{N-1}^S\}$.
I^C	The IDs held by P_C, $I^C = \{id_0^C,\ldots,id_{n-1}^C\}$.
X	The features held by P_S, $X = X_0\|\|\ldots\|\|X_{m-1}$.
Y	The labels held by P_C.
$\mathcal{X}$	The features in the intersection.
$\mathcal{Y}$	The labels in the intersection.
D^S	The data held by P_S, $D^S = I^S\|\|X$.
D^C	The data held by P_C, $D^C = I^C\|\|Y$.
d	The degree of LFHE polynomial modulo.
p	The LFHE plaintext module.
q	The LFHE ciphertext module.
t	The modulus of the finite field in ECC.
$\langle\cdot\rangle^S, \langle\cdot\rangle^C$	The shares held by P_S and P_C, respectively.

B Improved Protocols

Improved Get New ID Protocol. For the binary classification problem, i.e. the label is binary, we proposed an improved get new ID protocol $\prod_{\text{GetNewID2}}$ (Protocol 8) to reduce the communication overhead. Our improvements mainly stem from replacing AHE with ECC, which has shorter cryptographic keys. Compared with $\prod_{\text{GetNewID}}$, $\prod_{\text{GetNewID2}}$ has the same processes except for the computations related to the label. Therefore, we only describe the operations of the labels here and the detailed protocol is shown in Protocol 8.

Protocol 8: $\prod_{\text{GetNewID2}}$

Input: P_S inputs its data $D^S = I^S||X$, P_C inputs its data $D^C = I^C||Y$.

Output: P_S outputs its new IDs I^S_{new} and P_C outputs its new IDs I^C_{new} in the intersection. Besides, P_S and P_C output label shares $\langle\mathcal{Y}\rangle^S$ and $\langle\mathcal{Y}\rangle^C$ in the intersection, respectively.

Parameters: Hash function $H(\cdot)$ used for hashing an element to a point on the elliptic curve.

Setup:

1: P_S shuffles its data locally, and then sets its new ID as $I^S_{new} = \{0,\ldots,N-1\}$.
2: P_S generates an ECC private key α and computes $\alpha H(I^S) \leftarrow \{\alpha H(id_i^S)\}_{i\in[N]}$.

Online:

1: P_C shuffles its data locally.
2: P_C generates an ECC private key β and ECC key pair (η,κ), as well as a point e on the elliptic curve. Besides, P_C computes $\beta H(I^C) \leftarrow \{\beta H(id_j^C)\}_{j\in[n]}$ and $\mathcal{L} \leftarrow \{\ell_j\}_{j\in[n]}$, where $\ell_j = (2y_j-1)e$. Then P_C encrypts $\mathcal{L}$ to $\text{E.Enc}_\eta(\mathcal{L}) \leftarrow \{\text{E.Enc}_\eta(\ell_j)\}_{j\in[n]}$.
3: P_C sends $\beta H(I^C)$ and $\text{E.Enc}_\eta(\mathcal{L})$ to P_S.
4: P_S computes $\alpha\beta H(I^C)$ and $\mathcal{B}\text{E.Enc}_\eta(\mathcal{L})$ where $\mathcal{B} = \{2b_j-1\}_{j\in[n]}$ and $b_j \in \{0,1\}$. Then P_S shuffles them to $\alpha\beta H(I^C)_\pi$ and $(\mathcal{B}\text{E.Enc}_\eta(\mathcal{L}))_\pi$, where π is a permutation function. P_S sends $\alpha\beta H(I^C)_\pi$, $(\mathcal{B}\text{E.Enc}_\eta(\mathcal{L}))_\pi$ and $\alpha H(I^S)$ to P_C.
5: P_C computes $\alpha H(I^C)_\pi \leftarrow \beta^{-1}\alpha\beta H(I^C)_\pi$ and $\mathcal{B}_\pi\mathcal{L}_\pi \leftarrow \text{E.Dec}_\kappa((\mathcal{B}\text{E.Enc}_\eta(\mathcal{L}))_\pi)$, i.e. $\mathcal{B}_\pi\mathcal{L}_\pi \leftarrow \{(2b_{\pi(j)}-1)l_{\pi(j)}\}_{j\in[n]}$. After that, P_C computes $\{b'_{\pi(j)} \leftarrow \frac{1}{2} - \frac{1}{2}l_{\pi(j)}\}_{j\in[n]}$. Then, P_C sends $\Delta \leftarrow \{\pi(j) \mid \alpha H(id^C_{\pi(j)}) = \alpha H(id_i^S)\}_{j\in[n],i\in[N]}$ to P_S.
6: P_C returns $I^C_{new} \leftarrow \{i|\alpha H(id^C_{\pi(j)}) = \alpha H(id_i^S)\}_{j\in[n],i\in[N]}$, and label shares $\langle\mathcal{Y}\rangle^C \leftarrow \{b'_{\pi(i)}\}_{i\in I^C_{new}}$. P_S returns label shares $\langle\mathcal{Y}\rangle^S \leftarrow \{b_{\pi(j)}\}_{\pi(j)\in\Delta}$.

Specifically, rather than encrypting the label using AHE, P_C computes $\mathcal{L} = \{\ell_j\}_{j\in[n]}$ and encrypts it to $\text{E.Enc}_\eta(\mathcal{L}) = \{\text{E.Enc}_\eta(\ell_j)\}_{j\in[n]}$, where $\ell_j = (2y_j-1)e$, e is a point on the elliptic curve and η is the ECC public key. P_C sends $\text{E.Enc}_\eta(\mathcal{L})$ to P_S. Then, P_S generates n random bit $b_j \in \{0,1\}$ for $j \in [n]$ and computes $\mathcal{B}\text{E.Enc}_\eta(\mathcal{L})$, where $\mathcal{B} = \{2b_j-1\}_{j\in[n]}$. After that, P_S sends $\mathcal{B}\text{E.Enc}_\eta(\mathcal{L})$ to P_C. After shuffling it to $(\mathcal{B}\text{E.Enc}_\eta(\mathcal{L}))_\pi$, P_C sends $(\mathcal{B}\text{E.Enc}_\eta(\mathcal{L}))_\pi$ to P_S. P_C computes $\{b'_{\pi(j)} = \frac{1}{2} - \frac{1}{2}l_{\pi(j)}\}_{j\in[n]}$ and returns $\langle\mathcal{Y}\rangle^C = \{b'_{\pi(j)}\}_{i\in I^C_{new}}$. P_S returns $\langle\mathcal{Y}\rangle^S = \{b_{\pi(j)}\}_{\pi(j)\in[n]}$.

Improved Oblivious Polynomial Reduction Protocol. For the case of $n \le d/4$, where n is the smaller data size and d is the degree of LFHE polynomial modulo, we proposed

an improved oblivious polynomial reduction protocol $\prod_{\text{OPR2}}$ (Protocol 9). We set $\theta = \lceil d/2n \rceil$. Our improvements mainly stem from packing θ ciphertexts into one ciphertext. Here, we assume $m\%\theta = 0$, and for the case where $m\%\theta \neq 0$, it is similar, so we omit the details here.

Protocol 9: $\prod_{\text{OPR2}}$

Input: P_S inputs polynomials $f_k(x)$ with $deg(f_k(x)) \leq N-1$ for $k \in [m]$, P_C inputs LFHE key pair (fpk, fsk) and $g(x) = \prod_{i=0}^{n'-1}(x - z_i)$ for $i \in [n']$.

Output: P_S outputs $\sum_{j=0}^{\theta-1} x^j \text{F.Enc}_{fpk}(\bar{f}_{i\theta+j}(x))$ for $i \in [m/\theta]$.

1: P_S constructs $f_{k,j}(x)$ for $k \in [m]$.
2: P_C computes $h_j(x) \leftarrow x^{jn} \bmod g(x)$ for $j \in [\lceil N/n \rceil]$. Then P_C encrypts $h_j(x^\theta)$ to $\text{F.Enc}_{fpk}(h_j(x^\theta))$ for $j \in [\lceil N/n \rceil]$ and sends $\text{F.Enc}_{fpk}(h_j(x^\theta))$ to P_S.
3: P_S computes $\text{F.Enc}_{fpk}(\bar{f}_k(x^\theta))$ following Equation (8) for $k \in [m]$.
4: P_S computes $\sum_{j=0}^{\theta-1} x^j \text{F.Enc}_{fpk}(\bar{f}_{i\theta+j}(x))$ for $i \in [m/\theta]$.

The improved oblivious polynomial reduction protocol $\prod_{\text{OPR2}}$ is shown in Protocol 9. P_S constructs the polynomials $f_{k,j}(x)$ for $k \in [m]$ such that $f_k(x) = \sum_{j=0}^{\lceil N/n \rceil - 1} f_{k,j}(x) \cdot x^{jn}$ for $k \in [m]$. Then P_C computes $h_j(x) = x^{jn} \bmod g(x)$ for $j \in [\lceil N/n \rceil]$. After that P_C encrypts $h_j(x^\theta)$ to $\text{F.Enc}_{fpk}(h_j(x^\theta))$ for $j \in [\lceil N/n \rceil]$ and sends $\text{F.Enc}_{fpk}(h_j(x^\theta))$ for $j \in [\lceil N/n \rceil]$ to P_S. Subsequently, P_S computes $\text{F.Enc}_{fpk}(\bar{f}_k(x^\theta))$ following Equation (8) for $k \in [m]$. Finally, P_S computes

$$\text{F.Enc}_{fpk}(\bar{f}_0(x^\theta)) \boxplus \cdots \boxplus x^{\theta-1}\text{F.Enc}_{fpk}(\bar{f}_{\theta-1}(x^\theta)),$$
$$\text{F.Enc}_{fpk}(\bar{f}_\theta(x^\theta)) \boxplus \cdots \boxplus x^{\theta-1}\text{F.Enc}_{fpk}(\bar{f}_{2\theta-1}(x^\theta)),$$
$$\cdots$$
$$\text{F.Enc}_{fpk}(\bar{f}_{(m/\theta-1)\theta}(x^\theta)) \boxplus \cdots \boxplus x^{\theta-1}\text{F.Enc}_{fpk}(\bar{f}_{m-1}(x^\theta)).$$

That is, P_S computes $\sum_{j=0}^{\theta-1} x^j \text{F.Enc}_{fpk}(\bar{f}^S_{i\theta+j}(x))$ for $i \in [m/\theta]$ to pack θ ciphertexts into one ciphertext.

Protocol 10: $\prod_{\text{OPI2}}$

Input: P_S inputs $\{r_{i,k}\}_{i\in[n'],k\in[m]}$. P_C inputs LFHE key pair (pk, sk) and z_i for $i \in [n']$.

Output: P_S outputs $\sum_{j=0}^{\theta-1} x^j \text{F.Enc}_{fpk}(\bar{f}^S_{i\theta+j}(x))$ for $i \in [m/\theta]$, where $\bar{f}^S_k(z_i) = r_{i,k}$ for $i \in [n'], k \in [m]$.

1: P_C generates ι polynomials $RF_\mu(x)$ and ι polynomials $CF_\nu(x)$ for $\mu, \nu \in [\iota]$, where $\iota = \lceil \sqrt{n'} \rceil$, such that $RF_\mu(z_i) \cdot CF_\nu(z_i) = 1$ if and only if $i = \mu\iota + \nu$ for $\mu, \nu \in [\iota], i \in [n']$. P_C encrypts $RF_\mu(x^\theta)$ and $CB_\nu(x^\theta)$ to $\text{F.Enc}_{fpk}(RF_\mu(x^\theta))$ and $\text{F.Enc}_{fpk}(CF_\nu(x^\theta))$ for $\mu, \nu \in [\iota]$ respectively and sends them to P_S.
2: P_S computes $\text{F.Enc}_{fpk}(\bar{f}^S_k(x^\theta))$ following Equation (9).
3: P_S computes $\sum_{j=0}^{\theta-1} x^j \text{F.Enc}_{fpk}(\bar{f}^S_{i\theta+j}(x))$ for $i \in [m/\theta]$.

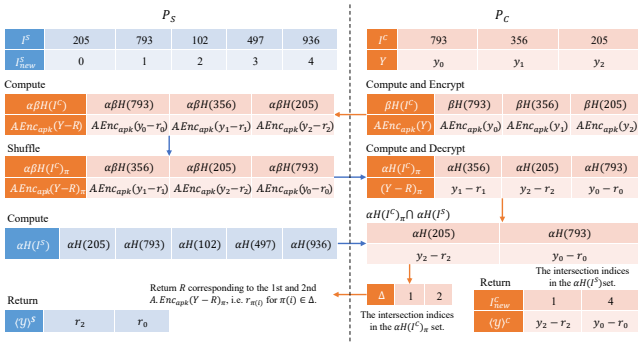

Figure 7: Example of the get new ID protocol

Improved Oblivious Polynomial Interpolation and Evaluation Protocols. The main ideas of the improved oblivious polynomial interpolation and evaluation protocols are similar to the improved oblivious polynomial reduction protocol. The detailed improved oblivious polynomial interpolation and evaluation protocols are shown in Protocol 10 and Protocol 11, respectively.

Protocol 11: $\prod_{\text{OPE2}}$

Input: P_S inputs $\sum_{j=0}^{\theta-1} x^j \text{F.Enc}_{fpk}(\bar{f}_{i\theta+j}(x))$ for $i \in [m/\theta]$. P_C inputs LFHE key pair (fpk, fsk), $g(x) = \prod_{i=0}^{n'-1}(x - z_i)$ and z_i for $i \in [n']$.

Output: P_C outputs $\mathcal{X} = \{x_{id'_i,k} | x_{id'_i,k} = \bar{f}_k(z_i)\}_{i\in[n'],k\in[m]}$.

1: P_C encrypts $g(x^\theta)$ to $\text{F.Enc}_{fpk}(g(x^\theta))$. Then P_C sends $\text{F.Enc}_{fpk}(g(x^\theta))$ to P_S.
2: P_S generates random polynomials $\gamma_i(x) \in \mathbb{F}[x]$ for $i \in [m/\theta]$ with $deg(\gamma_i(x)) < d - \theta n'$. Then P_S computes $\text{F.Enc}_{fpk}(\tilde{f}_i(x)) \leftarrow \sum_{j=0}^{\theta-1} x^j \text{F.Enc}_{fpk}(\bar{f}_{i\theta+j}(x)) \boxplus (\text{F.Enc}_{fpk}(g(x)) \boxtimes \text{F.Enc}_{fpk}(\gamma_i(x))) \boxplus \text{F.Enc}_{fpk}(0)$ for $i \in [m/\theta]$ and sends them to P_C.
3: P_C decrypts $\text{F.Enc}_{fpk}(\tilde{f}_i(x))$ to $\sum_{j=0}^{\theta-1} x^j f'_{i\theta+j}(x) \leftarrow \sum_{j=0}^{\theta-1} x^j \bar{f}_{i\theta+j}(x) + g(x)\gamma_i(x)$ for $i \in [m/\theta]$. P_C returns $x_{id'_i,k} \leftarrow f'_k(z_i)$ for $i \in [n'], k \in [m]$.

C Protocol Examples

We provide an example of the get new ID protocol $\prod_{\text{GetNewID}}$ in Figure 7. Besides, we provide an example of how to compute the reduced polynomial in Figure 8.

D Proof of Theorem 2

Proof. For $\mu, \nu \in [\iota]$, let $I_\mu := \{i | i/\iota = \mu\}_{i\in[\iota^2]}$, $I_\nu := \{i | i\%\iota = \nu\}_{i\in[\iota^2]}$, $J_\mu := [\iota^2] \setminus I_\mu$, $J_\nu := [\iota^2] \setminus I_\nu$. It is worth noting that $|I_\mu| = |I_\nu| = \iota$ and $|J_\mu| = |J_\nu| = \iota(\iota - 1)$.

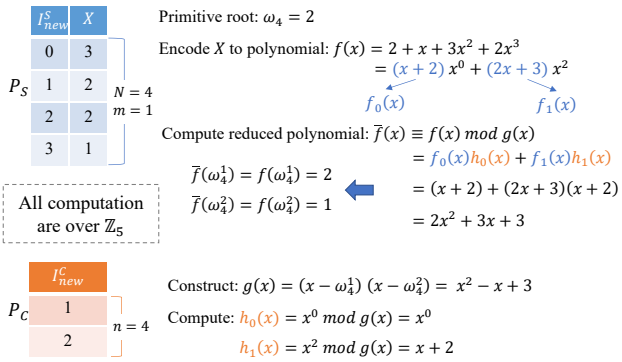

Figure 8: Example of how to compute reduced polynomial.

Then we can construct ι^2 polynomials $\{\tilde{RF}_\mu(x)|\tilde{RF}_\mu(x) = \prod_{i\in J_\mu}(x-z_i)\}_{\mu\in[\iota^2]}$ and ι^2 polynomials $\{\tilde{CF}_\nu(x)|\tilde{CF}_\nu(x) = \prod_{i\in J_\nu}(x-z_i)\}_{\nu\in[\iota^2]}$. These polynomials satisfy:

1. $deg(\tilde{RF}_\mu(x)) = deg(\tilde{CF}_\nu(x)) = \iota(\iota-1)$;
2. $\tilde{RF}_\mu(z_i) = 0$ if and only if $i \in J_\mu$;
3. $\tilde{CF}_\nu(z_i) = 0$ if and only if $i \in J_\nu$.

After that, we can construct two sets of interpolating polynomials $\{I\tilde{RF}_\mu(x)\}_{\mu\in[\iota]}, \{I\tilde{CF}_\nu(x)\}_{\nu\in[\iota^2]}$ which satisfy:

1. $deg(I\tilde{RF}_\mu(x)) < \iota, deg(I\tilde{CF}_\nu(x)) < \iota$;
2. $I\tilde{RF}_\mu(z_i) = \tilde{RF}_\mu(z_i)^{-1}$ for $i \in I_\mu$;
3. $I\tilde{CF}_\nu(z_i) = \tilde{RF}_\mu(z_i)^{-1}$ for $i \in I_\mu$.

At last, let $RF_\mu(x) = \tilde{RF}_\mu(x)I\tilde{RF}_\mu(x)$ and $CF_\nu(x) = \tilde{CF}_\nu(x)I\tilde{CF}_\nu(x)$ for $u, v \in [\iota]$, and they satisfy the properties 1, 2, and 3. □

E Security Proof

Theorem 3. *The protocol $\prod_{\text{Batch\_PIR}}$ (Protocol 3) securely computes $\mathcal{F}_{\text{Batch\_PIR}}$ (Figure 5) against a semi-honest adversary $\mathcal{A}$.*

Proof. We consider the following two cases:
Case 1: Corrupt P_S. We conduct the simulator $\mathcal{S}_S(D^S, \perp)$ that simulates the view of the corrupt P_S. It executes as follows:

- Encode X into polynomials $f_k(x)$ for $k \in [m]$.
- Construct $f_{k,j}(x)$ such that $f_k(x) = \sum_{j=0}^{\lceil N/n\rceil-1} f_{k,j}(x)\cdot x^{jn}$ for $k \in [m]$.
- Construct random polynomials $h_j(x)$ for $j \in [\lceil N/n\rceil]$ and encrypted it as $\text{F.Enc}_{fpk}(h_j(x))$.
- Compute $\text{F.Enc}_{fpk}(\bar{f}_k(x)) \leftarrow \sum_{j=0}^{\lceil N/n\rceil-1} \text{F.Enc}_{fpk}(f_{k,j}(x)) \boxtimes \text{F.Enc}_{fpk}(h_j(x))$ for $k \in [m]$.
- Construct random polynomial $g(x)$ and encrypt it to $\text{F.Enc}_{fpk}(g(x))$.
- Compute $\text{F.Enc}_{fpk}(f'_k(x))$ following the real protocol.

The view simulated by $\mathcal{S}_S(D^S, \perp)$ is computationally indistinguishable from the real one by the LFHE.
Case 2: Corrupt P_C. We conduct the simulator $\mathcal{S}_C(I^C_{new}, \mathcal{X})$ that simulates the view of the corrupt P_C. It executes as follows:

- Construct $g(x) \leftarrow \prod_{i=0}^{n'-1}(x-z_i)$ where $z_i = \omega_N^{id'_i}$ for $i \in [n']$.
- Compute $h_j(x) \leftarrow x^{jn} \bmod g(x)$ for $j \in [\lceil N/n\rceil]$ and encrypts $h_j(x)$ to $\text{F.Enc}_{fpk}(h_j(x))$.
- Construct random polynomials $\bar{f}_k(x)$ with $deg(\bar{f}_k(x)) \le 2(n-1)$, such that $\bar{f}_k(x) = x_{id'_i,k}$ for $i \in [n'], k \in [m]$, where $x_{id'_i,k}$ is P_C's output received by $\mathcal{S}_C$.
- Construct random polynomials $\gamma_k(x) \in \mathbb{F}[x]$ for $k \in [m]$ with $deg(\gamma_k(x)) < d - n'$.
- Compute $f'_k(x) \leftarrow \bar{f}_k(x) + g(x)\gamma_k(x)$ and encrypt it to $\text{F.Enc}_{fpk}(f'_k(x))$ for $k \in [m]$ to simulate the message from P_S.

The view simulated by $\mathcal{S}_C(I^C_{new}, \mathcal{X})$ is computationally indistinguishable from the real one due to the $f'_k(x)$ simulated by $\mathcal{S}_C(I^C_{new}, \mathcal{X})$ and the $f'_k(x)$ in the real world are both uniformly distributed on $(\bar{f}_k(x) + g(x)\gamma_k(x)) \in \mathbb{F}[x]$ for $k \in [m]$. □

Theorem 4. *The protocol $\prod_{\text{Suda}}$ (Protocol 1) securely computes $\mathcal{F}_{\text{Suda}}$ (Figure 4) against a semi-honest adversary $\mathcal{A}$.*

Proof. We consider the following two cases:
Case 1: Corrupt P_S. We conduct the simulator $\mathcal{S}_S(D^S, \mathcal{X}^S||\mathcal{Y}^S)$ that simulates the view of the corrupt P_S. It executes as follows:

- Shuffle its data locally.
- Compute $\alpha H(I^S)$.
- Simple n EC points $\beta H(I^C) \leftarrow \{\beta H(id^C_j)\}_{j\in[n]}$ uniformly.
- Simple n values y_j uniformly and encrypt to $\text{A.Enc}_{apk}(y_j)$ for $j \in [n]$.
- Compute $\text{A.Enc}_{apk}(R)$ following the real protocol.
- Compute $\alpha\beta H(I^C), \text{A.Enc}_{apk}(Y-R)$ and shuffle them to $\alpha\beta H(I^C)_\pi, \text{A.Enc}_{apk}(Y-R)_\pi$.
- Randomly draw n' values from $[N]$ without replacement to simulate Δ.
- Set $\langle\mathcal{Y}\rangle^S \leftarrow \{r_{\pi(j)}\}_{\pi(j)\in\Delta}$.
- Execute the same steps as the former four steps of simulator $\mathcal{S}_S(D^S, I^C_{new})$.

- Construct random polynomials $RF_\mu(x)$ and $CF_\nu(x)$ for $\mu,\nu \in [\lceil\sqrt{n'}\rceil]$ and encrypt them to $\text{F.Enc}_{fpk}(RF_\mu(x))$ and $\text{F.Enc}_{fpk}(CF_\nu(x))$ for $\mu,\nu \in [\lceil\sqrt{n'}\rceil]$ respectively.
- Compute $\text{F.Enc}_{fpk}(\bar{f}_k^S(x))$ following the real protocol.
- Compute $\text{F.Enc}_{fpk}(\bar{f}_k^C(x)) \leftarrow \text{F.Enc}_{fpk}(\bar{f}_k(x)) \boxminus \text{F.Enc}_{fpk}(\bar{f}_k^S(x))$.
- Construct random polynomial $g(x)$, encrypt it to $\text{F.Enc}_{fpk}(g(x))$.
- Compute $\text{F.Enc}_{fpk}(f'_k(x))$ following the real protocol.

The view simulated by $\mathcal{S}_S(D^S, \mathcal{X}^S || \mathcal{Y}^S)$ is computationally indistinguishable from the real one by the discrete logarithm assumption and encryption algorithms, including AHE, LFHE, and ECC.

Case 2: Corrupt P_C. We conduct the simulator $\mathcal{S}_C(D^C, \mathcal{X}^C || \mathcal{Y}^C))$ that simulates the view of the corrupt P_C. It executes as follows:

- Shuffle its data locally.
- Compute $\beta H(I^C)$ and $\text{A.Enc}_{apk}(Y)$ following the real protocol.
- Simple n EC points $\alpha\beta H(I^C)_\pi \leftarrow \{\alpha\beta H(id_\pi^C(j))\}_{j\in[n]}$ uniformly.
- Simple n values $R = \{r_j\}_{j\in[n]}$ uniformly and computes $\text{A.Enc}_{apk}(Y-R)_\pi$.
- Simple N EC points $\alpha H(I^S) \leftarrow \{\alpha\beta H(id_i^S)\}_{i\in[N]}$ uniformly.
- Compute I_{new}^C and $\langle\mathcal{Y}\rangle^C$ following the real protocol.
- Execute the same steps as the former two steps of simulator $\mathcal{S}_S(D^S, I_{new}^C)$.
- Simple random values $r_{i,k}$ for $i \in [n'], k \in [m]$ uniformly.
- Construct polynomials $RF_\mu(x)$ and $CF_\nu(x)$ following the real protocol and encrypt $RF_\mu(x)$ and $CB_\nu(x)$ to $\text{F.Enc}_{fpk}(RF_\mu(x))$ and $\text{F.Enc}_{fpk}(CF_\nu(x))$ for $\mu,\nu \in [\lceil\sqrt{n'}\rceil]$.
- Construct random polynomials $\bar{f}_k^C(x)$ with $deg(\bar{f}_k^C(x)) \le 2(n-1)$ such that $\{\bar{f}_k^C(z_i)\}_{i\in[n'],k\in[m]} = \langle\mathcal{X}\rangle^C$, where $\langle\mathcal{X}\rangle^C$ is P_C's output received by $\mathcal{S}_C$.
- Construct random polynomials $\gamma_k(x) \in \mathbb{F}[x]$ for $k \in [m]$ with $deg(\gamma_k(x)) < d - n'$.
- Compute $f'_k(x) \leftarrow \bar{f}_k^C(x) + g(x)\gamma_k(x)$ and encrypt it to $\text{F.Enc}_{fpk}(f'_k(x))$ for $k \in [m]$.

The view simulated by $\mathcal{S}_C(D^C, \mathcal{X}^C || \mathcal{Y}^C))$ is computationally indistinguishable from the real one due to the discrete logarithm assumption and the $f'_k(x)$ simulated by $\mathcal{S}_C(D^C, \mathcal{X}^C || \mathcal{Y}^C))$ and the $f'_k(x)$ in the real world are both uniformly distributed on $(\bar{f}_k^C(x) + g(x)\gamma_k(x)) \in \mathbb{F}[x]$ for $k \in [m]$. □

Table 7: Comparison between two settings of `iPrivJoin`.

N	n	Framework	Comm. (MB)	Time (s)
2^{20}	1024	`iPrivJoin` (1)	2655.32	182.90
		`iPrivJoin` (2)	27054.87	2674.89
	2048	`iPrivJoin` (1)	3012.62	207.49
		`iPrivJoin` (2)	89278.87	10272.04
	4096	`iPrivJoin` (1)	3389.94	221.58
		`iPrivJoin` (2)	367854.87	43741.82
2^{22}	1024	`iPrivJoin` (1)	9833.19	688.35
		`iPrivJoin` (2)	54198.08	3960.81
	2048	`iPrivJoin` (1)	9989.60	702.45
		`iPrivJoin` (2)	116422.08	11798.79
	4096	`iPrivJoin` (1)	10682.27	727.89
		`iPrivJoin` (2)	79020.80	45143.92
2^{24}	1024	`iPrivJoin` (1)	37913.63	2718.53
		`iPrivJoin` (2)	162770.58	10243.38
	2048	`iPrivJoin` (1)	39389.21	2815.29
		`iPrivJoin` (2)	224994.58	17710.65
	4096	`iPrivJoin` (1)	39464.18	2800.06
		`iPrivJoin` (2)	263005.50	51425.44

F Additional Evaluations

Comparisons between Different Settings of `iPrivJoin`. We evaluate the efficiency of two settings of `iPrivJoin`, i.e. `iPrivJoin` (1): P_S which holds the larger data performs simple hash and P_C which holds the smaller data performs cuckoo hash; and `iPrivJoin` (2): P_S performs cuckoo hash and P_C performs simple hash. The results (Table 7) show that `iPrivJoin` (1) outperforms `iPrivJoin` (2) in both communication size and running time, therefore we choose `iPrivJoin` (1) as our baseline.

Efficiency of `Suda` with Feature Dimension $m = 1000$. Besides, we conduct an experiment to demonstrate the efficiency of `Suda` with fixed feature dimension $m = 1000$ and across $N \in \{2^{20}, 2^{22}, 2^{24}\}$, $n \in \{1024, 2048, 4096\}$. The experimental results are shown in Table 8. We observe that the running time in the WAN setting is comparable to that in the LAN setting. This is because our communication size is small, so the network environment has little impact on our running time.

Table 8: Efficiency of `Suda` with feature dimension $m = 1000$.

N	n	Comm.(MB)	Time (s)	
			LAN	WAN
2^{20}	1024	194.00	690.07	703.02
	2048	316.53	1199.96	1208.36
	4096	565.85	3072.43	3119.86
2^{22}	1024	388.46	2628.86	2631.35
	2048	511.00	3137.61	3159.88
	4096	749.46	5104.59	5153.46
2^{24}	1024	1166.31	10198.90	10362.40
	2048	1288.84	10713.00	10919.70
	4096	1527.30	12593.10	12705.10