STARLIT: Secure federaTed Analytics on veRticalLy partItioned daTa

Aydin Abadi[⋆1] Steven J. Murdoch[⋆⋆1] Mohammad Naseri[⋆⋆⋆1] George Theodorakopoulos[†2] Suzanne Weller[‡3] Sasi Murakonda[§3] Bradley Doyle[¶3]

[1] University College London
[2] University of Cardiff
[3] Privitar

1 Introduction

1.1 Problem Statement

SWIFT holds a dataset containing transactions between ordering and beneficiary bank accounts. Separately, a set of banks each holds a dataset of account information. Whilst SWIFT is able to train a machine learning model to detect anomalous transactions using their own data, this paper considers a federated setting in which the SWIFT dataset is augmented with account details from each bank for both the ordering and beneficiary account holders. The goal is to produce a model that is as good as, if not better, than one trained on SWIFT data alone. Further, no sensitive data should be shared between SWIFT and the individual banks, and the solution should be efficient enough to be used in practice.

This is a specific example of federated learning on vertically partitioned data in which each SWIFT transaction is associated with an ordering account at a given bank and a beneficiary account at another bank. Given that SWIFT does not hold information about exactly the same individuals as the two banks whose features must contribute to each transaction, this is doubly-asymmetric federated learning. Complexity is added by the fact that some features relevant to the model are the result of a combination of SWIFT and bank features.

[⋆] aydin.abadi@ucl.ac.uk
[⋆⋆] s.murdoch@ucl.ac.uk
[⋆⋆⋆] mohammad.naseri.19@ucl.ac.uk
[†] theodorakopoulosg@cardiff.ac.uk
[‡] suzanne.weller@privitar.com
[§] sasi.murakonda@privitar.com
[¶] bradley.doyle@privitar.com

1.2 System Design

In line with the principle of privacy by design, we ensure that the information shared between parties is minimal and appropriate for the task at hand. Our system augments the SWIFT dataset with two types of features using the banks' datasets:

1. **Discrepancy features**: computed based on the name and address information held by SWIFT and the equivalent information held by the bank of the ordering and beneficiary account respectively - just two binary values (one representing whether information about the ordering account matches and the other for the beneficiary account)
2. **Account flag features**: taken directly from the dataset of the ordering and beneficiary accounts respectively - just two binary values (one representing whether the flag value is zero or not for the ordering account and the other for the beneficiary account)

Banks sharing only binary values (about whether the information at account and transaction levels is anomalous or not) complies with the principle of **data minimisation**, as this data is adequate, necessary, and sufficient for achieving the objective at hand. To align with the principle of **integrity and confidentiality**, we compute the transaction-level discrepancy features using a *secure private set intersection* protocol and protect the flag values using *local differential privacy.*

Our system ensures that the data at the banks (such as names and addresses) is never disclosed in plain text form to SWIFT. And by design, banks only share minimal information that is necessary for improving anomaly detection models. Moreover, for a desired level of improvement in accuracy, our framework finds and applies the mechanism that provides the maximum protection to flag values at the banks.

Solution Architecture: Fig. 1 shows a high level overview of the architecture of our approach, which consists of a feature extraction phase followed by a training phase. During the feature extraction phase, certain features require cooperation between SWIFT and the banks, which is facilitated by a third party, the Flag Collector, used to aggregate features held within the banks' datasets. This drastically simplifies the training phase from n-party down to two-party vertical federated learning, which will allow the system to scale to large numbers of banks with little to no impact.

2 Threat Model and Defenses

The three types of clients we have in our system are: SWIFT, Bank Clients, and Flag Collector (a separate client or an independent entity being run at

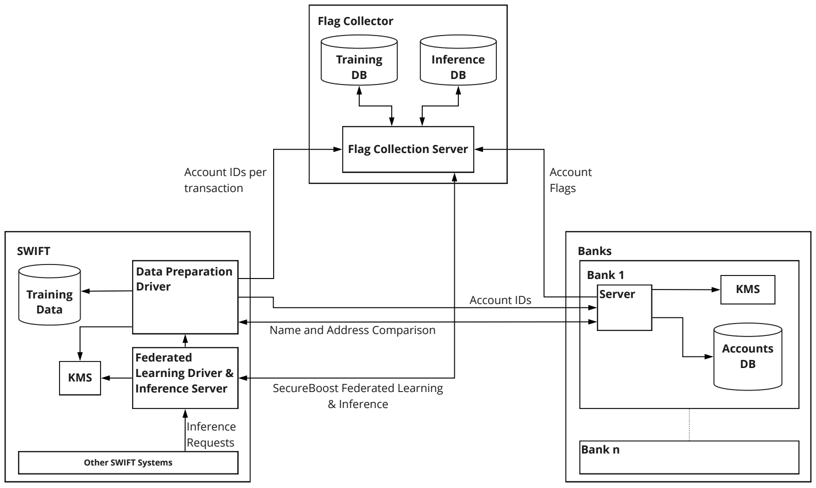


Fig. 1: High level architecture diagram showing the various components involved and the channels of communication between them. Our approach involves a feature extraction phase, where banks and SWIFT participate in a secure two-party protocol to enhance the feature set at SWIFT with information comparing the names and addresses of ordering and beneficiary accounts with data from banks. We also assume the existence of a third service, the Flag Collection Server, that collects the flag values corresponding to the accounts involved in a transaction. The Flag Collection Server and SWIFT engage in a two-party vertical federated learning protocol to train the anomaly detection model.

one of other two clients). We assume that all the participants are honest-but-curious parties and hence follow the protocols without any deviation. We consider it a privacy violation if the information of about any individual at one client is learned by another during the model training (including pre-processing) or deployment phase.

We specifically focus on ensuring that SWIFT cannot infer individual level information at the banks (the exact flag value) i.e., **protecting banks against attribute inference attacks by SWIFT**. We rely on a combination of techniques to thwart the possibility of inference attack in different scenarios.

As per our flag collection protocol (section 4.2), SWIFT only transmits account numbers and message IDs to Banks and only transmits message IDs to the Flag Collector. Banks only transmit flags that are paired with these account numbers to the Flag Collector. Neither the Banks nor the Flag Collector can make useful inferences from these values.

This still leaves the chance of an inference attack during model training/deployment. To address this issue and the scenario where Flag Collector cannot be assumed to be independent of SWIFT (e.g., due to the possibility of collusion), we use local differential privacy, where any flag values that leave the bank are obfuscated via a randomisation strategy, providing the strongest possible privacy guarantees.

Details on the mechanisms we used to achieve local differential privacy for the flag values are provided below. Note that this protection is an additional layer on top of what's already offered by the SecureBoost protocol, which only shares encrypted (aggregated) gradient information.

2.1 Local Differential Privacy

As proposed in Wang et al. [11], we consider two generalised mechanisms on binary attributes for achieving local differential privacy. The first one uses randomised response and the second one relies on adding Laplace noise with post-processing (applying a threshold of 0.5) for binarising the values.

Below, we list the corresponding transformation matrices for a given privacy level ϵ, where element at position (i, j) represents the probability of responding with value j if the true value is i. Note that although we list the matrices only for binary attributes here, both mechanisms generalise to the case of categorical variables with more than two values.

Randomised Response:

$$\begin{pmatrix} \frac{e^\epsilon}{1+e^\epsilon} & \frac{1}{1+e^\epsilon} \\ \frac{1}{1+e^\epsilon} & \frac{e^\epsilon}{1+e^\epsilon} \end{pmatrix} \tag{1}$$

Laplace noise with post-processing:

$$\begin{pmatrix} 1-\frac{1}{2}e^{\frac{-\epsilon}{2}} & \frac{1}{2}e^{\frac{-\epsilon}{2}} \\ \frac{1}{2}e^{\frac{-\epsilon}{2}} & 1-\frac{1}{2}e^{\frac{-\epsilon}{2}} \end{pmatrix} \tag{2}$$

2.2 Mechanisms for Optimal Inference Privacy

Any randomisation mechanism for the obfuscating the flags while sharing can offer some protection against inference attacks by SWIFT. Given a value of epsilon in LDP, there can be many mechanisms that satisfy the constraint of differential privacy, of which two can be found using equation 1 (for RR) and equation 2(for Laplace). These mechanisms always assign an equal probability of converting a 0 to 1 and 1 to 0. Moreover, they need not be the optimal transformation matrices that provide maximum inference privacy (i.e., maximum protection against SWIFT's ability to infer the flag values). As a key contribution in this work, **we developed a framework to explore the entire privacy mechanism space and find optimal mechanisms that maximise inference privacy, under the given constraints on utility and local differential privacy**. The main advantage of formulating the construction of privacy mechanism as an optimization problem is that we can automatically explore a large solution space to discover optimal mechanisms that are not expressible in closed form (such as the Laplace or Gaussian mechanism), so human intuition would not be able to find it. More details about the game construction and solution are provided in Section 6.

3 Empirical Results

We evaluated our solution (SecureBoost with Local Differential Privacy) using the development data set provided. Below we present the empirical privacy-utility trade-off that can be achieved with our solution and compare it with a baseline centralised solution without any privacy protections. We also discuss the efficiency and scalability aspects of our solution in the subsequent section.

Features used: In this work, we *do not* focus on feature exploration or hyper-parameter tuning for improving the accuracy of model. We just use the example features extracted from SWIFT (in the notebook provided by the organisers) along with four binary values computed using the bank data.

The features extracted from SWIFT for training the model are: [Settlement Amount, Instructed Amount, Hour, Sender hour frequency, Sender currency frequency, Sender currency amount average, Sender receiver frequency]. The four binary flags that we add represent whether SWIFT and the banks agree on the sender and receivers address details, and whether or not the sending and receiving accounts have a flag of 0.

3.1 Privacy-Utility Trade-off

Baseline:

Our benchmark for analysing the trade-off between utility and privacy is a centralised model, constructed at SWIFT, where all the data from banks is revealed without any privacy protection. We extract the same features listed above and train a standard XGBoost model with 30 trees and 5-fold cross-validation with average precision score as the metric. We use default values for all the hyper-parameters when training the model.

Evaluation procedure:

For analysing the AUPRC that can be achieved at a given level of privacy, we modify the flag values that banks send using an appropriate privacy mechanism and construct XGBoost models with these noisy features. We use the same parameters as in the baseline model (30 trees and 5-fold cross-validation) and measure the average precision score for the final model on training and test data (averaged over 5 runs to account for the randomness of the privacy mechanism and the training process).

We chose XGBoost for the baseline and also in our experiments to understand the privacy-utility trade-off as it can be theoretically proved that it will achieve the same accuracy as our federated SecureBoost solution, given the same set of parameters and input data. SecureBoost does the same computation as XGBoost while constructing the trees albiet on encrypted gradients. Hence, the additional cost will not be on accuracy but rather on performance (which we discuss in section 3.2). We also observed this to be the case from our experimental results.

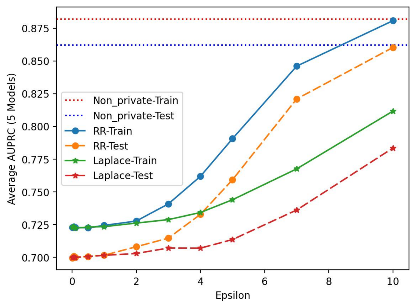
It is worth noting that in an alternative configuration of our solution, since the flag values are protected with local differential privacy, the banks could even agree to send all the features directly to SWIFT instead of the flag collector, which implies SWIFT could directly train the model using standard XGBoost without the need for implementing SecureBoost.

Evaluation procedure:

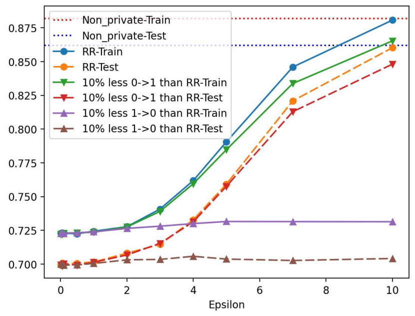
For analysing the AUPRC that can be achieved at a given level of privacy, we modify the flag values that banks send using an appropriate privacy mechanism and construct XGBoost models with these noisy features. We measure the average precision score for the final model on training and test data (averaged over 5 runs to account for the randomness of the privacy mechanism and the training process).

Key takeaways:

Figure 2 provides a summary of our utility-privacy trade-off analysis. As expected, randomised response mechanism offers better utility-privacy compared with Laplace mechanism. From the sample asymmetric transformations explored here, we can clearly see that the cost of converting a zero to one is much higher



(a) Randomised Response vs Laplace



(b) Randomised Response vs Asymmetric matrices

Fig. 2: We protect flag values with local differential privacy (LDP). Plot(a) compares the effect on AUPRC of the model when using Randomised Response (RR) and Laplace mechanism with post-processing for achieving LDP. As expected from the optimality results in [11] and [5], RR provides a better utility-privacy trade-off compared to the Laplace mechanism. Both RR and Laplace mechanisms result in symmetric transformation matrices i.e., equal probability for converting a 0 to 1 and a 1 to 0. Plot(b) compares the effect on AUPRC when using RR and privacy mechanisms at the same value of epsilon but with the constraint of 10% less probability of converting 0 to 1 (1 to 0) than what's recommended by RR (can be found using our game framework). The results clearly show that increasing the probability of converting a zero flag to non-zero even slightly has a massive effect on the performance of the model, which is intuitive given the large fraction of zero flag values.

than that of converting a one to zero. Although randomised response offers better utility than the other privacy mechanisms considered here for the problem at hand, our game framework can find better solutions, if they exist, by helping explore the entire privacy mechanism space.

3.2 Efficiency and Scalability

Experiments were run using AWS ECS cloud with the docker environment provided for the competition. The docker containers are provisioned with 56GB RAM and 8 vcpu. The FATE secureboost implementation uses multiprocessing to operate on table-like objects. We set the partitions setting to 5, which means operations on tables are performed with a parallelism of 5. We ran the scenario on the provided dev dataset with Swift and one bank.

Baseline

Our solution can be decomposed in two main steps: feature creation and training (or prediction). The following charts show the runtime characteristics of our solution. The secureboost training was configured with 10 trees, tree depth of 3, a sampling rate of the dataset of 40% and goss sampling of 0.1. Efficiency results for this baseline are given in Table 1. The baseline was used to explore the impact of different configurations on efficiency.

Table 1: Efficiency metrics for baseline performance investigation

AURPC	0.4715
total_training_time_full-data-federated (h)	1.1
peak_training_memory_gb_full-data-federated	12.38
network_disk_volume_full-data-federated (G)	4.98
network_file_volume_full-data-federated	993

Although our solution uses a large number of flower rounds, many of the last rounds are just empty rounds. This is required because the parties SWIFT and Flag Collector decide together when training and prediction are finished but the Flower framework imposes us to pre-set a number of rounds. Similarly, messages need to go through the Flower server when instead a peer to peer communication protocol would have been more efficient. This has an impact on the network metrics. The cpu utilization of our solution can be seen in Fig. 3.

The SecureBoost training is very CPU intensive, this is due to the encryption required by the algorithm to encrypt the gradient. In the chart we only used 5 CPUs out of 8 to mimic the setup in the submission. No CPU specific library was used to run Paillier encryption. The timing it takes to train or predict can

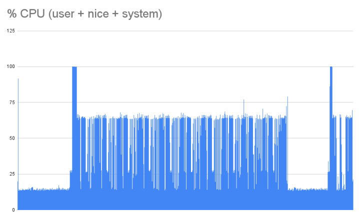


Fig. 3: CPU utilisation for baseline run

be reduced by increasing parallelism or using hardware specific libraries like the Intel Paillier Cryptosystem Library.

Efficiency Results

We analysed the efficiency of the solution with different SecureBoost configurations. The results are shown in Table reftable::scalability-table. Being a well established protocol and model, SecureBoost provides a lot of options that can be used for different settings. Sampling and goss sampling provide a way to cut down the network and memory overhead by reducing the amount of data processed in any round of training. Tree depth is an effective parameter to improve accuracy while keeping a good efficiency in terms of training time and memory consumption. Our FATE/Flower integration allows to split large messages into chunks.

Table 2: The efficiency of the overall solution using various training parameters

Sample	40%	100%	40%	40%	40%	40%
goss top_rate	0.1	0.1	disabled	0.3	0.1	0.1
AURPC	0.4715	0.5786	0.47	0.5965	0.4715	0.6520
total_training_time_full-data-federated (h)	1.1	2.21	2.83	1.5	1	1.13
peak_training_memory_gb_full-data-federated	12.38	17.48	18.39	13.66	12.22	16.40
network_disk_volume_full-data-federated (G)	4.98	14.51	16.61	7.84	4.34	5.10
network_file_volume_full-data-federated	993	1270	1256	1035	927	1316
max message size	100MB	100MB	100MB	100MB	1GB	100MB
tree depth	3	3	3	3	3	5

Scalability

The model training phase remains independent of the number of bank clients, as the training happens only between SWIFT and the flag collector party.

The feature extraction scales with the number of bank accounts involved rather than the number of banks, aside from a small overhead due to additional Flower communication. Table 3 demonstrates our experimental results confirming this.

Table 3: The runtime of feature extraction depending of the number of bank clients

number of bank clients	2	5	9
feature extraction runtime (seconds)	833	808	821

3.3 Submission parameter selection

Our final selection of parameters was influenced by the privacy-utility analysis in section 3.1 (in selecting epsilon) and the efficiency analysis in section 3.2 in choosing the model hyper parameters such as number of trees, sampling rate, etc).

For the centralised solution, as far as possible, we matched the parameters used in the centralised XGBoost model with those used in the federated solution. Specifically for number of trees, tree depth and the L2 regularisation parameter.

The parameters selected for the submission were:

Centralised: number_of_trees=10, tree_depth=5, L2_regularisation=0.1

Federated: number_of_trees=10, tree_depth=5, L2_regularisation=0.1, epsilon=10

4 Technical Details - Feature Extraction

4.1 Comparing names and addresses

For each transaction, SWIFT holds six string columns: *Name* plus two address columns, *CityCountryZip* and *Street* for both the Ordering account and the Beneficiary account. Each bank also holds corresponding *Name*, *CityCountryZip*, and *Street* information per account.

Exploration of the provided synthetic dataset indicated that discrepancies in these columns between SWIFT and the banks could provide information about anomalous transactions. Including discrepancy features is therefore likely to improve model accuracy. To check that the information (e.g., Name, Street, CityCountryZip) that the banks hold is the same as the information SWIFT holds while preserving bank's and SWIFT's privacy, we use a privacy preserving cryptographic protocol called Private Set Intersection (PSI).

The result of the PSI (matches between different string columns of an account) is fed to the federated machine learning model as an additional feature.

Private Set Intersection (PSI) In this work, initially we focused on and implemented the RSA-based PSI proposed in [1], due to its simplicity (that would lead to easier security analysis). However, after conducting concrete run-time analysis we noticed that this protocol takes a long time to terminate.

Therefore, we adjusted and used the python-based implemented of the PSI in [7] which yields much lower run-time than the one in [1]. The new protocol mainly relies on efficient symmetric key primitives, such as Oblivious Pseudorandom Functions and Cuckoo hashing. The security of this PSI has been proven in a standard simulation-based (semi-honest) model and the related paper has been published at a top-tier conference. The efficiency of the PSI protocol mainly stems from the efficient "Oblivious Pseudorandom Function" (OPRF) that Kolesnikov *et al.* constructed which itself relies on the Oblivious Transfer (OT) extension proposed earlier in [6].

Our experiments were done with each party having 2^n items and compared the run-time of the PSI in [1] with the one in [7]. We concluded that the PSI in [7] is around 10-11 times faster than the one in [1]. In the (UK-US PETs Prize) competition provided data-set, there are approximately 1M unique entries, and therefore we estimate that our chosen PSI protocol will take around 12 minutes to run (before added overhead introduced by the flower framework). An actual run took 13 minutes which included loading the data. If we had stayed with the RSA-based PSI, we estimate it would have taken at least 3 hours. Table 4 summarises the run-time comparison between the two PSIs.[4] Note that any efficient (traditional/delegated) PSI can be used in our proposed analytic framework.

Table 4: The run-time comparison between the RSA-based PSI in [1] and our choice of PSI proposed in [7] (in sec.).

Protocol	Set Cardinality										
	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}
[1]	4.10	9.32	16.56	32.78	65.45	132.97	252.56	524.32	1059.49	-	-
[7]	0.84	1.18	1.84	3.23	6.06	11.63	23.75	45.80	99.09	183.79	367.93

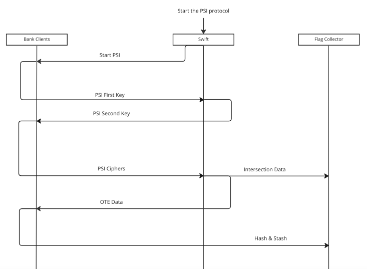
The actual implementation in Flower is sketched next.

We use the Flower adaptor to communicate between the PSI clients (at SWIFT) and PSI servers (at the bank clients), See Figure 4a. The PSI result is found by comparing encrypted data generated by the PSI clients and servers. We send a dataframe of encrypted data from SWIFT to the flag collector, the banks also send their encrypted data to the flag collector. The flag collector does not

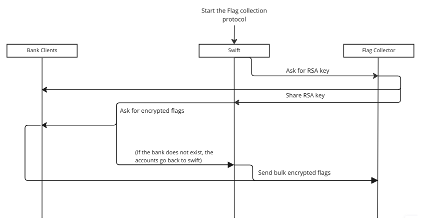
[4] The experiments were conducted on a laptop with an 8 core, 2.4Ghz i9 CPU and 64GB of memory. We did not take advantage of parallelisation.

receive the encryption key (or even the encryption algorithm), it only checks if an (encrypted) value in a dataframe also appears in the (encrypted) data from the banks. This means that the flag collector has no way of learning what the unencrypted data is, and the banks and SWIFT do not even receive the result of the PSI protocol.

PSI checking is limited to a single check per account by concatenating Bank + Account + Name + Street + CountryCityZip to a single comparison. This increases privacy and efficiency, with a possible utility cost. (Although no cost to accuracy was observed on the provided synthetic dataset.) Note that the reason we include the bank name in the protocol is down to how the datasets are partitioned. If there was only one bank at each bank client, there would be no need for this step.



(a) between SWIFT, the bank clients and the flag collector for the PSI protocol


Start the Flag collection protocol
Bank Clients
Swift
Flag Collector
Ask for RSA key
Share RSA key
Ask for encrypted flags
(If the bank does not exist, the accounts go back to swift)
Send bulk encrypted flags

(b) between SWIFT, the bank clients and the flag collector for the flag collection protocol

Fig. 4: Order of communication for feature extraction

4.2 Flag collector

For each account the bank holds a flag providing extra information that can help detect anomalies. However the flags are private information, so cannot be shared directly with SWIFT. We propose the following method to align the flags with the SWIFT dataset without sharing them with SWIFT.

SWIFT sends message IDs and associated accounts to the relevant banks, as well as the information if the account is sending or receiving in each transaction. The banks then use their account information to pair each message ID with the correct flag and send this data to the flag collector. Finally, the flag collector uses the data from the bank to create the dataframe of flags, F. This dataframe can then be used as the input data for the ML model. If we were in the centralized setting, we would be able to join this matrix to the database SWIFT has along message ID.

This is not done in the clear, the flag collector client generates an RSA key pair, and sends the public key to each bank client (and SWIFT). The bank clients then create a symmetric AES key, and encrypt this key with the public RSA key. They then encrypt their dataframe of flags with the symmetric key and send the encrypted dataframe and encrypted symmetric key to the flag collector. The flag collector can then decrypt the symmetric key and then the dataframes of flags.

Our implementation also treats each bank client as a single bank (even if composed of multiple banks from the data). We therefore additionally send across the bank that the account is held at. If each bank client only held one bank, this would not be required. For the sake of the privacy analysis, we have assumed that each Flower bank client only holds the data of one bank. If the bank does not exist at that client, the accounts are sent back to SWIFT which encrypts them with a new flag, that represents "bank not found". Similarly, the bank clients add an "account not found" flag if required. See Figure 4b for the communications used for the flag collection protocol.

Privacy when collecting flags The following information is learnt by each party after this process has been completed.

– The banks learn which of their accounts are in use and how many transactions each account is a part of - we assume that they already know this.
– The banks may receive rows for accounts that it does not manage, but SWIFT erroneously thinks they do.
– The flag collector learns which message IDs come from each bank. It can use this information to calculate how many transactions are between each pair of banks. Finally it also knows the values of the flags coming from each bank.

Importantly, the banks do not learn anything about other banks, or accounts held at other banks.
The flag collector has no way to find the accounts participating in a transaction, it only knows the message ID of the transaction and the two associated flags. From this, the flag collector can not learn anything about the account x_i.

4.3 Model Training and Inference

After the feature extraction phase, we have two parties (SWIFT and the Flag collector) that together hold all the data required for training the anomaly detection model. SWIFT has the features it could extract from its own data, while Flag collector has the features that compare names and addresses of ordering and beneficiary accounts with data from banks and the (privacy-protected) flag values associated with these accounts. This is the classic vertical federated learning setup with only one party (SWIFT) holding the labels to predict, which opens up multiple off-the-shelf protocols that can be used for training an ML model [3,4,?,?,?,?,?,?,?,?,?].

We use the SecureBoost algorithm in [4], which involves exchange of encrypted (aggregate) gradients and hessians between SWIFT and the Flag collector during the training phase. SWIFT can decrypt the gradients and hessians for determining the best feature to split on. Once the model is trained, each party owns the part of the tree that uses the features they hold. Hence, when using the distributed inference protocol in [4], SWIFT coordinates with the Flag collector to determine the split condition to be used. It might be possible for SWIFT to infer the flag value based on the final prediction, which points towards an interesting and **potentially inevitable trade-off between model interpretability and privacy**. SWIFT can either clearly explain the impact of the flag values on it's prediction (required for interpretability but offers no protection to flag values) or not learn the exact effects of the flag value on final prediction (good for privacy but bad for interpretability).

Note on improvements proposed in phase-I:

In our whitepaper submission during phase-I, we identified that it might be possible for SWIFT to infer the flag values for certain accounts during the training phase (by observing the split decisions and gradients) or the during the inference phase (by observing the label). We proposed a number of ideas that we wanted to explore to reduce this information leakage. Although we have implemented some of the techniques, they are ad-hoc approaches that do not provide any theoretical guarantees. So, we switched directions towards local differential privacy and developed a whole new game-theoretic framework for protecting the flag values. This approach is substantially more rigorous than the techniques we proposed in phase-I and has a broader applicability beyond this problem.

5 Flag protection - Game

Notation	Definition
$\mathcal{V}$	INPUT: Set of flag values EXAMPLE: $\mathcal{V} = \{0, 1, 2, .., 10\}$
$\pi(v), v \in \mathcal{V}$	INPUT: Prior probability of value v (SWIFT's prior knowledge) EXAMPLE: $\pi(0) = 0.9, \pi(1) = \pi(2) = \cdots = \pi(10) = 0.01$.
$d_p(\hat{v}, v) : \mathcal{V} \times \mathcal{V} \rightarrow \mathcal{R}$	INPUT: privacy metric (attacker's error when estimating v as $\hat{v}$) EXAMPLE: If $\hat{v} = v, d_p(\hat{v}, v) = 0$. If $\hat{v} \neq v, d_p(\hat{v}, v) = 1$
$f(v'\|v) : \mathcal{V} \times \mathcal{V} \rightarrow [0, 1]$	OUTPUT: privacy mechanism

The problem of finding a Privacy Mechanism (PM) that offers optimal flag privacy to the Bank given the knowledge of the adversary (SWIFT), is an instance of a Bayesian Stackelberg game. In a Stackelberg game the leader, in our case the Bank, plays first by choosing a PM (a transformation matrix), and commits to that by running it on the actual values of the flags; and the follower, in our case SWIFT, plays next estimating the flag value, knowing the PM the Bank has committed to. It is a Bayesian game because SWIFT has incomplete information about the true flag values and plays according to its prior information about these values. Inspired by similar work in location privacy protection games [9,8], we now proceed to define the game for a single flag value, but the transformation matrix computed will be used for each value:

Step 0 Nature selects a flag value $v \in \mathcal{V}$ for the Bank according to a probability distribution $\pi(.)$, the *flag profile*. That is, flag value v is selected with probability $\pi(v)$. This encodes the relative proportions of the flag values in the dataset.

Step 1 Given v, the Bank runs the PM $f(v'|v)$ to select a replacement value $v' \in \mathcal{V}$.

Step 2 Having observed v', SWIFT selects an estimated flag value $\hat{v} \sim g(\hat{v}|v'), \hat{v} \in \mathcal{V}$. SWIFT knows the probability distribution $f(v'|v)$ used by the PM, and the Bank's flag profile $\pi(.)$, but not the true flag value v.

Final Step The game outcome is the number $d_p(\hat{v}, v)$, which is the Bank's privacy for this iteration of the game. This number represents SWIFT's error in estimating the true value of the flag.

The above description is common knowledge to both SWIFT and the Bank. SWIFT tries to minimize the expected game outcome via its choice of g, while the Bank tries to maximize it via its choice of transformation matrix f.

As changing the flag values distorts the data for training the ML algorithm, we impose upper bounds $p^{\max}(v', v)$ on the probabilities $f(v'|v)$.

Finally, and independently of the above considerations, we want the PM to be ϵ-differentially private.

6 Flag protection - Optimisation problem

We now explain how to construct a concrete optimization problem that encodes the above description and that we can solve to obtain the optimal PM $f()$, given $\pi(), d_p, p^{\max}(v', v)$, and ϵ.

SWIFT knows the function $f(v'|v)$ implemented by the PM, and thus can form a posterior distribution $\Pr(v|v')$ on the true flag value, conditional on the observation v'. Then, SWIFT chooses $\hat{v}$ to minimize the conditional expected privacy, where the expectation is taken under the posterior distribution:

$$\min_{\hat{v}} \sum_{v} \Pr(v|v') d_p(\hat{v}, v). \tag{3}$$

If there are multiple minimizing values of $\hat{v}$, then SWIFT may randomize among them. This randomization is expressed through $g(\hat{v}|v')$, and in this case (3) would be rewritten as

$$\sum_{v,\hat{v}} \Pr(v|v')g(\hat{v}|v')d_p(\hat{v}, v), \tag{4}$$

but it should be noted that the value of this equation would be the same as the value of (3) for any minimizing value of $\hat{v}$.

As $\pi(v)$ and $f(v|v')$ are known to SWIFT, so is

$$\Pr(v|v') = \frac{\Pr(v, v')}{\Pr(v')} = \frac{f(v'|v)\pi(v)}{\sum_v f(v'|v)\pi(v)}. \tag{5}$$

So, for a given v', the Bank's conditional privacy is given by (3). The probability that v' is reported is $\Pr(v')$. Hence, the unconditional expected privacy of the Bank is

$$\sum_{v'} \Pr(v') \min_{\hat{v}} \sum_{v} \Pr(v|v')d_p(\hat{v}, v) = \sum_{v'} \min_{\hat{v}} \sum_{v} \pi(v)f(v'|v)d_p(\hat{v}, v) \tag{6}$$

To facilitate computations, we define

$$x_{v'} \triangleq \min_{\hat{v}} \sum_{v} \pi(v)f(v'|v)d_p(\hat{v}, v). \tag{7}$$

Incorporating $x_{v'}$ into (6), the unconditional expected privacy of the Bank can be rewritten as

$$\sum_{v'} x_{v'}, \tag{8}$$

which the Bank aims to maximize by choosing $f(v'|v)$. The minimum operator makes the problem non-linear, which is undesirable, but (7) can be transformed to a series of linear constraints:

$$x_{v'} \leq \sum_{v} \pi(v)f(v'|v)d_p(\hat{v}, v), \forall \hat{v}. \tag{9}$$

Indeed, maximizing (8) under (7) is equivalent to maximizing (8) under (9). For every v', there must be some $\hat{v}$ for which (9) holds as a strict equality: Otherwise, we could increase one of the $x_{v'}$, so the value of (8) would increase.

From (8) and (9), the linear program for the Bank is constructed:

Choose $f(v'|v), x_{v'}, \forall v, v'$ to solve the following linear programming problem.

$$\textbf{Maximize} \sum_{v'} x_{v'} \tag{10}$$

subject to

$$x_{v'} - \sum_{v} \pi(v) f(v'|v) d_p(\hat{v}, v) \leq 0, \forall \hat{v}, v' \tag{11}$$

$$f(v'|v) \leq p^{\max}(v', v), \forall v, v' \tag{12}$$

$$\sum_{v'} f(v'|v) = 1, \forall v \tag{13}$$

$$f(v'|v) \geq 0, \forall v, v' \tag{14}$$

$$\frac{f(v'|v_1)}{f(v'|v_2)} \leq \exp(\epsilon), \forall v', v_1, v_2 \tag{15}$$

Constraints (13) and (14) reflect that $f(v'|v)$ is a probability distribution function for each v. Constraint (15) is the constraint that enforces differential privacy.

6.1 Alternative quality-privacy tradeoffs

The above formulation encodes the privacy-accuracy tradeoff in one particular way – maximize inference privacy, subject to a differential privacy constraint and an accuracy-related constraint on the probabilities $f(v'|v)$. The general framework is flexible enough to accommodate other tradeoffs.

For example, instead of introducing constraints $p^{\max}(v', v)$ on $f(v'|v)$, we can introduce an Accuracy Loss (AL) matrix with entries $AL_{v'v}$ that quantify the loss in accuracy when replacing value v with v'. Then, instead of (12), we can upper bound the total expected accuracy loss that is caused by a given transformation matrix f with the following constraint:

$$AL(f) := \sum_{v} \pi(v) \sum_{v'} f(v'|v) AL_{v'v} \leq AL^{\max}.$$

Alternatively, in a more radical departure from the original formulation, rather than aiming to maximize the Bank's privacy (inference privacy) subject to AL constraints, we could instead aim to minimize the accuracy loss $AL(f)$ subject to a lower bound on inference privacy, i.e. $\sum_{v'} x_{v'} \geq PR^{\min}$.

In general, the main benefit of formulating the construction of the transformation matrix as an optimization problem is that we can automatically explore a large solution space to discover optimal probability distributions $f(v'|v)$ that are not expressible in closed form (such as the Laplace or Gaussian mechanism), so human intuition would not be able to find it.

7 Solution Development

7.1 Integration With Flower

The Flower framework quite heavily clashed with our proposed architecture in which there was no central “server” party controlling the process, instead clients would directly communicate with one another to send required information and the Swift party would also act as a coordinator responsible for initiating the feature extraction and learning phases. In Flower, the central server runs a number of training rounds, each of which consists of a series of steps involving sending instructions to, and receiving results from, the clients in the system.

This process makes it difficult to do any pre-processing of data before entering into the iterative training process, which was needed for us to complete our feature extraction phase. It also requires the server's strategy to know exactly what every client should be doing on each round of communication.

To work around these issues, we created a server strategy implementation which simply acts as a message router for the clients in the system. This design allowed us to maintain our initial design of Swift acting as a coordinator party and allowing the server Strategy to remain agnostic of what needed to happen on each round of messages which hugely simplified the server logic. Additionally this change greatly **improves the extensibility of the prototype and its adaptability to further use cases**.

7.2 Integration with FATE

Our solution uses the SecureBoost model implemented in the FATE (Federated AI Technology Enabler) open source framework. The protocol is conducted jointly over two parties that can progress in parallel and exchange secure messages between each other to progress in the work. The protocol is inherently peer to peer, and in our setting it involves SWIFT and the Flag Collector.

Significant effort has gone in re-adjusting the protocol to work with the Flower API under which the parties can't communicate with each other directly. We solved this problem by routing the messages through the Flower server.

Moreover, the flower clients are stateless. In order to integrate with FATE without needing to snapshot the who system for every Flower round, we wrote a new implementation of the FATE federation API that uses python multiprocessing Queues to send and receive messages. With this component we are able to spawn longer lived standalone FATE processes for training and prediction. Each Flower client simply proxies messages to the respective FATE process via python multiprocessing Queues as shown in Fig. 5. The end result each party can communicate to the other party (e.g. Flag Collector) by routing the messages through the Flower server as required.

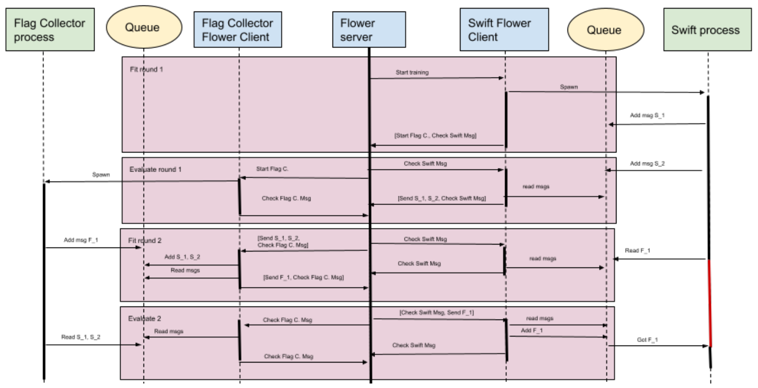


Fig. 5: In order to integrate the FATE federated AI framework with the Flower framework used in the competition, our solution runs a separate FATE process for each of the parties involved in the vertically partitioned learning. The Flower Client communicates with the FATE process using python multiprocessing queues. All messages between the parties are routed via Flower.

8 Adaptability and other use cases

Our work is highly adaptable to many use cases that involve a wide range of collaborative analysis tasks such as cross-sector partnerships between crime agencies and human rights NGOs enabling efficient collective intelligence without the need for direct data sharing [10]. Another example is analysis across healthcare data joined with local authority data to enable research into and optimisation of complex hospital discharges and community care [2].

Through our solution, we have demonstrated privacy preserving creation of features that are a function of data held by multiple parties (e.g. the string matching features on account names and addresses held by SWIFT and the banks). This flexible feature creation is an essential building block of any federated learning problem.

We have also developed a general purpose framework which can be used to select optimal obfuscation mechanisms for protecting flags (any categorical variables) that maximise inference privacy under some stated constraints on utility and local differential privacy. The framework is flexible in how the utility constraint is stated and can even be modified to maximise utility under any given constraints on inference privacy and guarantees if local differential privacy.

Our modular design and integration between the FATE and Flower federated frameworks enables not just SecureBoost but a broad range of FATE functions to be executed via Flower. Finally, the flower adapter architecture we built contributes towards extending the use of Flower to vertically partitioned use cases.

References

1. Agrawal, R., Evfimievski, A., Srikant, R.: Information sharing across private databases. In: Proceedings of the 2003 ACM SIGMOD international conference on Management of data. pp. 86–97 (2003)
2. Boniface, M., Carmichael, L., Hall, W., Pickering, B., Stalla-Bourdillon, S., Taylor, S.: A blueprint for a social data foundation: Accelerating trustworthy and collaborative data sharing for health and social care transformation (2020), `https://southampton.ac.uk/~assets/doc/wsi/WSI%20white%20paper%204%20social%20data%20foundations.pdf`
3. Ceballos, I., Sharma, V., Mugica, E., Singh, A., Roman, A., Vepakomma, P., Raskar, R.: Splitnn-driven vertical partitioning. arXiv preprint arXiv:2008.04137 (2020)
4. Cheng, K., Fan, T., Jin, Y., Liu, Y., Chen, T., Papadopoulos, D., Yang, Q.: Secureboost: A lossless federated learning framework. IEEE Intelligent Systems 36(6), 87–98 (2021)
5. Kairouz, P., Oh, S., Viswanath, P.: Extremal mechanisms for local differential privacy. Advances in neural information processing systems 27 (2014)
6. Kolesnikov, V., Kumaresan, R.: Improved OT extension for transferring short secrets. In: Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II. Lecture Notes in Computer Science, vol. 8043, pp. 54–70. Springer (2013)
7. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: CCS (2016)
8. Shokri, R.: Privacy games: Optimal user-centric data obfuscation. arXiv preprint arXiv:1402.3426 (2014)
9. Shokri, R., Theodorakopoulos, G., Troncoso, C., Hubaux, J.P., Le Boudec, J.Y.: Protecting location privacy: optimal strategy against localization attacks. In: Proceedings of the 2012 ACM conference on Computer and communications security. pp. 617–627 (2012)
10. The Royal Society: From privacy to partnership: The role of privacy enhancing technologies in data governance and collaborative analysis, `https://royalsociety.org/-/media/policy/projects/privacy-enhancing-technologies/From-Privacy-to-Partnership.pdf?la=en-GB&hash=4769FEB5C984089FAB52FE7E22F379D6`
11. Wang, Y., Wu, X., Hu, D.: Using randomized response for differential privacy preserving data collection. In: EDBT/ICDT Workshops. vol. 1558, pp. 0090–6778 (2016)