SVFL: Efficient Secure Aggregation and Verification for Cross-Silo Federated Learning

Fucai Luo, Saif Al-Kuwari, and Yong Ding

Abstract—Cross-silo federated learning (FL) allows organizations to collaboratively train machine learning (ML) models by sending their local gradients to a server for aggregation, without having to disclose their data. The main security issues in FL, that is, the privacy of the gradient and the trained model, and the correctness verification of the aggregated gradient, are gaining increasing attention from industry and academia. A popular approach to protect the privacy of the gradient and the trained model is for each client to mask their own gradients using additively homomorphic encryption (HE). However, this leads to significant computation and communication overheads. On the other hand, to verify the aggregated gradient, several verifiable FL protocols that require the server to provide a verifiable aggregated gradient were proposed. However, these verifiable FL protocols perform poorly in computation and communication. In this paper, we propose SVFL, an efficient protocol for cross-silo FL, that supports both secure gradient aggregation and verification. We first replace the heavy HE operations with a simple masking technique. Then, we design an efficient verification mechanism that achieves the correctness verification of the aggregated gradient. We evaluate the performance of SVFL and show, by complexity analysis and experimental evaluations, that its computation and communication overheads remain low even on large datasets, with a negligible accuracy loss (less than 1%). Furthermore, we conduct experimental comparisons between SVFL and other existing FL protocols to show that SVFL achieves significant efficiency improvements in both computation and communication.

Index Terms—Machine learning, secure aggregation, federated learning, verifiable, privacy-preserving

1 Introduction

Federated learning (FL) [29] is a promising collaborative machine learning (ML) framework allowing models to be trained on sensitive real-world data while preserving its privacy. The main feature of FL is that the training data does not need to leave its local repositories. That is, FL enables participating entities to train the model on their data independently and in parallel, which greatly reduces data privacy risks and improves training efficiency/scalability. Not surprisingly, FL has been widely used in various applications such as disease outbreak discovery [32], intrusion detection for Internet of Things (IoT) [40], next word prediction [35], and autonomous driving [47].

In FL, each client trains a copy of a global model locally on their data and computes a local gradient vector, which is then sent to a centralized server (i.e., aggregator). The server combines these gradient vectors and obtains an aggregated gradient, which is then sent back to all clients. Upon receiving the aggregated gradient, each client updates the global model and proceeds to the next training iteration. However, this FL process raises at least the following three important privacy concerns. 1) The server can learn information about the clients' local training data by analyzing their gradient vectors. This type of attacks is often referred to as *inference attacks* [51]. 2) The server may manipulate the global model at will by providing each client with a malformed aggregated gradient. In particular, a "lazy" server may reduce the aggregation operation to save computational cost, or worse, maliciously forge an aggregated gradient. 3) The server may know the trained model. This is because the server obtains the aggregated gradient from which it can learn some information about the trained model. In some cases, clients may wish to keep the trained model private. In response to the above privacy issues, the concepts of *secure aggregation* [8], *verifiable* FL [57], and *cross-silo* FL [63] were proposed. In the following, we will elaborate on these concepts in sequence.

1.1 Secure Aggregation

Secure aggregation entails computing a multiparty sum where no client reveals its local gradient vector in the clear (even to the server). In fact, the secure aggregation problem has been a research hotspot, and it has been addressed by different approaches including secure multiparty computation (MPC) [9], partially/fully homomorphic encryption [39], [41], functional encryption [5], and double-masking [8], [10]. We refer to Related Work for details about these solutions.

- *Fucai Luo is with the Department of New Networks, Peng Cheng Laboratory, Shenzhen, Guangdong Province 518055, China. E-mail: lfucai@126.com.*
- *Saif Al-Kuwari is with the College of Science and Engineering, Hamad Bin Khalifa University, Doha 999043, Qatar. E-mail: smalkuwari@hbku.edu.qa.*
- *Yong Ding is with the Guangxi Key Laboratory of Cryptography and Information Security, School of Computer Science and Information Security, Guilin University of Electronic Technology, Guilin 541004, China. E-mail: stone_dingy@126.com.*

Manuscript received 7 June 2022; revised 7 October 2022; accepted 1 November 2022. Date of publication 4 November 2022; date of current version 5 December 2023.
This work was supported in part by the Major Key Project of PCL under Grants PCL2021A02, PCL2021A09, and PCL2022A03, in part by the National Key R & D Program of China under Grant 2020YFB1006003, in part by the National Natural Science Foundation of China under Grant 62172119, and in part by the Guangxi Natural Science Foundation under Grant 2019GXNSFGA245004.
(Corresponding author: Yong Ding.)
Digital Object Identifier no. 10.1109/TMC.2022.3219485

Based on different application scenarios, FL can be roughly divided into two categories: *cross-device* FL, where clients are a large number of mobile or edge devices (e.g., smartphones, personal computers, and IoT devices) with unreliable communication and limited computing power [60], and *cross-silo* FL, where clients are a small number of organizations (e.g., hospitals, financial companies, and research institutions) with sufficient computing resources and reliable communications [27]. From the security perspective, the main difference between cross-device FL and cross-silo FL is that the server in cross-device FL can see the trained model, while in cross-silo FL only clients can see the trained model; that is, no external party, including the server, is allowed to access to the trained model (even the training model). In addition, in the cross-silo setting, we usually do not need to consider dropouts, which are common in the cross-device setting [60].

Most of the existing research on FL is mainly addressing cross-device FL, but in recent years, cross-silo FL is receiving increasingly wide attention, especially for healthcare applications, such as medical imaging [26], diagnostic and prognostic biomarkers [50], and epidemic detection [13]. In this paper, we focus on cross-silo FL.

Recently, WeBank [4] developed an opensource cross-silo FL framework named FATE [1], which has built-in support for Pailler cryptosystem [41], a well-studied additively homomorphic encryption (HE) scheme. Subsequently, by replacing the HE in FATE with an efficient batching encryption scheme, Zhang et al. [63] proposed BatchCrypt, which substantially improves training speed and reduces the communication overhead over FATE.

1.2 Verification

Another related concern is the verifiability of the aggregated gradient. As mentioned above, the server aggregates local gradients sent by the clients and obtains an aggregated gradient, which is then sent back to each client. This raises a natural question: how to verify the correctness of the aggregated gradient? Clearly, if we cannot verify the correctness of the aggregated gradient, how can privacy issue 2) above be avoided? This led to the proposals of verifiable FL [17], [21], [57], which address the issue of correctness verification of the aggregated gradient. In verifiable FL, besides the masked gradient, each client also needs to send some auxiliary information to help the server generate a "proof" of correctness of the aggregated gradient when it is correctly computed. Then, the server returns an aggregated gradient and the "proof" to each client. Finally, each client checks whether the "proof" is correct on the aggregated gradient.

1.3 Contributions

In this paper, we propose SVFL, the first cross-silo FL protocol supporting secure aggregation and correctness verification of the aggregated gradient, with reduced communication and computation overheads compared to the previous solutions. Table 1 compares SVFL with the previous secure aggregation protocols, cross-silo FL, and verifiable FL. Technically, in the cross-silo setting, we use a simple masking technique called masking with one-time pads (**Motp**) to guarantee the privacy of the local gradient and trained model. We also adopt a secure homomorphic network coding signature scheme to realize the correctness verification of the aggregated gradient. Our contributions can be summarized as follows:

TABLE 1
Functional Comparison Between SVFL and the Previous Related Solutions

Approach	Privacy of gradient	Privacy of model	Verification[†]
Secure Aggregation [8], [10], [7] [25], [52], [15], [6], [59], [48]	✓	✗	✗
BatchCrypt [63]	✓	✓	✗
VFL [17]	✓	✓	✓
VerifyNet [57]	✓	✗	✓
VeriFL [21]	✓	✗	✓
SVFL (This work)	✓	✓	✓

[†] *Each client can verify the correctness of the aggregated gradient returned by the server.*

- In the cross-silo setting, we replace the additively HE used by BatchCrypt [63] with a simple masking technique (**Motp**) to protect the privacy of the local gradient and the trained model. Compared with BatchCrypt, SVFL accelerates the model training by about 414 to 605 times without adding any communication overhead. Furthermore, to achieve the correctness verification of the aggregated gradient, we design an efficient verification mechanism using an efficient homomorphic network coding signature scheme (**HNsig** [18]).
- We provide a comprehensive security analysis for SVFL, which demonstrates the privacy of the local gradient and the trained model as well as the verifiability (i.e., the correctness verification of the aggregated gradient).
- We provide a complexity analysis of computational and communication costs for SVFL.
- We conduct extensive experiments on convolutional neural network (CNN) with MNIST dataset and AlexNet [31] with CIFAR10 dataset. The experimental results show that SVFL is efficient in terms of both computation and communication, with a negligible accuracy loss (less than 1%). In particular, we compare SVFL with BatchCrypt [63], VFL [17], VerifyNet [57] and VeriFL [21]. The comparison results show that our SVFL has significant efficiency advantages in both computation and communication.

We remark that SVFL is also suitable for cross-device settings by using a Trusted Authority (TA) to initialize the ML model and generate the required parameters, which is commonly used in verifiable FL. We leave this as an extension to this work.

1.4 Organization

The remainder of the paper is organized as follows. We briefly outline related work in Section 2. In Section 3, we recall a homomorphic network coding signature over integers and present a simple masking technique called masking with one-time pads. We present the system architecture and the concrete construction of SVFL in Section 4, followed by the correctness and security analysis in Section 5. We evaluate the efficiency of SVFL from theoretical and experimental perspectives in Sections 6 and 7, respectively. In Section 8, we compare SVFL with some previous solutions including cross-silo FL and verifiable FL. Finally, Section 9 concludes the paper.

2 Related Work

In this section, we survey the existing secure aggregation protocols, cross-silo FL and verifiable FL.

2.1 Secure Aggregation

Secure Aggregation with Double-Masking. The first secure aggregation protocol was proposed by Bonawitz et al. [8], who used the double-masking technique (using pseudorandom values), Shamir's secret sharing [49], key agreement protocol [11], and symmetric encryption to protect the privacy of the local gradient and handle dropouts. However, their secure aggregation protocol requires at least 4 rounds of communication between each client and the server in every iteration. For resource-limited clients connected over a WAN, this communication overhead can be prohibitive.

Based on the framework proposed by Bonawitz et al. [8], Bell et al. [7] and Choi et al. [10] proposed secure aggregation protocols with polylogarithmic communication and computation overheads. Their protocols achieve better computational and communication efficiency than the secure aggregation protocol of [8]. The key idea behind [7] and [10] is to replace the complete communication graph of [8] with a sparse random graph and to use secret sharing only for a subset of clients instead of for all clients.

To address such efficiency challenge, So et al. [52] recently proposed Turbo-Aggregate, which uses a circular communication topology to reduce the computation and communication overheads of the secure aggregation protocol of [8]. Beguier et al. [6] proposed SAFER, a secure aggregation protocol between multiple servers with low computation and communication overheads. SAFER achieves low computational and communication costs by using model update compression and arithmetic sharing.

Kadhe et al. [25] proposed FastSecAgg, a secure (private) aggregation protocol based on Fast Fourier Transform multi-secret sharing. FastSecAgg is secure against adaptive adversaries where clients can adaptively be corrupted during the execution of the protocol. Similarly, Fereidooni et al. [25] proposed SAFELearn, a generic design for secure (private) aggregation. SAFELearn can be instantiated with MPC or HE and only needs 2 rounds of communication in each training iteration.

Secure Aggregation with Encryption. Homomorphic Encryption (HE) allows certain operations (e.g., addition) to be performed directly on encrypted data. Such property is exactly what is needed for secure aggregation. A number of proposals [12], [23], [34], [42], [45], [53], [58] have been made to build a secure aggregation protocol for FL using additively HE and multi-key HE.

Recently, Sav et al. [48] proposed POSEIDON, using a multiparty lattice-based homomorphic encryption scheme [38]. To improve the efficiency of POSEIDON, the authors provided a generic packing approach so that single-instruction-multiple-data (SIMD) operations can be efficiently performed on encrypted data.

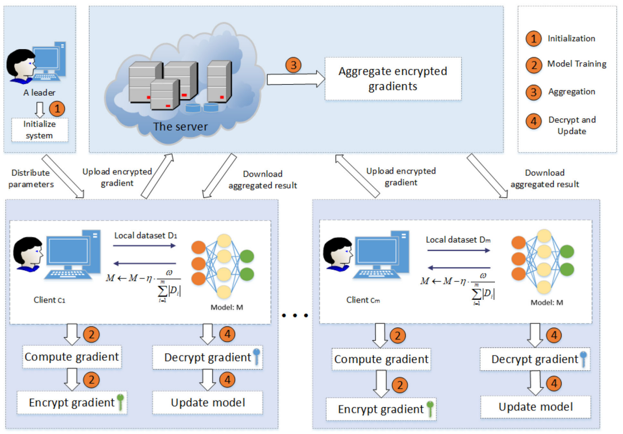


Fig. 1. The architecture of cross-silo FL.

Similarly, Xu et al. [59] proposed Hybridalpha, using a multi-input functional encryption scheme [5] and differential privacy (DP) [14]. With multi-input functional encryption, each client obtains a public key and uses it to encrypt the local gradient, while the server obtains a function key and uses it to compute the average cumulative sum of the clients' gradients.

MPC and/or DP. Other solutions [19], [37], [43], [54], [55] combine MPC techniques and/or DP with ML. However, these protocols often use some heavy cryptographic primitives and are usually customized for specific ML algorithms, which limits their flexibility and scalability. Therefore, such approaches are not suitable for FL.

2.2 Cross-Silo FL

Fig. 1 depicts a typical architecture of cross-silo FL [60], [63], [64]. In cross-silo FL, a public/private key pair of a HE scheme will be distributed to each client, and the public key will be sent to the server. In each training iteration, the server will randomly select a client as the leader who generates and distributes the public/private key pair of the HE scheme. The leader also initializes the ML model and sends the model parameters to all other clients. Upon receiving the model parameters and public/private key pair, each client trains the model locally, computes the gradient, encrypts it with the public key, and sends the encrypted gradient to the server. The server aggregates all encrypted gradients and returns an aggregated encrypted-gradient to all clients. Each client decrypts the aggregated encrypted-gradient and updates the local ML model.

Based on the above framework, WeBank [4] developed an opensource cross-silo FL called FATE [1], where HE is implemented as a pluggable module. Recently, Zhang et al. [63] proposed BatchCrypt using an efficient batch encryption technique. Compared to FATE, BatchCrypt achieves $23\times$-$93\times$ training speedup while reducing the communication cost by $66\times$-$101\times$. Their key idea is to use a new batch encoding scheme to encode a batch of quantized gradients into a long integer and encrypt it in one go. Compared with full precision encryption of a single gradient, this batch encryption substantially reduces encryption overhead and data transmission. However, compared with the plain FL, the batch encryption is still expensive in terms of computation and communication.

Remark 1. In cross-silo FL, the goal of the clients is to train a common ML model that is accessible only to the clients. Hence, it is usually assumed that the clients are honest-but-curious, i.e., they are following the protocol honestly, but will attempt to infer information about other clients' local gradients by colluding with other clients or even the server. However, if one of the clients does not follow the protocol honestly, then they cannot collectively train the desired model, which is to no one's advantage. In other words, active adversaries that deviate from the protocol, sending incorrect and/or arbitrarily chosen messages to honest clients, omitting messages, aborting, and sharing their entire view of the protocol with each other, and also with the server (if the server is an active adversary) are usually not considered in the cross-silo setting. Therefore, choosing any client as the leader will not lead to higher security risk. In fact, such active adversary scenarios are challenging not only in the cross-silo setting, but also in the cross-device setting. The main reason is that the defenses (if they exist) can easily clash with the data privacy requirement; indeed, such defenses always need to have access to the clients' data to detect the clients' behaviors. In this work, we also consider choosing the leader in this way.

2.3 Verifiable FL

The first verifiable FL was proposed by Xu et al. in [57], called VerifyNet. The authors used the double-masking technique proposed by Bonawitz et al. [8], Shamir's secret sharing, key agreement protocol, and symmetric encryption to protect the privacy of the local gradient and handle dropouts, which is similar to what Bonawitz et al. did in their secure aggregation protocol [8]. To verify the correctness of the aggregated gradient, they combined homomorphic hash functions [62] with Pseudorandom Functions [16]. However, VerifyNet requires huge computational and communication costs, where the communication overhead increases linearly with the gradient size. Another limitation of VerifyNet is that it relies on TA.

Another verifiable FL is proposed by Guo et al. in [21] called VeriFL, which reduces the computational and communication overhead required by VerifyNet. Unlike VerifyNet, VeriFL combines a linearly homomorphic hash with a commitment scheme to achieve the correctness verification of the aggregated gradient. VeriFL not only relies on TA, but also requires another 3 rounds of communication for the correctness verification of the aggregated gradient.

Recently, Fu et al. [17] proposed VFL using Lagrange interpolation, which not only supports secure aggregation and correctness verification of the aggregated gradient, but also protects the trained model from being leaked to the server. However, due to the use of the Lagrange interpolation, VFL incurs a large communication overhead that increases linearly with the degree of the polynomial involved in the Lagrange interpolation. In addition, VFL does not consider dropouts. Like VerifyNet and VeriFL, VFL also relies on TA.

3 Preliminaries

In this section, we review the secure network coding signature over integers by Gennaro et al. [18] that will be used as a building block of SVFL, and present a simple masking technique.

3.1 Secure Homomorphic Network Coding Signature Over Integers

Network coding [33] provides an alternative, decentralized approach to traditional multi-cast routing. Gennaro et al. [18] focused on random linear network coding and proposed the first RSA-based homomorphic network coding signature scheme in the random oracle model.

Definition 2 (RSA Assumption [46]). *Given a composite number $N = pq$ and an exponent e prime to the $\varphi(N)$, where p, q are distinct primes and $\varphi(\cdot)$ is the Euler's totient function, the RSA assumption is defined to efficiently find the e-th root modulo N of a random $y \leftarrow \mathbb{Z}_N^*$. For large RSA key sizes, e.g., 2048 bits, the efficient method to solve this problem is not known at present. The subgroup of the squares is denoted by $\mathbf{QR}_N$, where $\mathbf{QR}_N = \{y \in \mathbb{Z}_N^* | \exists x \in \mathbb{Z}_N^*, y = x^2 (\mathrm{mod}\ N)\}$.*

Definition 3. *A homomorphic network coding signature scheme contains a tuple of algorithms* $\mathbf{HNsig} = (\mathbf{Setup}, \mathbf{Sign}, \mathbf{Vrfy}, \mathbf{Combine})$, *defined as follows:*

- ***Setup***$(1^\lambda, 1^N)$. *On input a security parameter λ and a RSA modulus $N = pq$, where p, q are distinct safe primes, which makes $\mathbf{QR}_N$ cyclic and random elements in $\mathbf{QR}_N$ the generators of $\mathbf{QR}_N$ with high probability. Output a public key $pk = (N, e, g_1, \ldots, g_n, H)$ and a private signing key $sk = (d)$, where $e \in \mathbb{Z}_{\varphi(N)}^*$ is an exponent prime to the $\varphi(N)$, $(g_1, \ldots, g_n)$ are random generators of $\mathbf{QR}_N$, $H : \{0,1\}^* \rightarrow \mathbb{Z}_N^*$ is a collision-resistant hash function, and $d \in \mathbb{Z}_{\varphi(N)}^*$ satisfies $ed = 1(\mathrm{mod}\ \varphi(N))$.*
- ***Sign***$(sk, \mathsf{fid}, \mathbf{v}^{(i)})$. *On input a signing key d, a random file identifier $\mathsf{fid} \in \{0,1\}^*$, and an augmented vector $\mathbf{w}^{(i)} = \mathbf{u}^{(i)} || \mathbf{v}^{(i)} = (u_1^{(i)}, \ldots, u_m^{(i)}, v_1^{(i)}, \ldots, v_n^{(i)}) \in \mathbb{Z}^{m+n}$, where $\mathbf{u}^{(i)}$ is the i-th unit vector, output a signature*

$$\sigma = \left(H(i, \mathsf{fid}) \cdot \prod_{j=1}^{n} g_j^{v_j^{(i)}} \right)^d (\mathrm{mod}\ N).$$

In particular, for any vector $\mathbf{w} = (u_1, \ldots, u_m, v_1, \ldots, v_n) \in \mathbb{Z}^{m+n}$, it outputs a signature $\sigma = (\prod_{i=1}^m h_i^{u_i} \cdot \prod_{j=1}^n g_j^{v_j})^d (\mathrm{mod}\ N)$, where $h_i = H(i, \mathsf{fid})$ for $i \in [m]$, where $[m] \triangleq \{1, \ldots, m\}$.
- ***Vrfy***$(pk, \mathbf{w}^{(i)}, \mathsf{fid}, \sigma)$. *On input a public key pk, a vector $\mathbf{w}^{(i)} = (u_1^{(i)}, \ldots, u_m^{(i)}, v_1^{(i)}, \ldots, v_n^{(i)})$, a file identifier fid, and a signature σ, accept the signature σ (i.e., set* $\mathbf{Vrfy}(pk, \mathbf{w}^{(i)}, \mathsf{fid}, \sigma) = 1$*) if and only if*

$$\sigma^e = \prod_{k=1}^{m} h_k^{u_k^{(i)}} \cdot \prod_{j=1}^{n} g_j^{v_j^{(i)}} \ (\mathrm{mod}\ N),$$

where $h_k = H(k, \mathsf{fid})$ for $k \in [m]$. Note that if $(u_1^{(i)}, \ldots, u_m^{(i)}) = \mathbf{u}^{(i)}$, then check $\sigma^e \stackrel{?}{=} H(i, \mathsf{fid}) \cdot \prod_{j=1}^{n} g_j^{v_j^{(i)}} \pmod N$.

- ***Combine***$(pk, \{(\mathbf{w}^{(i)}, \sigma_i, \alpha_i)\}_{i=1}^{\ell})$. *On input a public key pk and ℓ triples $(\mathbf{w}^{(i)}, \sigma_i, \alpha_i)$ for $\ell \in [m]$, where $\alpha_i \in \mathbb{Z}$, check if* $\mathbf{Vrfy}(pk, \mathbf{w}^{(i)}, \mathsf{fid}, \sigma_i) = 1$ *holds for $i \in [\ell]$. For those $(\mathbf{w}^{(i)}, \sigma_i, \alpha_i)$ that pass verification, e.g., $(\mathbf{w}^{(1)}, \sigma_1, \alpha_1), \ldots, (\mathbf{w}^{(\ell')}, \sigma_{\ell'}, \alpha_{\ell'})$, set $\mathbf{w} = \sum_{i=1}^{\ell'} \alpha_i \mathbf{w}^{(i)}$, and output a signature on $\mathbf{w}$:*

$$\sigma = \prod_{i=1}^{\ell'} \sigma_i^{\alpha_i} (\bmod\ N).$$

The correctness of the above signature scheme is straightforward. The security of the above signature scheme says that any PPT adversary, given access to the public key and signature queries (adaptively) for a set of vectors $\bar{\mathbf{v}}^{(1)}, \ldots, \bar{\mathbf{v}}^{(m)}$ (corresponding to the augmented vectors $\bar{\mathbf{w}}^{(1)}, \ldots, \bar{\mathbf{w}}^{(m)}$), cannot forge a valid signature σ^* on a vector $\mathbf{w}^*$ under a file identifier $\mathsf{fid}^* \in \{0,1\}^*$, where fid^* is one of the identifiers chosen by the challenger (signer) and the vector $\mathbf{w}^*$ is not in the linear span of the vectors $(\bar{\mathbf{w}}^{(1)}, \ldots, \bar{\mathbf{w}}^{(m)})$. We refer the interested reader to [18] for details about the security definition, and here we only recall the following theorem.

Theorem 4 ([18]). *Under the RSA assumption, the above homomorphic network coding signature scheme **HNsig** is secure against any PPT adversary in the random oracle model.*

Lemma 5. *Given $\mathbf{w}^{(i)} = \mathbf{u}^{(i)} || \mathbf{v}^{(i)} \in \mathbb{Z}^{m+n}$, where $\mathbf{u}^{(i)}$ is the i-th unit vector, for any vector $\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i \mathbf{w}^{(i)}$ and $\mathbf{w}' = \sum_{i=1}^{\ell} \beta_i \mathbf{w}^{(i)}$ for $\ell \in [m]$, if $\mathbf{w} = \mathbf{w}'$, then we have $\alpha_i = \beta_i$ for $i \in [\ell]$.*

Proof. Since $\mathbf{u}^{(i)}$ is the i-th unit vector, we have

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i \mathbf{w}^{(i)} = \left(\alpha_1, \ldots, \alpha_\ell, 0, \ldots, 0, \sum_{i=1}^{\ell} \alpha_i v_1^{(i)}, \ldots, \sum_{i=1}^{\ell} \alpha_i v_n^{(i)} \right)$$

and

$$\mathbf{w}' = \sum_{i=1}^{\ell} \beta_i \mathbf{w}^{(i)} = \left(\beta_1, \ldots, \beta_\ell, 0, \ldots, 0, \sum_{i=1}^{\ell} \beta_i v_1^{(i)}, \ldots, \sum_{i=1}^{\ell} \beta_i v_n^{(i)} \right).$$

Therefore, if $\mathbf{w} = \mathbf{w}'$, the equality $\alpha_i = \beta_i$ holds. □

The Lemma 5 indicates that any vector $\mathbf{w}$ in the linear span of the vectors $(\bar{\mathbf{w}}^{(1)}, \ldots, \bar{\mathbf{w}}^{(m)})$ can be represented as a unique linear combination of the vectors $(\bar{\mathbf{w}}^{(1)}, \ldots, \bar{\mathbf{w}}^{(m)})$. Combining Theorem 4 with Lemma 5, we can obtain the following lemma.

Lemma 6. *For secure **HNsig** scheme, given a series of valid vector-signature pairs $(\bar{\mathbf{w}}^{(1)}, \sigma_1), \ldots, (\bar{\mathbf{w}}^{(m)}, \sigma_m)$ under an identifier $\mathsf{fid}^* \in \{0,1\}^*$, if σ^* is a valid signature on a vector $\mathbf{w}^* = (\gamma_1, \ldots, \gamma_m, w_1, \ldots, w_n)$ under the file identifier $\mathsf{fid}^* \in \{0,1\}^*$, then the vector $\mathbf{w}^*$ must be a unique linear combination of the vectors $(\bar{\mathbf{w}}^{(1)}, \ldots, \bar{\mathbf{w}}^{(m)})$, i.e., $\mathbf{w}^* = \sum_{i=1}^{m} \gamma_i \bar{\mathbf{w}}^{(i)}$.*

Proof. First, by Theorem 4, if σ^* is a valid signature on a vector $\mathbf{w}^* = (\gamma_1, \ldots, \gamma_m, w_1, \ldots, w_n)$ under a file identifier $\mathsf{fid}^* \in \{0,1\}^*$, then the vector $\mathbf{w}^*$ must be in the linear span of the vectors $(\bar{\mathbf{w}}^{(1)}, \ldots, \bar{\mathbf{w}}^{(m)})$. By Lemma 5, if $\mathbf{w}^*$ is in the linear span of the vectors $(\bar{\mathbf{w}}^{(1)}, \ldots, \bar{\mathbf{w}}^{(m)})$, then the vector $\mathbf{w}^*$ must be a unique linear combination of the vectors $(\bar{\mathbf{w}}^{(1)}, \ldots, \bar{\mathbf{w}}^{(m)})$. □

The Lemma 6 ensures that we can derive a unique linear combination of the vectors $(\bar{\mathbf{w}}^{(1)}, \ldots, \bar{\mathbf{w}}^{(m)})$ for the vector $\mathbf{w}^*$ from a valid vector-signature pair $(\sigma^*, \mathbf{w}^*)$. This plays a crucial role in our SVFL.

3.2 Masking With One-Time Pads

We present a simple masking technique that we call masking with one-time pads, or **Motp** for short. **Motp** is similar to the encryption *one-time pad* (OTP) [44] and that of [8], [17], [57].

Given a vector $\mathbf{x} \in \mathbb{Z}_R^n$ for some $R \in \mathbb{N}$, choose a uniformly random vector $\mathbf{r} \in \mathbb{Z}_R^n$, then, we mask the vector $\mathbf{x}$ as follows:

$$\mathbf{z} = \mathbf{x} + \mathbf{r} (\bmod\ R).$$

Given the mask vector $\mathbf{r} \in \mathbb{Z}_R^n$, we can unmask the vector $\mathbf{z} \in \mathbb{Z}_R^n$ as follows:

$$\mathbf{x} = \mathbf{z} - \mathbf{r} (\bmod\ R).$$

Lemma 7. *For any $\ell \geq 1$, fix ℓ vectors $\mathbf{x}_1, \ldots, \mathbf{x}_\ell \in \mathbb{Z}_R^n$, choose $\ell + 1$ uniformly random vectors $\mathbf{r}_1, \ldots, \mathbf{r}_\ell \in \mathbb{Z}_R^n$ and $\mathbf{y} \in \mathbb{Z}_R^n$, compute $\mathbf{z} = \sum_{i=1}^{\ell} (\mathbf{x}_i + \mathbf{r}_i) (\bmod\ R)$. Then, we have*

$$\mathbf{z} \equiv \mathbf{y},$$

where "$\equiv$" indicates that the distributions are identical.

Proof. The proof is straightforward by mathematical induction. First, when $\ell = 1$, let $\mathbf{z}_1 = \mathbf{x}_1 + \mathbf{r}_1 (\bmod\ R)$, then we have $\mathbf{z}_1 \equiv \mathbf{y}_1$ due to the property of uniform distribution, where $\mathbf{y}_1 \xleftarrow{\$} \mathbb{Z}_R^n$. Assume that $\mathbf{z}_k \equiv \mathbf{y}_k$ holds when $\ell = k$, where $\mathbf{z}_k = \sum_{i=1}^{k} (\mathbf{x}_i + \mathbf{r}_i) (\bmod\ R)$, $\mathbf{y}_k \xleftarrow{\$} \mathbb{Z}_R^n$ and $k \in \mathbb{N}$, then we have

$$\begin{aligned} \mathbf{z}_k + \mathbf{x}_{k+1} + \mathbf{r}_{k+1} (\bmod\ R) &\equiv \mathbf{y}_k + \mathbf{x}_{k+1} + \mathbf{r}_{k+1} (\bmod\ R) \\ &\equiv \mathbf{y}_k + \mathbf{y}_{k+1} (\bmod\ R), \end{aligned}$$

where $\mathbf{y}_{k+1} \xleftarrow{\$} \mathbb{Z}_R^n$. On the other hand, we have

$$\mathbf{y}_k + \mathbf{y}_{k+1} (\bmod\ R) \equiv \mathbf{y},$$

where $\mathbf{y} \xleftarrow{\$} \mathbb{Z}_R^n$. Therefore, we have

$$\mathbf{z}_{k+1} \equiv \mathbf{y},$$

where $\mathbf{z}_{k+1} = \sum_{i=1}^{k+1} (\mathbf{x}_i + \mathbf{r}_i) (\bmod\ R)$. □

To reduce the communication cost, we use a secure Pseudorandom Generator [61] (PRG) to generate a mask vector $\mathbf{r} = \mathbf{PRG}(\mathbf{s}) \in \mathbb{Z}_R^n$, where $\mathbf{s} \xleftarrow{\$} \{0,1\}^l$ is a uniformly random seed. By the security of the PRG, we have that $\mathbf{r} \overset{comp}{\approx} \mathbf{r}'$, where $\mathbf{r}' \xleftarrow{\$} \mathbb{Z}_R^n$ and $X \overset{comp}{\approx} Y$ indicates that X and Y are computationally indistinguishable, as long as $\mathbf{s}$ is kept secret from the PPT distinguisher.

To mask a vector $\mathbf{x} \in \mathbb{Z}_R^n$, we compute:

$$\mathbf{z} = \mathbf{x} + \mathbf{PRG}(\mathbf{s}) (\bmod\ R).$$

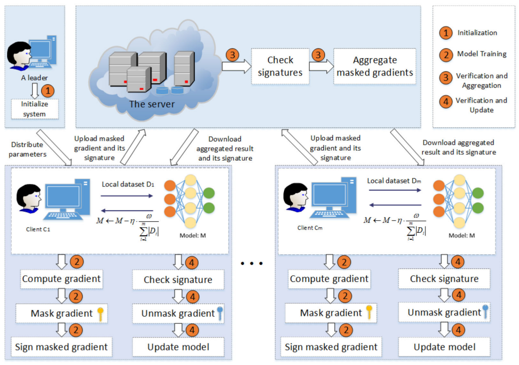


Fig. 2. The architecture of SVFL.

Given the seed $\mathbf{s}$, we can unmask the vector $\mathbf{z} \in \mathbb{Z}_R^n$:

$$\mathbf{x} = \mathbf{z} - \mathbf{PRG}(\mathbf{s}) (\bmod\ R).$$

Combining Lemma 7 with the security of the PRG yields the following lemma.

Lemma 8. *For any $\ell \geq 1$, fix ℓ vectors $\mathbf{x}_1, \ldots, \mathbf{x}_\ell \in \mathbb{Z}_R^n$, choose $\ell + 1$ uniformly random vectors $\mathbf{s}_1, \ldots, \mathbf{s}_\ell \in \{0,1\}^l$ and $\mathbf{y} \in \mathbb{Z}_R^n$, compute $\mathbf{z} = \sum_{i=1}^{\ell}(\mathbf{x}_i + \mathbf{PRG}(\mathbf{s}_i))(\bmod\ R)$. Then, we have $\mathbf{z} \overset{comp}{\approx} \mathbf{y}$.*

4 The Proposed Protocol

In this section, we construct SVFL, an efficient cross-silo FL protocol supporting secure aggregation and verification, using **HNsig** scheme and our **Motp** scheme.

4.1 System Architecture

Fig. 2 depicts SVFL, where the **HNsig** and **Motp** schemes are implemented as pluggable modules. SVFL contains two entities: the client and the server.

- **Client.** The role of a client includes: 1) initializing the system by generating initial model parameters and some related parameters (one of the clients); 2) training the model locally, masking the local gradient, and signing the masked gradient; and 3) checking the correctness of the aggregated gradient.
- **Server.** The role of the server includes: 1) randomly selecting a leader from the clients; 2) aggregating the masked gradients; and 3) returning an aggregated masked-gradient along with an aggregated signature ("proof").

4.2 The SVFL Protocol

Let $C = \{C_1, \ldots, C_m\}$ be a set of clients with a corresponding local datasets $D = \{D_i, \ldots, D_m\}$. SVFL consists of four phases: **Initialization**, **Model Training**, **Aggregation** and **Verification and Update**:

- **Initialization**. In each training iteration, the server will randomly select a client as the leader to initialize the model and generate m seeds of **PRG** and a public/secret key pair of **HNsig** scheme. The leader also chooses a random string and m weights. Then, the leader sends the model parameters, the seeds, the public/secret key pair, the random string and m weights to all other clients, while sending the public key, the random string and m weights to the server.
- **Model Training**. Upon receiving the model parameters, m seeds, the public/secret key pair, the random string and m weights, each client trains the model locally, computes the local gradient, masks it using **Motp** scheme, signs the masked gradient using **HNsig**, and sends the (masked) gradient-signature pair to the server.
- **Aggregation**. Upon receiving the gradient-signature pair from each client, the server aggregates all masked gradients and computes an aggregated signature on the aggregated masked-gradient using **HNsig**. Finally, the server returns the aggregated masked-gradient and its aggregated signature to each client.
- **Verification and Update**. After receiving the aggregated masked-gradient and its aggregated signature from the server, each client first validates the signature, then unmasks the aggregated masked-gradient, updates the local model, and starts the next iteration. Otherwise, the client notifies the other clients and terminates SVFL.

Clearly, our SVFL applies to cross-device setting by assuming a TA, which initializes the model and generates the required parameters. We provide a general description of SVFL in Fig. 3. We also provide details about the masking and signing algorithm (see Algorithm 1), the verification and aggregation algorithm (see Algorithm 2), and the verification and update algorithm (see Algorithm 3). We reiterate that SVFL is for cross-silo FL, so the clients are a handful of organizations with adequate computing resources and reliable communications. This means that we do not need to consider dropouts. Moreover, the public/secret key pair of **HNsig** and the m weights can always be reused, but the random string fid and the m seeds of **PRG** cannot be reused. Hence, in each iteration the server needs to re-randomly select a leader to re-generate the random string and the m seeds. Of course, we can assume that there is a TA to initialize the trained model and generate these parameters, but it is reasonable to assume that some randomly selected client plays this role in the cross-silo FL setting (see Remark 1). Indeed, After all, generating a public/secret key pair for **HNsig** is costly, but such cost is one-time; generating a random string and m seeds is a recurring task yet a simple task.

Algorithm 1. Masking and Signing Algorithm

```
Input: $\mathbf{W}_i \in \mathbb{Z}_R^n, \mathbf{s}_i \in \{0,1\}^l, (pk, sk)$, fid.
Output: $(\widehat{\mathbf{W}}_i, \sigma_i, i)$.
1. Mask the gradient $\mathbf{W}_i$:
      $\widehat{\mathbf{W}}_i = \mathbf{W}_i + \mathbf{PRG}(\mathbf{s}_i)(\bmod\ R);$
2. Sign the masked gradient $\widehat{\mathbf{W}}_i = (\hat{W}_{i,1}, \ldots, \hat{W}_{i,n})$:
      $\sigma_i = \left( H(i, \mathsf{fid}) \cdot \prod_{j=1}^{n} g_j^{\hat{W}_{i,j}} \right)^d (\bmod\ N);$
3. Return $(\widehat{\mathbf{W}}_i, \sigma_i, i)$.
```

SVFL Protocol

- **Initialization**: The server randomly selects a client as the leader. The leader proceeds as follows:
  - Initialize the model and obtain the model parameters $\mathbf{M}^t$, where t is the number of iterations and we set $t = 0$ in the initialization phase.
  - Generate m random seeds $\mathbf{s}_1, \ldots, \mathbf{s}_m \overset{\$}{\leftarrow} \{0,1\}^l$ of **PRG** for **Motp** scheme and run **Setup**$(1^\lambda, 1^N) \to (pk = (N, e, g_1, \ldots, g_n, H), sk = d)$ for **HNsig** scheme.
  - Choose a random string $\mathsf{fid} \in \{0,1\}^*$ and m weights $\alpha_1, \ldots, \alpha_m \in \mathbb{N}$.
  - Send $(\mathbf{M}^t, (\mathbf{s}_1, \ldots, \mathbf{s}_m), pk, sk, \mathsf{fid}, (\alpha_1, \ldots, \alpha_m))$ to all other clients, while sending $(pk, \mathsf{fid}, (\alpha_1, \ldots, \alpha_m))$ to the server.
- **Model Training**: Upon receiving $(\mathbf{M}^t, (\mathbf{s}_1, \ldots, \mathbf{s}_m), pk, sk, \mathsf{fid}, (\alpha_1, \ldots, \alpha_m))$, each client C_i for $i \in [m]$ proceeds as follows:
  - Choose a random dataset $D_i^t \subseteq D_i$ of size $\alpha_i = |D_i^t|$, train the model locally and compute the local gradient $\mathbf{W}_i \in \mathbb{Z}^n$. For simplicity, we assume that the dimension of the gradient $\mathbf{W}_i$ is n and the elements of vector $\alpha_i \mathbf{W}_i$ are in $\mathbb{Z}_R$ for some $R \in \mathbb{N}$.
  - Mask the local gradient $\mathbf{W}_i$ by computing
  $$\widehat{\mathbf{W}}_i = \mathbf{W}_i + \mathbf{PRG}(\mathbf{s}_i) \pmod{R},$$
  and sign the masked gradient $\widehat{\mathbf{W}}_i$ by computing
  $$\sigma_i = \mathbf{Sign}(sk, \mathsf{fid}, \widehat{\mathbf{W}}_i).$$
  - Send $(\widehat{\mathbf{W}}_i, \sigma_i, i)$ to the server. See Algorithm 1 for details.
- **Aggregation**: After receiving $(\widehat{\mathbf{W}}_i, \sigma_i, i)$, the server proceeds as follows:
  - Aggregate all masked gradients by computing:
  $$\widehat{\mathbf{W}} = \textstyle\sum_{i=1}^m \alpha_i \widehat{\mathbf{W}}_i.$$
  - Compute an aggregated signature:
  $$\sigma = \mathbf{Combine}(pk, \{(\widetilde{\mathbf{W}}_i, \sigma_i, \alpha_i)\}_{i=1}^m),$$
  where $\widetilde{\mathbf{W}}_i = \mathbf{u}_i || \widehat{\mathbf{W}}_i \in \mathbb{Z}^{m+n}$ and $\mathbf{u}_i$ is the i-th unit vector.
  - Return $(\widehat{\mathbf{W}}, \sigma)$ to each client. See Algorithm 2 for details.
- **Verification and Update**: Upon receiving $(\widehat{\mathbf{W}}, \sigma)$, each client C_i proceeds as follows:
  - Check whether the aggregated signature is valid:
  $$\mathbf{Vrfy}(pk, \widetilde{\mathbf{W}}, \mathsf{fid}, \sigma) \overset{?}{=} 1,$$
  where $\widetilde{\mathbf{W}} = (\sum_{i=1}^m \alpha_i \mathbf{u}_i) || \widehat{\mathbf{W}}$. If the signature is invalid, then reject the aggregated masked-gradient.
  - Otherwise, unmask the aggregated masked-gradient $\widehat{\mathbf{W}}$ by computing
  $$\mathbf{W} = \widehat{\mathbf{W}} - \textstyle\sum_{i=1}^m \alpha_i \cdot \mathbf{PRG}(\mathbf{s}_i) \pmod{R},$$
  and update the local model parameters:
  $$\mathbf{M}^{t+1} \leftarrow \mathbf{M}^t - \eta \cdot \frac{\mathbf{W}}{\sum_{i=1}^m \alpha_i},$$
  where η is a learning rate parameter, and proceed to the next iteration until the termination condition is met. See Algorithm 3 for details.

Fig. 3. Detailed description of SVFL.

Algorithm 2. Aggregation Algorithm

```
Input: $\{(\widehat{\mathbf{W}}_i, \sigma_i, i)\}_{i=1}^m, pk, (\alpha_1, \ldots, \alpha_m)$.
Output: $(\widehat{\mathbf{W}}, \sigma)$.
1. Aggregate all masked gradients corresponding to $\{(\alpha_i, i)\}_{i=1}^m$:
    $\widehat{\mathbf{W}} = \sum_{i=1}^m \alpha_i \widehat{\mathbf{W}}_i$;
2. Combine all signatures corresponding to $\{(\alpha_i, i)\}_{i=1}^m$:
    $\sigma = \prod_{i=1}^m \sigma_i^{\alpha_i} \pmod{N}$;
3. Return $(\widehat{\mathbf{W}}, \sigma)$.
```

5 Correctness and Security Analysis

In this section, we analyze the correctness and security of SVFL.

5.1 Correctness of SVFL

The correctness of SVFL requires that each client obtain correct aggregated gradient and valid aggregated signature, as long as each client and the server run SVFL protocol honestly.

Theorem 9 (Correctness). *We assume that the elements of weighted gradients (i.e., $\alpha_i \widehat{\mathbf{W}}_i$) are in the same range $[0, B-1]$ (e.g., 32 bits long) and $R = m(B-1)+1$, where $B \in \mathbb{N}$. Let λ be the security parameter and $(\mathbf{s}_1, \ldots, \mathbf{s}_m) \in \{0,1\}^l$ be m random seeds of **PRG**, where we let $l = \lambda$. Then, if each client and the server execute SVFL honestly, each client will get correct aggregated gradient and valid aggregated signature.*

Proof. It is not hard to prove this due to the correctness of **Motp** and **HNsig** schemes. Specifically, given $\{(\widehat{\mathbf{W}}_i, \sigma_i, \alpha_i, i)\}_{i=1}^m$, where $\widehat{\mathbf{W}}_i = \mathbf{W}_i + \mathbf{PRG}(\mathbf{s}_i) \pmod{R}$, the server can compute an aggregated masked-gradient and a corresponding aggregated signature by running $\widehat{\mathbf{W}} \leftarrow \sum_{i=1}^m \alpha_i \widehat{\mathbf{W}}_i$ and $\sigma \leftarrow \mathbf{Combine}(pk, \{(\widetilde{\mathbf{W}}_i, \sigma_i, \alpha_i)\}_{i=1}^m)$, respectively. By the correctness of the **HNsig** scheme, the signature σ is valid on $\widehat{\mathbf{W}}$. On the other hand, given $(\widehat{\mathbf{W}}, \{(\alpha_i, i)\}_{i=1}^m)$, we have

$$\widehat{\mathbf{W}} = \sum_{i=1}^m \alpha_i \widehat{\mathbf{W}}_i = \sum_{i=1}^m \alpha_i \mathbf{W}_i + \sum_{i=1}^m \alpha_i \cdot \ \mathbf{PRG}(\mathbf{s}_i) + KR$$

for some $K \in \mathbb{N}$. Therefore, we have

$$\mathbf{W} = \widehat{\mathbf{W}} - \sum_{i=1}^m \alpha_i \cdot \ \mathbf{PRG}(\mathbf{s}_i) = \sum_{i=1}^m \alpha_i \mathbf{W}_i (\mathrm{mod}\ R).$$

□

5.2 Threat Model of SVFL

In a cross-silo FL system, we want to prevent each client's local gradient from being leaked to other clients as well as both the local gradients and the trained model (aggregated gradient) from being leaked to the server, while ensuring that the server aggregates the masked gradients honestly. To preserve the privacy of the local gradients and the trained model, we consider the honest-but-curious setting, where both the clients and the server will follow the protocol honestly, but will attempt to infer information about the clients' local gradients.

Algorithm 3. Verification and Update Algorithm

```
Input: $(\widehat{\mathbf{W}}, \sigma), (\mathbf{s}_1, \ldots, \mathbf{s}_m), pk, (\alpha_1, \ldots, \alpha_m), \mathbf{M}^t, \eta$.
Output: $\mathbf{M}^{t+1}$.
  1. Check if the signature is valid:
  $\sigma^e = \prod_{i=1}^k h_i^{\alpha_i} \cdot \prod_{j=1}^n g_j^{W_j} (\mathrm{mod}\ N)$, where $h_i = H(i, \mathsf{fid})$
  and $\widehat{\mathbf{W}} = (\hat{W}_1, \ldots, \hat{W}_n)$;
  2. Unmask the vector $\widehat{\mathbf{W}}$:
        $\mathbf{W} = \widehat{\mathbf{W}} - \sum_{i=1}^k \alpha_i \cdot \ \mathbf{PRG}(\mathbf{s}_i)(\mathrm{mod}\ R)$;
  3. Update the local model parameters:
        $\mathbf{M}^{t+1} \leftarrow \mathbf{M}^t - \eta \cdot \frac{\mathbf{W}}{\sum_{i=1}^k \alpha_i}$;
  4. Return $\mathbf{M}^{t+1}$.
```

However, unlike cross-device FL, in cross-silo FL, we do not consider collusion between the server and the clients. Recall that all clients share the same information (i.e., the model parameters, the seeds, the public/secret key pair, the random string and m weights) distributed by the leader, while the server can only obtain the public key, the random string and m weights. If the server colludes with some clients, then the clients can only provide the server with the public key, the random strings and m weights that the server already has, as providing information other than these will result in the disclosure of not only the trained model (it is the common interest of all clients in the cross-silo setting) but also their own local gradients (it follows from the construction of SVFL). In other words, in cross-silo FL, the collusion between the server and the clients does not provide any additional information to the server and therefore makes no sense to the server. But if the server is willing to collude with some clients despite information asymmetry (i.e., the server cannot get any useful information from the clients), the clients can compute other clients' local gradients by unmasking their masked gradients provided by the server.

Based on the above analysis, coupled with the fact that protecting the local gradients and the trained model from being learned by any external parties (including the server) is the primary purpose of cross-silo FL, it is reasonable to not consider the collusion between the server and the clients. For completeness, we discuss some potential approaches to defend against such collusion attacks in Remark 10. In fact, such collusion was also not considered in HE-based FL (e.g., [12], [23], [45], [53]). Therefore, we consider two types of attacks: one is initiated by the honest-but-curious clients, where up to $m-2$ clients collude to infer information about the other clients' local gradients, and the other is initiated by the honest-but-curious server whose goal is to infer information about the clients' local gradients and the trained model.

As for the security of verification, like the previous verifiable FL [17], [21], [57], we consider an active-adversary setting, where the malicious server may not aggregate the masked gradient honestly, instead tries to forge an aggregated masked-gradient and its corresponding "proof". We point out that the concrete threat model relies on the specific technique used to achieve this.

Remark 10. One potential defense against such collusion attacks (note that here we do not consider the security of verification) is to use a TA to generate m weights $\{\alpha_i \in \mathbb{N}\}_{i\in[m]}$ and $m+1$ random masked vectors $\{\mathbf{s}_i \in \{0,1\}^l\}_{i\in[m]}, \mathbf{r} \in \mathbb{Z}_R^n$ such that $\sum_{i=1}^m \alpha_i \ \mathbf{PRG}(\mathbf{s}_i) = \mathbf{r}$ and distribute $(\mathbf{s}_i, \mathbf{r})$ to the corresponding client C_i for $i \in [m]$. Then, by the **Motp** scheme, each client C_i uses the masked vector $\mathbf{PRG}(\mathbf{s}_i)$ to mask its local gradient. Upon receiving the aggregated masked-gradient from the server, each client uses the vector $\mathbf{r}$ to unmask the gradient and obtains the desired sum. This approach is highly efficient, but it relies on the TA. Another potential defense mechanism is to have the clients generate the above m weights and vectors $(\mathbf{s}_i, \mathbf{r})$ themselves by running an MPC protocol. While the efficiency of this approach is acceptable, it requires more communication rounds. Yet another potential defense mechanism is to use some distributed cryptographic primitives, such as multi-key homomorphic encryption scheme. This approach may not require more communication rounds, but it is often inefficient.

5.3 Security of SVFL

We first show that SVFL can protect the privacy of clients' local gradients and aggregated gradient. Then, we show that SVFL can prevent the malicious server from forging aggregated masked-gradient and its signature.

Privacy. In the honest-but-curious setting of our threat model, each client and the server will follow the protocol honestly, but try to infer the clients' local gradients and aggregated gradient. Therefore, we can use a simulation-based proof which is standard for MPC protocol to prove the privacy of the clients' local gradients and aggregated gradient.

We first consider the privacy against the honest-but-curious clients, who have their own local gradients and the aggregated gradient. Given any subset $\mathcal{U} \subseteq \mathcal{C}$ of the clients, where $\mathcal{C}$ is the set of all clients ($|\mathcal{C}| = m$), let $\mathbf{REAL}_{\mathcal{U}}^{\mathcal{C},\lambda}(\{(\alpha_i, \mathbf{W}_i)\}_{i\in\mathcal{C}}, (\mathbf{s}_1, \ldots, \mathbf{s}_m))$ be a random variable representing the joint view of the clients in $\mathcal{U}$. It suffices to prove that the joint view of the clients in $\mathcal{U}$ can be simulated given only the gradients of the clients in $\mathcal{U}$ and the aggregated gradient. This indicates that these honest-but-curious clients learn nothing more than the sum of gradients of the other clients and their own gradients.

Then, we consider the privacy against the honest-but-curious server, who receives m masked gradients and their signatures from clients. Note that the server does not collude with any client according to our threat model. In addition, the signatures are for the masked gradients, they do not reveal any information about the local gradients. Therefore, it suffices to show that these masked gradients and any aggregated masked-gradient can be simulated given only some public information. This indicates that the server learns nothing more than some random vectors. Let $\mathbf{REAL}_{\mathcal{S}}^{\mathcal{C},\lambda}(\{(\alpha_i, \mathbf{W}_i)\}_{i\in\mathcal{C}}, (\mathbf{s}_1, \ldots, \mathbf{s}_m))$ be a random variable representing the view of the server $\mathcal{S}$.

Theorem 11 (Privacy Against Honest-but-Curious Clients). *Under the same parameter selections as Theorem 9, there exists a PPT simulator* $\mathbf{SIM}$ *such that for all* λ, $(\mathbf{s}_1, \ldots, \mathbf{s}_m)$, $\mathcal{C}$, $\mathcal{U}$ *such that* $\mathcal{U} \subseteq \mathcal{C}$, *and inputs* $\{(\alpha_i, \mathbf{W}_i)\}_{i\in\mathcal{C}}$, *the output of* $\mathbf{SIM}_{\mathcal{U}}^{\mathcal{C},\lambda}$ *is computationally indistinguishable from the output of* $\mathbf{REAL}_{\mathcal{U}}^{\mathcal{C},\lambda}$:

$$\begin{aligned}&\mathbf{SIM}_{\mathcal{U}}^{\mathcal{C},\lambda}(\mathbf{W}, \{(\alpha_i, \mathbf{W}_i)\}_{i\in\mathcal{U}}, (\mathbf{s}_1, \ldots, \mathbf{s}_m))\\ &\overset{comp}{\approx} \ \mathbf{REAL}_{\mathcal{U}}^{\mathcal{C},\lambda}(\{(\alpha_i, \mathbf{W}_i)\}_{i\in\mathcal{C}}, (\mathbf{s}_1, \ldots, \mathbf{s}_m)),\end{aligned}$$

where $\mathbf{W} = \sum_{i\in\mathcal{C}} \alpha_i \mathbf{W}_i$.

Proof. Since the server is not involved, the joint view of the clients in $\mathcal{U}$ does not depend on the local gradients of the clients not in $\mathcal{U}$. Given $(\mathbf{W}, \{(\alpha_i, \mathbf{W}_i)\}_{i\in\mathcal{U}})$, where $\mathcal{U} \subseteq \mathcal{C}$ and $\mathbf{W} = \sum_{i\in\mathcal{C}} \alpha_i \mathbf{W}_i$, the simulator $\mathbf{SIM}$ can generate the simulated view of the clients in $\mathcal{U}$ by running the clients in $\mathcal{U}$ on their real local gradients $\{\mathbf{W}_i\}_{i\in\mathcal{U}}$ and the seeds $\{\mathbf{s}_i\}_{i\in\mathcal{U}}$, and the clients in $\mathcal{C} \setminus \mathcal{U}$ on some uniformly random vectors. More specifically, the simulator $\mathbf{SIM}$ generates the real masked gradient $\{\widehat{\mathbf{W}}_i\}_{i\in\mathcal{U}}$ using $\{(\mathbf{s}_i, \alpha_i, \mathbf{W}_i)\}_{i\in\mathcal{U}}$ for the clients in $\mathcal{U}$. For all honest clients in $\mathcal{C} \setminus \mathcal{U}$ with the local gradients $\{\mathbf{W}_j\}_{j\in\mathcal{C}\setminus\mathcal{U}}$, instead of computing the masked gradient $\widehat{\mathbf{W}}_j = \mathbf{W}_j + \mathbf{PRG}(\mathbf{s}_j) \pmod R$ for $j \in [|\mathcal{C} \setminus \mathcal{U}|]$, the simulator $\mathbf{SIM}$ selects $|\mathcal{C} \setminus \mathcal{U}|$ uniformly random vectors $\mathbf{Z}_i \in \mathbb{Z}_R^n$ subject to $\mathbf{W} - \sum_{i\in\mathcal{U}} \alpha_i \mathbf{W}_i = \sum_{i\in\mathcal{C}\setminus\mathcal{U}} \mathbf{Z}_i \pmod R$ as the masked gradients. By Lemma 8, we have

$$\mathbf{Z}_j \overset{comp}{\approx} \widehat{\mathbf{W}}_j \tag{1}$$

for $j \in [|\mathcal{C} \setminus \mathcal{U}|]$, and hence we conclude that the simulator $\mathbf{SIM}$ successfully simulates the joint view of the clients in $\mathcal{U}$. □

Theorem 12 (Privacy Against Honest-but-Curious Server). *Under the same parameter selections as Theorem 9, there exists a PPT simulator* $\mathbf{SIM}$ *such that for all* λ, $\mathbb{Z}_R^n$, $(\mathbf{s}_1, \ldots, \mathbf{s}_m)$, $\mathcal{C}$, $\mathcal{S}$ *and inputs* $\{(\alpha_i, \mathbf{W}_i)\}_{i\in\mathcal{C}}$, *the output of* $\mathbf{SIM}_{\mathcal{S}}^{\mathcal{C},\lambda}$ *is computationally indistinguishable from the output of* $\mathbf{REAL}_{\mathcal{S}}^{\mathcal{C},\lambda}$:

$$\begin{aligned}&\mathbf{SIM}_{\mathcal{S}}^{\mathcal{C},\lambda}(\{\alpha_i\}_{i\in\mathcal{C}}, \mathbb{Z}_R^n) \overset{comp}{\approx}\\ &\mathbf{REAL}_{\mathcal{S}}^{\mathcal{C},\lambda}(\{(\alpha_i, \mathbf{W}_i)\}_{i\in\mathcal{C}}, (\mathbf{s}_1, \ldots, \mathbf{s}_m)).\end{aligned}$$

Proof. Since the server can only obtain $\{(\widehat{\mathbf{W}}_i, \sigma_i, \alpha_i)\}_{i=1}^m$ from clients, where $\widehat{\mathbf{W}}_i = \mathbf{W}_i + \mathbf{PRG}(\mathbf{s}_i) \pmod R \in \mathbb{Z}_R^n$, the simulator $\mathbf{SIM}$ can simulate the view of the honest-but-curious server by simply choosing m uniformly random vectors $\mathbf{Z}_i \in \mathbb{Z}_R^n$ for $i \in [m]$. Then, by Lemma 8, we have

$$\mathbf{Z}_i \overset{comp}{\approx} \widehat{\mathbf{W}}_i \tag{2}$$

for $i \in [m]$.

In addition, note that the server can aggregate any number of masked gradients. Then, by Eq.(2), we have

$$\sum_{i=1}^{k} \alpha_i \mathbf{Z}_i \overset{comp}{\approx} \sum_{i=1}^{k} \alpha_i \widehat{\mathbf{W}}_i$$

for any $k \in [m]$. □

Verifiability. Following our threat model, we now show that the malicious server cannot pass each client's checking.

Theorem 13. *Under the same parameter selections as Theorem 9, each client can detect forged aggregated masked-gradient.*

Proof. Let $(\widehat{\mathbf{W}}_i, \sigma_i, \alpha_i)$ for $i \in [m]$ be (masked) gradient-signature tuples that the malicious server got from clients. Recall that any signature satisfies

$$\sigma_i^e = H(i, \mathsf{fid}) \cdot \prod_{j=1}^{n} g_j^{\hat{W}_{i,j}} \pmod N,$$

where $\widehat{\mathbf{W}}_i = (\hat{W}_{i,1}, \ldots, \hat{W}_{i,n})$. Let $(\overline{\mathbf{W}}, \bar{\sigma})$ be an aggregated gradient-signature pair that the malicious server sends to each client during the *Verification and Update* phase. Each client checks if the signature is valid, i.e., $\mathbf{Vrfy}(pk, \overline{\mathbf{W}}', \mathsf{fid}, \bar{\sigma}) \overset{?}{=} 1$, where $\overline{\mathbf{W}}' = (\sum_{i=1}^m \alpha_i \mathbf{u}_i)||\overline{\mathbf{W}}$. Assume that $(\overline{\mathbf{W}}', \bar{\sigma})$ is a forgery, then we have that the signature $\bar{\sigma}$ can pass each client's checking but $\overline{\mathbf{W}} \neq \sum_{i=1}^m \alpha_i \widehat{\mathbf{W}}_i$.

However, if the signature passes each client's checking, then we have

$$\bar{\sigma}^e = \prod_{i=1}^{m} h_i^{\alpha_i} \cdot \prod_{j=1}^{n} g_j^{\bar{W}_j} \pmod N, \tag{3}$$

where $h_i = H(i, \mathsf{fid})$ and $\overline{\mathbf{W}} = (\bar{W}_1, \ldots, \bar{W}_n)$ (by Algorithm 3). This means that $\bar{\sigma}$ is a valid signature on the vector $\overline{\mathbf{W}}' = (\alpha_1, \ldots, \alpha_m, \bar{W}_1, \ldots, \bar{W}_n) \in \mathbb{Z}^{m+n}$. Therefore, by Lemma 6, the vector $\overline{\mathbf{W}}'$ must be a unique linear combination of the vectors $(\mathbf{u}_1||\widehat{\mathbf{W}}_1, \ldots, \mathbf{u}_m||\widehat{\mathbf{W}}_m)$, that is,

$$\overline{\mathbf{W}}' = \sum_{i=1}^{m} \alpha_i (\mathbf{u}_i||\widehat{\mathbf{W}}_i),$$

and hence we have

$$\overline{\mathbf{W}} = \sum_{i=1}^{m} \alpha_i \widehat{\mathbf{W}}_i,$$

which contradicts our earlier assumption.

This means that if the signature on an aggregated masked-gradient returned by the server passes each client's checking, then the aggregated gradient is correct. In other words, any forged aggregated gradient-signature pair cannot pass each client's checking. □

6 Complexity Analysis

In this section, we provide a concrete complexity analysis of SVFL.

6.1 Performance Analysis of Client

Computation Cost. The computation cost of each client consists of: 1) masking the local gradient, which takes $O(n)$ time; 2) signing the masked gradient, which takes $O(n)$ time; 3) verifying the signature returned by the server, which takes $O(m+n)$ time; 4) unmasking the aggregated masked-gradient, which takes $O(n)$ time; 5) updating the model, which takes $O(n)$ time. Overall, each client's computation is $O(m+n)$. In addition, the client selected as the leader needs to initialize the system, which takes $O(n)$ time.

Communication Cost. Each client only needs to send a masked gradient and its signature to the server, which requires an overall communication cost of $n\lceil\log_2 R\rceil + \lceil\log_2 N\rceil$ bits (see the parameter setting in Theorem 9). Moreover, the client selected as the leader also needs to send the public/secret keys, m seeds, the initialized model parameters, the random string and m weights to all other clients, and send the public key, the random string and m weights to the server, which needs an overall communication cost of $n\lceil\log_2 R\rceil + 2(n+1)\lceil\log_2 N\rceil \quad +3\lceil\log_2\varphi(N)\rceil + (m+2)\lambda + 2m\lceil\log_2 B\rceil$ bits. Therefore, compared with plain FL, this implies a communication expansion factor of $(n\lceil\log_2 R\rceil + \lceil\log_2 N\rceil)/n\lceil\log_2 B\rceil$. Since $R = m(B-1)+1$ (see Theorem 9), for $B = 2^{32}$ (i.e., 32 bits), $n = 2^{20}$ elements, $N = 2^{1024}$, and $m = 100$ clients, the expansion factor is 1.21.

Storage Cost. Besides the local dataset, each client must store the model parameters, public/secret keys, m seeds, a random string and m weights, with a total storage cost of $n\lceil\log_2 R\rceil + (n+1)\lceil\log_2 N\rceil + 2\lceil\log_2\varphi(N)\rceil \quad +m\lceil\log_2 B\rceil + (m+1)\lambda$ bits.

6.2 Performance Analysis of Server

Computation Cost. The computation cost of the server[1] consists of: 1) aggregating all masked gradients, which takes $O(m+n)$ time; 2) computing an aggregated signature, which takes $O(m)$ time. Overall, the server's computation is $O(m+n)$.

Communication Cost. The server only needs to send the aggregated masked-gradient and its signature to each client, with an overall communication cost of $mn\lceil\log_2 R\rceil + m\lceil\log_2 N\rceil$ bits. Therefore, compared with plain FL, this implies a communication expansion factor of $(n\lceil\log_2 R\rceil + \lceil\log_2 N\rceil)/n\lceil\log_2 B\rceil$. For $B = 2^{32}$ (i.e., 32 bits), $n = 2^{20}$ elements, $N = 2^{1024}$, and $m = 100$ clients, the expansion factor is 1.21.

1. The process in which the server randomly selects a client as a leader does not involve a specific calculation, and the time required is quite small (it is just a matter of flipping a coin), which makes it negligible compared to other specific calculations (e.g., aggregation). Thus, we omit it.

Storage Cost. The server must store the public key, the random string and m weights, which is $(n+1)\lceil\log_2 N\rceil + \lceil\log_2\varphi(N)\rceil + m\lceil\log_2 B\rceil + \lambda$ bits, along with the masked gradients and their signatures buffer, which is $mn\lceil\log_2 R\rceil + m\lceil\log_2 N\rceil$ bits, such that the server can aggregate the masked gradients upon arrival.

7 Performance Evaluation

In this section, we evaluate the performance of SVFL in real scenarios. In particular, we train two practical ML models using SVFL and evaluate the accuracy of the results, as well as the incurred computational and communication overhead. The experimental results conform with the complexity analysis presented in Section 6.

7.1 Experimental Setup

Considering that secure aggregation and verification can be implemented as plug-in modules for SVFL, we run single-threaded simulations on a Linux virtual machine installed on a laptop with an Intel(R) Core(TM) i5-8250U CPU @1.60GHz and 16.0GB RAM. We implement SVFL in python on TensorFlow Federated [2] and conduct the experiments on: 1) a convolutional neural network, consisting of two 5×5 convolution layers, a fully connected layer with 512 units, and a softmax output layer, on the popular MNIST dataset, and 2) an AlexNet [31], consisting of five convolutional layers and three fully connected layers, on the popular CIFAR10 dataset. In all the experiments below, we randomly partition the training dataset over all clients; for example, assuming the number of clients is 10, for MNIST dataset with 70,000 images (60,000 training data and 10,000 test data) of 28×28 pixels, the client C_i may be randomly assigned 10,000 training data and 1,000 test data. In addition, we use SHA-256 hash to represent the hash function H and AES in counter mode to achieve PRG. We use the standard SGD algorithm for training, where the learning rate is set to 10^{-3} with a learning rate decay 10^{-6}, the batch size is 100, and the number of local epochs is 10.

7.2 Model Accuracy

Compared to the plain FL with the floating-point gradients, SVFL requires the floating-point gradient value to be represented as a 32-bit integer, which is the only factor that may lead to degraded model accuracy. Prior work [22] shows that deep networks can be trained using only 16-bit wide fixed-point number representation to achieve near-lossless accuracy. Therefore, from a theoretical point of view, the accuracy loss due to 32-bit integer representation of the gradient is negligible.

In this experiment, we simulate the training with 10 clients ($m = 10$) and compare the test accuracy of SVFL with that of the plain FL where no secure aggregation and verification are involved. Fig. 4 shows the comparison results. The experimental results demonstrate that the accuracy achieved by SVFL is very comparable to the ones achieved by the plain FL; reaching 96.75% as compared to 97.23% for MNIST dataset and 78% as compared to 79.2% for CIFAR10 dataset at the 500^{th} round.

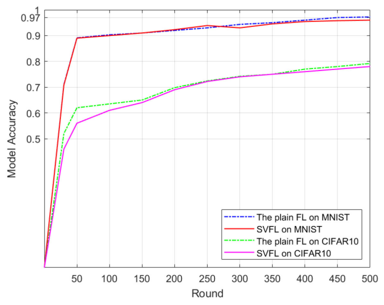


Fig. 4. Accuracy comparison between SVFL and the plain FL in [36].

7.3 Computational Overhead

We evaluate the computational cost of our SVFL on five processes: masking the gradients, signing the masked gradients, aggregating the masked gradients, unmasking the aggregated masked-gradients and verification. Note that we omit the running time for local model training and update (it depends on specific training models and the use of advanced optimizers such as Adam [28]), as this work focuses on secure aggregation and verification. Improving local and global models of training are beyond the scope of the current work.

Table 2 shows the running time per round for both a single client and the server. As illustrated in Table 2, masking and unmasking consume a very short time compared to that spent on signing and verification. Fig. 5 depicts the relationship between the total running time per round and the number of clients and the gradient vector size. As can be seen from Fig. 5, the running times of the client and the server increase linearly with both the number of clients and the gradient vector size. Also, the figure shows that the gradient vector size has a greater impact on the computation costs of both the client and server than the number of clients.

7.4 Communication Overhead

We evaluate our experiment on a 50Mbps communication channel. Table 3 shows the communication overhead and its corresponding time per round for both a single client and the server (to a single client). From Table 3, we observe that the communication costs of a single client and the server are almost the same. Fig. 6 describes the relationship between the total amount of data transferred per round and the gradient vector size and the number of clients. As shown in Fig. 6, the communication overhead per client increases linearly with the gradient vector size, but increases logarithmically with the number of clients. We do not plot the communication cost for the server, as it is essentially m times the communication cost of the client.

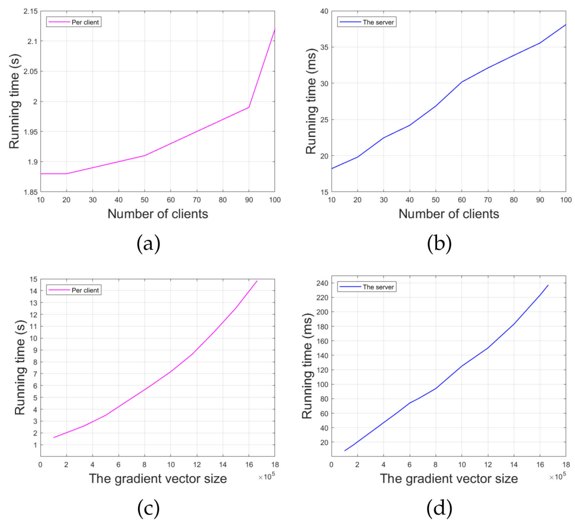


Fig. 5. Running time per round. (a) Running time per client, where the gradient vector size is fixed to 166.337K entries. (b) Running time for the server, where the gradient vector size is fixed to 166.337K entries. (c) Running time per client, where the number of clients is set to 10. (d) Running time for the server, where the number of clients is set to 10.

8 Performance Comparison With Similar Protocols

In this section, we compare the performance (computation and communication) of SVFL with that of BatchCrypt [63], VFL [17], VerifyNet [57] and VeriFL [21].

8.1 Comparison With Cross-Silo FL

By Table 1, we know that BatchCrypt [63] does not support correctness verification of the aggregated gradient, so we compare the computational costs of the masking, aggregation and unmasking (i.e., **Motp** scheme) of SVFL with the encryption, aggregation and decryption overhead (i.e., Pailler cryptosystem [41]) of BatchCrypt. In [63], the authors trained a 3-layer fully-connected neural network over FMNIST dataset [56] (with 101.77K gradients), an AlexNet model [31] over CIFAR10 dataset [30] (with 1.25M gradients), and a LSTM model [24] over Shakespeare dataset [3] (with 4.02M gradients). Thus, to compare SVFL and BatchCrypt in the same experimental environment, we evaluate the computational overhead of the masking, aggregation and unmasking when the gradient vector size is set to 101.77K, 1.25M, and 4.02M, respectively. As depicted in Fig. 7, the computational costs of the masking, aggregation and unmasking of SVFL are much lower than the encryption, aggregation and decryption of BatchCrypt. Specifically, the masking consumes 533-799 times less computation time than the encryption, the aggregation of SVFL consumes 5.7-17 times less computation time than the aggregation of BatchCrypt, and the unmasking

TABLE 2
Running Time Per Round

	Masking	Signing	Aggregation	Unmasking	Verification	Total
Client	2.64ms	1229.13ms	–	23.31ms	651.46ms	1906.54ms
Server	–	–	18.2ms	–	–	18.2ms

Each entry represents the average over 10 iterations. The number of clients is set to $m = 10$ and the gradient vector size is 166.337K entries. "Aggregation" includes aggregation of the masked gradients and combination of the signatures.

TABLE 3
Communication Overhead and Runtime Per Round

Num. Clients		Communication	Time
10	Client	0.7005MB	1.24s
	Server	0.7005MB	1.17s
100	Client	0.7664MB	1.46s
	Server	0.7664MB	1.38s

The gradient vector size is fixed to 166.337K entries. The time includes network latency.

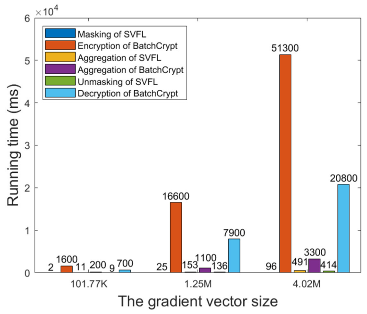


Fig. 7. Comparison of the computation overhead between SVFL and BatchCrypt.

consumes 49-77 times less computation time than the decryption. This means that SVFL (without verification) is at least 74-112.6 times faster than BatchCrypt. In fact, as shown in [63], for Paillier with 3074-bit key, the encryption time required to encrypt a 6.87MB plaintext is 3111.14s and the decryption time is 993.8s. Even with the maximum speedup of $93\times$ claimed in [63], it would take 33.45s of encryption time and 10.69s of decryption time. By contrast, for any 6.87MB plaintext, the execution time for the masking and unmasking of SVFL is 0.047s and 0.185s, respectively.

In terms of communication, BatchCrypt provides the network footprint incurred in one iteration, which is much greater than the communication overhead. Thus, we compare the communication cost between SVFL and BatchCrypt using an analytical method. As noted in [63], Pailler encryption produces substantially larger ciphertexts, which expands the amount of data transfer by more than $150\times$ compared to plain FL. In addition, compared with FATE [1], BatchCrypt reduces the network footprint by up to $101\times$ for LSTM model over the Shakespeare dataset, which is the best reduction that BatchCrypt could achieve. Thus, BatchCrypt requires at least $1.48\times$ more communication amount than plain FL. By contrast, by the theoretical analysis of communication cost given in Section 6, compared with plain FL, the communication expansion factor of SVFL is $1.21\times$ for both the client and the server. Therefore, SVFL requires less traffic than BatchCrypt.

Remark 14 (Security Comparison with HE-based FL). In Section 5.3, we described two types of attacks applicable to SVFL: one is due to the honest-but-curious clients, and the other is due to the honest-but-curious server, we then proved that our SVFL is secure against both attacks based on the security of the underlying masking with one-time pads. In comparison, the benchmarking HE-based FL protocol [45] (and other follow-up works, e.g. [12], [23], [53], [63]) only considered and proved the privacy against the honest-but-curious server, under CPA-secure (chosen plaintext attacks; we refer to [20] for its definition) of the underlying HE scheme. We remark that considering only honest-but-curious server is relevant to the scenario in which clients are considered as organizations such as hospitals and financial institutions under the jurisdiction of federal governments.

8.2 Comparison With Verifiable FL

By Table 1, VFL [17], VerifyNet [57] and VeriFL [21] support validation for the correctness of the aggregated gradient. As noted in **Related Work** (see Section 2), VFL, VerifyNet and VeriFL add the verification mechanism on top of [8], and they all use (double) masking and unmasking techniques similar to our **Motp** scheme for masking the local gradients. Therefore, we are only concerned with verification-related comparisons.

SVFL versus VFL. In VFL [17], the computational costs related to masking and verification are included in its encryption, verification and decryption, so we compare the runtime of the "masking"+"signing" and "verification"+"unmasking" of SVFL with the runtime of the encryption and "verification"+ "decryption" of VFL, respectively. Fig. 8 reports on the experimental results and shows that SVFL performs better than VFL.

On the other hand, the communication overhead of VFL increases linearly with the degree of the polynomial (VFL uses Lagrange interpolation to achieve the correctness verification of the aggregated gradient), which is associated with its security. By contrast, each client and the server of SVFL only need to send a signature, which is less than 1KB

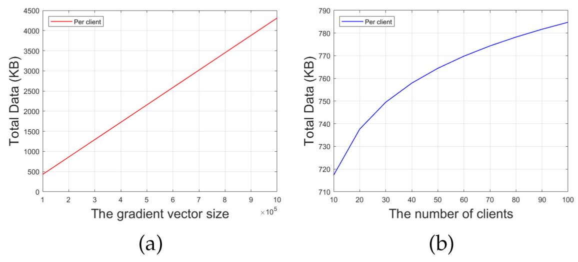


Fig. 6. Total data transfer per round. (a) Total data transfer, as the gradient vector size increases, where the number of clients is set to 10. (b) Total data transfer, as the number of clients increases, where the gradient vector size is fixed to 166.337K entries.

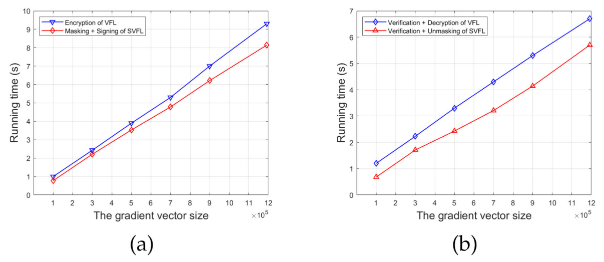


Fig. 8. Comparison of the computation overhead between SVFL and VFL, where the number of clients is fixed to 10.

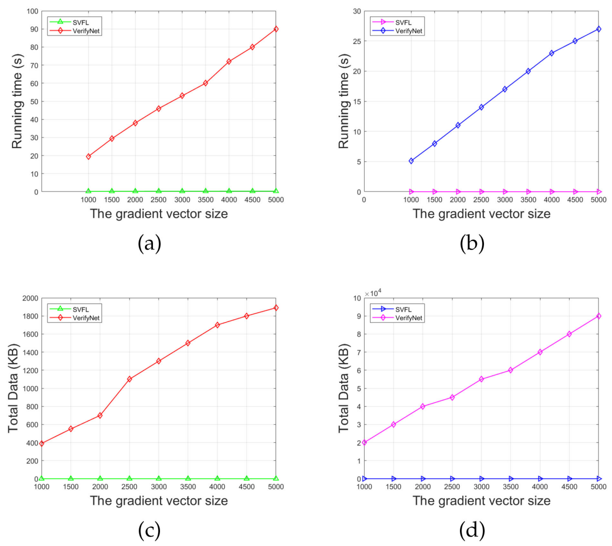


Fig. 9. Comparison of the computation and communication overheads between SVFL and VerifyNet, where the number of clients is fixed to 100. (a) Running time per round of the client. (b) Running time per round of the server. (c) Total data transfer for the client. (d) Total data transfer for the server.

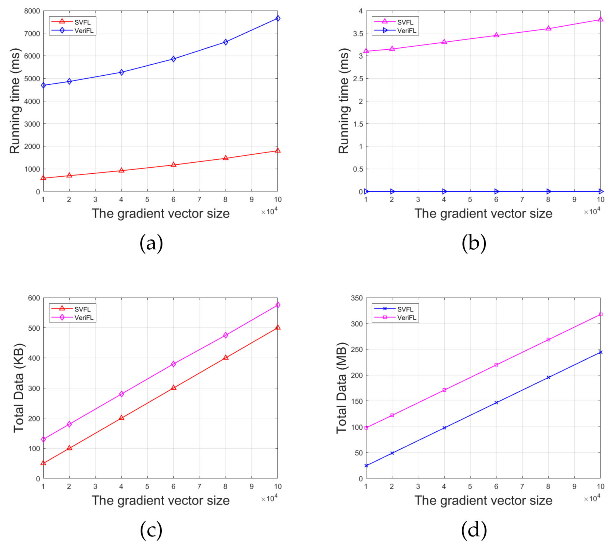


Fig. 10. Comparison of the computation and communication overheads between SVFL and VeriFL, where the number of clients is fixed to 500. (a) Running time per round of the client. (b) Running time per round of the server. (c) Total data transfer for the client. (d) Total data transfer for the server.

in size, regardless of the number of clients and the gradient vector size. Therefore, we conclude that SVFL requires a lower communication cost than VFL.

SVFL versus VerifyNet. Recall that in order to achieve the correctness verification of the aggregated gradient, the server needs to generate a "proof" and each client verifies the "proof". In VerifyNet [57], the author compared the verification costs of each client and the server (i.e., compute the "proof") to the total cost. Thus, we compare the verification overhead of the client ("signing"+"verification") and the server (computing an aggregated signature) of SVFL with the verification overhead of the client and the server of VerifyNet (no dropouts), respectively. In terms of communication, we compare the communication overhead associated with the verification of SVFL with VerifyNet. As shown in Fig. 9, SVFL is way superior to VerifyNet in terms of verification and communication overhead. This is mainly because VerifyNet relies on expensive bilinear operations, and the communication overhead of the client and the server in VerifyNet increase linearly with the gradient vector size. By contrast, in SVFL, each client only needs to generate a signature and verify an aggregated signature (where the size of any signature is less than 1KB regardless of the gradient vector size), while the server only needs to generate the aggregated signature; to maintain fair comparison, we do not consider mutual verification.

SVFL versus VeriFL. VeriFL [21] divides the aggregation and verification phases into four and three rounds respectively, among which we observe that the second and fourth rounds of the aggregation phase and all rounds of the verification phase are related to the correctness verification of the aggregated gradient. Therefore, we focus on these rounds and compare the computation and communication overheads associated with the verification of SVFL with VeriFL (no dropouts). As illustrated in Fig. 10, SVFL performs better than VeriFL in terms of verification efficiency including verification and communication overhead. This is mainly because in the verification process of VeriFL, each client needs to verify the commitments of all clients, which takes a considerable amount of time and increases linearly with the number of clients. By contrast, in SVFL, each client only needs to verify an aggregated signature smaller than 1KB in size, which is independent of the number of clients. Note that there is no verification overhead for the server in VeriFL when we do not consider dropouts; in comparison, the server in our SVFL needs to compute an aggregated signature for verification, but its computational cost is very small (see Fig. 10b).

9 Conclusion

In this paper, we proposed SVFL, a cross-silo federated learning protocol that provides secure aggregation and correctness verification of the aggregated gradient, while achieving both improved computation and communication overheads compared to existing cross-silo FL and verifiable FL protocols. We showed, by complexity analysis and extensive experimental evaluation, that the computation and communication overheads of our SVFL are very low, making it ideal for practical applications. The techniques we use in this work are implemented as pluggable modules, which can be truncated depending on different application scenarios and requirements. Thus, we believe that SVFL is suitable for the more practical case of non-i.i.d. data in the cross-silo setting. However, since this work mainly focuses on secure aggregation and verification, we leave it as future work to demonstrate how SVFL can be applied to scenarios with non-i.i.d. data in the cross-silo setting. In addition, SVFL can also be used in the cross-device federated learning setting, but it needs to rely on a TA. We believe it would be interesting to explore how to build a cross-device FL that supports secure aggregation and verification without relying on TA.

REFERENCES

[1] Fate (federated ai technology enabler), 2019. [Online]. Available: https://github.com/FederatedAI/FATE

[2] Tensorflow federated, 2019. [Online]. Available: https://www.tensorflow.org/federated

[3] Text generation with an RNN, 2019. [Online]. Available: https://www.tensorflow.org/tutorials/text/text_generation

[4] Webank, 2019. [Online]. Available: https://www.webank.com/en/

[5] M. Abdalla, D. Catalano, D. Fiore, R. Gay, and B. Ursu, "Multi-input functional encryption for inner products: Function-hiding realizations and constructions without pairings," in *Proc. Annu. Int. Cryptol. Conf.*, 2018, pp. 597–627.

[6] C. Beguier and E. W. Tramel, "Safer: Sparse secure aggregation for federated learning," 2020, *arXiv:2007.14861*.

[7] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly) logarithmic overhead," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2020, pp. 1253–1269.

[8] K. Bonawitz et al., "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1175–1191.

[9] E. Boyle, K. Chung, and R. Pass, "Large-scale secure computation: Multi-party computation for (parallel) RAM programs," in *Proc. 35th Annu. Cryptol. Conf.*, 2015, pp. 742–762.

[10] B. Choi, J. Sohn, D. Han, and J. Moon, "Communication-computation efficient secure aggregation for federated learning," 2020, *arXiv:2012.05433*.

[11] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Informat. Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976.

[12] Y. Dong, X. Chen, L. Shen, and D. Wang, "Eastfly: Efficient and secure ternary federated learning," *Comput. Secur.*, vol. 94, 2020, Art. no. 101824.

[13] Q. Dou et al., "Federated deep learning for detecting covid-19 lung abnormalities in CT: A privacy-preserving multinational validation study," *NPJ Digit. Med.*, vol. 4 no. 1, pp. 1–11, 2021.

[14] C. Dwork et al., "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, no. 3/4, pp. 211–407, 2014.

[15] H. Fereidooni et al., "SAFELearn: Secure aggregation for private federated learning (full version)," in *Proc. IEEE Secur. Privacy Workshops*, 2020, pp. 56–62.

[16] D. Fiore, R. Gennaro, and V. Pastro, "Efficiently verifiable computation on encrypted data," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 844–855.

[17] A. Fu, X. Zhang, N. Xiong, Y. Gao, H. Wang, and J. Zhang, "VFL: A verifiable federated learning with privacy-preserving for big data in industrial IoT," *IEEE Trans. Ind. Informat.*, vol. 18, no. 5, pp. 3316–3326, May 2022.

[18] R. Gennaro, J. Katz, H. Krawczyk, and T. Rabin, "Secure network coding over the integers," in *Proc. 13th Int. Conf. Pract. Theory Public Key Cryptography*, 2010, pp. 142–160.

[19] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 201–210.

[20] O. Goldreich, *The Foundations of Cryptography - Volume 2: Basic Applications*, Cambridge, U.K.: Cambridge Univ. Press, 2004.

[21] X. Guo et al., "VeriFL: Communication-efficient and fast verifiable aggregation for federated learning," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 1736–1751, 2021.

[22] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 1737–1746.

[23] M. Hao, H. Li, G. Xu, S. Liu, and H. Yang, "Towards efficient and privacy-preserving federated deep learning," in *Proc. IEEE Int. Conf. Commun.*, 2019, pp. 1–6.

[24] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[25] S. Kadhe, N. Rajaraman, O. O. Koyluoglu, and K. Ramchandran, "FastSecAgg: Scalable secure aggregation for privacy-preserving federated learning," 2020, *arXiv:2009.11248*.

[26] G. A. Kaissis, M. R. Makowski, D. Rückert, and R. F. Braren, "Secure, privacy-preserving and federated machine learning in medical imaging," *Nature Mach. Intell.*, vol. 2, no. 6, pp. 305–311, 2020.

[27] L. U. Khan, W. Saad, Z. Han, E. Hossain, and C. S. Hong, "Federated learning for internet of things: Recent advances, taxonomy, and open challenges," 2020, *arXiv:2009.13012*.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Representations*, 2015, pp. 101–112.

[29] J. Konecný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016, *arXiv:1610.05492*.

[30] A. Krizhevsky et al., "Learning multiple layers of features from tiny images," 2009.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Informat. Process. Syst. 26th Annu. Conf. Neural Informat. Process. Syst.*, 2012, pp. 1106–1114.

[32] V. Lampos, A. C. Miller, S. Crossan, and C. Stefansen, "Advances in nowcasting influenza-like illness rates using search query logs," *Sci. Rep.*, vol. 5, no. 1, pp. 1–10, 2015.

[33] S. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Trans. Inf. Theory*, vol. 49, no. 2, pp. 371–381, 2003.

[34] J. Ma, S. Naas, S. Sigg, and X. Lyu, "Privacy-preserving federated learning based on multi-key homomorphic encryption," 2021, *arXiv:2104.06824*.

[35] J. Stremmel and A. Singh, "Pretraining federated text models for next word prediction," in *Proc. Future Inform. Commun. Conf.*, 2021, pp. 477–488.

[36] H. B. McMahan, E. Moore, D. Ramage, and B. A. Y. Arcas, "Federated learning of deep networks using model averaging," 2016, *arXiv:1602.05629*.

[37] P. Mohassel and P. Rindal, "ABY3: A mixed protocol framework for machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 35–52.

[38] C. Mouchet, J. R. Troncoso-Pastoriza, and J.-P. Hubaux, "Multiparty homomorphic encryption: From theory to practice," *IACR Cryptol. ePrint Arch.*, vol. 2020, 2020, Art. no. 304.

[39] P. Mukherjee and D. Wichs, "Two round multiparty computation via multi-key FHE," in *Proc. 35th Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2016, pp. 735–763.

[40] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A. Sadeghi, "DÏoT: A federated self-learning anomaly detection system for IoT," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 756–767.

[41] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, 1999, pp. 223–238.

[42] J. Park and H. Lim, "Privacy-preserving federated learning using homomorphic encryption," *Appl. Sci.*, vol. 12, no. 2, 2022, Art. no. 734.

[43] A. Patra and A. Suresh, "Blaze: Blazing fast privacy-preserving machine learning," 2020, *arXiv:2005.09042*.

[44] C. Phillips, "The american solution of a german one-time-pad cryptographic system (G-OTP)," *Cryptologia*, vol. 24, no. 4, pp. 324–332, 2000.

[45] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Trans. Inf. Forensics Secur.*, vol. 13 no. 5, pp. 1333–1345, May 2018.

[46] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[47] S. Samarakoon, M. Bennis, W. Saad, and M. Debbah, "Federated learning for ultra-reliable low-latency V2V communications," in *Proc. IEEE Glob. Commun. Conf.*, 2018, pp. 1–7.

[48] S. Sav et al., "POSEIDON: Privacy-preserving federated neural network learning," in *Proc. 28th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1012–1026.

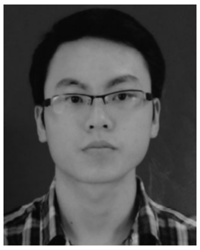
[49] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[50] M. J. Sheller et al., "Federated learning in medicine: Facilitating multi-institutional collaborations without sharing patient data," *Sci. Rep.*, vol. 10, no. 1, pp. 1–12, 2020.

[51] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 3–18.

[52] J. So, B. Güler, and A. S. Avestimehr, "Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning," *IEEE J. Sel. Areas Informat. Theory*, vol. 2, no. 1, pp. 479–489, Mar. 2021.

[53] S. Truex et al., “A hybrid approach to privacy-preserving federated learning,” in *Proc. 12th ACM Workshop Artif. Intell. Secur.*, 2019, pp. 1–11.
[54] S. Wagh, D. Gupta, and N. Chandran, “SecureNN: Efficient and private neural network training,” *IACR Cryptol. ePrint Arch.*, vol. 2018, 2018, Art. no. 442.
[55] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, “Falcon: Honest-majority maliciously secure framework for private deep learning,” 2020, *arXiv:2004.02229*.
[56] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms,” 2017, *arXiv:1708.07747*.
[57] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, “VerifyNet: Secure and verifiable federated learning,” *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 911–926, 2020.
[58] P. Xu, M. Hu, T. Chen, W. Wang, and H. Jin, “LaF: Lattice-based and communication-efficient federated learning,” *IEEE Trans. Inf. Forensics Secur.*, vol. 17, pp. 2483–2496, 2022.
[59] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, and H. Ludwig, “Hybridalpha: An efficient approach for privacy-preserving federated learning,” in *Proc. 12th ACM Workshop Artif. Intell. Secur.*, 2019, pp. 13–23.
[60] Q. Yang, Y. Liu, T. Chen, and Y. Tong, “Federated machine learning: Concept and applications,” *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 12:1–12:19, 2019.
[61] A. C. Yao, “Theory and applications of trapdoor functions (extended abstract),” in *Proc. 23rd Annu. Symp. Found. Comput. Sci.*, 1982, pp. 80–91.
[62] A. Yun, J. H. Cheon, and Y. Kim, “On homomorphic signatures for network coding,” *IEEE Trans. Comput.*, vol. 59, no. 9, pp. 1295–1296, 2010.
[63] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, “Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning,” in *Proc. USENIX Annu. Tech. Conf.*, 2020, pp. 493–506.
[64] N. Zhang, Q. Ma, and X. Chen, “Enabling long-term cooperation in cross-silo federated learning: A repeated game perspective,” *IEEE Trans. Mobile Comput.*, early access, Feb. 04, 2022, doi: 10.1109/TMC.2022.3148263.



Fucai Luo received the PhD degree in cyber security from the state key laboratory of information security, School of Cyber Security, University of Chinese Academy of Sciences, China, in 2019. He is currently a postdoctoral fellow with the Department of New Networks, Peng Cheng Laboratory, Shenzhen, China. His research interests are lattice-based cryptography, especially fully homomorphic encryption. His recent work has focused on secure multiparty computation, machine learning, and the connections between them.

Saif Al-Kuwari received the bachelor’s of Engineering in computers and networks from the University of Essex, UK in 2006, and two PhD’s from the University of Bath and Royal Holloway, University of London in Computer Science, both in 2011. He is currently an assistant professor with the College of Science and Engineering, Hamad Bin Khalifa University, and an adjunct research assistant professor with Kindi Center for computing research, Qatar University. His research interests include applied cryptography, quantum computing, computational forensics, and their connections with machine learning.

Yong Ding received the PhD degree in cryptography from the School of Communication Engineering, Xidian University, Xi’an, China, in 2005. From 2008 to 2009, he was a research fellow for Computer Science with the City University of Hong Kong. He is now a professor with the School of Computer and Information Security of Guilin University of Electronic Science and Technology. His current research interests include cryptography and information security.