Skefl: Single-Key Homomorphic Encryption for Secure Federated Learning

Dongfang Zhao
University of Washington
dzhao@uw.edu

Abstract

Homomorphic encryption (HE) is widely adopted in untrusted environments such as federated learning. A notable limitation of conventional single-key HE schemes is the stringent security assumption regarding collusion between the parameter server and participating clients: Adversary clients are assumed not to collude with the server, as otherwise, the parameter could transmit the ciphertext of one client C_0 to another client C_1, who shares the same private key and could recover the local model of C_0. One plausible solution to alleviate this strong assumption is multi-key HE schemes, which, unfortunately, prove impractically slow in production systems. In this work, we propose a new protocol that achieves the balance between security and performance: We extend single-key HE schemes with efficient secret sharing, ensuring that collusion between the parameter server and any compromised clients cannot reveal any local model. We term this protocol Skefl: Single-key homomorphic encryption for secure federated learning. The key idea behind Skefl is the secret-sharing of homomorphic *ciphertexts* generated by multiple clients using the same pair of secret and public keys. We will substantiate the security claims of the proposed protocol using the well-known simulation framework in cryptography. Additionally, we will report on the practical performance of the Skefl protocol.

1 Introduction

Background The application of homomorphic encryption (HE) [Gentry, 2009] to federated learning (FL) [McMahan *et al.*, 2017] has garnered significant research attention. HE enables the parameter server to aggregate encrypted local models without gaining knowledge of model weights, thereby preserving the privacy of sensitive training data [Zhu *et al.*, 2019]. Among existing HE schemes [Fan and Vercauteren, 2012; Brakerski *et al.*, 2012; Cheon *et al.*, 2017], CKKS [Cheon *et al.*, 2017] is considered the most promising, supporting approximate encryption on floating-point numbers. Furthermore, CKKS facilitates the encryption of vectors, a highly desirable feature for machine learning work-

loads. Notable libraries for CKKS and other HE schemes include HElib [HElib, Accessed 2022], SEAL [SEAL, 2021], and OpenFHE [Badawi *et al.*, 2022].

Motivation The utilization of the conventional Homomorphic Encryption (HE) scheme in Federated Learning (FL) presents challenges in both security and performance.

- **Security:** The conventional HE scheme issues a pair of private and public/evaluation keys, enabling untrusted parties to use the public/evaluation key for algebraic operations on ciphertexts, while data owners use the private key to recover the algebraic results. In FL, however, this necessitates that all clients share the same private key. If the parameter server (PS) and any semi-honest client collude, the local model's weights can be compromised: the PS provides the ciphertext, and the semi-honest client provides the private key.

- **Performance:** One plausible solution to the aforementioned security issue is to adopt multi-key HE schemes [López-Alt *et al.*, 2012; Chen *et al.*, 2019; Kim *et al.*, 2023]. The concept of multi-key HE (MKHE) involves every client generating a distinct pair of private and public keys, such that the algebraic operations on ciphertexts incorporate n public/evaluation keys, where n denotes the total number of parties. Consequently, the ciphertext result can only be decrypted with all the n private keys. Thus, collusion between the parameter server and a subset of compromised clients (i.e., not all the n clients are compromised, in which case the system cannot be secure anyway) cannot recover the original model weights. However, the most well-known MKHE scheme [Kim *et al.*, 2023] incurs computational costs with a linear growth in the number of clients, exacerbating the performance challenge of (single-key) HE [T. Tawose *et al.*, 2023] and rendering it impractical for federated learning, where the number of clients could reach tens of millions on mobile devices [McMahan *et al.*, 2017].

1.1 Proposed Approach

In this work, we introduce a novel protocol designed to achieve a balance between security and performance. We extend single-key homomorphic encryption (HE) schemes with efficient secret sharing to prevent collusion between

the parameter server and compromised clients from revealing any local model. We refer to this innovative protocol as Skefl: Single-key homomorphic encryption for secure federated learning. The core concept of Skefl lies in the secret-sharing of homomorphic *ciphertexts* generated by multiple clients using the same pair of secret and public keys. We term this secret-sharing of ciphertext as *asymmetric threshold secret sharing* (ATSS), signifying that the parameter server and compromised clients cannot recover the local model without the ciphertext share of the client who owns it.

We emphasize key distinctions between Skefl and existing protocols for applying HE schemes in FL systems:

- Skefl employs secret sharing of ciphertext messages, where the homomorphic aggregation of ciphertext shares equals the original encryption of a local model. This is distinct from the secret sharing of plaintext models, where reconstruction can be trivially achieved through arithmetic summation.

- Skefl necessitates homomorphic encryption of (shares of) local models, as the parameter server could otherwise unilaterally recover the global model. That is, a straightforward secret-sharing scheme of plaintext models is insufficient.

1.2 Contributions

In summary, this paper's technical contributions include:

- We present a new protocol, Skefl, designed for the efficient and secure application of homomorphic encryption in federated learning systems.

- We establish the provable security of Skefl using the classical simulation framework in cryptography.

- We implement Skefl within a practical federated learning framework tailored for collaborative scientific computing, presenting promising results across various benchmarks.

2 Preliminaries and Related Work

2.1 Secure Aggregation and Poisoning Attacks

Secure aggregation algorithms often assume a centralized parameter server for effective collaboration. In a study by [Yin *et al.*, 2018], robust distributed gradient descent algorithms rooted in median and trimmed mean operations were analyzed. Additionally, [Li *et al.*, 2019] introduced a class of robust stochastic sub-gradient methods for distributed learning from heterogeneous datasets while addressing an unknown number of adversarial workers, with a master machine assumption. These algorithms typically operate under the *semi-honest* (or *honest-but-curious*) attack model, wherein participants adhere to the protocol but may passively probe, store, or analyze data. The semi-honest model is a prevalent threat assumption in production systems [Zhang *et al.*, 2020].

Beyond the semi-honest model, another crucial perspective assumes *malicious* participants who could deviate from the protocol and even manipulate data. For instance, the deliberate introduction of compromised samples to the training dataset constitutes a *data poisoning* attack [Alfeld *et al.*, 2016]. Similarly, a malicious client might upload an arbitrary model, referred to as a *model poisoning* attack [Li *et al.*, 2020]. Numerous well-known attacks fall within this category, including backdoor attacks [Yang *et al.*, 2019; Bhagoji *et al.*, 2019; Bagdasaryan *et al.*, 2020; Xie *et al.*, 2020]. The security community has proposed diverse solutions to counter these poisoning attacks. In [Fang *et al.*, 2020], methods to poison clients' local models and corresponding defense strategies were discussed. Recent research along this trajectory includes [Data and Diggavi, 2021; Yang and Li, 2021; Sun *et al.*, 2021; Shejwalkar and Houmansadr, 2021], further advancing the understanding and mitigation of these security threats.

2.2 Homomorphic Encryption and Secret Sharing

An encryption function $f()$ is deemed *additive homomorphic* when the equation $f^{-1}(f(a) \oplus f(b)) = a + b$ holds, where $f^{-1}()$ denotes decryption and $\oplus$ signifies the binary operation within the range of $f()$. A parallel property exists for the multiplicative operation. When a function supports both additive and multiplicative homomorphism, it is termed *fully homomorphic encryption* (FHE) [Gentry, 2009]. While diverse FHE schemes like BFV [Fan and Vercauteren, 2012] and CKKS [Cheon *et al.*, 2017] have been proposed, their broader adoption in HEFL is hindered by substantial computational overhead. Instead, HE schemes supporting only additive (or multiplicative) homomorphism, such as Paillier encryption [Paillier, 1999], have found efficiency in PPML systems [Savvides *et al.*, 2020], including adoption in works like [Hardy *et al.*, 2017; Zhang *et al.*, 2020].

Conversely, information-theoretic techniques like threshold secret sharing (TSS) [Shamir, 1979; Rabin and Ben-Or, 1989] offer an alternative to cryptographic methods. TSS protects data confidentiality by fragmenting a message into n trunks, each assigned to one of n participants. A (t, n) TSS protocol ensures that t or more chunks suffice to reconstruct the original message, while t or fewer chunks reveal no original message information. Despite various implementations of (t, s) TSS protocols, like the Lagrange interpolating polynomial and the Chinese Remainder Theorem (CRT), the core concept remains consistent—sufficient participants collectively and uniquely solve a system of equations. TSS serves as a fundamental element in secure multiparty computation (MPC), facilitating collaborative computation of functions by a group of participants, such as aggregating local models in PPML and FL. MPC finds application in multiple PPML frameworks like DeepSecure [Rouhani *et al.*, 2018], SecureML [Mohassel and Zhang, 2017], and ABY [Demmler *et al.*, 2015].

3 Methodology

3.1 ATSS: Asymmetric Threshold Secret Sharing

Notations

Let $f > 0$ denote the maximal number of semi-honest clients in an FL system, in which there are a single semi-honest server $\mathcal{S}$ and a set of n clients $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$. Therefore, there are at least $(n - f)$ honest clients. We assume that the majority of clients are honest, implying that $2f < n$. If

the context is clear, we will use $2f + 1 = n$ to replace the above inequality; this is just to simplify the analysis a bit and is not a technical requirement. This also implies that $\mathcal{S}$ can collude with up to f clients. We use $\mathbf{W}_i$ to denote the local model (weights) trained at client C_i. In production systems, a model usually comprises layers of weights and biases; in our discussion, we simply the application-specific implementation by a real-valued vector $\mathbf{W}$. Since the primitive somehow decompose $\mathbf{W}_i$ into a list of sub-models, we use superscripts to denote the sub-models: $\mathbf{W}_i^j$ denotes the j-th sub-model, or trunk, of $\mathbf{W}_i$, $1 \leq j \leq i$. Obviously, $\mathbf{W}_i = \mathbf{Rec}_{j=1}^i(\mathbf{W}_i^j)$, where $\mathbf{Rec}()$ denotes the reconstruction function over sub-models to reconstruct the original model. Without loss of generality, we can think of C_j stores the j-th sub-model of the original model trained by C_i. By convention, we use $d \xleftarrow{\$} U$ to denote that an element d is randomly selected from set U. We use $\mathbb{Z}^+$ to denote the set of positive integers.

Decomposition
Each $\mathbf{W}_i$ is decomposed into $(f + 1)$ sub-models. We choose $(f + 1)$ to reflect the balance between security and performance. If the number of sub-models is less than $(f + 1)$, it is then possible for f clients to jointly recover the ciphertext of $\mathbf{W}_j$. On the other hand, the overall number of chunks being transmitted in the entire FL system, and thus the network overhead thereof, is proportional to the number of sub-models; therefore, we want to keep the number of sub-models as small as possible. As a result, $(f + 1)$ is the minimal number that guarantees the confidentiality of the primitive.

The indices of clients (i.e., j's) to receive the sub-models from $\mathbf{W}_i$ are chosen randomly except for C_i itself. That is, C_i will always hold a chunk and there are additional f clients holding the remaining f chunks. As a result, there are $\binom{n}{f}$ possibilities for the set of clients assigned to hold the chunks of a specific local model. In contrast with the conventional threshold secret-sharing (TSS) scheme, client C_i keeps not only $\mathbf{W}_i^k$, $k \in J$ where J denotes the set of client indices j's, but also the intact $\mathbf{W}_i$. Therefore, C_i has an asymmetric role in the above secret-sharing scheme: C_i could re-decompose $\mathbf{W}_i$ by canceling out the previous trunks and re-broadcasting the newly split chunks to a different set of j's. We call the above primitive *asymmetric threshold secret-sharing* (ATSS). The point of keeping the original $\mathbf{W}_i$ will be further elaborated on later in the security analysis.

ATSS decomposes $\mathbf{W}_i$ using an additive secret-sharing scheme into a set of shares stored locally on the current client. Unsurprisingly, the share assigned to C_i itself is calculated as follows:

$$enc(w^i) = \left(\bigoplus_{j \neq i} enc(w)^j \right) \oplus enc(w),$$

where enc denotes the homomorphic encryption, $\oplus$ denotes the homomorphic addition operation between two encrypted models, and $\bigoplus$ denotes the consecutive $\oplus$'s among a set of ciphertext models.

The complexity of ATSS decomposition is as follows. Let m denote the cardinality of tensor W_i in the FL model. Each

client sends out $m \cdot f$ messages to the system. Therefore, in each FL training round, there are a total of $\mathcal{O}(nmf)$ messages. If we assume $2f + 1 = n$, then the overall message complexity of each FL round is $\mathcal{O}(mn^2)$. The overall computational complexity is also $\mathcal{O}(mn^2)$: there are n clients, each of which computes the $\oplus$ operation f times (i.e., $\mathcal{O}(n)$) for each of m elements.

Reconstruction
The reconstruction from the chunks is straightforward because $\oplus$ is symmetric. To reconstruct $\mathbf{W}_i$, ATSS computes each of its elements w as follows:

$$enc(w) = \bigoplus_{j \in J} enc(w)^j,$$

where j denotes the j-th chunk of $\mathbf{W}_i$ and J is the set of chunk indices. The requester can convert the bit string back to a float number as an element of the original vector. Consequently, the aggregation function $\mathbf{Rec}()$ can be defined as the extension of the above element-wise operation.

The complexity of ATSS aggregation is similar to decomposition. Both the time and message complexity are $\mathcal{O}(mn^2)$ because all the operations are exactly the reverse of decomposition.

Verification
For clients or server $\mathcal{S}$ who wants to verify the validity of an encrypted $enc(W_i)$ generated by client C_i, C_i publishes the hash value of W_i, $H(enc(W_i))$, where $H()$ denotes a cryptographic hash function, such as SHA256. Please note that the verification is for the completeness of the ATSS primitive and unnecessarily involved in the FL workloads.

The message complexity of verifying a single model is $\mathcal{O}(n)$ because the requester would need to collect f chunks from other clients with f messages. Since we assume $2f + 1 = n$, it implies that the message complexity is linear to the number of clients. The time complexity of verifying a single model, however, is constant $\mathcal{O}(1)$ because the requester only needs to compare the published hash value of the model with the locally reconstructed one.

Primitives
We conclude this section with a list of formally defined ATSS primitives as follows.

- **ATSS.Split**() is a function:

$$\{0, 1\}^\ell \rightarrow \left(\{0, 1\}^\ell \right)^{f+1},$$

where the input *in* is a string of length l (in bits) and the output *out* is $(f + 1)$ l-bit strings whose XOR equals the input.

- **ATSS.Merge**() is a function:

$$\left(\{0, 1\}^\ell \right)^{f+1} \rightarrow \{0, 1\}^\ell,$$

where the input *in* is a set of l-bit strings and the output *out* is a single l-bit string. Note that **ATSS.Split**(x) = $y \implies$ **ATSS.Merge**(y) = x, and the converse does not hold in general.

- **ATSS.Verify**() is a function:

$$\{0,1\}^{\ell} \times \mathbb{Z}^{+} \rightarrow \{0,1\},$$

where the first input is the encryption of a model $enc(W_i)$, the second input is the client index verifying the encrypted model, and the output indicates success (1) or failure (0).

3.2 Secure Aggregation with Homomorphic Secret-Sharing Ciphertext

This section depicts the protocol adopting the ATSS primitive into a single-key homomorphic encryption scheme for federated learning, namely Skefl. A number of building blocks of Skefl are built upon the ATSS primitives. We start by the extended primitive of **ATSS.Split**().

Weighted Sub-models for Non-IID Data

While the vanilla **ATSS.Split**() can decompose a single vector, it has to be extended to take care of the data heterogeneity in federated learning, also known as not independent and identically distributed (non-IID). For instance, in the original aggregation algorithm called **FedAvg**(), the global model is calculated as the weighted average of all local models:

$$W = \textbf{FedAvg}(1, n, W_i) \overset{\text{def}}{=} \sum_{i=1}^{n} \frac{N_i}{N} W_i,$$

where N denotes the total number of data samples, N_i denotes the number of data samples on client C_i, and n denotes the number of clients.

The extended decomposition algorithm is presented in Alg. 1. Line 2 calls the **ATSS.Split**() primitive to generate a list of f random sub-models and one calculated sub-model. It does not matter which one of the $(f + 1)$ is the calculated one because the remainder of the algorithm will assume the first f sub-models will be adjusted and sent to f distinct clients than C_i—the host where the algorithm is being executed. Line 5 modifies each of the f sub-models by calling the aggregation function on the sub-model itself. This can be trivially done by setting the first two arguments as identical integers, such as is. The effect of doing so is to adjust the weight of each sub-model; without this step, the global aggregation with Skefl would be inconsistent with the vanilla **Aggr**() function. Lines 6–8 pick an unused client C_d and send the modified sub-model to it. Finally, client C_i calculates its own sub-model that will be sent to the aggregator.

The complexity of Alg. 1 is as follows. Lines 4–9 take $\mathcal{O}(n)$ iterations, each of which incurs the aggregation that costs another $\mathcal{O}(n)$. Line 2 itself costs $\mathcal{O}(mn^2)$. Therefore, the overall time complexity of Alg. 1 is $\mathcal{O}(mn^3)$ when **ATSS.Split**() is used as a black-box function. However, if **ATSS.Split**() is open-source then the key modifications in Lines 5–7 can be integrated into the internals of **ATSS.Split**(), which would make Alg. 1 result in the same asymptotic complexity of **ATSS.Split**(), i.e., $\mathcal{O}(mn^2)$.

Garbled Homomorphism of Local Models

After each client C_i runs **Skefl_Dist()**, a local aggregation is performed to strengthen the confidentiality of the original model W_i. The goal of doing so is to garble the sub-models

---

Algorithm 1: Skefl Distribute **Skefl_Dist**()

Input: A local model W_i trained by C_i; The aggregation algorithm **Aggr**$(1, n, W_i)$ taken by the FL system (e.g., **FedAvg**); The number of semi-honest clients f; The total number of clients n; The bit-string length of data item l;

Output: A set of sub-models $enc(W_i^j)$, $1 \le j \le f + 1$, satisfying **Aggr**$(enc(W_i))$ = $enc(W)$;

1 **for** $i = 1; i \le n; i + +$ **do**
2    $(enc(W_i^1), \ldots, enc(W_i^{f+1})) = $ **ATSS.Split**(W_i)
3    $U = \{1, \ldots, n\} \setminus \{i\}$
4   **for** $j = 1; j \le f; j + +$ **do**
5      $enc(W_i^j) = $ **Aggr**$(j, j, enc(W_i^j))$
6      $d \xleftarrow{\$} U$
7   C_i sends $enc(W_i^j)$ to C_d
8      $U = U \setminus \{d\}$
9   **end**
10    $enc(W_i^{f+1}) = $
     $\left(\bigoplus_{j=1}^{f} enc(W_i^j) \right) \oplus \textbf{Aggr}(1, 1, enc(W_i))$
11 **end**

---

generated by **Skefl_Dist**() such that semi-honest clients cannot collude with the semi-honest server to possibly identify the sub-models. Specifically, each client C_i applies the aggregation function **Aggr**() over all of the sub-models it receives (including the one sent by itself). Formally, we defined the garbled local model $\widehat{W}_i$ as

$$\widehat{W}_i \overset{\text{def}}{=} \textbf{Aggr}(1, n, W_k^*),$$

where we use * to denote an arbitrary index of sub-model from C_k. If there are no sub-models sent from C_k to C_i, we set the default value as the zero vector $W_k^* = 0$.

Client C_i then applies the homomorphic encryption algorithm **Enc**() to $\widehat{W}_i$ to even protect the garbled (summation of) sub-models. We define a function for the local garbling aggregation as a function:

$$\textbf{Skefl\_Garble}(i) \overset{\text{def}}{=} \textbf{enc}(\widehat{W}_i). \quad (1)$$

On average, each client C_i would receive n sub-models and the local aggregation involves up to nm operations. Therefore, the time complexity of **Skefl_Garble**() is $\mathcal{O}(mn)$.

Secure Aggregation with Skefl

The server $\mathcal{S}$ receives from each client C_i the encryption of a garbled model $\textbf{enc}(\widehat{W}_i)$. Because **enc**() is homomorphic, the aggregation function **Aggr**() can be converted into a series of algebraic operations over the ciphertexts. Recall that a homomorphic function implies that:

$$\textbf{dec}_{sk}(\textbf{enc}_{pk}(x) \boxplus \textbf{enc}_{pk}(y)) = x + y,$$

where **dec**() is the reverse function of homomorphism **enc**(). Therefore, if we apply the corresponding aggregation over

the ciphertexts, the result would be decrypted into the aggregation of the plaintext, which is exactly what the clients compute with the private/secret key sk. If we extend the homomorphic addition $\boxplus$ to a set of inputs, then $\boxed{+}_{i=1}^{n} \mathbf{enc}_{pk}(\widehat{\boldsymbol{W}}_i)$ denotes the consecutive summation of the set of cryptographically garbled models from all clients.

Let $\mathbf{Skefl\_Aggr}()$ denote the above equation:

$$\mathbf{Skefl\_Aggr}(\{\boldsymbol{W}_i\}) \overset{\text{def}}{=} \boxed{+}_{i=1}^{n} \mathbf{enc}_{pk}(\widehat{\boldsymbol{W}}_i),$$

where $1 \leq i \leq n$. We claim that

$$\mathbf{dec}_{sk}(\mathbf{Skefl\_Aggr}(\{\boldsymbol{W}_i\})) = \mathbf{Aggr}(1, n, \boldsymbol{W}_i). \quad (2)$$

We skip the formal proof and only highlight the key step for verifying the above equation (omitting the pk and sk subscripts):

$$\mathbf{dec}\left(\boxed{+}_{i=1}^{n} \mathbf{enc}\left(\widehat{\boldsymbol{W}}_i\right)\right) = \mathbf{dec}\left(\mathbf{enc}\left(\mathbf{Aggr}(1, n, \boldsymbol{W}_i)\right)\right),$$

which essentially states that we can push the aggregation inside the ciphertext due to the homomorphic property.

The time complexity of $\mathbf{Skefl\_Aggr}()$ is straightforward. The naive way of applying $\boxed{+}$ is obviously $\mathcal{O}(n)$. However, if we can leverage multiple cores to parallelize the summation through a binary tree, then the time complexity can be reduced to $\mathcal{O}(\log n)$ in practical systems.

3.3 Security Analysis

Threat Model
We assume all adversaries are *efficient*, meaning that they run polynomial algorithms in the bit length of the input. Roughly speaking, this implies that the adversaries do not have access to unlimited computational power. We also assume all adversaries are *semi-honest*, which is also known as *honest-but-curious*. This entails two assumptions:

- All clients C_i's and the server $\mathcal{S}$ follow the aggregation algorithm $\mathbf{Aggr}()$ without tempering with the models $\boldsymbol{W}_i$'s;
- Some (up to f) clients and the server $\mathcal{S}$ may probe and store data during the execution of the FL workloads.

Specifically, server $\mathcal{S}$ could collude with a client C_j to hopefully recover a local model trained by another client $C_i, j \neq i$. We will show that Skefl can resist such attacks in the next section.

We assume the building block of the homomorphic encryption scheme is *computational* secure under the *chosen-plaintext attack* (CPA). In practice, there are multiple CPA-secure schemes, such as Paillier [Paillier, 1999] and Symmetria [Savvides *et al.*, 2020]. There are two ways to define the *indistinguishability* under CPA (IND-CPA) when an adversary $\mathcal{A}$ obtains a polynomial number of plaintext-ciphertext pairs $(m, \mathbf{enc}(m))$'s: (i) $\mathcal{A}$ cannot distinguish a specific ciphertext $\mathbf{enc}(m)$ from a random string; and (ii) $\mathcal{A}$ cannot distinguish $\mathbf{enc}(m_1)$ from $\mathbf{enc}(m_2), m_1 \neq m_2$. The former definition implies the latter one, although the latter definition is

more widely adopted. In this work, we will take the second definition. That is, we assume the probability for $\mathcal{A}$ to guess the right message from $\mathbf{enc}(m_1)$ and $\mathbf{enc}(m_2)$ is *negligibly* higher than $\frac{1}{2}$ after learning about a polynomial number of $(m, \mathbf{enc}(m))$ pairs. By *negligibly*, we mean the advantage of $\mathcal{A}$ is a function that decreases faster than the inverse of *any* polynomial functions.

IND-CPA of Skefl
Let l denote the bit-string length of the model values. Without loss of generality, we assume two clients C_1 and C_2 upload two (encrypted and garbled) models $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ to the aggregator. Let o denote the output by the adversary $\mathcal{A}$, indicating $\mathcal{A}$'s guess about whether the original plaintext model is from either C_1 or C_2. If by assumption $\mathbf{enc}()$ is IND-CPA, then the following holds:

$$\begin{cases} Pr\left[o = 1 | \mathbf{enc}(\boldsymbol{W}_1)\right] \leq \frac{1}{2} + \mathcal{O}(\frac{1}{2^l}) \\ Pr\left[o = 2 | \mathbf{enc}(\boldsymbol{W}_2)\right] \leq \frac{1}{2} + \mathcal{O}(\frac{1}{2^l}) \\ Pr[\mathbf{enc}(\boldsymbol{W}_1)] = Pr[\mathbf{enc}(\boldsymbol{W}_2)] = \frac{1}{2} \end{cases} \quad (3)$$

where $Pr[\ ]$ denotes the probability (under the condition of observing a ciphertext). The last term $\mathcal{O}(\frac{1}{2^l})$ is often called a *negligible function* in the cryptographic literature.

Our goal is to show that the encrypted garbling of $\boldsymbol{W}_i$'s are IND-CPA. That is, the best $\mathcal{A}$ can do is the following:

$$\begin{cases} Pr\left[o = 1 | \mathbf{Skefl\_Garble}(1)\right] \leq \frac{1}{2} + \mathcal{O}(\frac{1}{2^l}) \\ Pr\left[o = 2 | \mathbf{Skefl\_Garble}(2)\right] \leq \frac{1}{2} + \mathcal{O}(\frac{1}{2^l}) \\ Pr[\mathbf{Skefl\_Garble}(1)] = Pr[\mathbf{Skefl\_Garble}(2)] = \frac{1}{2} \end{cases} \quad (4)$$

after sending $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ to the semi-honest aggregator. To prove the above equation, we will use the popular *reduction* technique, which is essentially proof by contradiction.

Theorem 1 (Skefl is IND-CPA). *If $\mathbf{enc}()$ is IND-CPA, then $\mathbf{Skefl\_Garble}()$ defined in Eq. (1) is IND-CPA.*

Proof (sketch). We use the standard simulation framework to prove the IND-CPA of the proposed Skefl protocol. Suppose there is a perfectly secure oracle that can privately take in a local model for aggregation and would never collude with any clients. Without loss of generality, we will focus on the $o = 1$ case as $o = 2$ is similar. Let μ denote the negligible (asymptotic) function $\mathcal{O}(\frac{1}{2^l})$. Because

$$Pr[o = 1 | \mathbf{enc}(\boldsymbol{W}_1)] \leq \frac{1}{2} + \mu,$$

we know

$$\frac{Pr[o = 1 \wedge \mathbf{enc}(\boldsymbol{W}_1)]}{Pr[\mathbf{enc}(\boldsymbol{W}_1)]} \leq \frac{1}{2} + \mu,$$

which is equivalent to

$$Pr[o = 1 \wedge \mathbf{enc}(\boldsymbol{W}_1)] \leq \frac{1}{4} + \frac{\mu}{2}. \quad (5)$$

Now, suppose

$$Pr\left[o = 1 | \mathbf{Skefl\_Garble}(1)\right] > \frac{1}{2} + \mu,$$

which means (similar to before):

$$Pr\left[o = 1 \wedge \textbf{Skefl\_Garble}(1)\right] > \frac{1}{4} + \frac{\mu}{2}.$$

According to Eq. (1), the following holds:

$$Pr\left[o = 1 \wedge \textbf{enc}(\widehat{\boldsymbol{W}}_1)\right] > \frac{1}{4} + \frac{\mu}{2}.$$

From the perspective of $\mathcal{A}$ who outputs $o = 1$, the encryption of the original local model $\boldsymbol{W}_i$ is indistinguishable from the encryption of the garbled local model $\widehat{\boldsymbol{W}}_i$ (required by the IND-CPA property of the encryption). Therefore, the last equation implies a contradiction to Eq. (5), which completes the proof. □

4 Evaluation

4.1 Implementation

We have implemented the proposed Skefl protocol on top of FedML [He *et al.*, 2020] using the message-passing interface (MPI) [Open MPI, Accessed 2021] interface. FedML supports multiple geographically-distributed clusters under the same namespace (i.e., an *MPI_COMM_WORLD*) and yet allows each client to be assigned with a single CPU core (i.e., a *rank* in MPI) to save cost. The early release of FedML does not support a persistence layer for intermediate data; an experimental feature of blockchain-based data provenance for intermediate models is enabled in our evaluation. FedML is a pure Python implementation and invokes low-level C++ MPI calls through mpi4py [MPI4PY, Accessed 2021]. The homomorphic computation over tensors is implemented through TenSEAL [Benaissa *et al.*, 2021] and we pick the CKKS [Cheon *et al.*, 2017] scheme for our implementation because it is the only one currently supporting homomorphic encryption over float numbers.

4.2 Experimental Setup

Test Bed

We have deployed Skefl to CloudLab [Duplyakin *et al.*, 2019]. Our testbed is a cluster of 10 nodes, each of which has 40 Intel Xeon Silver 4114 cores at 2.20 GHz, 192 GB ECC DDR4-2666 Memory, and an Intel DC S3500 480 GB SSD. All nodes are connected to two networks: a 10-Gbps experiment network (Dual-port Intel X520-DA2 10 Gb NIC) and a 1-Gbps control network (Onboard Intel i350 1 Gb). We use the experiment network for evaluation. All experiments are repeated at least three times, and we report the average numbers along with the standard errors.

Federated Learning (Hyper-)Parameters

As for the setting of federated learning, we set the numbers of local and global epochs both as 10. The fraction of clients for each round of model updating is 0.1. The local batch size is set to 50. The learning rate is 0.05 and the SGD momentum is 0.8. The number of clients is equal to the number of nodes in our test bed, 10. The aggregation algorithm is the original FedAvg [McMahan *et al.*, 2017].

Local Machine Learning Models

We pick two machine learning models to train local neural networks: convolutional neural network (CNN) and multi-layered perception (MLP). The CNN model comprises two convolutional layers: the first being from input data to 10 with kernel size 5 and the second being from 10 to 20 with kernel size 5. The CNN model then applies two linear layers: $320 \rightarrow 50 \rightarrow 10$. For the MLP model, there is a 64-neuron hidden layer: $784 \rightarrow 64 \rightarrow 10$. Both models use ReLU and SoftMax as the activation functions.

Workloads

We chose four of the most widely used data sets: MNIST [MNIST Dataset, Accessed 2020], Fashion-MNIST [Xiao *et al.*, 2017], CIFAR-10 [Krizhevsky, 2009], and extended SVHN-extra [Netzer *et al.*, 2011], all of which can be downloaded from PyTorch.

4.3 Experimental Results

Overall Accuracy

In this subsection, we delve into the heart of our experimental evaluation by presenting the overall accuracy performance of the proposed Skefl protocol within the context of the Federated Machine Learning (FedML) paradigm. We engage in a rigorous comparison, pitting the accuracy performance of Skefl against two well-established baseline aggregation algorithms: FedAvg and FedHE. The culmination of these investigations is vividly depicted in Fig. 1, where the trajectory of accuracy is meticulously traced over multiple rounds of aggregation. The graphical representation unequivocally showcases the convergence trends, revealing that Skefl, FedAvg, and FedHE all exhibit rapid convergence, achieving accuracy levels that soar past the 80% threshold within a remarkably scant 10 aggregation rounds. This compelling outcome unambiguously underscores the potency of Skefl in adeptly harmonizing the twin objectives of accuracy preservation and privacy assurance during the aggregation process. The capability of Skefl to uphold accuracy at a competitive level vis-à-vis conventional methodologies stands as a compelling testament to its pragmatic utility and relevance.

ATSS Performance

Transitioning to the meticulous assessment of the Asymmetric Threshold Secret Sharing (ATSS) primitives, meticulously delineated in Fig. 2, we meticulously scrutinize the temporal dynamics of the execution times associated with the three constituent ATSS primitives: **ATSS.Split**(), **ATSS.Combine**(), and **ATSS.Aggregate**(). The observed trends in execution times delineate a discernible pattern. For datasets characterized by relative simplicity, exemplified by MNIST and FMNIST, all three ATSS primitives promptly culminate their execution in a matter of mere seconds. Nevertheless, the narrative changes when grappling with more intricate datasets, typified by CIFAR-10 and SVHN, where execution times elongate to span tens of seconds. Evidently, **ATSS.Split**() emerges as the harbinger of the longest execution durations among the ATSS primitives.
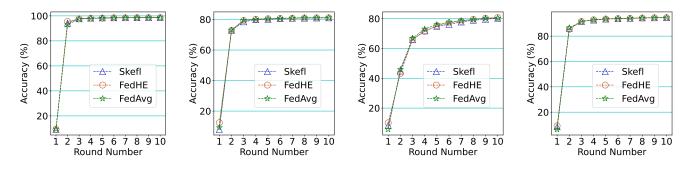
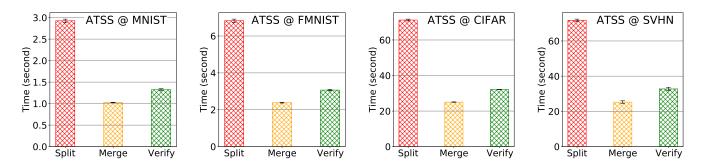Figure 1: Skefl accuracy for MNIST, FMNIST, CIFAR-10, and SVHN

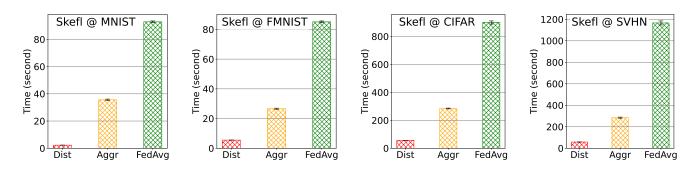Figure 2: Performance of ATSS primitives on MNIST, FMNIST, CIFAR-10, and SVHN

Figure 3: Skefl overhead of FedAvg on MNIST, FMNIST, CIFAR-10, and SVHN

Skefl Overhead

Embarking on a distinct avenue of analysis, we delve into the nuanced exploration of the overhead incurred by the Skefl protocol, encapsulated within Fig. 3. This analysis delves into the computational costs borne by the two pivotal functions embedded within the Skefl protocol: **Skefl_Dist**() and **Skefl_Aggr**(). The findings elegantly elucidate the delicate equilibrium that characterizes the trade-off between the gains in privacy bestowed by secret sharing and the attendant computational overhead. Astonishingly, **Skefl_Dist**() shines forth as a paragon of efficiency, incurring negligible overhead that essentially aligns with the streamlined efficiency of the pristine FedAvg protocol. On a divergent trajectory, **Skefl_Aggr**() exacts a relatively heftier computational toll, accounting for a range spanning from 24% to 38% of the temporal expenditure entailed by the analogous FedAvg operation.

5   Conclusion

Conventional single-key HE schemes used in federated learning assume non-collusion between the parameter server and participating clients, a vulnerability wherein an adversary client could compromise the local model of another by intercepting ciphertexts. To address this, our proposed Skefl protocol extends single-key HE schemes with efficient secret sharing, ensuring that collusion between the parameter server and any compromised clients does not reveal local models. The security of Skefl is rigorously proven using a well-established simulation framework in cryptography. Additionally, the practical performance of Skefl is reported, highlighting its effectiveness in achieving a robust balance between security and efficiency in secure federated learning scenarios.

References

[Alfeld *et al.*, 2016] Scott Alfeld, Xiaojin Zhu, and Paul Barford. Data poisoning attacks against autoregressive models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.

[Badawi *et al.*, 2022] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. Openfhe: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915, 2022. https://eprint.iacr.org/2022/915.

[Bagdasaryan *et al.*, 2020] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In Silvia Chiappa and Roberto Calandra, editors, *The 23rd International Conference on Artificial Intelligence and Statistics (AIStat)*, 2020.

[Benaissa *et al.*, 2021] Ayoub Benaissa, Bilal Retiat, Bogdan Cebere, and Alaa Eddine Belfedhal. Tenseal: A library for encrypted tensor operations using homomorphic encryption, 2021.

[Bhagoji *et al.*, 2019] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning*, 2019.

[Brakerski *et al.*, 2012] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *The 3rd Innovations in Theoretical Computer Science Conference*, 2012.

[Chen *et al.*, 2019] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, 2019.

[Cheon *et al.*, 2017] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *23rd International Conference on the Theory and Applications of Cryptology and Information Security (AsiaCrypt)*. Springer, 2017.

[Data and Diggavi, 2021] Deepesh Data and Suhas Diggavi. Byzantine-resilient high-dimensional sgd with local iterations on heterogeneous data. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2478–2488. PMLR, 18–24 Jul 2021.

[Demmler *et al.*, 2015] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015.

[Duplyakin *et al.*, 2019] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, July 2019.

[Fan and Vercauteren, 2012] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. https://eprint.iacr.org/2012/144.

[Fang *et al.*, 2020] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Local model poisoning attacks to byzantine-robust federated learning. In *Proceedings of the 29th USENIX Conference on Security Symposium*, USA, 2020. USENIX Association.

[Gentry, 2009] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing (STOC)*, 2009.

[Hardy *et al.*, 2017] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *CoRR*, abs/1711.10677, 2017.

[He *et al.*, 2020] Chaoyang He, Songze Li, Jinhyun So, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. Fedml: A research library and benchmark for federated machine learning. *Advances in Neural Information Processing Systems, Best Paper Award at Federate Learning Workshop*, 2020.

[HElib, Accessed 2022] HElib. https://github.com/homenc/HElib, Accessed 2022.

[Kim *et al.*, 2023] Taechan Kim, Hyesun Kwak, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. Asymptotically faster multi-key homomorphic encryption from homomorphic gadget decomposition. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 726–740, New York, NY, USA, 2023. Association for Computing Machinery.

[Krizhevsky, 2009] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[Li *et al.*, 2019] Liping Li, Wei Xu, Tianyi Chen, Georgios B. Giannakis, and Qing Ling. Rsa: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI)*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019.

[Li *et al.*, 2020] Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. A review of applications in federated learning. *Computers & Industrial Engineering*, 149:106854, 2020.

[López-Alt *et al.*, 2012] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '12, page 1219–1234, New York, NY, USA, 2012.

[McMahan *et al.*, 2017] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In Aarti Singh and Xiaojin (Jerry) Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AIStat)*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 2017.

[MNIST Dataset, Accessed 2020] MNIST Dataset. http://yann.lecun.com/exdb/mnist/, Accessed 2020.

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

[MPI4PY, Accessed 2021] MPI4PY. https://mpi4py.readthedocs.io/en/stable/intro.html, Accessed 2021.

[Netzer *et al.*, 2011] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[Open MPI, Accessed 2021] Open MPI. http://www.open-mpi.org/, Accessed 2021.

[Paillier, 1999] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, page 223–238, Berlin, Heidelberg, 1999. Springer-Verlag.

[Rabin and Ben-Or, 1989] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC '89, page 73–85, New York, NY, USA, 1989. Association for Computing Machinery.

[Rouhani *et al.*, 2018] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: Scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, New York, NY, USA, 2018. Association for Computing Machinery.

[Savvides *et al.*, 2020] Savvas Savvides, Darshika Khandelwal, and Patrick Eugster. Efficient confidentiality-preserving data analytics over symmetrically encrypted datasets. *Proc. VLDB Endow.*, 13(8):1290–1303, April 2020.

[SEAL, 2021] Microsoft SEAL (release 3.7). https://github.com/Microsoft/SEAL, September 2021. Microsoft Research, Redmond, WA.

[Shamir, 1979] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, nov 1979.

[Shejwalkar and Houmansadr, 2021] Virat Shejwalkar and Amir Houmansadr. Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning. In *Network and Distributed Systems Security (NDSS) Symposium 2021*, 2021.

[Sun *et al.*, 2021] Jingwei Sun, Ang Li, Louis DiValentin, Amin Hassanzadeh, Yiran Chen, and Hai Li. Fl-wbc: Enhancing robustness against model poisoning attacks in federated learning from a client perspective. *Advances in Neural Information Processing Systems*, 2021.

[T. Tawose *et al.*, 2023] Olamide T. Tawose, Jun Dai, Lei Yang, and Dongfang Zhao. Toward efficient homomorphic encryption for outsourced databases through parallel caching. *Proceedings of the ACM on Management of Data (SIGMOD)*, May 2023.

[Xiao *et al.*, 2017] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[Xie *et al.*, 2020] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *International Conference on Learning Representations*, 2020.

[Yang and Li, 2021] Yi-Rui Yang and Wu-Jun Li. Basgd: Buffered asynchronous sgd for byzantine learning. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11751–11761. PMLR, 18–24 Jul 2021.

[Yang *et al.*, 2019] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.*, 10(2), jan 2019.

[Yin *et al.*, 2018] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 5650–5659. PMLR, 10–15 Jul 2018.

[Zhang *et al.*, 2020] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. BatchCrypt: Efficient homomorphic encryption for Cross-Silo federated learning. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, July 2020.

[Zhu *et al.*, 2019] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.