Privacy-Preserving Feature Extraction for Detection of Anomalous Financial Transactions*

Martine De Cock,[†] Zekeriya Erkin,[‡] Steven Golob,[†] Dean Kelley,[†] Ricardo Maia,[§] Anderson Nascimento,[†] Sikha Pentyala,[†] Célio Porsius Martins,[‡] Jelle Vos[‡]

[†]University of Washington Tacoma
[‡]Delft University of Technology
[§]University of Brasilia

Abstract

We propose a cross-silo federated architecture in which a payment network system (PNS) denoted by $\mathcal{S}$ has labeled data to train a model $\mathcal{M}$ for detection of anomalous payments. The other entities in the federation are banks $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$ that collaborate with $\mathcal{S}$ to create feature values to improve the utility of $\mathcal{M}$. To jointly extract feature values in a privacy-preserving manner, $\mathcal{S}$ and the banks engage in cryptographic protocols to perform computations over their joint data, without the need for $\mathcal{S}$ and the banks to disclose their data in an unencrypted manner to each other, i.e. our solution provides *input privacy* through encryption, with mathematically verifiable guarantees. To the best of our knowledge, such joint privacy-preserving feature extraction in a federation with horizontally and vertically partitioned data is novel.

Furthermore, to prevent the model from memorizing instances from the training data, the model is trained with a machine learning (ML) algorithm that provides Differential Privacy (DP). Our overall solution therefore provides both *input privacy*, as none of the entities in the federation ever sees the data of any of the other entities in an unencrypted manner, and *output privacy*, as the model and any inferences with that model avoid information leakage about the underlying training data under DP guarantees.

For the privacy-preserving feature extraction we propose a custom protocol based on elliptic curve-based ElGamal and oblivious key-value stores (OKVS). The model is a neural network trained with DP-SGD. We prove that our overall solution is secure in the honest-but-curious setting. Experimental results demonstrate that our solution is efficient and scalable, and that it yields accurate models while preserving input and output privacy.

1 Background

We propose a solution for the problem of privacy-preserving detection of anomalous transactions as put forward in "Track A: Financial Crime" of the Privacy-Enhancing Technologies (PETs) Prize Challenge,[1] hence referred to as "the competition". In the competition's problem setting, there is an entity $\mathcal{S}$ – a payment network system (PNS) – that holds information about financial transactions. Each transaction known to $\mathcal{S}$ has, apart from other information, details of the ordering and beneficiary accounts, and the financial institutions involved in making the transaction. These financial institutions $\mathcal{B}_i$ $(i = 1, \ldots, n)$ are PNS's partner banks which hold information about the ordering and beneficiary accounts. The task is to use the sensitive information residing with all the

*Authors listed in alphabetical order

[1] `https://petsprizechallenges.com/`

entities, i.e. the PNS and the banks, to train a machine learning (ML) model to detect anomalies based on, for example, an unexpected amount or currency, unusual corridors (senders/receivers), unusual timestamps, or unusual account information. As the trained model is evaluated in terms of AUPRC, it should, for each transaction, output a predicted probability that the transaction is an anomaly. The challenge is to train such a model in a privacy-preserving manner, i.e. the training data held by the PNS cannot be disclosed outside its premises, and likewise, the account information held by each bank cannot be disclosed outside that bank (*input privacy*). Furthermore, future inferences with the trained model may be made public, which means that they should prevent sensitive information disclosure about the training data held by the PNS and the banks (*output privacy*). Hereafter, the data held by the PNS or the banks refers to the synthetic dataset provided by SWIFT[2] for the competition purposes, implying that all the transactions that we refer to are synthetic in nature.

Federated Learning (FL) [1] has emerged as a popular paradigm to train global machine learning models over data held my multiple entities (clients). Privacy is preserved to some extent in FL because, instead of raw data, the clients only need to disclose gradients or trained model parameters. Nearly all state-of-the-art FL algorithms and applications assume scenarios in which the data is horizontally partitioned, i.e. each client holds one or more instances. In the competition task however, the data is split both horizontally and vertically, making the mainstream FL paradigm difficult to use as is [2]. Among other things, each of the banks has only relatively few features, and no ground truth labels. Orthogonal to this, it is also well understood by now that FL is not truly privacy-preserving, as information about the clients' training data may leak from the gradients or model parameters (see e.g. [2, 3, 4, 5]). Existing works that use a combination of PETs in the context of FL to provide end-to-end privacy address, to the best of our knowledge, only the scenario of horizontally distributed data [6, 7, 8, 9, 10, 11].

A relevant technique for cross-silo FL is **Secure Multiparty Computation (MPC)**, an umbrella term for cryptographic approaches that allow two or more parties to jointly compute a specified output from their private information in a distributed fashion, without revealing the private information to each other [12]. While MPC typically comes with a substantial computation and communication overhead, a major advantage is that it can easily be applied to scenarios where the data is partitioned horizontally, vertically, or in any other mixed way. Nearly all of the MPC protocols proposed in the literature for model training by us and by others (e.g. [13, 14, 15, 16, 17, 18, 19]) however only protect *input privacy*, i.e. they enable training of a model over data that is distributed among data holders without requiring those data holders to disclose the data. These approaches do not provide sufficient protection if the trained model is to be made publicly known, or even if – as in our scenario – it is only made available for black-box query access, because information about the model and its training data is leaked through the ability to query the model (see e.g. [20, 21, 22, 23]). Formal privacy guarantees in this case can be provided by **Differential Privacy (DP)** [24]. It is however well known that local DP (letting each data holder add noise to its information before sending it out) causes severe utility loss, while global DP (letting each data holder send its information to a trusted curator who is responsible for adding the noise) introduces a single point of failure (namely the curator) and would violate the rules of the competition that sensitive information cannot leave the PNS or the banks in an unprotected manner.

When entering the competition we envisioned a solution that combines MPC with DP for FL, through leveraging the underexplored idea of replacing the trusted curator from the global DP paradigm with an MPC protocol for noise generation. Our proposed solution built upon an approach with which we obtained the highest accuracy in the iDASH2021 Track III challenge on confidential

[2] `www.swift.co`

computing, which consisted of an MPC protocol for training a logistic regression (LR) model with data from multiple hospitals, combined with an MPC protocol to generate noise to perturb the model parameters to provide DP guarantees [25]. While this method [25] can handle data that is horizontally or vertically distributed, it is not directly applicable to the data from the PETs Prize competition because our existing method [25] assumes that all feature values are readily available in the federation. In the PETs Prize competition data however, there are features that carry high signal but have yet to be constructed, by combining information from the PNS and from the banks (e.g. a Boolean feature that represents whether the name of an account holder in a transaction known to the PNS matches with the name of that account holder in the bank's data). For the PETs Prize challenge we therefore envisioned (1) an MPC protocol for joint feature extraction from the data held by the PNS and the banks and (2) an MPC protocol for training of a global model over the output from the PNS's model and the joint features. These MPC protocols, together with an MPC protocol for DP noise generation, would be ran jointly by the PNS $\mathcal{S}$, the banks, and an aggregrator $\mathcal{A}$, in a federated architecture setup.

In Phase 2 of the competition, the requirement to use the *Flower* framework [26] for the implementation was introduced. *Flower* does not support client peer-to-peer communication, which is an important part of MPC. While in *Flower* it is still possible to enable communication between clients by routing all messages through the aggregator as a mediator, we decided to adapt our approach to a solution that lends itself better to the restrictions of the *Flower* framework. The core of our final solution, as presented in this report, consists of a model $\mathcal{M}$ with DP guarantees trained by the PNS on transaction data and a cryptographic protocol for joint feature extraction executed by the PNS and the banks. In our solution, the PNS and the banks (or the nodes that hold data from multiple banks) assume the role of *clients* in the *Flower* framework. During model training, none of the clients nor the aggregator sees any data from any of the other clients. In addition, the model is trained with DP to prevent memorization of instances from the training data. For the privacy-preserving feature extraction step, we propose a custom cryptographic protocol based on elliptic curve-based homomorphic ElGamal and oblivious key-value stores (OKVS) [27]. This protocol allows PNS and the banks to extract features from their combined data with few interactions and without requiring each of the clients to disclose their data in an unencrypted manner. With almost all of the attention in the privacy-preserving machine learning (PPML) literature going to the model training phase, our proposal contributes to filling an important gap on data preprocessing, namely private feature extraction.

Our overall solution provides *input privacy* as well as *output privacy*. From a technical point of view, our private feature extraction is built on top of a novel private set membership protocol that is especially efficient when performing many sequential queries. Moreover, typical protocols for private set membership or private set intersection based on oblivious key-value stores [27] and built using OT extension protocols [28] reveal the protocol's output in the clear. This is problematic when the result of the protocol needs to be used in other private computations. We provide an extension of our protocol that overcomes this limitation by outputting ElGamal encryptions of the data and using a custom private equality test that uses its homomorphic properties. We are not aware of similar constructions in the literature. To make the protocol efficient, we instantiated the cryptosystem over an elliptic curve and implemented it in Rust (with a Python wrapper). The resulting protocol has low computational and communication demands.

2 Threat model

2.1 General cryptographic assumptions and communication infrastructure. In our solution, we model all participants as polynomial time Turing machines (PPT). Our adversaries are also modeled as polynomial time Turing machines. We assume the security of the AES block cipher

(i.e. AES is a secure pseudo-random function). For the elliptic curve cryptography, we assume the decisional Diffie-Helman problem to hold over Curve25519 [29] and Elligator2 [30] to be statistically indistinguishable from randomness.

We implement authenticated and private communication channels between the PNS and the nodes holding the banks' data by using the authenticated encryption mode of operation EAX. To do so, we predistribute symmetric keys among the respective parties.

2.2 Adversarial behavior. We assume the adversaries to be honest-but-curious. That means they follow the protocol specifications, but try to obtain as much information as possible about private information from their inputs, messages exchanged and internal randomness. Our protocols (Sec. 3.2) are designed to prevent adversaries from learning such information (see Sec. 4 for the proofs). We work with static adversaries. Our solutions can be generalized to stronger adversarial models (fully malicious/active adversaries) at the cost of having a reduced efficiency.

2.3 Data privacy for the payment network system. In our solution, the PNS's training data never leaves the PNS, not even in encrypted form. The privacy of the PNS's data is guaranteed even if the result of the financial transaction classification (the predicted probability that the transaction is anomalous) is made public. This privacy guarantee follows directly from the fact that the model is generated from the data with an algorithm that provides Differential Privacy (DP) guarantees (Sec. 3.1). This means that the probability that the algorithm generates a specific model from the data is very similar to the probability of generating that model if a particular transaction had been left out of the data. The latter implies that what the model has memorized about individual transactions is negligible. Obviously, if the result of the classification is not made public by the PNS, no information whatsoever leaks about the PNS's data (a result that follows from our secure distributed feature extraction protocol).

2.4 Data privacy for the banks. In our solution, the banks' data never leaves the banks in plaintext form. In the feature extraction protocol, the banks encrypt their data as ElGamal ciphertexts and encode the ciphertexts in oblivious key-value stores (OKVS), which they send to the PNS. The PNS and the banks perform computations over this data while it is stays encrypted. At the end of the protocol, (1) the PNS and the banks can jointly decrypt the result (Sec. 3.2) and open it to the PNS, or (2) use a protocol extension to compute linear functions (such as generalized linear machine learning models) on the encrypted data.

To adapt to the requirement to use the Flower framework, our implementation uses the former approach, so the PNS learns one bit of information. This bit is 0 if the account information seems to be correct and 1 if there is any indication of unusual account information in either the sender or the receiver account. We note that in this protocol, the PNS does not learn whether the account problem is with the sender or with the receiver, nor what kind of "unusual account problem" (mismatches, flags) is the culprit.

2.5 Threats outside the scope of this work. We do not consider side channel attacks in our proposal, but Protocols 3.1 and 3.3 have been designed using constant-time primitives, and the only variable-time operations do not reveal information about the inputs. However, we do not investigate the security of our solution against these attacks, and give no guarantees about our current implementation's resistance to them.

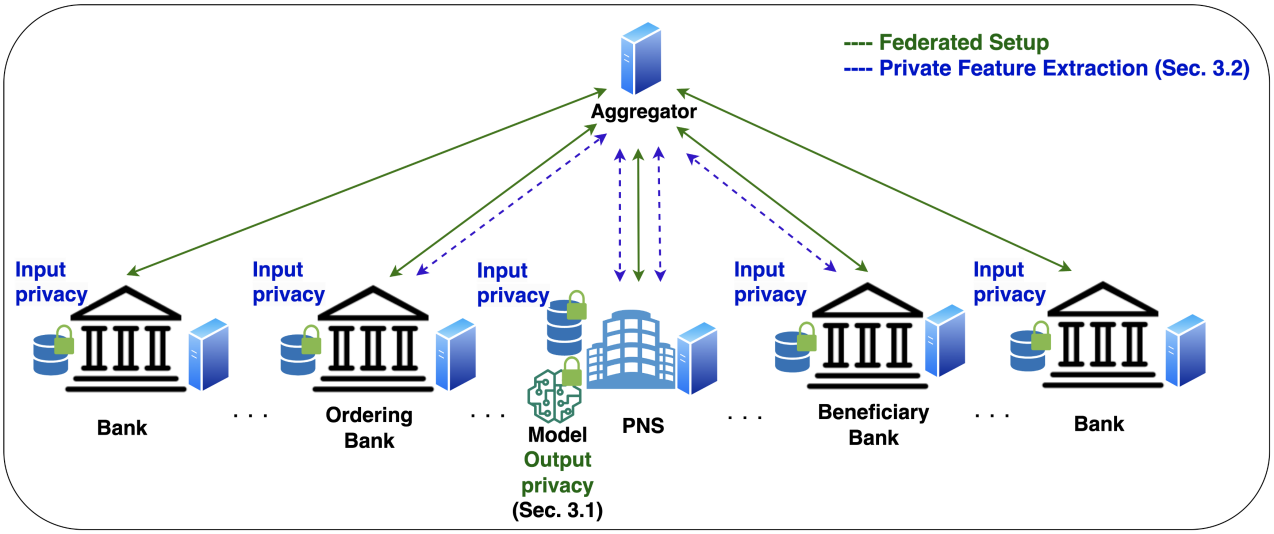

Figure 1: Architecture diagram of the federation

3 Technical approach

3.1 Model training

General. The entities in our solution are the payment network system (PNS) denoted by $\mathcal{S}$, the n banks denoted by $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$, and an aggregator denoted by $\mathcal{A}$. The PNS $\mathcal{S}$ and the banks $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$ are clients in a federated setup, as illustrated in Fig. 1. Any communication between the clients in this setup has to flow through the aggregator $\mathcal{A}$. Each client other than the PNS corresponds to a node that holds the data from one or more banks. For ease of readability, and without loss in generality, we do not make this distinction between nodes and banks here. In our solution, and in our explanation below, each "bank" can also be thought of as a node that holds the data from one or more banks.

$\mathcal{S}$ has a training dataset with labeled transactions. $\mathcal{S}$ represents each transaction $\boldsymbol{x}$ as a feature vector $(x_1, x_2, \ldots, x_m)$ with m features and a binary label y, namely $y = 1$ for an anomalous transaction and $y = 0$ otherwise. Relevant features include the instructed and settlement amount of the transaction, the time at which the transaction is initiated, etc. All m features are known to $\mathcal{S}$ or can be computed locally by $\mathcal{S}$ without consulting any information source outside of $\mathcal{S}$. $\mathcal{S}$ trains a classifier $\mathcal{M}$ based on the training data that $\mathcal{S}$ holds. For a query transaction, i.e. a new transaction that needs to classified as anomalous or not, the model $\mathcal{M}$ outputs a predicted probability $\mathcal{M}(x_1, x_2, \ldots, x_m) \in [0, 1]$ that the transaction is anomalous. To prevent leakage of information from the predicted probabilities about the transactions in the training data, we use an ML model training algorithm that provides Differential Privacy (DP). In this way, our solution gives formal guarantees that the trained model $\mathcal{M}$, and hence predictions made with $\mathcal{M}$, are negligibly affected by the inclusion of any particular transaction in the training data, thereby offering privacy through plausible deniability [24].

Formally, a randomized algorithm $\mathcal{F}$ provides (ϵ, δ)-DP if for all pairs of neighboring datasets D and D' (i.e. datasets that differ in one entity), and for all subsets S of $\mathcal{F}$'s range

$$\mathrm{P}(\mathcal{F}(D) \in S) \leq e^{\epsilon} \cdot \mathrm{P}(\mathcal{F}(D') \in S) + \delta \tag{1}$$

The parameter $\epsilon \geq 0$ denotes the *privacy budget* or privacy loss, while $\delta \geq 0$ denotes the probability of violation of privacy, with smaller values indicating stronger privacy guarantees in both cases.

ϵ-DP is a shorthand for $(\epsilon, 0)$-DP. In our context, D and D' are training datasets with transactions, $\mathcal{F}$ is an algorithm to induce an ML model from a dataset, and $\mathcal{F}(D)$ is an ML model. An ϵ-DP algorithm $\mathcal{F}$ is usually created out of an algorithm $\mathcal{F}^*$ by adding noise that is proportional to the *sensitivity* of $\mathcal{F}^*$, in which the sensitivity measures the maximum impact a change in the underlying dataset can have on the output of $\mathcal{F}^*$.

A variety of (ϵ, δ)-DP ML model training algorithms have been proposed in the literature, including for logistic regression, tree ensembles, and neural networks [31, 32, 33, 34]. The noise that is added by each of these algorithms to perturb model coefficients or gradients to provide privacy typically has a negative impact on the utility of the models. Our overall solution is general enough to allow for any (ϵ, δ)-DP ML model training algorithm to be used by the PNS to train its model $\mathcal{M}$. For our federated submission we used DP-SGD [31]. In Sec. 5 we compare with other DP model training algorithms as well.

Features. We made the following observations in deciding on features for model training:

- **Unexpected amount.** We did not observe any notable difference between the InstructedAmount and the SettlementAmount in the transactions in the train data (synthetic development data). We include `InstructedAmount` as a feature.
- **Unexpected currency.** Most transactions in the train data (synthetic development data) have the same InstructedCurrency and SettlementCurrency. The few transactions that involve two different currencies are all anomalous. We include `SameCurrency` as a feature. The value of this feature is 1 if both currencies in a transaction are the same, and 0 otherwise.
- **Unusual corridors (senders/receivers).** Including sender banks, receiver banks, or sender-receiver bank pairs as features tended to lead to overfitting in our experiments. In a notebook that was made available by the competition organizers, frequency features (such as sender_receiver_freq) were suggested. After initial experimentation, *we decided not to include them because extracting these frequency feature values relies on making many queries over the training data with potential differential privacy violations if one is not careful.* Indeed, if one transaction would be omitted from the dataset, then the frequency feature values of other transactions – and hence the model trained over those extracted values – may change as well. Correctly training a DP model involving such frequency based features would e.g. involve adding noise to the frequency counts and pay out of the privacy budget ϵ for this. We decided not to pursue this and omit said features.
- **Unusual timestamps.** We found the Timestamp and the SettlementDate to be strong indicators of anomalous transactions in the synthetic development data. We encode this as a feature `InterimTime` which is the Timestamp subtracted from the SettlementDate expressed in total number of seconds. We also include a feature `difference_days_absolute` which is a more coarse grained version of InterimTime, rounded up to the number of days.
- **Unusual account information.** We leverage abnormalities in synthetic account information in a feature extraction process between the PNS and the banks, as described in Sec. 3.2.

Differentially private feature normalization. The neural network and logistic regression models that we train benefit from normalizing the feature values in a preprocessing step. Common techniques for feature normalization in ML are based on statistics from the training data, such as the mean and the standard deviation. If one would remove an instance from the training data, then the values of these statistics can change, which has a downstream influence on normalized feature values and inferred labels for test instances.

To make the PNS's model $\mathcal{M}$ end-to-end differentially private, including the feature normalization step, we therefore propose to compute a differential private estimation of the mean of each feature. For a feature x_i and a dataset of size n, the mean is defined as $\mu = \sum_i \frac{x_i}{n}$. We use the Laplace

mechanism for computing differentially private queries of $\sum_i(x_i)$ and n separately, and compute their ratio. We bound the sensitivity of the numerator of μ by clipping the dataset as per Eq. 2, where c is a given publicly known clipping threshold.

$$clip(x_i) = \begin{cases} c & \text{if } x_i > c \\ x_i & \text{if } x_i \leq c \end{cases} \tag{2}$$

We note that the threshold c can be determined by public information about the feature or by looking into a differentially private histogram of the data (at a modest cost to the privacy budget) as to minimize the effect of clipping. In the subsequent steps of our protocol, we work with the clipped dataset. Denoting the DP-private mean by $\tilde{\mu}$, we have:

$$\tilde{\mu} = \frac{\sum_i x_i + Lap(\max_i x_i/\epsilon_1)}{n + Lap(1/\epsilon_2)} = \frac{\sum_i x_i + Lap(c/\epsilon_1)}{n + Lap(1/\epsilon_2)}. \tag{3}$$

Lap denotes the Laplace distribution. Once the DP-private mean for each feature has been determined, we divide the value of each entry in the dataset by its respective DP-private mean.

3.2 Inference

During inference, $\mathcal{S}$ has to infer the probability to which each new transaction $\boldsymbol{x}$ is anomalous. Our solution leverages information held by the banks $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$ to improve the accuracy of the predictions made by model $\mathcal{M}$. This requires communication between the PNS and each of the banks. To comply with the Flower framework, in our implementation we route any messages from the PNS to – and from – a bank through the aggregator $\mathcal{A}$. We pre-distribute symmetric keys among the respective parties to enable such private message passing between the PNS and each of the banks (see Sec. 2.1). The messages in the feature extraction protocol are already encrypted, so we do not encrypt them again.

We assume that $\mathcal{S}$ has a list of valid bank IDs – in our implementation $\mathcal{S}$ receives this information from the clients in a set-up phase – so for each transaction, $\mathcal{S}$ is able to verify locally whether the `Sender` and `Receiver` banks listed in the transaction are valid. If either of them is not, then $\mathcal{S}$ pre-labels the transaction as anomalous with probability 1.0. If all bank IDs in a transaction are valid, then $\mathcal{S}$ identifies the Ordering Bank $\mathcal{B}_s$ (i.e. the Sender bank) and the Beneficiary Bank $\mathcal{B}_r$ (i.e. the Receiver bank)[3] and calls upon these banks to engage in a protocol to create a Boolean feature $B(\boldsymbol{x})$ derived from:

- information from Ordering bank ("Sender" bank) $\mathcal{B}_s$ indicating
  - whether the account ID of the ordering entity as listed in the transaction $\boldsymbol{x}$ is a valid ID known to the ordering bank
  - whether the name of the ordering entity as listed in the transaction $\boldsymbol{x}$ is the same as the name known to the ordering bank
  - whether the street address of the ordering entity as listed in the transaction $\boldsymbol{x}$ is the same as the street address known to the ordering bank
  - whether the country/city/zip of the ordering entity as listed in the transaction $\boldsymbol{x}$ is the same as the country/city/zip known to the ordering bank
  - whether the ordering bank has flagged the ordering entity's account for any reason (e.g. account closed, account frozen,...)
- information from Beneficiary bank $\mathcal{B}_r$ ("Receiver" bank) indicating the same as above for the account of the beneficiary entity

[3]We note that $\mathcal{B}_s$ can be the same as $\mathcal{B}_r$ for some transactions.

The Boolean feature $B(\boldsymbol{x})$ is 0 if the account information appears correct and 1 if there is any indication of inconsistency or unusual account information in either the sender or the receiver account. The final inference result for transaction $\boldsymbol{x}$ is computed as $\max(\mathcal{M}(\boldsymbol{x}), B(\boldsymbol{x}))$, in which $\mathcal{M}(\boldsymbol{x})$ denotes the probability predicted by the model trained by a DP algorithm (see Sec. 3.1) and $B(\boldsymbol{x})$ to denotes the boolean feature (value 0 or 1) jointly extracted by the PNS and the banks. Below we explain how in our solution $B(\boldsymbol{x})$ is computed in a privacy-preserving manner.

Privacy-preserving feature extraction protocol (plaintext result)

The purpose of the feature extraction is to compute whether an appropriate entry for the orderer in transaction $\boldsymbol{x}$ exists in the database of $\mathcal{B}_s$ and an appropriate entry for the beneficiary in $\boldsymbol{x}$ exists in the database of $\mathcal{B}_r$. In other words, the ideal functionality is the logical `AND` between two multi-party private membership queries. Importantly, this functionality will be called numerous times both in parallel and sequentially. One established primitive for this purpose are multi-party private set intersections (MPSI), which perform many queries in parallel. However, these protocols are not efficient when executed sequentially. Here, we present a novel protocol that combines a key-idea from two state-of-the-art MPSI protocols.

Our new protocol combines oblivious key-value store (OKVS), which are the corner stone of the most efficient MPSI protocols when the number of elements is large (in the millions) [27], and composed `AND` operations over elliptic curve points [35], which enable MPSI protocols in the star topology, scaling linearly with the number of parties. To be precise, we use OKVS to let each bank send their data in encrypted format once, after which each query is very cheap to compute and requires low bandwidth. By encoding pairs of curve points into the OKVS instead of secret shares, we transform the protocol to the star topology, so the banks do not need to communicate with each other. This has three concrete benefits: it is more communication-efficient, it fits the centralized communication model of the Flower framework, and preserves privacy of the banks as a network observer cannot tell how many transactions flow between each pair of banks.

Conceptually, an OKVS is nothing more than a dictionary that outputs a random-looking value for each key. If the key was encoded in the OKVS, then the value always corresponds to the value that went with it. We choose to work with the PaXoS [36] OKVS with the xxh3 statistical hash function [37]. Note that this hash function does not have to be cryptographically secure, as the security of OKVSs only relies on the statistical indistinguishability of the encoded values. The OKVS function $\mathsf{decode}(D, q)$ returns the value corresponding to the key q in OKVS D. Regardless if the key was encoded in the OKVS, the properties of the OKVS ensure that the returned value is indistinguishable from randomness.

The main challenge that one faces encoding curve points in an OKVS is that the OKVS requires those points to be indistinguishable from random bits. The typical compressed representation of curve points certainly does not satisfy this requirement. For example, a compressed Montgomery point is simply its X coordinate, which must satisfy the strict curve equation. As a result, not every set of random bits is interpretable as the scalar representation of X. Instead, we use the Elligator2 map [30] to map between random-looking bits and curve points. Consequently, we work over Montgomery and twisted Edwards curves.

In our protocols, we work over Curve25519. We denote the scalar group by $\mathbb{Z}_q$ and the curve group by $E(\mathbb{Z}_q)$. Here, $q = 2^{255} - 19$ and $\#E(\mathbb{Z}_q) = 2^{252} + 27742317777372353535851937790883648493$. All parties have access to a generator G. While Curve25519 is defined as a Montgomery curve, it is birationally equivalent to a twisted Edwards curve. Unless specified, we use the twisted Edwards model. We also introduce two functions $\mathsf{ToMontgomery}(x)$ and $\mathsf{ToEdwards}(x, s)$, which switch x between the two curve models. Here, s denotes the sign of the twisted Edwards point, as the

Montgomery point only contains the X-coordinate of the point. We use the Montgomery model to apply the Elligator2 map given by the function $\psi : \mathbb{Z}_q \mapsto E(\mathbb{Z}_q)$ and its inverse ψ^{-1}. The inverse only returns a representative for half of the points in $E(\mathbb{Z}_q)$; otherwise, it returns $\perp$.

We present Algs. 1 and 2 to encode curve points from ElGamal ciphertexts as bytes and back.

During the training phase, each node (which may contain multiple banks) sets up a single OKVS and key pair using Protocol 3.1. This protocol generalizes to any type of data as its only requirement is that the data is hashable. In our implementation, we let bank $\mathcal{B}_i$ encode R_i in the OKVS, which contains the `["Account", "Name", "Street", "CountryCityZip"]` columns. The bank omits any flagged entries from R_i (that is, where `flag != "0"`). The PNS performs a different setup, only generating a single key pair.

1: **procedure** ToBytes(p)
2: $p_{\mathcal{E}} \leftarrow$ ToMontgomery(p)
3: ▷ Choose + or - representative for p
4: $s \in_R \{+, -\}$
5: $B \in \psi^{-1}(p_{\mathcal{E}}, s)$
6: **if** $B = \perp$ **then**
7: **return** $\perp$
8: **if** FromBytes(B) $= p$ **then**
9: **return** B
10: ▷ Encode the sign in the MSB
11: $B[31] \leftarrow B \vee 128$
12: **return** B

Algorithm 1: Encodes a twisted Edwards point p in 32 bytes indistinguishable from randomness.

1: **procedure** FromBytes(B)
2: ▷ Extract the sign from the MSB
3: $s \leftarrow B[31] \gg 7$
4: **return** ToEdwards($\psi(B), s$)

Algorithm 2: Decodes a twisted Edwards point from 32 bytes B.

Input: Set R_i containing the rows of the bank $\mathcal{B}_i$'s database.
Output: Public key $pk_i \in E(\mathbb{Z}_q)$, secret key $sk_i \in \mathbb{Z}_q$, and OKVS D_i.

1. Bank $\mathcal{B}_i$ randomly generates $sk_i \in_R \mathbb{Z}_q$ and computes $pk_i \leftarrow sk_i\, G \in E(\mathbb{Z}_q)$.
2. Bank $\mathcal{B}_i$ generates $v_j \leftarrow$ ToBytes($r_j\, G$) || ToBytes($r_j\, pk_i$) where $r_j \in_R \mathbb{Z}_q$ for $j = 1, \ldots, |R_i|$. If ToBytes returns $\perp$, the bank resamples r_j.
3. Bank $\mathcal{B}_i$ encodes OKVS D_i where the keys are the rows in R_i, and the values are v_j for $j = 1, \ldots, |R_i|$.
4. Bank $\mathcal{B}_i$ sends pk_i and D_i to the PNS.

Protocol 3.1: One-time setup for each bank: creation of oblivious key-value stores (OKVS)

Protocol 3.2, as described below, is the feature extraction protocol used in our submitted code. The PNS must generate two queries: one for the ordering bank, and one for the beneficiary. The selected fields are `["OrderingAccount", "OrderingName", "OrderingStreet", "OrderingCountryCityZip"]` and `["BeneficiaryAccount", "BeneficiaryName", "BeneficiaryStreet", "BeneficiaryCountryCityZip"]`, respectively. The PNS must also know the OKVSs of

these two banks. After finishing the protocol, the PNS is the only party who receives the output.

Protocol 3.2 is based on ElGamal encryptions over the twisted Edwards curve. The intuition behind it is that each bank's OKVS will only output valid encryptions of the identity $\mathcal{O}$ when the query actually matches a row in that bank's database. Steps 2–4 of the protocol are there to ensure that the encrypted value is multiplied by a random scalar by all three parties, so no information is leaked apart from the Boolean result. In steps 5–6, the three parties collaboratively decrypt the result such that only PNS receives the output. We note that whenever curve points are sent between the parties, we compress them to a single X coordinate for space-efficiency.

Input: OKVS D_s and D_r representing the databases and queries q_s and q_r.
Output: True if $q_s \in R_s \wedge q_r \in R_r$, otherwise false with overwhelming probability.

1. PNS computes $\hat{x}_s \,||\, \hat{y}_s \leftarrow \mathsf{decode}(D_s, q_s)$ and $\hat{x}_r \,||\, \hat{y}_r \leftarrow \mathsf{decode}(D_r, q_r)$. It transforms them into a set of Edwards points:
$$x_s \leftarrow \mathsf{FromBytes}(\hat{x}_s)\ ,\ y_s \leftarrow \mathsf{FromBytes}(\hat{y}_s)\ ,\ x_r \leftarrow \mathsf{FromBytes}(\hat{x}_r)\ ,\ y_r \leftarrow \mathsf{FromBytes}(\hat{y}_r)\ .$$
2. PNS generates $z \in_R \mathbb{Z}_q$ and computes:
$$a \leftarrow z x_s\ ,\quad b \leftarrow z x_r\ ,\quad c \leftarrow zG\ ,\quad d \leftarrow z(y_s + y_r + pk_{\mathcal{S}})\ .$$
It sends (a, b, c, d) to banks $\mathcal{B}_s$ and $\mathcal{B}_r$.
3. Banks $\mathcal{B}_i$ for $i \in \{s, r\}$ generate $z_i \in_R \mathbb{Z}_q$ and compute:
$$\hat{a}_i \leftarrow z_i a\ ,\quad \hat{b}_i \leftarrow z_i b\ ,\quad \hat{c}_i \leftarrow z_i c\ ,\quad \hat{d}_i \leftarrow z_i d\ .$$
They then send $(\hat{a}_i, \hat{b}_i, \hat{c}_i, \hat{d}_i)$ to PNS.
4. PNS computes:
$$\alpha \leftarrow \hat{a}_s + \hat{a}_r\ ,\quad \beta \leftarrow \hat{b}_s + \hat{b}_r\ ,\quad \gamma \leftarrow \hat{c}_s + \hat{c}_r\ ,\quad \delta \leftarrow \hat{d}_s + \hat{d}_r\ .$$
It sends α to $\mathcal{B}_s$ and β to $\mathcal{B}_r$.
5. Bank $\mathcal{B}_s$ computes $\hat{\alpha} \leftarrow sk_s \alpha$, bank $\mathcal{B}_r$ computes $\hat{\beta} \leftarrow sk_r \beta$, and they send $\hat{\alpha}$ and $\hat{\beta}$ to PNS.
6. PNS checks if $\delta \stackrel{?}{=} \hat{\alpha} + \hat{\beta} + sk_{\mathcal{S}}\, \gamma$.

Protocol 3.2: Checks a transaction's consistency. $\mathcal{B}_s$ is the sender's bank, and $\mathcal{B}_r$ the receiver's.

Privacy-preserving feature extraction protocol (encrypted result)

In the protocol above, the PNS receives the Boolean result of the computed feature, revealing that this account is valid. We propose an extension of the protocol to minimize this leakage. We note, however, that this approach does not allow our method to output a confidence score between 0 and 1. Instead, the method outputs 0 or 1. This is an inherent limitation of the ideal functionality. For the purpose of the competition, we have not implemented this extension in the code submission. In Section 4.3, we provide an argument why the leaked Boolean feature is permissible with regards to the bank's privacy (and that of the account holder).

In this extension, the inference computation changes from $\max(\mathcal{M}(\boldsymbol{x}), B(\boldsymbol{x}))$ to $\mathcal{M}(\boldsymbol{x}) \geq t \vee B(\boldsymbol{x})$ for some pre-defined threshold t. Apart from $\mathcal{M}(\boldsymbol{x}) \geq t$, the inference can be entirely computed under encryption by simply adding $\mathcal{M}(\boldsymbol{x}) \geq t$ in encrypted form to the ElGamal ciphertext in step 2 of Protocol 3.2.

The PNS can also perform inference entirely in the encrypted domain by training a differentially-private model on both the Boolean feature and some other features. It would do so by omitting steps

5–6 of Protocol 3.2 and instead performing a secure comparison operation to check if the ElGamal encryption equals the identity $\mathcal{O}$. This is a 3-out-of-3 threshold ElGamal, which is essentially a triple encrypted standard ElGamal encryption. To decrypt, $\mathcal{B}_s$ must multiply a by sk_s, $\mathcal{B}_r$ must do so for b, and $\mathcal{S}$ for c. We present a custom secure equality operation for this purpose in Protocol 3.3. Unlike typical equality protocols, it does not require full decryptions (we only check if the decryption is the identity) and no conversions to secret shares. It functions in the multi-party setting and scales linearly with the number of parties. After running this protocol, the PNS can run any quantized linear model over the resulting ElGamal ciphertext (linear over the Boolean feature, the other features are plaintexts). The three parties proceed to finish the protocol using steps 5–6 in Protocol 3.2 to decrypt the final result.

Input: Ciphertext c encrypting either the identity $\mathcal{O}$ or another point in $E(\mathbb{Z}_q)$.
Output: Ciphertext c' encrypting $\mathcal{O}$ if c encrypted $\mathcal{O}$, otherwise c' encrypts G.

1. The first party $\mathcal{P}_1$ creates a set $C \leftarrow \{c\}$ and ciphertexts c_0 and c_1 encrypting $\mathcal{O}$ and G, respectively.
2. Each party $\mathcal{P}_i$ for $i = 1, \ldots, p$, in turn, does the following:
   - It flips k coins $r_{i,j} \in_R \{0, 1\}$.
   - It swaps c_0 and c_1 if $r_i = \sum_{j=1}^{k} r_{i,j} = 1 \pmod 2$, setting $c_0 \leftarrow c_{r_i}$ and $c_1 \leftarrow c_{1-r_i}$.
   - For $j = 1, \ldots, k$, it adds a ciphertext encrypting $\mathcal{O}$ to C if $r_{i,j} = 0$, or a ciphertext encrypting randomness otherwise.
   - It multiplies each ciphertext in C by some random scalar from $\mathbb{Z}_q$.
   - It shuffles set C and sends the elements to party $\mathcal{P}_{i+1}$.
3. Parties $\mathcal{P}_1, \ldots, \mathcal{P}_p$ collaboratively decrypt the ciphertexts in C and count the number of non-identity elements as t.
4. The first party $\mathcal{P}_1$ outputs ciphertext $c' \leftarrow c_{t \pmod 2}$ (without decrypting it).

Protocol 3.3: Secure equality protocol between multiple parties $\mathcal{P}_1, \ldots, \mathcal{P}_p$ for threshold additively homomorphic encryptions of points in $E(\mathbb{Z}_q)$.

The intuitive understanding of the above protocol is that each party either adds an even or odd number of encryptions of $\mathcal{O}$ to the set C. So long as one party does not collude, it is not clear from the decrypted ciphertexts whether the remaining party added an even or odd number of identity encryptions. The security is decided by the number of ciphertexts k that each party adds. We explain the correctness of this protocol in more detail in Section 4.2, along with our choice of k and its impact on the protocol's security.

3.3 Centralized solution

In our centralized solution (i.e. the baseline solution, which is not privacy-preserving) all the banks give their data to the PNS without any encryption. This enables the PNS to extract the Boolean feature $B(\boldsymbol{x})$ without further involvement of the banks and without the need for the custom protocol from Sec. 3.2. Furthermore, the PNS trains a model $\mathcal{M}'$ on its labeled training instances using a traditional ML algorithm, without DP. In our centralized submission, PNS uses a Random Forest (RF) to this end. For more details about this RF in the centralized setting, see Sec. 5.

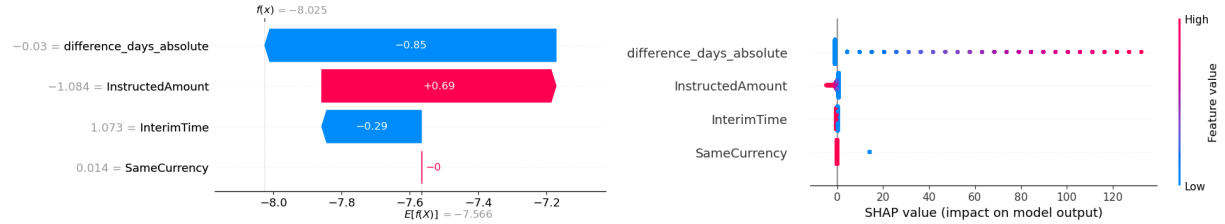

Figure 2: Visualization of SHAP values of an LR model trained with DP guarantees (see LR-DP in Sec. 5) to illustrate the effect of individual features on the model output.

3.4 Privacy vs. utility, scalability, and explainability tradeoffs of our solution

Our solution provides both *input privacy* through the use of elliptic curve-based ElGamal and oblivious key-value stores and *output privacy* through the use of DP. Compared to the centralized solution, which does not offer any privacy, our technique for *input privacy* preserves the same utility while increasing the computational cost, while our technique for *output privacy* affects the utility while having negligible impact on the computational cost.

- **Impact on utility.** The introduction of DP in our federated solution leads to some decrease in utility, which is inversely proportional to the privacy, i.e. the larger the privacy budget ϵ, the higher the utility. The introduction of cryptographic protocols on the other hand does not lead to any further decrease in utility, because the results of the computations over encrypted data are the same as the results obtained in the clear, i.e. in the centralized setup where no encryption is used.
- **Impact on efficiency.** Training the model with DP guarantees leads to a small increase in runtime when compared to model training with no privacy guarantees. However, the difference in runtimes due to this is negligible considering the total runtime of our privacy-preserving solution. Moreover, the cryptographic feature extraction protocol only moderately increases runtime and memory usage as it works over the fast and compact Curve25519, and only requires a dozen curve operations per query. This is all made possible since the majority of the effort can be done once, at setup, by letting each bank encode their dataset in a space-efficient OKVS.
- **Impact on scalability.** In our solution, the number of clients only plays a role in the inference step (Sec. 3.2). One of the more demanding operations is encoding an OKVS. Fortunately, there is only one OKVS per client, which may hold the data of several banks if there are several banks at that client, and it only has to be computed once at setup. Since the cryptographic protocols always only involve 3 parties, their complexity does not increase when more clients join the system.
- **Usability and explainability of the model.** As $\mathcal{S}$ holds a differentially private model $\mathcal{M}$, it can explain the inference w.r.t the contribution from features from the PNS [38] for the particular transaction (see e.g. Fig. 2). Any further explanation on the features held by the PNS [39] can be provided by the PNS at the risk of privacy leakage which can be further controlled by red tape processes.

4 Proof of privacy

Our method contains the following components: a model $\mathcal{M}$ trained by the PNS with a DP algorithm (Sec. 3.1), and a protocol for privacy-preserving feature extraction from data across the PNS and the banks (Sec. 3.2). So, we break down the arguments for proving the security of our solution into two parts: a proof that DP guarantees are correctly added to the model, and a proof that the feature

extraction protocol is secure. After that, we provide a privacy argument for the method as a whole.

4.1 DP model training

The differential privacy guarantees of our solution for the PNS data follows from differentially private data normalization. The differentially private data normalization $\tilde{\mu}$ is as specified in Eq. 3 for each feature. Privacy of $\tilde{\mu}$ follows from the Laplace mechanism, the sensitivities of the numerator (c), and the denominator (1, since it is a count query), and the post-processing property of DP. The spent privacy budget is $(\epsilon_1 + \epsilon_2)$ times the total number of normalized features. This follows from sequential composition of pure DP.

The privacy of the model training follows from the differential privacy guarantees of DP-SGD [31]. Denoting the privacy budget spent by DP-SGD by ϵ_{train}, we arrive at an (ϵ, δ) solution where $\epsilon = |f| \cdot (\epsilon_1 + \epsilon_2) + \epsilon_{train}$, where $|f|$ is the number of normalized features used in our model and δ comes from DP-SGD. δ is empirically set to $\frac{1}{|D|}$, where $|D|$ is the cardinality of the training dataset.

Because of the DP guarantees of our model, and the post-processing property of DP, the privacy of the training dataset is protected even when results inferred with the model are made public. Our solution has the interesting aspect of using private features for inference. These private features are the result of secure computations performed with PNS and bank data. It is clear that the output of the classifier will (and should) reveal information about these private features if they are useful at all for the classification. Thus, we do not offer privacy of the input that is classified. We only provide privacy for the training data during the inference phase.

4.2 Feature extraction

At the core of feature extraction is Protocol 3.2, which the PNS and the banks perform for each transaction in the inference phase (in our implementation, this is done in parallel). Here, we first provide a proof of correctness of this protocol and then show that it is secure in the semi-honest model. We also provide a short security argument for the secure equality operation presented in Protocol 3.3.

Proof of correctness (core protocol)

Claim 1. *Protocol 3.2 returns true when $q_s \in R_s \wedge q_r \in R_r$.*

Proof. Working backwards through the protocol:

$$\delta = sk_s\alpha + sk_r\beta + sk_{\mathcal{S}}\gamma \ , \qquad \text{(Step 5)}$$

$$\hat{d}_s + \hat{d}_r = sk_s(\hat{a}_s + \hat{a}_r) + sk_r(\hat{b}_s + \hat{b}_r) + sk_{\mathcal{S}}(\hat{c}_s + \hat{c}_r) \ , \qquad \text{(Step 4)}$$

$$\cancel{(z_s + z_r)}\hat{d} = sk_s\cancel{(z_s + z_r)}\hat{a} + sk_r\cancel{(z_s + z_r)}\hat{b} + sk_{\mathcal{S}}\cancel{(z_s + z_r)}\hat{c} \ , \qquad \text{(Step 3)}$$

$$\not{z}(y_s + y_r + pk_{\mathcal{S}}) = sk_s\not{z}a + sk_r\not{z}b + sk_{\mathcal{S}}\not{z}G \ . \qquad \text{(Step 2)}$$

Since $q_s \in R_s$ and $q_r \in R_r$, then $y_s = sk_s x_s$ and $y_r = sk_r x_r$ by the functionality of an OKVS. Moreover, the setup implies $pk_{\mathcal{S}} = sk_{\mathcal{S}}G$. It follows that:

$$y_s + y_r + pk_{\mathcal{S}} = sk_s x_s + sk_r x_r + sk_{\mathcal{S}}G \ , \qquad (4)$$

$$y_s + y_r + pk_{\mathcal{S}} = y_s + y_r + pk_{\mathcal{S}} \ . \qquad (5)$$

□

Claim 2. *Protocol 3.2 returns false with overwhelming probability when $q_s \notin R_s \vee q_r \notin R_r$.*

Proof. Given (4), it must hold with overwhelming probability that:

$$y_s + y_r + pk_{\mathcal{S}} \neq sk_s x_s + sk_r x_r + sk_{\mathcal{S}} G \ .$$

Let us assume that $q_s \notin R_s$ (the argument follows the same when $q_r \notin R_r$). Then, $x_s \neq sk_s a$ with probability $1 - |E(\mathbb{Z}_q)|^{-1}$, where $|E(\mathbb{Z}_q)| = 2^{252} + 27742317777372353535851937790883648493$. As a result, (5) only holds with negligible probability. □

Proof of privacy (core protocol)

In Protocol 3.1 the banks only encode an OKVS and generate an ElGamal keypair. The security of this keypair, which PNS also generates, is implied by the decisional Diffie-Hellman assumption (or more precisely, by the discrete log problem). The security of the OKVS is defined by its indistinguishability from randomness. We achieve this by encoding curve points as strings that are indistinguishable from random bytes, as proposed in Algs. 1 and 2.

Given that Protocol 3.2 is correct (see above) and the ideal functionality is deterministic, what remains is to show that the protocol privately computes the ideal functionality in the semi-honest model [40]. We do so by showing that there exists a simulator that given the input and output can replicate the view of a party without having access to the data of other parties. To be precise, the family of simulated views is computationally indistinguishable from those of actual protocol executions. Our protocol relies on the Diffie-Hellman assumption:

Lemma 1. *The decisional Diffie-Hellman assumption implies that, for a generator point G, unknown scalars $a, b \in_R \mathbb{Z}_q$, and random point $C \in_R E(\mathbb{Z}_q)$:*

$$(a\,G, b\,G, ab\,G) \stackrel{c}{\equiv} (a\,G, b\,G, C) \ .$$

We provide two privacy proofs: one that proves PNS's view $\mathsf{view}_{\mathcal{S}}$ to be simulatable and one for a bank's view $\mathsf{view}_{\mathcal{B}}$. We keep these proofs short. We direct the reader to the work by Vos et al. [35] for a more detailed proof of a comparable protocol. To simplify notation, we do not explicitly pass the source of randomness as an input to the simulator. For the purpose of our arguments, we consider the OKVSs D_s and D_r to be public. Given that their contents are statistically indistinguishable from randomness, this only leaks their size. This first proof shows that the protocol remains private when PNS is corrupted.

Claim 3. *There exists a simulator $\mathsf{Sim}_{\mathcal{S}}$ for PNS in Protocol 3.2, such that:*

$$\{\mathsf{Sim}_{\mathcal{S}}(1^\lambda, q_s, q_r, o)\}_{q_s \in Q,\, q_r \in Q,\, o \in \{0,1\}} \stackrel{c}{\equiv} \{\mathsf{view}_{\mathcal{S}}(q_s, q_r, \lambda)\}_{q_s \in Q,\, q_r \in Q,\, o \in \{0,1\}} \ ,$$

for security parameter $\lambda = 128$, queries q_s and q_r from query space Q, and output o.

Proof. Function $\mathsf{view}_{\mathcal{S}}$ returns inputs q_s and q_r, output o, and all incoming messages . Simulator $\mathsf{Sim}_{\mathcal{S}}$ generates an indistinguishable view by outputting the inputs and output, and randomly sampling messages $\hat{a}_s, \hat{b}_s, \hat{c}_s, \hat{d}_s, \hat{a}_r, \hat{b}_r, \hat{c}_r, \hat{d}_r, \hat{\alpha}, \hat{\beta} \in_R E(\mathbb{Z}_q)$. These messages are indistinguishable from those received in actual executions:

- In step 3, $\hat{a}_i = z_i a = z_i p_a G$ for some p_a unknown to PNS. Given Lemma 1, $\hat{a}_i$ is computationally indistinguishable from randomness, even when given $a = p_a G$ and $z_i G$ (the latter is not actually given). The same argument applies to $\hat{b}_i$, $\hat{c}_i$, and $\hat{d}_i$.
- In step 5, PNS receives $\hat{\alpha} = sk_s \alpha = sk_s p_\alpha G$ and $\hat{\beta} = sk_r \beta = sk_r p_\beta G$. Given Lemma 1, α is computationally indistinguishable from randomness, even when given $pk_r = sk_r G$ and $\beta = p_\beta G$. The same argument applies to $\hat{\beta}$. □

Next, we prove that the protocol remains private when either bank is corrupted. The proof below applies to the sending bank, but the argument for the receiving bank is almost identical (replacing α with β). We assume that the two banks are not colluding.

Claim 4. *There exists a simulator* $\mathsf{Sim}_{\mathcal{B}}$ *for bank* $\mathcal{B}_s$ *in Protocol 3.2, such that:*

$$\{\mathsf{Sim}_{\mathcal{B}}(1^{\lambda})\} \stackrel{c}{\equiv} \{\mathsf{view}_{\mathcal{S}}(\lambda)\} ,$$

for security parameter $\lambda = 128$ *(the banks do not output anything).*

Proof. Function $\mathsf{view}_{\mathcal{B}}$ returns all incoming messages of bank $\mathcal{B}_s$. Simulator $\mathsf{Sim}_{\mathcal{B}}$ generates an indistinguishable view by randomly sampling messages $a, b, c, d, \alpha \in_R E(\mathbb{Z}_q)$. These messages are indistinguishable from those received in actual executions:

- In step 2, the bank receives $a = zx_s = zp_aG$, $b = zx_r = zp_bG$, $c = zG$, and $d = z(y_s + y_r + pk_{\mathcal{S}}) = zp_dG$. Given Lemma 1, a is computationally indistinguishable from randomness, even when given $c = zG$ and p_aG (which may be guessed by this bank). The same argument applies to b and d. Since z is random, $c = zG$ is statistically indistinguishable from randomness.
- In step 4, the bank receives $\alpha = \hat{a}_s + \hat{a}_r$, which is indistinguishable from randomness since $\hat{a}_r$ is unknown to bank $\mathcal{B}_s$ given that the queried banks are not colluding. □

We note that one might also give a proof that proves that the protocol remains private when two of the three parties collude. This would require a more sophisticated simulator, which looks similar to that in the work by Vos et al. [35].

Correctness and privacy of the secure equality protocol

Finally, we move on to prove the security of our secure equality protocol 3.3.

Claim 5. *Protocol 3.3 correctly and privately computes an equality.*

Proof. Verifying correctness of the secure equality protocol (Protocol 3.3) comes down to verifying its behavior depending on whether a party's coin tosses come out to an even or odd number of 1s. We study the case where there is one party, but the argument extends trivially to multiple parties.

- If $r_1 = 0$, then c_0 encrypts $\mathcal{O}$ and c_1 encrypts G. If c encrypts $\mathcal{O}$, then $c' = c_0$, which is correct. Otherwise, $c' = c_1$, which is also correct.
- If party $\mathcal{P}_i$ has $r_i = 1$, then c_0 encrypts G and c_1 encrypts $\mathcal{O}$. If c encrypts $\mathcal{O}$, then $c' = c_1$, which is correct. Otherwise, $c' = c_0$, which is also correct.

Next, we analyze the security of Protocol 3.3. We do not consider the security of the ElGamal scheme, which we discussed previously. In the protocol, each party both randomizes and shuffles the set of ciphertexts C. As a result, the only meaningful information that is revealed when the set is decrypted is the number of identity points. We refer to the number of non-identity points as t.

If a party would only perform random coin flips, the number of non-identity points t is given by $P(\boldsymbol{t} = t) = \binom{k}{t}0.5^k$. However, since we are inserting ciphertext c, this changes to $P(\boldsymbol{t} = t) = P(c = \mathcal{O})\binom{k-1}{t}0.5^{k-1} + P(c \neq \mathcal{O})\binom{k-1}{t-1}0.5^{k-1}$, where we use $c \neq \mathcal{O}$ to denote that c does not encrypt $\mathcal{O}$.

Using this, we derive the posterior probability that $c \neq \mathcal{O}$ given the number of points t.

$$\begin{aligned} P(c \neq \mathcal{O} | \boldsymbol{t} = t) &= \frac{P(\boldsymbol{t} = t | c \neq \mathcal{O}) P(c \neq \mathcal{O})}{P(\boldsymbol{t} = t)} \,, \\ &= \frac{\binom{k-1}{t-1} \cancel{0.5^{k-1}} P(c \neq \mathcal{O})}{(1 - P(c \neq \mathcal{O}))\binom{k-1}{t} \cancel{0.5^{k-1}} + P(c \neq \mathcal{O})\binom{k-1}{t-1} \cancel{0.5^{k-1}}} \,, \\ &= \frac{\binom{k-1}{t-1} P(c \neq \mathcal{O})}{(1 - P(c \neq \mathcal{O}))\binom{k-1}{t} + P(c \neq \mathcal{O})\binom{k-1}{t-1}} \,. \end{aligned}$$

An adversary's strongest attack guesses $c = \mathcal{O}$ when $P(c \neq \mathcal{O} | \boldsymbol{t} = t) < \frac{1}{2}$ and $c \neq \mathcal{O}$ otherwise. This leads to the following expected guessing chance:

$$\sum_{t=0}^{k} P(\boldsymbol{t} = t) \underbrace{\left(\frac{1}{2} + \left| \frac{1}{2} - P(c \neq \mathcal{O} | t = t) \right| \right)}_{\text{Guess based on the posterior}} \,. \tag{6}$$

An adversary who does not have access to the protocol's result can only guess using its knowledge about the prior probability of c. In other words, it will only succeed with probability $\frac{1}{2} + \left| \frac{1}{2} - P(c = \mathcal{O}) \right|$. Using (6), we formulate the advantage of an adversary using our Protocol 3.3, and restrict it to 2^{-40}, which we deem negligible:

$$\left(\sum_{t=0}^{k} P(\boldsymbol{t} = t) \left(\frac{1}{2} + \left| \frac{1}{2} - P(x = 1 | t = t) \right| \right) \right) - \left(\frac{1}{2} + \left| \frac{1}{2} - P(c \neq \mathcal{O}) \right| \right) \leq 2^{-40} \,. \tag{7}$$

We very conservatively estimate $P(c \neq \mathcal{O}) \leq 0.05$. Then, the first k for which (7) holds is $k = 44$. □

4.3 Privacy overall

Since our submitted model reveals the Boolean feature computed by Protocol 3.2, the PNS can tell with certainty that its queries match the data from the banks. So, by the definition of the ideal functionality, the PNS can tell with full certainty whether the data from its transactions are members of the banks' databases. We note, however, that an adversary without the Boolean feature can perform such an attack with extremely high likelihood given access to PNS's database. For example, when analyzing the PNS's training data (synthetic development data), $\frac{46631}{47218} = 98.76\%$ of its unique (account, name, address) tuples are members of the banks' databases. As such, the concrete advantage of this attack when the PNS has access to the Boolean feature is only 1.24%. Since we assume the semi-honest model, the PNS can also not perform queries on data that is not in its list of transactions, meaning that the feature is barely exploitable. In the malicious model, the PNS could perform membership queries on data that is not in its database, but it would have to guess the other fields to be succesful, which makes this attack hard to analyze.

5 Experimental results

Unless otherwise specified, all experiments in this section are performed on the *synthetic development data* made available during Phase 2 of the competition. This data includes a *synthetic train dataset* with 2,990,349 negative and 3,521 positive instances, and a *synthetic test dataset* with 1,002,395 negative and 1,279 positive instances.

AUPRC	privacy	RF	LR	MLP	LR_{best}
with DP	$\epsilon = 0.5$	0.667	0.550	0.741	0.935
	$\epsilon = 1.0$	0.742	0.749	0.771	0.935
	$\epsilon = 5.0$	0.671	0.757	0.776	0.941
without DP	$\epsilon = \infty$	0.976	0.803	0.776	0.943

with DP:
– RF-DP: Fletcher et al. [34]
– LR-DP: objective perturbation method [33]
– MLP-DP, LR_{best}-DP: DP-SGD [31]
without DP:
– RF, LR: sklearn
– MLP: TensorFlow
– LR_{best}: TensorFlow

Table 1: Utility-privacy tradeoff of models augmented with the account validity feature extracted jointly by the PNS and the banks. Higher values of ϵ denote a larger privacy budget, i.e. less privacy. The AUPRC results are independent of the number of clients/partitions. All reported results are an average over 5 runs. The highlighted cells correspond to best performing models in centralized and federated settings on our local evaluation.

5.1 Utility-privacy tradeoffs

The utility results in Tab. 1 are obtained by fitting models on the train dataset and evaluating them on the test dataset in terms of AUPRC (area under the precision-recall curve). We trained 3 kinds of models: Random Forest (RF), Logistic Regression (LR), and Multilayer Perceptron (MLP). The first three models in Tab. 1 are trained on the feature set `[InstructedAmount, SameCurrency, InterimTime, difference_days_absolute]` (Sec. 3.1). The fourth model (LR_{best}) is trained on the feature set `[bin_features, SameCurrency]` where `bin_features` is a discretized representation of the `InterimTime` feature as we explain in more detail below. The predictions made by all the models in Tab. 1 are augmented with the Boolean feature about the validity of the bank accounts that is extracted from data of the PNS and the banks (Sec. 3.2).

The RF and LR models in the centralized setting are trained with sklearn,[4] and the MLP with TensorFlow. For the first model – RF – in Tab. 1, in the centralized setting we used 20 trees with max_depth = 10. For the LR model and the MLP model (second and third model in Tab. 1), we preprocessed the appropriate features with a StandardScaler (i.e. normalization by replacing feature values with z-scores). For the MLP we used one hidden layer with 250 nodes and RELU activation, and 1 output layer with sigmoid activation. For the fourth model – LR_{best} – we used the one hot encoded bin features and the Boolean currency based feature.

For ease of reference, we denote the models that we trained in the federated setting by appending "-DP" to indicate that they are differentially private. To train the RF-DP and LR-DP models in Tab. 1 we used the diffprivlib library.[5] For MLP-DP and LR-$_{best}$-DP we used the implementation of DP-SGD in tensorflow_privacy.[6] Similar to the centralized setting, RF-DP is trained with 20 trees and max_depth = 10. The way in which trees are constructed in this RF-DP approach [34] is quite different from the standard RF algorithm in sklearn that we used in the centralized approach. While in the standard RF algorithm each node in each tree is selected by evaluating it against the data, in the RF-DP approach, intermediate nodes and threshold values for these nodes are generated at random, to limit the number of queries needed against the data and stretch the privacy budget further. While in the centralized setting we obtained our best results with RF, this was no longer the case with RF-DP because some of our features (such as the `InterimTime` feature) only really pay off for well chosen thresholds. As mentioned earlier, the RF algorithm in the centralized setting was able to find and pick up those thresholds, while the RF-DP approach with all its random

[4] `https://scikit-learn.org/`
[5] `https://diffprivlib.readthedocs.io/en/latest/`
[6] `https://www.tensorflow.org/responsible_ai/privacy`

guessing of thresholds was not. As a result, in the federated setting, the LR-DP (based on objective perturbation [33]) and the MLP-DP (based on DP-SGD [31]) approaches took over in terms of better utility, and, as we observed, were most more stable across different runs.

One complication in using neural network based models such as LR or MLP is that they typically fare much better when the input features are normalized. As explained in Sec. 3.1, doing such normalization while providing DP required us to add noise to the feature values and having it count towards the privacy budget ϵ. To this end, we used $\epsilon_1 = 0.03, \epsilon_2 = 0.003$ in Eq. 3 and the remaining privacy budget was alloted for the model training. When using DP-SGD, δ was empirically set to $\frac{1}{|D|}$, where $|D|$ is the cardinality of the training data set.

Discretization of `InterimTime` into `bin_features`. We observed the `InterimTime` to be crucial for identifying anomalous transactions. Based on our observations in the centralized setting (the bottom row in Tab. 1) with no privacy, i.e. an infinite privacy budget, RF yields high AUPRC because the underlying decision tree learning algorithm has a built-in technique to find good thresholds for dynamic discretization of the `InterimTime` feature during tree construction. The LR and MLP training algorithms cannot detect such thresholds with the same ease. To mitigate this issue, in LR_{best}, we statically discretize the `InterimTime` feature into bins. We replace the `InterimTime` feature in each transaction by its corresponding bin number. The bin numbers in the training dataset are then one hot encoded. We refer to these one hot encoded features as `bin_features`.

To avoid any privacy leakage, we make the entire process of binning *differentially-private*. To do so, we first compute the DP mean of the `InterimTime` feature for benign transactions. The DP mean computations first clips the individual values to be summed based on clipping bounds publicly available based on the domain knowledge. The privacy of computation of mean follows from the Laplace mechanism, the clipping bounds, and the post-processing property of DP. A privacy budget of ϵ_3 is spent towards such computation. We then divide the feature set into two regions based on the above computed mean. Each region contains a distinct peak of benign samples. For each region, we compute the DP min and max value (percentile) of the `InterimTime` feature for the benign transactions and then generate 100 uniformly distributed bins for each percentile. The privacy of computation of percentile is due to [41] and a privacy budget of ϵ_4 is spent for computation of one percentile. [7] The total privacy sepnt for computation of DP statistics for binning process is $(\epsilon_3 + 2 \cdot \epsilon_4)$ which follows from sequential composition of DP.Once the bins are generated for both the regions, these are one hot encoded. Each value of the feature `InterimTime` is then mapped to the corresponding one hot encoded bin number that it falls into

As Tab. 3 shows, we obtained our best results on the competition leaderboard with LR_{best}-DP, trained with DP-SGD over `[bin_features, SameCurrency]`. The results in the last row of Tab. 3 are for $\epsilon = 5$, where a budget of $\epsilon_3 = 0.01, \epsilon_4 = 0.3$ was used for bin feature extraction and the remaining privacy budget was alloted for the training of LR_{best}-DP with DP-SGD.

5.2 Efficiency and scalability

We performed efficiency and scalability experiments on a desktop Intel i7 6700k at 4.2GHz, 64GB memory, and GTX1080 GPU. The results in Tab. 2 are based on the RF approach for the centralized setting, and on the MLP-DP trained with DP-SGD approach for the federated setting. In addition to model training, the reported runtimes also include the cost of normalizing the data (relevant for MLP-DP) and the cost of checking the validity of the accounts in the transactions against the bank data which, in the federated set-up is done in a privacy-preserving manner with the protocols from

[7] We use implementations available in `https://github.com/IBM/differential-privacy-library` to compute DP mean and DP min and max (using the percentile function) and we provide bounds for clipping that are independent of the data and depend on the given problem.

	Time			Memory		Communication	
	Total	PNS	node	PNS	node	PNS	node
scenario 1 PNS + 2 nodes	1596s	1198s	228s	3.50GB	1.95GB	1052B	1584B
scenario 2 PNS + 4 nodes	1581s	1173s	234s	3.92GB	2.01GB	1200B	3168B
scenario 3 PNS + 9 nodes	2701s	2215s	243s	4.36GB	1.85GB	2236B	7128B

Table 2: Efficiency and scalability results on development data.

	Entry	Method		C	N1	N2	N3
centralized	1	RF 8 features	OKVS 2 fields	0.8841			
	2	RF 4 features	OKVS 2 fields	0.9739			
	3	RF 4 features	OKVS 4 fields	0.9801			
federated	1	MLP with DP-SGD ($\epsilon = 5$) 4 features	OKVS 2 fields		0.8195	0.8235	0.8074
	2	LR with DP-SGD ($\epsilon = 5$) `bin_features`, `SameCurrency`	OKVS 4 fields		0.9494	0.9610	0.9477

- 8 features: `InstructedAmount`, `InterimTime`, `SettlementAmount`,`hour`,`sender_hour_freq`, `sender_currency_freq`,`sender_currency_amount_average`,`sender_receiver_freq`
- 4 features: `InstructedAmount`, `SameCurrency`, `InterimTime`, `difference_days_absolute`
- 2 fields: `Account`, `Name`
- 4 fields: `Account`, `Name`, `Street`, `CountryCityZIP`
- C: centralized; N1: federated with 2 nodes; N2: federated with 4 nodes; N3: federated with 9 nodes

Table 3: Leaderboard AUPRC results from official submissions

Sec. 3.2.

It can be seen in Tab. 2 that as the number of nodes increases, the runtime, memory usage, and communication costs of the PNS increase. This is expected since PNS's computation cost is largely dependent on the number of sender-receiver client pairs, which grows superlinearly with the amount of clients. However, the results also show that the total bank client runtime and memory resources remain fairly constant, and are mostly a function of the amount of banking data rather than number of client partitions. This can be explained simply by each clients' proportional computational load to its data size. The total clients' communication cost on the other hand scales superlinearly with the amount of clients, since, like in PNS, this computation depends more on the amount of sender-receiver pairs.

The results that we obtained on the public leaderboard are summarized in Tab. 3. Within each federated submission we see variation of the AUPRC results among the different scenarios (N1, N2, N3). This is because of the random noise that is added in our algorithms to provide DP guarantees. It is not caused by the partitioning.

6 Applicability of proposed approach beyond the PETs Prize

Our solution is customized to the financial crime detection track of the PETs Prize and the required use of the Flower framework. Our approach can be used and adapted to other applications in which there is an entity $\mathcal{S}$ who has a dataset with labeled training instances for an ML task, and

other entities $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$ who have data that can augment $\mathcal{S}$'s trained models. Our proposed solution is sufficiently generic to work with any kind of ML model that can be trained in a DP way [31, 32, 33, 34] so it can be used for many different use cases and ML tasks. $\mathcal{S}$ could for example be a display advertising company interested in click prediction. In this use case, $\mathcal{S}$ has a dataset with features about users and the advertisements that they clicked on, while $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$ are companies who each have a list with details about their customers. Leveraging knowledge from the companies would increase click and sales prediction. Our proposed solution could also for instance be used for detection of fraud in healthcare insurance claims. In this use case $\mathcal{S}$ is a company that processes healthcare insurance claims, and $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$ are healthcare providers each with their own data about what services were provided to patients. Our solution could also be used for surveillance applications in which an intelligence agency $\mathcal{S}$ would benefit from leveraging knowledge contained in lists of suspected terrorists known to other countries $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$. All these applications have a clear privacy narrative, calling for the kind of PETs that we proposed and used in our work here.

Acknowledgement: We would like to thank the organizers, the reviewers, and the judges for their tremendous efforts in making the PETs Prize Challenge a success. We recognize that organizing a competition of this kind and scale is a substantial endeavor, and value its effect as a catalyst for future research on the development of PETs.

References

[1] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, pages 1273–1282, 2017.

[2] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[3] Franziska Boenisch, Adam Dziedzic, Roei Schuster, Ali Shahin Shamsabadi, Ilia Shumailov, and Nicolas Papernot. When the curious abandon honesty: Federated learning is not private. *arXiv preprint arXiv:2112.02918*, 2021.

[4] Jinhyun So, Ramy E Ali, Basak Guler, Jiantao Jiao, and Salman Avestimehr. Securing secure aggregation: Mitigating multi-round privacy leakage in federated learning. *arXiv preprint arXiv:2106.03328*, 2021.

[5] Ahmed Roushdy Elkordy, Jiang Zhang, Yahya H Ezzeldin, Konstantinos Psounis, and Salman Avestimehr. How much privacy does federated learning with secure aggregation guarantee? *arXiv preprint arXiv:2208.02304*, 2022.

[6] Bargav Jayaraman, Lingxiao Wang, David Evans, and Quanquan Gu. Distributed learning without distress: privacy-preserving empirical risk minimization. In *Advances in Neural Information Processing Systems 31*, pages 6346–6357, 2018.

[7] Manas A Pathak, Shantanu Rane, and Bhiksha Raj. Multiparty differential privacy via aggregation of locally trained classifiers. In *Advances in Neural Information Processing Systems 23*, pages 1876–1884, 2010.

[8] Melissa Chase, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, and Peter Rindal. Private collaborative neural network learning. *Cryptology ePrint Archive*, 2017.

[9] David Byrd and Antigoni Polychroniadou. Differentially private secure multi-party computation for federated learning in financial applications. In *Proceedings of the First ACM International Conference on AI in Finance*, 2020.

[10] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, pages 1–11, 2019.

[11] Xiaolan Gu, Ming Li, and Li Xiong. Precad: Privacy-preserving and robust federated learning via crypto-aided differential privacy. *arXiv preprint arXiv:2110.11578*, 2021.

[12] Ronald Cramer, Ivan Damgard, and Jesper Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press Print, New York, 2015.

[13] Samuel Adams, Chaitali Choudhary, Martine De Cock, Rafael Dowsley, David Melanson, Anderson Nascimento, Davis Railsback, and Jianwei Shen. Privacy-preserving training of tree ensembles over continuous data. *Proceedings on Privacy Enhancing Technologies*, 2:205–226, 2022.

[14] Anisha Agarwal, Rafael Dowsley, Nicholas D. McKinney, Dongrui Wu, Chin-Teng Lin, Martine De Cock, and Anderson C. A. Nascimento. Protecting privacy of users in brain-computer interface applications. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 27(8):1546–1555, 2019.

[15] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *IEEE Symposium on Security and Privacy (SP)*, pages 19–38, 2017.

[16] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, (3):26–49, 2019.

[17] Martine De Cock, Rafael Dowsley, Anderson C. A. Nascimento, Davis Railsback, Jianwei Shen, and Ariel Todoki. High performance logistic regression for privacy-preserving genome analysis. *BMC Medical Genomics*, 14(23), 2021.

[18] Chuan Guo, Awni Hannun, Brian Knott, Laurens van der Maaten, Mark Tygert, and Ruiyu Zhu. Secure multiparty computations in floating-point arithmetic. *arXiv:2001.03192*, 2020.

[19] Marcel Keller and Ke Sun. Secure quantized training for deep learning. In *International Conference on Machine Learning*, pages 10912–10938, 2022.

[20] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, 2015.

[21] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction APIs. In *25th USENIX Security Symposium*, pages 601–618, 2016.

[22] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. Machine learning models that remember too much. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 587–601, 2017.

[23] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th USENIX Security Symposium*, pages 267–284, 2019.

[24] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[25] Sikha Pentyala, Davis Railsback, Ricardo Maia, Rafael Dowsley, David Melanson, Anderson Nascimento, and Martine De Cock. Training differentially private models with secure multiparty computation. Cryptology ePrint Archive, Report 2022/146, 2022. `https://ia.cr/2022/146`.

[26] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D. Lane. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[27] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In *Annual International Cryptology Conference*, pages 395–425. Springer, 2021.

[28] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions. *Journal of Cryptology*, 30(3):805–858, 2017.

[29] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In *9th International Conference on Theory and Practice of Public-Key Cryptography (PKC 2006)*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.

[30] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In *2013 ACM SIGSAC Conference on Computer and Communications Security*, pages 967–980, 2013.

[31] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318, 2016.

[32] Kamalika Chaudhuri and Claire Monteleoni. Privacy-preserving logistic regression. In *Advances in Neural Information Processing Systems*, pages 289–296, 2008.

[33] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12(3), 2011.

[34] Sam Fletcher and Md Zahidul Islam. Differentially private random decision forests using smooth sensitivity. *Expert Systems with Applications*, 78:16–31, 2017.

[35] Jelle Vos, Mauro Conti, and Zekeriya Erkin. Fast multi-party private set operations in the star topology from secure ANDs and ORs. *IACR Cryptol. ePrint Arch.*, page 721, 2022.

[36] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from PaXoS: Fast, malicious private set intersection. In *39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 739–767. Springer, 2020.

[37] Yann Collet. xxHash – Extremely fast non-cryptographic hash algorithm. https://cyan4973.github.io/xxHash/, 2021.

[38] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems 30*, pages 4765–4774. 2017.

[39] Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, 2(1):2522–5839, 2020.

[40] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. In Yehuda Lindell, editor, *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer International Publishing, 2017.

[41] Adam Smith. Privacy-preserving statistical estimation with optimal convergence rates. In *Proceedings of the 43th Annual ACM symposium on Theory of Computing*, pages 813–822, 2011.