SCOTCH: An Efficient Secure Computation Framework for Secure Aggregation

Yash More[1], **Prashanthi Ramachandran**[2], **Priyam Panda**[1], **Arup Mondal** [1], **Harpreet Virk**[1], **Debayan Gupta**[1]

[1] Ashoka University
[2] Brown University

yash.more@alumni.ashoka.edu.in, pramach3@cs.brown.edu, priyam.panda_asp22@ashoka.edu.in, arup.mondal_phd19@ashoka.edu.in, harpreet.virk@alumni.ashoka.edu.in, debayan.gupta@ashoka.edu.in

Abstract

Federated learning enables multiple data owners to jointly train a machine learning model without revealing their private datasets. However, a malicious aggregation server might use the model parameters to derive sensitive information about the training dataset used. To address such leakage, differential privacy and cryptographic techniques have been investigated in prior work, but these often result in large communication overheads or impact model performance. To mitigate this centralization of power, we propose SCOTCH, a decentralized *m-party* secure-computation framework for federated aggregation that deploys MPC primitives, such as *secret sharing*. Our protocol is simple, efficient, and provides strict privacy guarantees against curious aggregators or colluding data-owners with minimal communication overheads compared to other existing *state-of-the-art* privacy-preserving federated learning frameworks. We evaluate our framework by performing extensive experiments on multiple datasets with promising results. SCOTCH can train the standard MLP NN with the training dataset split amongst 3 participating users and 3 aggregating servers with 96.57% accuracy on MNIST, and 98.40% accuracy on the Extended MNIST (digits) dataset, while providing various optimizations.

Introduction

Standard machine learning environments often rely on large amounts of sensitive data to achieve a high level of performance (Halevy, Norvig, and Pereira 2009). However, preparing a central repository of data is laborious, making secure-collaborative training expensive. Outsourcing the data to a central server that performs model training for the users is a potential solution, but is often not feasible in privacy-sensitive settings. Secure aggregation of data using multiparty computation frameworks (MPC) (Yao 1982; Mood et al. 2016; Perry et al. 2014; Di Crescenzo et al. 2014; Gupta et al. 2016; Lindell 2020; Goldreich 1998; Goldreich, Micali, and Wigderson 2019) has been explored in recent works, but they significantly impact framework efficiency due to added computational overheads (Phong et al. 2018). Moreover, centralized aggregation creates a single point of failure in the framework that can potentially compromise the security and privacy of the training data if the server is malicious, or prone to adversarial attacks by colluding participants (Chen et al. 2021; Kairouz et al. 2021).

A recently proposed alternative for privacy-preserving training, without data outsourcing, is *Federated Learning* (FL) (McMahan et al. 2016). FL has emerged as a promising approach to collaboratively train a model by exchanging model parameters with a central aggregator (or server), instead of the actual training data. However, parameter exchange may still leak a significant amount of private data (Zhu, Liu, and Han 2019). Several approaches have been proposed to overcome this leakage problem based on differential privacy (DP) (Shokri and Shmatikov 2015; Papernot et al. 2018), MPC (Bonawitz et al. 2017; Ryffel et al. 2018), HE (Truex et al. 2019), etc. While DP-based learning aims to mitigate inference attacks, it significantly degrades model utility as the training of accurate models requires high privacy budgets (Jayaraman and Evans 2019). Cryptographic techniques provide improved privacy protection but remain too slow for practical use due to the extensive cryptographic operations. Hence, there arises a need for a secure, decentralized FL framework that protects user privacy, while allowing seamless training of ML models. This requires strong cryptographic protection of the intermediate model updates during the model aggregation and the final model weights.

In this work, we propose SCOTCH, a *practical* framework that enables secure *m-party* aggregation in a distributed *n-server* setting. It provides end-to-end protection of the parties' training data, intermediate model weights, and the final resulting model by combining secure multiparty computation (MPC) primitives based on *secure outsourced computation* and *secret sharing* to enable decentralized FL. Our contributions have been described in further detail in the following section.

Our Contributions

In this paper, we introduce a one-of-its-kind framework for privacy-preserving federated learning with primitives from conventional machine learning and multiparty computation (MPC). Specifically,

- We propose SCOTCH, a simple, fast, and efficient federated learning framework that allows for decentralized gradient aggregation using *secure outsourced computation* and *secret sharing* while ensuring strict privacy guar-

antees of the training data (Mohassel and Zhang 2017; Wagh, Gupta, and Chandran 2019).

- We evaluate the efficiency of our proposed *secret sharing*-based FL protocol against existing *state-of-the-art* frameworks. To the best of our knowledge, SCOTCH is the only approach for decentralized privacy-preserving FL with the least possible cryptographic computational overheads – only $O(2mn)$ crypto-related operations required in each training round, where m is the number of participants and n is the number of aggregators (See Table 1).
- We implement SCOTCH and perform extensive experiments on multiple standard datasets such as MNIST, EMNIST, and FMNIST with promising results: SCOTCH has efficiency improvements both in training time and communication cost while providing similar model performance and privacy guarantee as other approaches.

For ease of access, all of our code and experiments are available at: https://github.com/arupmondal-cs/SCOTCH.

Technical Background

Federated Learning. FL (McMahan et al. 2016) is a distributed ML approach that enables model training on a large corpus of decentralized data with myriad participants. It is an example of the more general approach of "bring code to data, not data to code". In FL, each party trains a model locally and exchanges only model parameters with an FL *server* or *aggregator*, instead of the private training data.

The participants in the training processes are *parties* and the *FL server*, which is a cloud-based distributed service. Devices agreement to the server that they are ready to run an *FL task* for a given *FL population*. An FL population is specified by a globally unique name which identifies the learning problem, or application, which is worked upon. An FL task is a specific computation for an FL population, such as training to be performed with given hyperparameters, or evaluation of trained models on local device data. After finishing the local computation on its local dataset then each device updates the model parameters (e.g. the weights of a neural network) to the FL server. The server incorporates these updates into its global state of the global model.

Secure Multiparty Computation. Secure multiparty computation (MPC) (Yao 1982; Mood et al. 2016; Perry et al. 2014; Di Crescenzo et al. 2014; Gupta et al. 2016; Lindell 2020; Goldreich 1998; Goldreich, Micali, and Wigderson 2019) is the universal cryptographic functionality, allowing any function to be computed obliviously by a group of mutually distrustful parties. There exist a number of different techniques for MPC (*e.g.,* garbled circuits, functional encryption, and homomorphic encryption, etc.). In this work, we have considered MPC based on secret sharing (Shamir 1979).

Secret Sharing. In cryptography, secret sharing (Shamir 1979; Blakley 1979) refers to the process of splitting a secret among n parties such that each party does not learn anything about the whole secret from the share it holds. The secret can be reconstructed only if a certain minimum number of parties, greater than or equal to a threshold, t, combine their shares. The scheme is known as the (t, n) threshold scheme or t-out-n secret sharing. In this work, we use additive secret sharing, which uses addition as the way to combine shares. We use the notation $[a]_j$ to denote the j^{th} share of a secret a.

Proposed Framework

In this section, we describe the proposed framework SCOTCH, an efficient distributed secure-computation approach for secure outsourced aggregation based on MPC primitives. The distributed federated averaging algorithm has been described in Algorithm 2. Algorithm 1 briefly describes one iteration of our protocol. The steps given in this algorithm have been illustrated in Figure 1.

Threat Model

We assume a passively secure threat model. A *passive* (honest-but-curious) adversary follows the protocol specifications but may try to learn information about the private input data by inspecting the shared inputs. Both the participants, data owners (or clients), and the aggregators (or servers) are *honest-but-curious*. SCOTCH ensures that aggregators (collude with any subset of participants and aggregators) can't learn any information about the private inputs of the honest participants. Similarly, it also ensures that any subset of colluding participants cannot learn any information about the private inputs or outputs of the honest participants by inspecting the messages exchanged with the aggregators or the final model. We also assume any encryption broadcast to the network in Algorithm 1 is re-randomized to avoid leakage about parties' confidential data by two consecutive broadcasts. We omit this operation in Algorithm 1 for clarity. Finally, *attacks that aim to create denial-of-service attacks or inject malicious model updates are beyond the scope of this short paper*.

SCOTCH Framework

We assume a set of n honest-but-curious aggregators, $\mathcal{S}$ and a set of m clients, $\mathcal{C}$, where each client $\mathcal{C}_i$ for $i \in \{1, \cdots, m\}$ holds its own private dataset $\mathcal{D}_i$. We defer more details about the threat model and security of the framework to 'threat model' and 'privacy guarantees' section. The clients in $\mathcal{C}$ agree upon a model architecture, NN_{arch}, for local training prior to the runtime of the framework. The underlying concept in this framework is n-out-of-n-additive-secret-sharing-based MPC, which provides protocols for n aggregators and is secure against a passive adversary that corrupts at most $m - 1$ clients.

Local Training. At the beginning of every iteration, the function `local_training` is invoked by client $\mathcal{C}_i$ in $\mathcal{C}$ with input $\mathcal{D}_i$. This function allows clients to train local models on their private datasets using the pre-decided model architecture, NN_{arch}. In the first iteration, initial weights are sampled and stored in w. For subsequent iterations, the aggregated weights from the previous iteration are used as initial weights. Each client samples a randomly-permuted (without replacement) subset d from the dataset

$\mathcal{D}_i$ in each iteration. Functions `permute_indices` and `choose_subset` help with the same. In each iteration, each client trains a model on NN_{arch} with inputs w and d. The clients then split the model weights into n-out-of-n additive secret shares by invoking `split_secret_shares`. These shares are then sent to the n aggregators.

Secure Aggregation. Having received a total of m shares from clients in $\mathcal{C}$, each server $\mathcal{S}_j$ for $\{1, \cdots, n\}$ adds its local shares and divides the sum by the total number of aggregators to obtain the value σ_j by invoking `federated_sum`. One can observe that σ_j is an n-out-of-n additive secret share of the federated average of the local models of the clients. Each server then sends σ_j to clients in $\mathcal{C}$ so that they can obtain the final model.

Computing the Final Model. Having obtained additive secret shares of the federated average from the n aggregators, each client locally adds up the shares to obtain the federated average of their models by invoking the function `compute_final_model`. Clients set the value of variable w as the federated average obtained in this iteration. If the current iteration is the final one, w is returned as the final output. If not, w is used as the initial weights in `local_training` for the subsequent iteration.

To enable seamless integration between machine learning primitives (which generally use floating-point), and MPC primitives (which generally use integers), we use integer ring arithmetic in our implementation. To enable conversions between the float and integer realms, we use functions `float_to_int`, `int_to_float`, and `truncate` based on primitives provided in (Mohassel and Zhang 2017). After training its local model, each client embeds its weights onto the integer ring by invoking `float_to_int`[1]. The rest of the operations are performed in the integer ring realm. At places where two values in the integer ring are multiplied, the product is truncated by invoking `truncate`. Finally, at the end of every iteration, the aggregated weights are converted back to float by invoking `int_to_float` in order to facilitate any further local training on them.

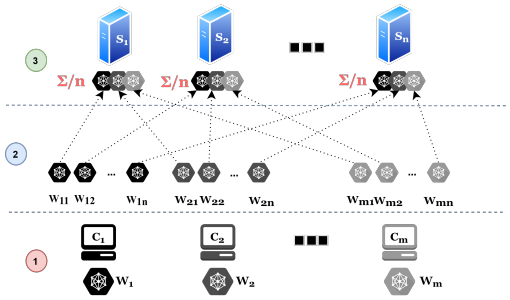

Figure 1: Schematic Diagram illustrating Algorithm 1.

[1] `float_to_int` converts a floating-point value into an l-bit integer by allocating l_x bits to the integer part, l_f bits to the fractional part, and 1 bit to the sign of the value, such that $l = l_x + l_f + 1$. Note that l_f represents the maximum *precision* of the value. Refer to (Wagh, Gupta, and Chandran 2019) for further details.

Algorithm 1: Secure Outsourced Aggregation

Input: Client $\mathcal{C}_i$ for $i \in \{1, \cdots, m\}$ holds its private dataset $\mathcal{D}_i$.
Output: Client $\mathcal{C}_i$ for $i \in \{1, \cdots, m\}$ obtains the final aggregated global model, $\mathbb{M}_{\text{agg}}$.

1. Client $\mathcal{C}_i$ for $i \in \{1, \cdots, m\}$ trains local model $\mathbb{M}_i$ on a random subset of its private dataset $\mathcal{D}_i$. Note that all the clients use the same model architecture.
2. Client $\mathcal{C}_i$ for $i \in \{1, \cdots, m\}$ creates n additive secret shares of its model $\mathbb{M}_i$ and sends each share $[M_i]_j$ for $j \in \{1, \cdots, n\}$ to server $\mathcal{S}_j$.
3. Server $\mathcal{S}_j$ for $j \in \{1, \cdots, n\}$ adds up the received shares from all clients and divides the sum by n to obtain σ_j and sends σ_j to all the clients.
   Each client locally computes $\mathbb{M}_{\text{agg}} = \sum_{j=1}^{n}(\sigma_j)$.

Algorithm 2: SCOTCH Framework

Input: Client C_i in $\mathcal{C}$ possesses private dataset $\mathcal{D}_i$ for $i \in \{1, \cdots, m\}$. $iter :=$ the total number of global iterations for aggregation len($\mathcal{D}_i$) represents the number of data points in the dataset $\mathcal{D}_i$. n is the total number of aggregators.
Output: Clients obtain the final aggregated model stored in w.

- **foreach** $k \in \{1, \cdots, iter\}$
  - **foreach** $i \in \{1, \cdots, m\}$
    - `local_training`($\mathcal{C}_i$, $\mathcal{D}_i$);

  - **Procedure** `local_training` ($\mathcal{C}_i$, $\mathcal{D}_i$):
    - **If**($k == 1$)
      - $w \leftarrow$ `random_init()`; // randomly sample initial weights for n.
    - $d =$`permute_indices`($\mathcal{D}_i$);
    - $d =$`choose_subset`(d, len(D_i) / $iter$);
    - $W_i \leftarrow$ train(w, d);
    - $\mathbb{M}_i \leftarrow$ `float_to_int`(W_i);
    - $\{[\mathbb{M}_i]_1, \cdots, [\mathbb{M}_i]_n\} \leftarrow$ `split_secret_shares`($\mathbb{M}_i$); // split the model into n-out-of-n additive secret shares.

  - **foreach** $j \in \{1, \cdots, n\}$
    - `federated_sum`(S_j, $[\mathbb{M}_i]_j$);

  - **Procedure** (S_j, $\{[\mathbb{M}_1]_j, \cdots, [\mathbb{M}_m]_j\}$):
    - $\sigma_j \leftarrow \sum_{i=1}^{m}([\mathbb{M}_i]_j) \times$ `float_to_int`$(1/n)$;
    - $\sigma_j \leftarrow$ `truncate`(σ_j);

  - **foreach** $i \in \{1, \cdots, m\}$

- `compute_final_model`($\mathcal{C}_i, \sigma_j$);

- **Procedure** ($\sigma_1, \cdots, \sigma_n$):
- $\mathbb{M}_{agg} \leftarrow \sum_{j=1}^{n}(\sigma_j)$;
- $w \leftarrow$ `int_to_float`($\mathbb{M}_{agg}$);

- **return** w;

Communication Complexity Table 1 describes the complexity of the secure aggregation protocol (refer to Algorithm 2). Since SCOTCH is a secure aggregation framework, the complexity of functions `local_training` and `compute_final_model` can be considered *offline*. As a result, we only consider `federated_sum()` as the *online* phase of the protocol.

Table 1: Summary of the complexity of Algorithm 2.

Complexity	Data Owners	Aggregator Servers
Computation	$O(2mn)$	$O(mn)$
Communication	$O(n)$	$O(m)$
Storage	$O(m)$	$O(n)$

Privacy Guarantees

SCOTCH achieves data privacy guarantees under a semi-honest adversary model with any subset of colluding aggregators. SCOTCH's infrastructure is designed using multi-input *secret sharing*-based MPC protocol to calculate the federated average of model gradients shared by participating clients. Private training data is not sent – participating entities get split "shares" of model gradients, or the generated averaged model, neither of which can be used to reconstruct sensitive information about the training dataset used. The security of these shares is guaranteed by standard MPC theorems (Goldreich 1998), and since the actual computations performed within the MPC setup (which can perform arbitrary computations and is agnostic in that sense).

Experimental Evaluation

Implementation Details

We simulate SCOTCH using socket, a low-level networking interface that can be accessed using Python. We rely on the Tensorflow library for the training and inference of machine learning models. All our experiments are performed on a local machine – a Linux machine with Intel i7-9700K CPU@3.60 GHz and GeForce RTX 2070 GPU with 32 GB RAM. All clients and servers are assumed to be running independent nodes and are connected via a virtual network.

Dataset and Model Configuration

MNIST (MNIST). The MNIST (MNIST) dataset comprises 60,000 handwritten digit character images, along with 10,000 testing images. The data is pre-processed by resizing each image, and one-hot encoding the labels. Each client uses a three-layer Multi-layered Perceptron to train on their local datasets. The architecture of the MLP is outlined in Figure 2.

EMNIST (EMNIST). The Extended MNIST (digits) (EMNIST) dataset contains 240,000 handwritten digit character images and 40,000 images for training and testing purposes respectively. The data is preprocessed by resizing each image, and one-hot encoding the labels. We use the same MLP architecture as used in MNIST, to train each local model.

FMNIST (FMNIST). Fashion-MNIST (FMNIST) is a dataset of Zalando's article images that contains 60,000 training images and 10,000 testing images. The data is preprocessed by resizing each image, and one-hot encoding the labels.

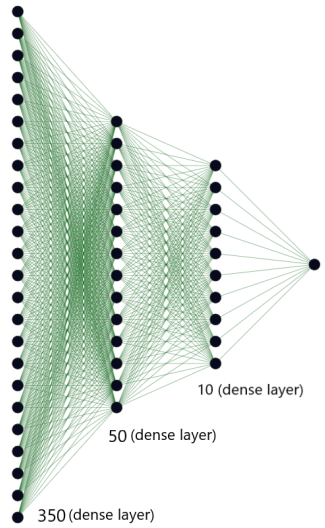

Figure 2: Multilayer Perceptron (MLP) architecture used across different experiments in SCOTCH.

Experimental Overview

SCOTCH's framework incorporates secure aggregation via *secret outsourced computation*. Each client takes part in federated learning by (a) locally training on their private data, and (b) sharing their gradients with servers via secret-sharing. Each server receives partial shares from the clients, which it aggregates and propagates back to all clients. This allows each client to recompute the global model gradients by averaging the shares received from the server(s).

Experimental Results

We evaluate SCOTCH in terms of three indicators: (a) Performance of the generated model with a different number of clients and servers, (b) Impact of varying precision while *secret sharing* on model performance, and (c) communication complexity (see Table 1).

Performance Analysis. We evaluate SCOTCH's performance on three standard datasets – MNIST, EMNIST, and FMNIST (refer Dataset and Model Configuration section)

with varying numbers of clients, in a 3-server setting. For each dataset, we use a three-layer MLP whose architecture has been outlined in §. We use a standard 70-30 train-test split, for each dataset, and the training data is equally divided amongst the clients. Each client locally trains on their individual dataset for 3-4 epochs, with a learning rate of 0.01. The results have been summarized in Table 3.

To test the effects of precision on training our global model, we compare the results of SCOTCH on MNIST dataset, with 16 and 32 bits of precision. The test accuracy comparison between these two is shown in Table 2. To support decimal arithmetic in an integer ring we use the solution proposed by (Mohassel and Zhang 2017). As we observe from our experiments, if we restrict the number of decimal places to 32 bits, we see a significant improvement in test accuracy as opposed to 16 bits. Therefore, we observe a direct correlation between the precision of floating-point numbers involved in network training and the resulting model. To understand the effects of precision, we trained a centralized FL server with a constraint – we round each weight update of the ML model with 32 bits of precision (restricting values up to 5 decimal places). We observed that there is a considerable decrease in model accuracy with decreasing precision. This underscores the importance of precision while training machine learning models. We summarize our observations in Table 4 (for further details, please refer to the Impact of Precision Length section).

We observe a *decrease in accuracy with increasing number of clients* because of the compounding errors in `float_to_int()` *and* `int_to_float()` conversions as a result of *limited precision*. These can be offset by an increase in precision. We plan to scale our existing framework to a larger number of clients and servers with the help of a reasonable increase in precision size in the near future.

Table 2: SCOTCH's performance accuracy on MNIST (MNIST), under 16-bit and 32-bit precision configuration. The number of global iterations for aggregation, *iter* (Algorithm 2) is set to 4. For the accuracy graph, see Figure 4.

Clients	2	3	4	5
MNIST-16	0.3	0.19	0.113	0.111
MNIST-32	0.975	0.965	0.74	0.53

Table 3: SCOTCH performance accuracy, as evaluated on three datasets: MNIST, EMNIST, and FMNIST with an increasing number of clients (under 32-bit precision). For the accuracy graph, see Figure 3.

Clients	MNIST	EMNIST	FMNIST
2	0.975	0.985	0.85
3	0.965	0.984	0.69
4	0.74	0.9	0.53
5	0.53	0.549	0.5

Table 4: Evaluating the performance accuracy of Centralized FL (1-server setting) on multiple precision configurations using the MNIST (MNIST) dataset.

Precision	4-bit	8-bit	16-bit	32-bit
Centralized FL	0.09	0.41	0.71	0.85

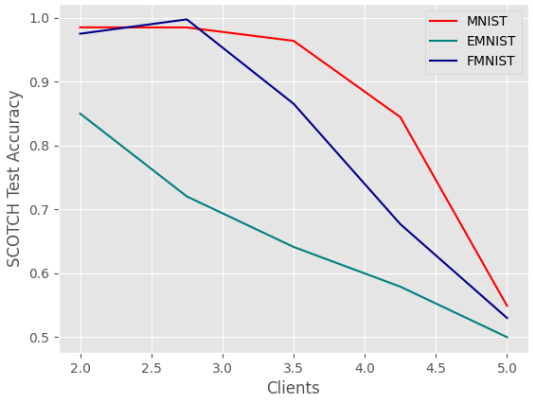

Figure 3: SCOTCH: Clients vs Performance accuracy on multiple datasets – MNIST, EMNIST, and FMNIST (3-server setting).

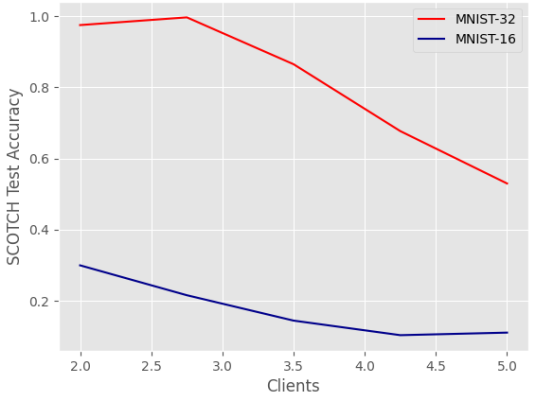

Figure 4: SCOTCH: Clients vs Performance accuracy on multiple precision settings (16-bits, 32-bits) on MNIST (3-server setting).

Impact of Precision Length

Most protocols in secure multiparty computation operate in integer rings. However, one needs to deal with decimal numbers while tackling computations in machine learning algorithms. To mitigate this, we use a mapping between fixed-point decimals and the integer ring (as used by state-of-the-art MPC frameworks such as SecureML (Mohassel and Zhang 2017)). The integer part of the decimal number is represented by l_w bits and the fractional part by l_f. To evaluate the effects of precision on our training and test-

ing accuracy, we replicate the precision settings used in SecureML (Mohassel and Zhang 2017) for logistic and linear regression. Comparing our tests with SecureML helps us to understand the effects of precision-length while training different machine learning models. Even though SecureML's experiments were restricted to 13–16 bits, they used much simpler models such as logistic regression and with a simpler dataset – (1000 to 1M samples of the MNIST dataset), and objective – Binary Classification. Through our experiments we observe that multi-class classification via Multilayer Perceptrons on much smaller dataset 70% of 60,000 MNIST images performs better if we increase the precision (refer to Table 4). One might notice how much of a difference does precision make on gradient updates while performing gradient descent using Neural Networks. This difference plays a role in our experiments as well.

Related Work

The existing privacy-preserving machine learning (PPML) works focus exclusively on training (generalized) linear models. They rely on *centralized* solutions where the learning task is securely outsourced to a server, notably using homomorphic encryption (HE) techniques. As such, these works do not solve the problem of privacy-preserving distributed ML, where multiple parties collaboratively train an ML model on their data. To address the latter, several works propose multi-party computation (MPC) (Yao 1982; Mood et al. 2016; Perry et al. 2014; Di Crescenzo et al. 2014; Gupta et al. 2016; Lindell 2020; Goldreich 1998; Goldreich, Micali, and Wigderson 2019) solutions where several tasks, such as clustering and regression, are distributed among 2, 3, or 4 servers (Mohassel and Zhang 2017; Wagh, Gupta, and Chandran 2019; Patra and Suresh 2020; Ramachandran et al. 2021; Wagh et al. 2021; Riazi et al. 2018; Demmler, Schneider, and Zohner 2015; Mohassel and Rindal 2018; Wagh et al. 2021). Although such approaches, however, limit the number of parties among which the trust is split, often assume an honest majority among the computing servers, and require parties to communicate (i.e., secret share) their data outside their premises. This might not be acceptable due to the privacy and confidentiality requirements and the strict data protection regulations.

A recently proposed alternative for privacy-preserving training – without data outsourcing – is *federated learning* (FL) (McMahan et al. 2016). FL has emerged as a promising approach to collaboratively train a model by exchanging model parameters with a central aggregator, instead of the actual training data. However, parameter exchange may still leak a significant amount of private data. Several approaches have been proposed to overcome this leakage problem based on differential privacy (DP) (Shokri and Shmatikov 2015; Papernot et al. 2018), MPC (Bonawitz et al. 2017; Ryffel et al. 2018), HE (Truex et al. 2019; Sav et al. 2020), Trusted Execution Environment (Mondal et al. 2021a,b), etc. Furthermore, in those settings, the aggregator is a central player, which also potentially represents a single point of failure (Kairouz et al. 2021) and due to the extensive use of cryptographic operations, these frameworks remain too slow for practical use. Finally, other works combine MPC with DP techniques to achieve better privacy guarantees (Truex et al. 2019; Xu et al. 2019; Pettai and Laud 2015). While DP-based learning aims to mitigate inference attacks, it significantly degrades model utility, as training accurate models requires high privacy budgets (Jayaraman and Evans 2019). Therefore, a *practical* distributed privacy-preserving federated learning approach requires strong cryptographic protection of the intermediate model updates during the model aggregation and the final model weights.

Conclusion

We propose SCOTCH, a decentralized *m-party*, *n-server* secure-computation framework for federated aggregation that utilizes MPC primitives. The protocol provides strict privacy guarantees against honest-but-curious aggregators or colluding data-owners; it offers the least communication overheads compared to other existing *state-of-the-art* privacy-preserving federated learning frameworks on standard datasets. In the near future, we plan to extend this framework to provide security against malicious servers and clients, scale it to a larger number of clients and servers, and finally deploy it via open-source channels for academic and industrial use-cases.

References

Blakley, G. R. 1979. Safeguarding cryptographic keys. In *1979 International Workshop on Managing Requirements Knowledge (MARK)*, 313–318. IEEE.

Bonawitz, K.; Ivanov, V.; Kreuter, B.; Marcedone, A.; McMahan, H. B.; Patel, S.; Ramage, D.; Segal, A.; and Seth, K. 2017. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 1175–1191.

Chen, H.; Asif, S. A.; Park, J.; Shen, C.-C.; and Bennis, M. 2021. Robust Blockchained Federated Learning with Model Validation and Proof-of-Stake Inspired Consensus. arXiv:2101.03300.

Demmler, D.; Schneider, T.; and Zohner, M. 2015. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation.

Di Crescenzo, G.; Feigenbaum, J.; Gupta, D.; Panagos, E.; Perry, J.; and Wright, R. N. 2014. Practical and privacy-preserving policy compliance for outsourced data. In *International Conference on Financial Cryptography and Data Security*, 181–194. Springer.

EMNIST. 2019. The EMNIST Dataset. https://www.nist.gov/itl/products-and-services/emnist-dataset. Accessed: 2020-08-01.

FMNIST. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. https://www.kaggle.com/zalando-research/fashionmnist. Accessed: 2020-08-01.

Goldreich, O. 1998. Secure multi-party computation. *Manuscript. Preliminary version*, 78.

Goldreich, O.; Micali, S.; and Wigderson, A. 2019. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 307–328.

Gupta, D.; Mood, B.; Feigenbaum, J.; Butler, K.; and Traynor, P. 2016. Using intel software guard extensions for efficient two-party secure function evaluation. In *International Conference on Financial Cryptography and Data Security*, 302–318. Springer.

Halevy, A.; Norvig, P.; and Pereira, F. 2009. The Unreasonable Effectiveness of Data. *IEEE Intelligent Systems*, 24: 8–12.

Jayaraman, B.; and Evans, D. 2019. Evaluating differentially private machine learning in practice. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 1895–1912.

Kairouz, P.; McMahan, H. B.; Avent, B.; Bellet, A.; Bennis, M.; Bhagoji, A. N.; and et al. 2021. Advances and Open Problems in Federated Learning. arXiv:1912.04977.

Lindell, Y. 2020. Secure Multiparty Computation (MPC). *IACR Cryptol. ePrint Arch.*, 2020: 300.

McMahan, H. B.; Moore, E.; Ramage, D.; Hampson, S.; et al. 2016. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*.

MNIST. 2010. The MNIST Database. http://yann.lecun.com/exdb/mnist. Accessed: 2020-08-01.

Mohassel, P.; and Rindal, P. 2018. ABY3: A Mixed Protocol Framework for Machine Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, 35–52. New York, NY, USA: Association for Computing Machinery. ISBN 9781450356930.

Mohassel, P.; and Zhang, Y. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, 19–38. IEEE.

Mondal, A.; More, Y.; Rooparaghunath, R. H.; and Gupta, D. 2021a. Flatee: Federated Learning Across Trusted Execution Environments. *arXiv preprint arXiv:2111.06867*.

Mondal, A.; More, Y.; Rooparaghunath, R. H.; and Gupta, D. 2021b. Poster: FLATEE: Federated Learning Across Trusted Execution Environments. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, 707–709. IEEE.

Mood, B.; Gupta, D.; Carter, H.; Butler, K.; and Traynor, P. 2016. Frigate: A validated, extensible, and efficient compiler and interpreter for secure computation. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, 112–127. IEEE.

Papernot, N.; Song, S.; Mironov, I.; Raghunathan, A.; Talwar, K.; and Erlingsson, Ú. 2018. Scalable private learning with pate. *arXiv preprint arXiv:1802.08908*.

Patra, A.; and Suresh, A. 2020. BLAZE: Blazing Fast Privacy-Preserving Machine Learning. In *NDSS*. NDSS.

Perry, J.; Gupta, D.; Feigenbaum, J.; and Wright, R. N. 2014. Systematizing secure computation for research and decision support. In *International Conference on Security and Cryptography for Networks*, 380–397. Springer.

Pettai, M.; and Laud, P. 2015. Combining differential privacy and secure multiparty computation. In *Proceedings of the 31st Annual Computer Security Applications Conference*, 421–430.

Phong, L. T.; Aono, Y.; Hayashi, T.; Wang, L.; and Moriai, S. 2018. Privacy-Preserving Deep Learning via Additively Homomorphic Encryption. *IEEE Transactions on Information Forensics and Security*, 13(5): 1333–1345.

Ramachandran, P.; Agarwal, S.; Mondal, A.; Shah, A.; and Gupta, D. 2021. S++: A Fast and Deployable Secure-Computation Framework for Privacy-Preserving Neural Network Training. *arXiv preprint arXiv:2101.12078*.

Riazi, M. S.; Weinert, C.; Tkachenko, O.; Songhori, E. M.; Schneider, T.; and Koushanfar, F. 2018. Chameleon: A hybrid secure computation framework for machine learning applications. 707–721.

Ryffel, T.; Trask, A.; Dahl, M.; Wagner, B.; Mancuso, J.; Rueckert, D.; and Passerat-Palmbach, J. 2018. A generic framework for privacy-preserving deep learning. *arXiv preprint arXiv:1811.04017*.

Sav, S.; Pyrgelis, A.; Troncoso-Pastoriza, J. R.; Froelicher, D.; Bossuat, J.-P.; Sousa, J. S.; and Hubaux, J.-P. 2020. POSEIDON: Privacy-Preserving Federated Neural Network Learning. *arXiv preprint arXiv:2009.00349*.

Shamir, A. 1979. How to share a secret. *Communications of the ACM*, 22(11): 612–613.

Shokri, R.; and Shmatikov, V. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 1310–1321.

Truex, S.; Baracaldo, N.; Anwar, A.; Steinke, T.; Ludwig, H.; Zhang, R.; and Zhou, Y. 2019. A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, 1–11.

Wagh, S.; Gupta, D.; and Chandran, N. 2019. SecureNN: 3-Party Secure Computation for Neural Network Training.

Wagh, S.; Tople, S.; Benhamouda, F.; Kushilevitz, E.; Mittal, P.; and Rabin, T. 2021. FALCON: Honest-Majority Maliciously Secure Framework for Private Deep Learning.

Xu, R.; Baracaldo, N.; Zhou, Y.; Anwar, A.; and Ludwig, H. 2019. HybridAlpha: An Efficient Approach for Privacy-Preserving Federated Learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, 13–23.

Yao, A. C. 1982. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, 160–164. IEEE.

Zhu, L.; Liu, Z.; and Han, S. 2019. Deep Leakage from Gradients. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32, 14774–14784. Curran Associates, Inc.