HybridAlpha: An Efficient Approach for Privacy-Preserving Federated Learning

Runhua Xu*
University of Pittsburgh
Pittsburgh, Pennsylvania, United States
runhua.xu@pitt.edu

Nathalie Baracaldo, Yi Zhou, Ali Anwar and Heiko Ludwig
IBM Almaden Research Center
San Jose, California, United States
{baracald,hludwig}@us.ibm.com,{yi.zhou,ali.anwar2}@ibm.com

ABSTRACT

Federated learning has emerged as a promising approach for collaborative and privacy-preserving learning. Participants in a federated learning process cooperatively train a model by exchanging model parameters instead of the actual training data, which they might want to keep private. However, parameter interaction and the resulting model still might disclose information about the training data used. To address these privacy concerns, several approaches have been proposed based on differential privacy and secure multiparty computation (SMC), among others. They often result in large communication overhead and slow training time. In this paper, we propose *HybridAlpha*, an approach for privacy-preserving federated learning employing an SMC protocol based on functional encryption. This protocol is simple, efficient and resilient to participants dropping out. We evaluate our approach regarding the training time and data volume exchanged using a federated learning process to train a CNN on the MNIST data set. Evaluation against existing crypto-based SMC solutions shows that *HybridAlpha* can reduce the training time by 68% and data transfer volume by 92% on average while providing the same model performance and privacy guarantees as the existing solutions.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; • **Computing methodologies → Distributed artificial intelligence**; *Neural networks.*

KEYWORDS

Federated learning, privacy, functional encryption, neural networks

ACM Reference Format:
Runhua Xu and Nathalie Baracaldo, Yi Zhou, Ali Anwar and Heiko Ludwig. 2019. HybridAlpha: An Efficient Approach for Privacy-Preserving Federated Learning. In *12th ACM Workshop on Artificial Intelligence and Security (AISec'19), November 15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3338501.3357371

*The work was done while interning at the IBM Research - Almaden.

AISec'19, November 15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6833-9/19/11…$15.00
https://doi.org/10.1145/3338501.3357371

1 INTRODUCTION

Machine learning (ML) has been widely applied in industry and academia to a wide variety of domains [23, 26]. While traditional ML approaches depend on a centrally managed training data set, privacy considerations drive interest in decentralized learning frameworks in which multiple participants collaborate to train a ML model without sharing their respective training data sets. Federated learning (FL) [25, 28] has been proposed as a decentralized process that can scale to thousands of participants. Since the training data does not leave each participant's domain, FL is suitable for use cases that are sensitive to data sharing. This includes health care, financial services and other scenarios of particular privacy sensitivity or subject to regulatory mandates.

In FL, each participant trains a model locally and exchanges only model parameters with others, instead of the active privacy-sensitive training data. An entity called *aggregator* merges the model parameters of different participants. Often, an aggregator is a central entity that also redistributes the merged model parameters to all participants but other topologies have been used as well, e.g., co-locating an aggregator with each participant. However, this approach still poses privacy risks: inference attacks in the learning phase have been proposed by [30]; deriving private information from a trained model has been demonstrated in [37]; and a model inversion attack has been presented in [19].

To address such privacy leakage, differential privacy [15, 17] has been proposed for a learning framework [1, 31], in which a trusted aggregator controls the privacy exposure to protect the privacy of the model's output. Similarly, [32] proposes to combine differential privacy techniques and secure multiparty computation (SMC) to support privacy-preserving analyses on private data from different data providers, whereas [6] combines secret sharing and authenticated encryption in a failure-robust protocol for secure aggregation of high-dimensional data.

Inspired from the hybrid methodology [32], a recent paper [38] also proposed a hybrid solution that provides strong privacy guarantees while still enabling good model performance. This hybrid approach combines a *noise-reduction* differential privacy approach with protection of SMC protocol, where the underlying security cornerstone is additive homomorphic encryption, i.e., threshold Paillier system [11]. Even though the hybrid approach has good model performance and privacy guarantees, it comes with long training time and high data transmission cost and cannot deal with participants dropping out during the FL process. In Table 1, we summarize existing privacy-preserving approaches from the perspectives of threat model, privacy guarantees, and offered features.

We believe a privacy-preserving FL framework should strive for strong privacy guarantees, high communication efficiency, and resilience to changes. As shown by Table 1, approaches that offer privacy guarantees incur a large number of communication rounds, substantially increasing the training time for FL systems.

To address the above-mentioned challenges, we propose *HybridAlpha*, an efficient approach for privacy-preserving FL. *HybridAlpha* employs *functional encryption* to perform SMC. Using *functional encryption*, we define a simple and efficient privacy-preserving FL approach, which also supports a participant group that is changing during the course of the learning process. We summarize our key contributions as follows:

- We propose *HybridAlpha*, an efficient privacy-preserving FL approach that employs a differential privacy mechanism and defines a SMC protocol from a multi-input functional encryption scheme. We adapt such scheme and include additional provisions to mitigate the risk that curious aggregators and colluding participants will infer private information.
- We implement and compare - both theoretically and experimentally - a functional encryption scheme to common, traditional cryptography schemes such as additive homomorphic encryption and its variants, which are typically used for SMC. Our benchmark results will guide future adoption of these cryptosystems in the selection of adequate SMCs for FL.
- We describe an implementation of the *HybridAlpha* approach and apply it to a convolutional neural network. The experimental results on the MNIST dataset show that our *HybridAlpha* framework has efficiency improvements both in training time and communication cost, while providing the same model performance and privacy guarantee as other approaches.
- At the same time, we demonstrate a solution to the dynamic participant group issue, which indicates that our proposed framework is robust to participants' dropout or addition. We also analyze the security and privacy guarantee of the *HybridAlpha* framework under our defined threat model within a trusted TPA, honest-but-curious aggregators, and partially dishonest participants.

To the best of our knowledge, this is the first approach for privacy-preserving federated learning that demonstrates how to make use of functional encryption to prevent certain inference attacks that would be possible by naively applying this cryptosystem. We demonstrate that our approach has better model performance, stronger privacy guarantee, lower training time and more efficient communication compared to existing solutions.

Organization. The rest of the paper is organized as follows. In X2, we introduce background and preliminaries. We propose our *HyrbidAlpha* framework and its underlying threat model in X3. The evaluation, as well as the security and privacy analysis are respectively presented in X4 and X5. We discuss related works in X6 and conclude the paper in X7.

2 BACKGROUND AND PRELIMINARIES

In this section, we introduce the background and explain the underlying building blocks of our proposed framework.

2.1 Privacy Preserving Federated Learning

The first FL design aimed to protect the data privacy by ensuring each participant would keep its data locally and uniquely transmit model parameters [28]. Although at first glance it may provide some level of privacy, attacks in the literature have demonstrated that it is possible to infer private information [19, 30, 37]. To fully protect the privacy of the training data from inference attacks, it is necessary to provide the *privacy of the computation* and the *output*.

Privacy of computation. Malicious participants involved in FL training may have an incentive to infer private information of others. Messages exchanged with the aggregator contain model updates that leak private information. For instance, if a bag of words is used as embedding to train a text-based classifier, inspecting gradients can help an adversary identify what words where used (e.g., non-zero gradients constitute words used). SMC protocols can be used to protect inference attacks at training time. These protocols ensure that individual results cannot be exposed while still allowing the computation of aggregated data.

Privacy of output. Machine learning models can also leak private information about the training data [19, 30, 37]. Here, adversaries can repeatedly query the model to identify if a particular observation was part of the training data. To prevent against these attacks, differential privacy has been proposed. In this case, noise is added to the model to protect individual records in the training dataset.

Existing approaches. Table 1 presents an overview of privacy preserving approaches (a more thorough description is presented in X6). Although some of them provide privacy guarantees for the computation and output, they lack relevant features for FL systems. In particular, approaches that increase the number of communication rounds can hinder the applicability of FL, as they augment the training time and amount of data exchanged. For large models such as neural networks, this is a major concern.

Another important feature should be provided by FL frameworks is the support for *dynamic participation*. In some scenarios, participants may leave the training process at any time, we refer to these as *dropouts*. As shown in Table 1, existing approaches cannot gracefully deal with dropouts and require re-doing an overall training round with new keys. New participants may also join the training process at any time. Existing approaches do not provide support for this dynamic flow and require full-re-keying.

Our proposed *HybridAlpha* reduces significantly the training time by limiting the number of messages exchanged to one by round - substantially less than existing approaches that offer privacy of computation. In what follows, we present in detail some of the basic building blocks that allow us to achieve this result.

2.2 Differential Privacy and Multiparty Computation

Differential privacy (DP) [15, 17] is a rigorous mathematical framework where an algorithm may be described as differentially private if and only if the inclusion of a single instance in the training dataset causes only statistically insignificant changes to the algorithm's output. The formal definition for DP is as follows:

Definition 2.1. (Differential Privacy [15]). A randomized function K gives (ϵ, δ)-*differential privacy* if for all data sets D and D'

Table 1: Comparison of privacy-preserving approaches in federated machine learning framework

	Threat Model		Privacy Guarantee		SMC	Features	
Proposed Approach	participant	aggregator	computation	output	type	communication [y]	dynamic participants
Shokri and Shmatikov[36]	honest	honest	✗	✓		1 round	✓
PATE [31]	honest	honest	✗	✓		1 round	
PySyft [34]	honest	HbC	✓	✓	HE	2 rounds[z]	
Bonawitz et al. [6]	dishonest	HbC	✓	✓	SS+AE	3 rounds[z]	dropout
Truex et al. [38]	dishonest	HbC	✓	✓	TP	3 rounds[z]	✗
HybridAlpha (our work)	dishonest	HbC	✓	✓	FE	1 round[z]	dropout + addition

"SS+AE"represents secret sharing techniques with key agreement protocol and authenticated encryption scheme; "HE" is homomorphic encryption scheme; "TP" is Threshold-Paillier system, a partially additive homomorphic encryption scheme; "FE" indicates functional encryption scheme; symbol indicates non-comparative option. HbC is the abbreviation of Honest but Curious.

[y] The count is based on one epoch at the training phase between the aggregator and the participant.

[z] The key distribution communication is not covered here.

differing on at most one element, and all S $\subseteq$ Range(K),

$$\Pr[K(D) \in S] \leq \exp(\epsilon) \Pr[K(D') \in S] + \delta.$$

The probability is taken over the coin tosses of K.

Note that ϵ*-differential privacy* can be treated as a special case of (ϵ, δ)*-differential privacy* where $\delta = 0$. To achieve DP, multiple mechanisms designed to inject noise to the algorithm's output have been proposed. These mechanisms add noise proportional to the sensitivity of the output, a measure of the maximum change of the output resulting by the inclusion of a single data point. Popular mechanisms include Laplacian and Gaussian mechanisms, where the Gaussian mechanism for a dataset D is defined as $M(D) = f(D) + N(0, S_f^2\sigma^2)$, where $N(0, S_f^2\sigma^2)$ is the normal distribution with mean 0 and standard deviation $S_f\sigma$. By applying the Gaussian mechanism to function f with sensitivity S_f satisfies (ϵ, δ)-differential privacy [16].

Noise Reduction through SMC. SMC allows multiple parties to compute a function over their inputs, without revealing their individual inputs [5, 10]. SMC can be achieved using different techniques such as garbled circuit with oblivious transfer, fully or partially homomorphic encryption, and functional encryption.

Prior work has shown that it is possible to maintain the same DP guarantee achieved by *local differential privacy* [24, 33], i.e., each party adds its own noise independently, and uses SMC to hide individual inputs. Concretely, using the Gaussian mechanism defined above, local differential privacy requires each participant to independently add $N(0, S_f^2\sigma^2)$. Considering n parties, the total noise adds up to n. However, when applying SMC each participant can add a fraction of the noise $N(0, \frac{1}{n}S^2\sigma^2)$ and then use a SMC technique to share the value for aggregation. As shown in [38], this ensures the same DP guarantee while reducing the amount of total noise injected by a factor of n.

2.3 Functional Encryption

HybridAlpha relies on Functional Encryption (FE), a public-key cryptosystem that allows parties to encrypt their data, meanwhile, an external entity can compute a specific function on the ciphertext without learning anything additional from the underlying plaintext data [8]. For this purpose, a *trusted third party* (TPA) is used to set up a master private key and a master public key that will be used to derive multiple public keys to one or more parties who intend to encrypt their data. Given a function $f(\cdot)$, a functional private key sk_f will be generated by the TPA who holds a master secret key. By processing the function related key sk_f a decryptor can compute $f_{\text{sk}_f}(x)$ from the encrypted data, $enc(x)$, without revealing the plaintext x.

To satisfy the distributed setting, a variant of functional encryption, i.e., Multi-Input Functional Encryption (MIFE) is proposed in [20], where a functional secret key sk_f can correspond to an n-ary function $f(\cdot)$ that takes multiple ciphertexts as input. In our federated machine learning framework, we adopt with modifications the MIFE scheme for inner-product proposed in [2, §4.4] due to its computational efficiency. Unlike other FE schemes, this construction provides a construction that does not require performance intensive pairing operations, and hence reduces the overall run-time.

We now present our construction of a MIFE scheme for federated learning, which derived from the construction in [2, §4.4] whose security relies on the Decisional Diffie-Hellman (DDH) assumption. This MIFE scheme relies on five algorithms, *Setup*, *PKDistribute*, *SKGenerate*, *Encrypt*, *Decrypt*, and three roles: the *third party authority (TPA)*, *participants* and *aggregators*. Algorithms *Setup, PKDistribute, SKGenerate* are run by *TPA*. Each *participant* runs *Encrypt* algorithm to encrypt their model parameters and the *aggregator* runs *Decrypt* algorithm to acquires the average sum of encrypted model parameters. Notice that comparing to the MIFE construction in [2], we add an additional algorithm, i.e., *PKDistribute*, to help deliver the public keys *pk* to *participants*. As a result, each *participant* will encrypt their data using its own *pk* rather than using the master key *msk* as described in [2]. This modification is beneficial because different parties do not need to share a single master key.

Suppose the inner-product functionality is defined as follows:

$$f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n, \mathbf{y}) = \sum_{i=1}^{\tilde{n}} \sum_{j=1}^{\tilde{m}} (x_{ij} y_{\left[\sum_{k=1}^{i-1} \eta_k + j\right]}) \text{ s.t. } |\mathbf{y}| = \sum_{i=1}^{\tilde{n}} \eta_i,$$

where n denotes the total number of input sources, η_i is the length of each input vector $\mathbf{x}_i$, and $dimension(\mathbf{y}) = \sum_{i=1}^{n} dimension(\mathbf{x}_i)$ The specific construction of MIFE is defined follows:

Setup$(1^\lambda, \mathsf{F}_n^{\tilde{\eta}})$: The algorithm first generates secure parameters as $G := (G, p, \partial)$ GroupGen(1^λ), and then generates several samples as $a \xleftarrow{R} Z_p$, $\mathbf{a} := (1, a, |, \forall i \in \{1, \ldots, n\}$:

$\mathbf{W}_i \xleftarrow{R} \mathbb{Z}_p^{\eta_i \times 2}$, $\mathbf{u}_i \xleftarrow{R} \mathbb{Z}_p^{\eta_i}$. Then, it generates the *master public key* and *master private key* as

$$\mathbf{mpk} := \{\mathbb{G}, [\![\mathbf{a}]\!]_1, [\![\mathbf{W}\mathbf{a}]\!]_1\}; \quad \mathbf{msk} := \{\mathbf{W}, \{\mathbf{u}_i\}_{i \in \{1,\ldots,n\}}\}.$$

PKDistribute$(\mathbf{mpk}, \mathbf{msk}, \mathrm{id}_i)$: It looks up the existing keys via id_i and returns the *public key* as $\mathbf{pk}_i := \{\mathbb{G}, [\![\mathbf{a}]\!]_1, \{\mathbf{W}\mathbf{a}\}_i, \{\mathbf{u}_i\}\}$.

SKGenerate$(\mathbf{mpk}, \mathbf{msk}, \mathbf{y})$: The algorithm first partitions $\mathbf{y}$ into $\{\mathbf{y}_1 || \mathbf{y}_2 || \cdots || \mathbf{y}_n\}$, where $|\mathbf{y}_i|$ is equal to η_i. Then it generates the function derived key as

$$\mathbf{sk}_{f,\mathbf{y}} := \{\mathbf{d}_i^{\top}\}_0 \quad \{\mathbf{y}_i^{\top} \mathbf{W}_i\}_{\mathbf{z}} \quad \tilde{O}\{\mathbf{y}_i^{\top} \mathbf{u}_i\}.$$

Encrypt$(\mathbf{pk}_i, \mathbf{x}_i)$: The algorithm first generates a random nonce $r_i \xleftarrow{R} \mathbb{Z}_p$, and then computes the ciphertext as

$$\mathbf{ct}_i := \{[\![\mathbf{t}_i]\!]_1 \quad [\![\mathbf{a}r_i]\!]_1, [\![\mathbf{c}_i]\!]_1 \quad [\![\mathbf{x}_i + \mathbf{u}_i + \mathbf{W}\mathbf{a}_i r_i]\!]_1\}.$$

Decrypt$(\mathbf{ct}, \mathbf{sk}_{f,\mathbf{y}})$: The algorithm first calculates as follows:

$$C := \frac{\hat{\Pi}_{i \in \{1,\ldots,n\}} [\![\mathbf{y}_i^{\top} \mathbf{c}_i]\!]_1 \bullet [\![\mathbf{d}_i^{\top} \mathbf{t}_i]\!]_1}{z};$$

and then recovers the function result as $f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n, \mathbf{y}) = \log_{\partial}(C)$.

Note that in the federated learning setting, the *aggregator* holds the vector $\mathbf{y}$, and each η_i is set as 1, which indicates the input from each *participant* is a single element instead of the vector as described in the MIFE scheme.

3 *HYBRID-ALPHA* FRAMEWORK

In this section, we present the specific construction of our *HybridAlpha* framework for privacy-preserving federated learning. Our framework prevents inference attacks from curious aggregators and limits the inference power of colluding participants, as detailed later in the threat model.

Figure 1 presents an overview of *HybridAlpha*. *Participants* want to collaboratively learn a machine learning model without sharing their local data with any other entity in the system. They agree on sharing only model updates with an aggregator. This entity is in charge of receiving model updates from multiple participants to build a common machine learning model.

Participants want to protect their data against any inference attack during the FL process and from the final model. For this purpose, they join a *HybridAlpha*, which has a *Third Party Authority (TPA)*. This entity provides a key management service that initiates the cryptosystem and provides functional encryption keys to all parties. To prevent potential leakage of information, *HybridAlpha* also includes an *Inference Prevention Module* that limits what type of functional encryption keys are provided. This module is designed to ensure that decryption keys cannot be obtained by curious aggregators and to limit potential collusion attacks. We detail this module in §3.2.2.

3.1 Threat Model

We consider the following threat model:

Honest-but-curious aggregator: We assume that the aggregator correctly follows the algorithm and protocols, but may try to learn private information inspecting the model updates sent by the participants in the process. This is a common assumption [6, 38].

Curious and colluding participants: We assume that participants may collude to try to acquire private information from other participants by inspecting the messages exchanged with the aggregator or the final model.

Trusted TPA: This entity is an independent agency which is widely trusted by the *participants* and the *aggregator*. In real scenarios, different sectors of the economy already have entities that can take such role. For instance, in the banking industry, central banks often play a fully trusted role, and in other sectors, a third company such as a service or consultant firm can embody the TPA. We also note that assuming such trusted and independent agency is a common assumption in existing cryptosystems that have employed the TPA as the underlying infrastructure [7, 8, 21]. The TPA is in charge of holding the master private and public key. The TPA is also trusted to perform public key distribution and function derived secret key generation. Similarly, *Inference Prevention Module* is fully trusted.

We assume that secure channels are used in all communications, thus, man-in-the-middle and trivial snooping attacks are prevented. We also assume a secure key-provisioning procedure such as Diffie-Hellman is in place to protect key confidentiality. Finally, attacks that aim to create denial of service attacks or inject malicious model updates are beyond the scope of this paper.

Based on the threat model above, our proposed privacy-preserving framework can ensure that (i) the semi-honest aggregator cannot learn additional information except for the expected output by the differential privacy mechanism, and (ii) the malicious colluding participants cannot learn the parameters of other honest participants. The specific security and privacy analysis are presented in §5.

3.2 HybridAlpha Detailed Operations

We now describe in detail the operations of *HybridAlpha* and begin by introducing the notation used. Let A be the aggregator and $\mathsf{S_P}$ be a set of n participants, where each participant P_i holds its own dataset D_i. We denote as L_{FL} the learning algorithm to be trained.

In this section, we first introduce the operations of the framework for non-adversarial settings, and then explain how additional features are used to protect against the inference attacks defined in the threat model section.

3.2.1 Non-adversarial setting. *HybridAlpha*'s operations under non-adversarial settings are indicated in Algorithm 1. As input, *HybridAlpha* takes the set of participants, the algorithm used for training, and the differential privacy parameter ϵ.

HybridAlpha initiates via the TPA setting up keys in the system. In particular, the TPA runs the *Setup* and *PKDistribute* algorithms presented in §2.3, so that each participant P_i has its own public key $\mathbf{pk}_i$ (lines 1-5). We note that *HybridAlpha* allows new participants to join the training process even if it has already started. To achieve this, the TPA provisions a larger number of keys than the initial set of participants (line 2). In this way, when new participants join

[1] Note that $[\![x]\!]_1 = \partial^x$. Here we implicitly adopt the bracket notation from [18], which is somewhat standard in the crypto community.

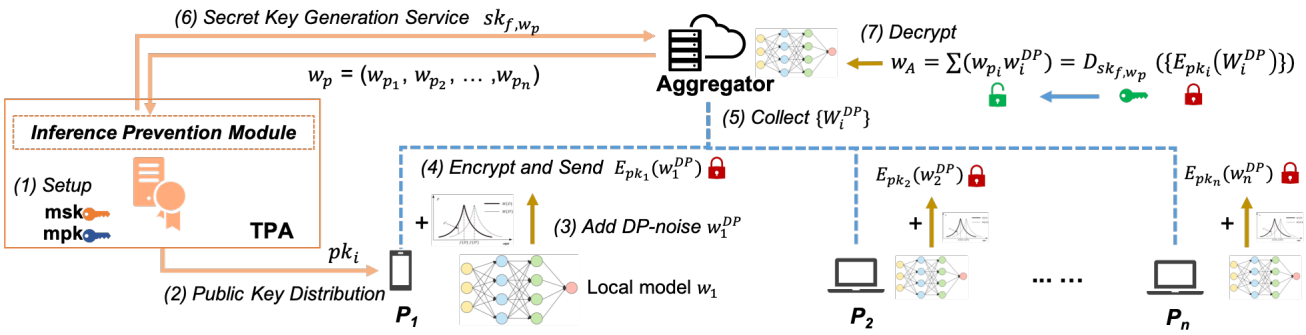


Figure 1: Framework overview of our proposed efficient approach for privacy-preserving federated learning. Note that we only present one epoch here. Each *participant* does the local training based on their owned dataset, and then sends out the model parameters using our proposed efficient privacy-preserving approach.

Algorithm 1: HybridAlpha

Input: L_{FL} := Machine learning algorithms to be trained; ϵ := privacy guarantee; S_P := set of participants, where $P_i \in S_P$ holds its own dataset D_i; N := maximum number of expected participants; t := minimum number of aggregated replies

Output: Trained global model M

1 **function** *TPA-initialization*(1^λ; N; S_P)
2 $\quad$ **mpk**; **msk** $\leftarrow$ MIFE.Setup(1^λ; F_N^1 s.t. $N \leftarrow |S_P|$;
3 $\quad$ **foreach** $P_i \in S_P$ **do**
4 $\quad\quad$ $\mathbf{pk}_i \leftarrow$ MIFE.PKDistribute(**mpk**; **msk**; P_i);
5 $\quad$ **end**
6 **function** *aggregate*(L_{FL}; S_P; t)
7 $\quad$ **foreach** $P_i \in S_P$ **do**
8 $\quad\quad$ asynchronously query P_i with $msg_{q;i} = (L_{FL}, |S_P|)$;
9 $\quad$ **end**
10 $\quad$ **do**
11 $\quad\quad$ $S_{msg_{recv}} \leftarrow$ collect participant response $msg_{r;i}$;
12 $\quad$ **while** $|S_{msg_{recv}}| < t$ ***and*** *still in max waiting time*;
13 $\quad$ **if** $|S_{msg_{recv}}| \geq t$ **then**
14 $\quad\quad$ specify w_P vector; request the $\mathbf{sk}_{f;w_P}$ from TPA;
15 $\quad\quad$ $M \leftarrow$ MIFE.Decrypt($\mathbf{sk}_{f;w_P}$; w_P; $S_{msg_{recv}}$);
16 $\quad$ **end**
17 $\quad$ **return** M
18 **function** *participant-train*(ϵ; t; $msg_{q;i}$; D_i; $\mathbf{pk}_i$)
19 $\quad$ $M_i \leftarrow L_{FL}(D_i)$;
20 $\quad$ $M_i^{DP} \leftarrow$ DP(ϵ; M_i; t);
21 $\quad$ $msg_{r;i} \leftarrow$ MIFE.Encrypt(M_i^{DP}; $\mathbf{pk}_i$);
22 $\quad$ sends $msg_{r;i}$ to aggregator;

the training process, they need to acquire the individual public key from the TPA, and then participate in the learning protocol; all this without requiring any changes for other participants.

To begin the learning process, the aggregator A asynchronously queries each participant P_i with a query to train the specified learning algorithm L_{FL} and the number of participant. Then, the aggregator collects the responses of each party P_i (lines 7-12).

When all responses are received, assuming there is quorum, A needs to request a key from the TPA corresponding to the weighted vector w_p that will be used to compute the inner product. That is, the aggregator requests private key $sk_{f;w_p}$ from the TPA based on w_p. For computation of average cumulative sum of each participant's model, w_p can be set as $w_p = (\frac{1}{n}; \frac{1}{n}; \dots; \frac{1}{n})$ s.t. $|w_p| = n$, where n is the number of received responses. Then, A updates the global model M by applying the decryption algorithm of the MIFE cryptosystem on collected ciphertext set $S_{msg_{recv}}$ and $sk_{f;w_p}$. Note that here we assume the aggregator A will get all responses from every participant. In the case of dropouts, n can be changed so that it reflects the number of participants that are being aggregated. In the next subsection, we show how *HybridAlpha* provides recommendations to set up t so that the number of allowed dropouts are limited for security reasons.

At the participant side, when a query for training is received by participant P_i, it trains a local model M_i using its dataset D_i. During the training process[2], the participant adds differential privacy noise to the model parameters according to the procedure presented in §2.2. Finally, P_i encrypts the resulting noisy model using the MIFE encryption algorithm and sends it to the aggregator (lines 18-22).

3.2.2 Inference Prevention Module. In our threat model, we assume an *honest-but-curious* aggregator that tries to infer private information during the training process. We consider multiple potential attacks where the aggregator manipulates the weighted vector to perform inference.

In particular, suppose that A wants to infer the model of P_i. A can try to launch an inference attack to obtain the model updates of participant k by setting the weighted vector as follow:

$$\forall k \in \{1, \dots, n\} : \mathbf{w}_p^0 = \begin{cases} w_{p_k} = 1; \text{if } k = i \\ w_{p_k} = 0; \text{if } k \neq i \end{cases}$$

If a malicious aggregator is allowed a key to perform the inner product of this vector with the model updates, the model updates of target user k would become visible; this follows because w_p^0 zeros-out the model updates of all other participants except for the

[2]The differential privacy mechanism depends on the machine learning model being trained. For simplicity, in Algorithm 1 we show the noise added after the training process takes place. However, we note that some DP mechanisms add noise during the training process e.g., to train a neural network with the DP mechanism in [1]

ones sent by target participant k. If this is not avoided, A would acquire $w_i + \frac{1}{n}N^1 0; S^2\sigma^{2o}$ as the decryption result of the MIFE cryptosystem. Here, the reduced noise $\frac{1}{n}N^1 0; S^2\sigma^{2o}$ does not provide the expected privacy guarantee to protect M_i of P_i because each honest participant is injecting noise, assuming its model update is aggregated privately with other n participants.

An honest but curious aggregator may also try to create a smaller weighted vector to exclude a subset of participants from the aggregation process. In the worst case, the malicious aggregator would try to shrink the weighted vector to include one single participant to uniquely "aggregate" the model updates of that participant.

Following this same attack vector, a malicious aggregator colluding with dishonest participants may try to build a w_p vector such that: (i) a target participant model update is included in the vectors; (ii) all other honest participants model updates are not aggregated, and (iii) updates of dishonest participants are included in the aggregation process. Since the aggregator is colluding with the dishonest participants included in the aggregation process and only the target participant is included in the aggregation, the model update of the target participant is easily reconstructed (its the single unknown variable in the average equation).

To prevent such inference attacks, we propose an additional component called *Inference Prevention Module* collocated with the TPA. This module intercepts and inspects requests for private keys for given weighted vectors to prevent a curious aggregator from obtaining a key that will allow for an inference-enabling inner product.

To this end, the Inference Prevention Module takes as input a parameter t that defines a threshold on the number of non-colluding participants, where $t \quad \frac{n}{2} + 1$, that is more than half of the participants should not be colluding. By running Algorithm 2 and using parameter t, it is possible to prevent the attacks previously described. In particular, the Inference Module enforces that keys are only provided to weighted vectors that have at least t non-zero elements and that the weight for each included model update is the same.

Threshold t has an impact on the number of dropouts allowed by the system. Mainly, it helps set up the minimum quorum of participants replying to the system. *HybridAlpha* allows a limited number of participants to dropout without requiring any re-keying; only the weighted vector sent by the aggregator needs to be updated by uniquely including the weights of model updates received.

We also note that t has an impact on how much differential privacy noise is added by each participant to achieve a pre-defined ϵ. Concretely, the number of aggregated replies is always at least t, so as explain in x2.2, the noise can be adapted to always account for t non-colluding participants contributing to the average, e.g., $N^1 0; \frac{1}{t}S^2\sigma^{2o}$. For this purpose, t needs to be communicated among all participants and the aggregator.

Underlying ML models of *HybridAlpha*. For simplicity we only use the neural networks as the underlying ML model in our FL framework for illustration and evaluation, however, the our *HybridAlpha* supports various ML algorithms. As functional encryption enables the computation of any inner-product based operation, any model that can be trained through a stochastic gradient descent (SGD)-based algorithm can be trained via our proposed *HybridAlpha*; models in this pool include SVMs, logistic regression, linear regression, Lasso, and neural networks, among others. Other models such as decision trees and random forests which require aggregating counts from each participant can also be trained by considering the counts sent to the aggregator as a vector.

Algorithm 2: Inference prevention filter

Input: $\mathbf{w}_p$:=A weighted vector to be inspected for inference attacks; t:= threshold of minimum number of dropouts and expected number of non-colluding participants

```
function inference-prevention-filter(w_p, t)
    c_nz  count the non-zero element in w_p;
    if c_nz < t then return "invalid w_p" ;
    foreach non-zero w_p_i  2 w_p do
        if w_p_i , 1/c_nz then return "invalid w_p";
    end
    forward w_p to the TPA;
```

4 EVALUATION

In this section we perform a detailed evaluation of our proposed approach to answer the following questions:

How does *HybridAlpha* perform theoretically when compared to existing techniques that have *similar* threat models? More specifically, how many crypto-related operations can be reduced by using *HybridAlpha*?

How does our proposed SMC perform under benchmarking? How does precision setting impact computation time compared with existing techniques? What impact do different numbers of participants have?

How does *HybridAlpha* compare to existing techniques in terms of performance efficiency?

4.1 Baselines and Theoretical Analysis

We compare the proposed *HybridAlpha* with two state of the art private-federated learning approaches: [38] and [34], which use different SMC techniques. A graphical overview and comparison of these baselines can be found in Figure 2, the steps performed by each approach are defined in this figure. We will use this notation to report our results. Additionally, we provide a brief description of our **baselines**:

We refer to the first baseline as **TP-SMC** [38]. This FL approach uses a threshold-based homomorphic cryptosystem that allows for a trusted parameter t that specifies the number of participants that are trusted not to collude.

We refer as **P-SMC** to our second baseline which is inspired by PySyft [34], an opensource system that uses SPDZ protocol [12, 13]. This construct supports homomorphic addition and multiplication. Because the SGD aggregation only requires addition, we opted for a additive homomorphic approach for the comparison, thus, the results reported for this baseline are representative yet faster than PySyft.

We note that the contrasted approaches follows a *similar threat model* to [38] with a honest-but-curious aggregator, and potentially

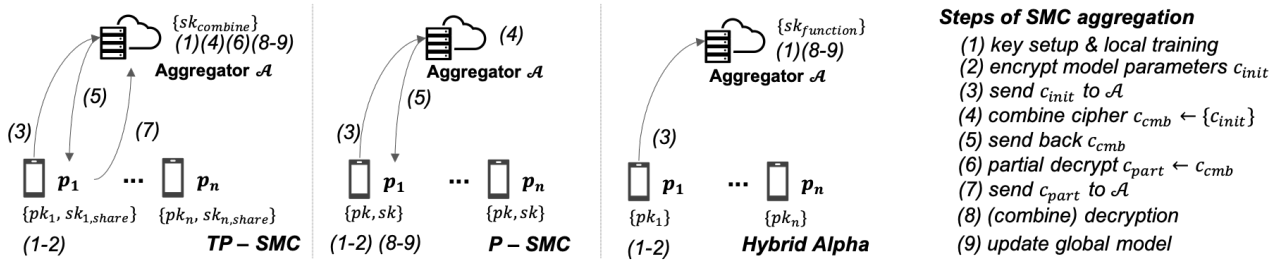


Figure 2: Illustration of aggregation via different crypto-based SMC solutions.

Table 2: The number of crypto-related operations required for each solution.

Communication	TP-SMC	P-SMC	*HybridAlpha*
Step (1)	n	n	$n+m$
Step (3)	n m	n m	n m
Step (5)	m t	n m	-
Step (7)	t m	-	-
TOTAL	$2mt+mn+n$	$2mn+n$	$mn+m+n$

colluding and malicious participants. However, they differ in the assumption of a TPA. We therefore, show how making use of a TPA, *HybridAlpha* can significantly reduce the training time of machine learning models.

Theoretical Comparison. We now theoretically compare the crypto-related communication steps associated with the contrasted approaches. Suppose that there are n participants and m aggregators in the FL framework, and the threshold for decryption of Threshold-Paillier cryptosystem is t. As shown in Table 2, In total, *HybridAlpha* reduces m^1n 1^o and m^12t 1^o operations compared to P-SMC and TP-SMC solutions, respectively. This is achieved because *HybridAlpha* doesn't require sending back encrypted aggregated model updates to the participants for decryption

In the following, we also provide the details of experimental results in x4.3. The experimental results are consistent with the theoretical analysis.

4.2 Experimental Setup

To benchmark the performance of *HybridAlpha*, we train a convolutional neural network (CNN) with the same topology as the one used in [38] to classify the publicly available MNIST dataset of handwritten digits [27]. The CNN has two internal layers of ReLU units, and a softmax layer of ten classes with cross-entropy loss. The first layer contains 60 neurons and the second layer contains 1000 neurons. The total number of parameters of this CNN is 118110. We also use the same hyperparameters reported in previous work: a learning rate of 0.1, a batch rate of 0.01. and for differential privacy we use a norm clipping of 4.0, and an epsilon of 0.5. We used noise-reduction method as in [38] as differential private mechanism. We run experiments for 10 participants, and each participant was randomly assigned 6,000 data points from the MNIST dataset. For model quality, we used the pre-defined MNIST test set. Our implementation uses Keras with a Tensorflow backend.

Cryptosystems Implementation We implement the contrasted cryptosystems in python based on the opensource integer group of the Charm framework [4]. Charm uses a hybrid design, where the underlying performance-intensive mathematical operations are implemented in native C modules, i.e., the GMP library [3], while cryptosystems themselves can be written in a readable, high-level language. Even though there exists Paillier implementation including its threshold variant using other programming languages, we re-implement them in a unified platform to allow for fair benchmarking and to enable easy integration with python-based machine learning frameworks such as Keras and Tensorflow.

In our implementation, we incorporated the following accelerating techniques. In *HybridAlpha*, as presented in x2.3, the final step of MIFE decryption is to compute the discrete logarithm of an integer, which is a performance intensive computation. An example would be how to compute f in $h = \partial^f$, where $h; \partial$ are big integers, while f is a small integer. To accelerate the decryption, we use a hybrid approach to solve the discrete logarithm problem. Specifically, we setup a hash table $T_{h;\partial;b}$ to store $^1h; f^o$ with a specified ∂ and a bound b, where b f b, when the system initializes. When computing discrete logarithms, the algorithm first looks up $T_{h;\partial;b}$ to find f, where the complexity is O^11^o. If there is no result in $T_{h;\partial;b}$, the algorithm employs the traditional *baby-step giant-step* algorithm [35] to compute f, where the complexity is $O^1n^{\frac{1}{2}o}$.

The second acceleration method we implemented modifies the encryption and decryption algorithms to allow for a one-shot encryption call of a tensor. Here, each generated random nonce is applied to the whole tensor instead of a single element. We note that a further performance enhancement technique that could be used is parallelizing the encryption/decryption implementation, however, we did not include this enhancement.

Experimental Setup. All the experiments are performed on a 2 socket, 44 core (2 hyperthreads/core) Intel Xeon E5-2699 v4 platform with 384 GB of RAM. Note that the FL framework is simulated (not run on the real distributed environment), hence the network latency issues are not considered in our experiment. However, we report a comparison of data transfer by contrasted approaches.

4.3 Experimental Results

Here, we first present the benchmark result of three contrasted approaches, and then show the experimental efficiency improvement.

Impact of Floating Point Precision. The parameters of a neural network (weights) are represented as floating point numbers. However, cryptosystems take them as input integers. Hence, the floating point parameters should be represented and encoded into integers. The *precision number* denotes the number of bits used after the

[3]The GNU Multiple Precision Arithmetic Library (https://gmplib.org/).

decimal point of a floating point number. In Table 3 we present the impact of the precision on the computation time of each crypto-based SMC. Based on our experimental results, the precision setting has no significant impact on operation time of each cryptosystem. To be specific, the time cost of encryption, decryption, and other ciphertext computations in each cryptosystem is stable, respectively, of length of the integer.

For encryption, the average time cost of 10 participants on 118110 gradients for *HybridAlpha* is around 4 seconds, while the time cost of P-SMC and TP-SMC under the same setting is about 35 seconds. For decryption, under the same setting, the cost time of HybridAlpha is about 30 seconds, while the time cost of P-SMC and TP-SMC are 31 and 88 seconds, respectively. Note that the decryption time of TP-SMC includes the share decryption by part of participants and the final combination decryption by the aggregator, without considering network latency of transmitting the partial decrypted ciphertext. We can conclude that our proposed approach has significant advantages on both encryption/decryption time cost comparing to P-SMC and TP-SMC solutions.

Finally, the number of decimal points used in the conversion impacts the overall accuracy of the trained model. In the remaining of the experiments, we used 6-digits which allows for good model and training time performance.

Impact of Number of Participants. We also measure the impact of the number of participants on the time cost for each crypto operation. The experimental results are shown in Table 4. We see two different trends on the participant and on the aggregator side. At the participant side, the encryption and decryption runtime stays the same for all of the evaluated approaches as the number of participants increases. In contrast, on the aggregator side, the time cost of ciphertext multiplication increases almost linearly with the increase in the number of participants (shown in italicized numbers in Table 4). However, we note a significant difference between *HybridAlpha* and TP-SMC. For *HybridAlpha* the decryption time increases approximately linearly with the increase of participants, while for TP-SMC, the decryption time increases exponentially as the number of participants increases.

Focusing on the TP-SMC, we also evaluate the impact of threshold t, which indicates the minimum number of participants who are required to do partial decryption. As shown in Table 5, only the final decryption has significant relationship with threshold t. For the same number of participants, the cost time of decryption increases linearly as the threshold number increase.

Model Quality, Training Time and Data Transmission . In this experiment, we evaluate the performance of *HybridAlpha* with respect to multiple techniques to perform FL. In particular, we assess the quality of models produced and the total training time. The contrasted approaches for this experiment include the following additional baselines: (i) "FL-no-privacy", where the neural network is trained without privacy considerations. This method provides a baseline for maximum possible performance in terms of model quality; (ii) "Local DP", where each participant applies differential privacy locally according to [24]; (iii) for "TP-SMC", "P-SMC" and "*HybridAlpha*", we report the results for two cases: adding differential privacy to protect privacy of the output and without adding differential privacy. When no differential privacy is added, we use "TP-SMC no DP", "P-SMC no DP" and "*HybridAlpha* no DP". For privacy-preserving approaches we use an $\epsilon = 0.5$. Finally, our experiments used $t = 5$ for *HybridAlpha* and TP-SMC. This experiment was run with 10 participants.

To measure quality of model performance, we report the F1-score (a measure that combines precision and recall) of the resulting models. The results are presented in Figure 3(a).

We see different trends depending on whether a particular approach protects privacy of the computation and of the output. As expected, approaches that do not protect the privacy of the final model - those that don't inject differential privacy noise- result in a in higher F1-score. In contrast, "Local DP" provides the lowest F1-score due to the high amount of noise injected by each participant. For approaches that use SMC to uniquely protect the privacy of the computation, "TP-SMC no DP", "P-SMC no DP" and "*HybridAlpha* no DP", we see higher F1-scores than for those that protect the privacy of the output. This shows the price of protecting against the risk of inference on the model. Finally, we see that approaches that combine differential privacy with SMC are capable of achieving higher F1-scores while protecting the privacy of the input and output.

We now analyze these approaches from the perspective of total training time presented in Figure 3(b). As it can be seen, our proposed *HybridAlpha* has very similar training time to "FL-no-privacy". In other words, the training time added by ensuring privacy of the input and output is negligent. In contrast, we see that the slowest approach is TP-SMC even though we set up t to a conservative 50% of the entire number of participants in the system. This result is due to the fact that TP-SMC requires more rounds of communication per global step. The high training time makes TP-SMC suitable for models that require limited number of interactions with the aggregator during training.

Beside the efficiency in training time, we also evaluate the efficiency of network transmission by measuring the volume of encrypted parameters transmitted over the network. In Figure 4, we present the total transmitted cipertext size under different crypto-based SMC approaches for one epoch. The green bar represents initial ciphertext size of model parameters, while the spotted orange bar indicates the size of subsequent ciphertext, including multiplied cipher, and partially decrypted ciphers. We can see that *HybridAlpha* provides the lowest transmission rate because it only performs one round of communication on encrypted data without any subsequent ciphertext transmission. Also, our proposed approach has smaller ciphertext size of initial parameters compared to contrasted approaches.

5 SECURITY AND PRIVACY ANALYSIS

We analyze the security and privacy of our proposed framework from three different perspectives: security offered by MIFE scheme, privacy guarantees of the framework, and prevention for different types of inference attacks.

5.1 Security of the Cryptographic Approach

The security of MIFE is critical to *HybridAlpha*, since it is the underlying infrastructure of SMC protocol that supports secure aggregation in *HybridAlpha*. In our adoption of MIFE, we add a *public key distribution* algorithm run by the TPA as a beneficial supplement

Table 3: The impact of precision on computation time (s) of three SMC approaches.

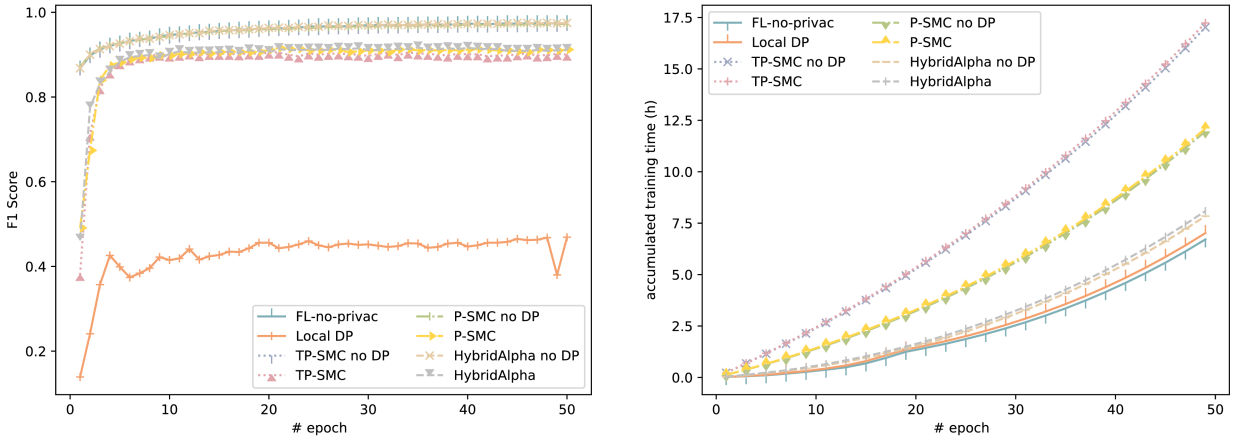
	TP-SMC ·				P-SMC			*HybridAlpha*	
precision	enc_{avg}	ct_{fuse}	$dec_{share,avg}$	$dec_{combine}$	enc_{avg}	ct_{fuse}	dec	enc_{avg}	dec
2	35.120	2.586	61.080	28.465	35.752	2.269	32.042	4.157	30.075
3	35.675	2.604	61.929	28.202	35.725	2.369	31.574	4.158	30.512
4	35.841	2.571	60.832	28.324	35.821	2.387	31.856	4.110	29.865
5	35.767	2.635	60.369	28.816	35.857	2.493	31.625	4.075	30.149
6	35.724	2.578	60.326	28.286	35.985	2.532	31.587	4.095	30.803

· The threshold parameter of Threshold-Paillier encryption system is set to half the number of participants.

Table 4: The impact of participant # on computation time (s) of three SMC approaches.

	TP-SMC ·				P-SMC			*HybridAlpha*	
participants	enc_{avg}	ct_{fuse}	$dec_{share,avg}$	$dec_{combine}$	enc_{avg}	ct_{fuse}	dec	enc_{avg}	dec
6	35.968	*1.375*	60.555	*22.184*	35.934	*1.332*	31.616	4.241	*20.246*
8	35.375	*1.843*	60.820	*23.980*	36.039	*1.859*	31.611	4.092	*25.349*
10	35.693	*2.358*	60.988	*28.401*	36.847	*2.611*	32.197	4.077	*31.782*
12	35.685	*2.759*	60.947	*34.684*	36.142	*2.959*	31.588	4.091	*36.884*
14	35.688	*3.215*	60.965	*39.838*	35.932	*3.330*	31.503	4.126	*42.683*
16	35.721	*3.694*	60.917	*46.849*	36.533	*4.481*	32.020	4.059	*47.435*
18	35.683	*4.170*	60.879	*53.441*	36.628	*5.368*	32.996	4.594	*56.519*
20	35.697	*4.764*	60.816	*97.224*	36.743	*5.765*	31.923	4.147	*59.823*

· The threshold parameter of TP-SMC is set to half the number of participants.



(a) F1 score of different approaches as epoch increases

(b) training time of different approaches as epoch increases

Figure 3: Model quality and time efficiency comparison for multiple FL approaches.

of the original MIFE scheme proposed in [2] to make it applicable to our FL framework.

Specifically, the additional algorithm is only responsible for distributing each participant's respective unique public key $\mathbf{pk}_i$. Unlike the original design of encryption algorithms where each participant encrypts the data using the master secret key **msk**, our encryption algorithm uses $\mathbf{pk}_i$ that is derived from the master keys **mpk** and **msk**. However, the core method in the encryption algorithm remains intact, and our design has no impact on other algorithms, e.g., *SKGenerate, Decrypt*. As a consequence, our adoption of MIFE does not change the security construction in the original MIFE scheme in [2]. It is then as secure as proved in [2]. To avoid redundancy, we do not present the correctness and security proofs to MIFE here, and readers can refer to [2] for more details.

5.2 Privacy of FL Framework

Our proposed FL framework can ensure the privacy of the output model and the aggregation computation.

5.2.1 Privacy of the Output Model. We provide ϵ*-differential privacy* guarantee via existing methods presented in previous works,

Table 5: Impact of threshold t for TP-SMC on computation time (s).

threshold[y]	enc_{avg}	ct_{fuse}	$dec_{share,avg}$	$dec_{combine}$
2	35.577	2.602	60.736	12.700
4	35.697	2.592	60.420	23.293
6	35.713	2.625	60.238	34.427
8	36.054	2.623	60.767	46.462
10	35.880	2.626	60.650	58.293

[y] The total participants number is set to 10 and the precision number is set to 6.

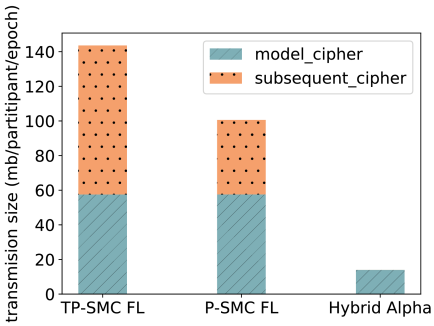


Figure 4: Total transmitted ciphertext size of different approaches for one epoch.

e.g., [1, 32, 38]. These papers have shown via theoretical analysis and experimental results that such a mechanism can achieve target privacy along with acceptable performance for the final trained model. As a consequence, our proposed framework can also achieve the same privacy guarantee for the output model as demonstrated in [32, 38].

5.2.2 Privacy of Computation. We exploit multi-input functional encryption as the underlying infrastructure for SMC protocol to compute the average of the weights of the participants' local trained models. As stated in x5.1, the MIFE scheme is secure so that any plaintext under its protection cannot be compromised by malicious attackers. The MIFE scheme also guarantees that the decryptor, the aggregator in our FL framework, can only acquire the function results, i.e., the average weight, but not the original data, i.e., weights of the participants' local models.

5.2.3 Inference Attack Prevention. Next, we consider inference attacks for two adversaries: (i) a curious aggregator, and (ii) malicious or colluding participants.

In x3.2.2, we have shown that a curious aggregator can launch an inference attack targeting a specific participant by manipulating the *weighted vector* w_p and subsequently requesting the function private key. To prevent such inference attacks, we add an additional module in TPA to filter requests for weighted vectors that are maliciously defined to isolate the reply of a single participant. Algorithm 2 verifies that at least t replies are used for aggregation, because there are at least $t > \frac{n \cdot 2}{} + 1$ non-colluding parties; even if the aggregator colludes with dishonest participants he cannot isolate the reply of a target participant.

Even if an adversary manages to collect other participants' encrypted data for a possible brute-force attack, this attack is not successful. In particular, suppose that there exits a malicious participant P_i with its own public key $\mathbf{pk}_{P_i}$, collected encrypted data $c_j = \mathrm{Enc}_{\mathbf{pk}_{P_{j,j,i}}}(m_j)$ from P_j, and its own original data set S. Here m_j is the corresponding plaintext of c_j, and m_j and any $m' \in S$ belong to a same integer group. The semantic security of the underlying MIFE scheme in our SMC protocol ensures that the adversary P_j does not have a non-negligible advantage to infer the original data m_j compared to the random guess. Furthermore, as we assume the existence of at least t honest participants where each participant does not share the same public key for encryption, the colluding participants cannot infer/identify private information using the output of the aggregator with their local models.

Note that based on the threat model defined in x3.1, we do not consider the DDoS attack on the aggregator where a malicious aggregator or a outside attacker will interrupt the network or replace a valid update from an honest participant.

6 RELATED WORK

Federated learning was proposed in [25, 28] to jointly learn a global model without sharing their data. Although this provides a basic privacy guarantee because privacy-sensitive data is not transmitted, it still suffers from inference attacks that use the final model or the model updates exchanged during the FL training to infer private information; examples of such attacks include [19, 29, 30, 37].

To thwart inference attacks, previous work has proposed to add differential privacy during the learning process e.g., [1]. However, directly applying such approaches in the FL setting results in poor model performance because each party trains its own model. For this reason, new approaches tailored to FL have been proposed.

Table 1 shows the threat model of existing approaches and the privacy guarantees they provide. Solutions proposed in [31, 36] rely on trusted aggregators and honest participants. PATE [31] proposes a "teacher-student" architecture where teachers train their models over their local datasets. Subsequently, a fully trusted aggregator is used to build a collective model. All these approaches differ from *HybridAlpha* in that they rely on a trusted aggregator.

A number of approaches propose to use SMC [6, 34, 38]. Among them, different SMC protocols are used to aggregate the global model (see Table 1). PySyft [34] employs the SPDZ protocol [12, 13]. Truex et al. [38] uses a threshold-based partially additive homomorphic encryption [11]. Bonawitz et al. [6] makes use of secret sharing that enables authenticated encryption. These approaches [6, 34, 38] also prevent or can be extended to prevent inference attacks on the final model by adding differential privacy. The main advantage *HybridAlpha* has over these approaches is the communication efficiency (see Table 1). While existing approaches need more than one round of communication between the aggregator and participants (excluding the key setup phase), *HybridAlpha* only incurs a single round. Hence *HybridAlpha* can be used to train machine learning models faster as demonstrated in the experimental section.

Most of the existing federated learning frameworks only work on the scenario of horizontally partitioned data. To tackle the issues and challenges in the case of vertically partitioned data, several methods are proposed in [9, 14, 22], which focus on entity resolution and simple machine learning models, like logistic regression. Such

vertically partitioned data cases beyond the research scope in this paper, and will be deferred to future work.

7 CONCLUSION

Federated learning promises to address privacy concerns and regulatory compliance such as for GDPR [39] and HIPAA [3]. However, extant approaches addressing privacy concerns in federated learning provide strong privacy guarantees and good model performance at the cost of higher training time and high network transmission. We propose *HybridAlpha*, a novel federated learning framework to address these issues. Our theoretical and experimental analysis shows that, compared to existing techniques, on average, *HybridAlpha* can reduce the training time by 68% and data transfer volume by 92% without sacrificing privacy guarantees or model performance. Using *HybridAlpha*, federated learning with strong privacy guarantees can be applied to use cases sensitive to training time and communication overhead, in particular for models with a large number of parameters.

REFERENCES

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

[2] Michel Abdalla, Dario Catalano, Dario Fiore, Romain Gay, and Bogdan Ursu. 2018. Multi-input functional encryption for inner products: function-hiding realizations and constructions without pairings. In *Annual International Cryptology Conference*. Springer, Springer, 597–627.

[3] Accountability Act. 1996. Health insurance portability and accountability act of 1996. *Public law* 104 (1996), 191.

[4] Joseph A. Akinyele, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, and Aviel D. Rubin. 2013. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering* 3, 2 (2013), 111–128. https://doi.org/10.1007/s13389-013-0057-3

[5] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, et al. 2009. Secure multiparty computation goes live. In *International Conference on Financial Cryptography and Data Security*. Springer, 325–343.

[6] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, ACM, 1175–1191.

[7] Dan Boneh and Matt Franklin. 2001. Identity-based encryption from the Weil pairing. In *Annual international cryptology conference*. Springer, 213–229.

[8] Dan Boneh, Amit Sahai, and Brent Waters. 2011. Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference*. Springer, Springer, 253–273.

[9] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, and Qiang Yang. 2019. SecureBoost: A Lossless Federated Learning Framework. *arXiv preprint arXiv:1901.08755* (2019).

[10] Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. 2015. *Secure multiparty computation*. Cambridge University Press.

[11] Ivan Damgård and Mads Jurik. 2001. A generalisation, a simpli. cation and some applications of paillier's probabilistic public-key system. In *International Workshop on Public Key Cryptography*. Springer, Springer, 119–136.

[12] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. 2013. Practical covertly secure MPC for dishonest majority–or: breaking the SPDZ limits. In *European Symposium on Research in Computer Security*. Springer, 1–18.

[13] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. 2012. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*. Springer, 643–662.

[14] Mentari Djatmiko, Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Maximilian Ott, Giorgio Patrini, Guillaume Smith, Brian Thorne, and Dongyao Wu. 2017. Privacy-preserving entity resolution and logistic regression on encrypted data. *Private and Secure Machine Learning (PSML)* (2017).

[15] Cynthia Dwork and Jing Lei. 2009. Differential privacy and robust statistics.. In *STOC*, Vol. 9. ACM, 371–380.

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

[17] Cynthia Dwork, Guy N Rothblum, and Salil Vadhan. 2010. Boosting and differential privacy. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*. IEEE, IEEE, 51–60.

[18] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Rafols, and Jorge Villar. 2017. An algebraic framework for Diffie–Hellman assumptions. *Journal of Cryptology* 30, 1 (2017), 242–288.

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

[20] Shafi Goldwasser, S Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. 2014. Multi-input functional encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, Springer, 578–602.

[21] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. 2006. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*. Acm, 89–98.

[22] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. 2017. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677* (2017).

[23] Michael I Jordan and Tom M Mitchell. 2015. Machine learning: Trends, perspectives, and prospects. *Science* 349, 6245 (2015), 255–260.

[24] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. 2014. Extremal mechanisms for local differential privacy. In *Advances in neural information processing systems*. 2879–2887.

[25] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).

[26] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436.

[27] Yann LeCun, Corinna Cortes, and Burges Christopher J.C. 2010. MNIST handwritten digit database. (2010). http://yann.lecun.com/exdb/mnist/

[28] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. 2016. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629* (2016).

[29] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2018. Machine learning with membership privacy using adversarial regularization. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, ACM, 634–646.

[30] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2019. Comprehensive Privacy Analysis of Deep Learning: Stand-alone and Federated Learning under Passive and Active White-box Inference Attacks. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, ACM.

[31] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Úlfar Erlingsson. 2018. Scalable private learning with PATE, In Proceedings of the 2018 Sixth International Conference on Learning Representations. *arXiv preprint arXiv:1802.08908*.

[32] Martin Pettai and Peeter Laud. 2015. Combining differential privacy and secure multiparty computation. In *Proceedings of the 31st Annual Computer Security Applications Conference*. ACM, ACM, 421–430.

[33] Zhan Qin, Yin Yang, Ting Yu, Issa Khalil, Xiaokui Xiao, and Kui Ren. 2016. Heavy hitter estimation over set-valued data with local differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 192–203.

[34] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. 2018. A generic framework for privacy preserving deep learning. In *Proceedings of Privacy Preserving Machine Learning Workshop with NeurIPS 2018*.

[35] Daniel Shanks. 1971. Class number, a theory of factorization, and genera. In *Proc. of Symp. Math. Soc., 1971*, Vol. 20. 41–440.

[36] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. ACM, ACM, 1310–1321.

[37] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 3–18.

[38] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, and Rui Zhang. 2018. A Hybrid Approach to Privacy-Preserving Federated Learning. *arXiv preprint arXiv:1812.03224* (2018).

[39] Paul Voigt and Axel von dem Bussche. 2017. *The EU General Data Protection Regulation (GDPR): A Practical Guide* (1st ed.). Springer Publishing Company, Incorporated.