FedTrust: Towards Building Secure Robust and Trustworthy Moderators for Federated Learning

Chih-Fan Hsu[†] Jing-Lun Huang[†] Feng-Hao Liu[‡] Ming-Ching Chang[*] Wei-Chao Chen[†]

[†] Inventec Corporation, Taipei 111, Taiwan
[‡] Florida Atlantic University, Boca Raton, FL 33431, USA
[*] University at Albany – State University of New York, Albany, NY 12065, USA

Abstract

Most Federated Learning (FL) systems are built upon a strong assumption of trust—clients fully trust the centralized moderator, which might not be feasible in practice. This work aims to mitigate the assumption by using appropriate cryptographic tools. Particularly, we examine various scenarios with different trust demands in FL, and then design the corresponding practical protocols with lightweight cryptographic tools. Three solutions for secure and trustworthy aggregation are proposed with increasing sophistication: (1) a single verifiable moderator, (2) a single secure and verifiable moderator, and (3) multiple secure and verifiable moderators, which can handle adversarial behaviors with different levels. We evaluate the performances of all our proposed protocols on the test accuracy and the training time, showing that our protocols maintain the accuracy with time overhead from 30% *to* 156% *depending on the secure and trustworthy levels. The protocols can be deployed in many practical FL settings with appropriate optimizations.*

1 Introduction

When data becomes the core to guarantee the performance of machine learning models to solve practical problems, allowing multiple parties with different data to jointly train a model becomes more attractive to communities. Federated Learning (FL) [10, 16] has shown growing success in decentralized multi-modal feature learning [16]. However, for the distributed nature of FL, security and privacy risks threaten the whole FL framework.

Most existing FL algorithms implicitly assume that the moderator (who aggregates gradients updated from clients) is trustworthy and honest and focus on the single moderator setting. However, requiring clients to completely trust the moderator implies significant vulnerabilities on the clients' side. Besides, the assumption is strong when considering data privacy from individual and business entities. It has been less studied how to protect the moderator against corruption or failure. In the former case, the moderator might not follow the prescribed aggregation procedure and thus result in a wrong model; in the latter case, the moderator might be disconnected/dropped out due to unstable networks or the running server. Moderator failure (even honest dropout) can abruptly stop the training process and then eventually damages the collaborative training.

The focus of this paper is to improve the *trustworthiness* and *robustness* of the moderator for the federated learning framework. We note that general cryptographic solutions and mechanisms (such as multi-party computation and verifiable computation [19, 7]) exist but they require heavy computation/communication overhead. Thus, a practical solution is not found in the literature. This work aims to identify suitable lightweight cryptographic protocols and evaluate their concrete performance/overhead in several FL settings with progressive secure levels. The result demonstrates the practicality of the protocols, which enhances the protection from the client's perspective and makes an important step toward developing a robust and trustworthy FL for current and future applications. The contribution of this paper is summarized as follows:

- We implemented several practical FL protocols to improve the security, trustworthiness and robustness of the FL moderator.
- We balance the usability and security of proposed protocols by using lightweight cryptographic tools.
- We evaluated the protocols on testing accuracy and training time overhead, which are commonly used benchmarks in implementation.

2 Background

We adopt several cryptographic tools to our solutions for building secure and trustworthy moderators. We first present the *additive homomorphic hash*, which is used to build a verifiable aggregation protocol at the moderator, such that clients can verify the correctness of gradient aggregation computation. We utilize *coin tossing*, a widely used tool in multi-party computation (MPC), to generate common random bits not controlled by any malicious user in MPC. We also use (robust threshold) *secret sharing* in the setting with multiple moderators. In the setting where the moderators are less honest, we use *Threshold Additive Homomorphic Encryption (TAHE)* to protect the privacy of users' gradients. We next briefly introduce necessary cryptographic tools.

Additively Homomorphic Hash (AHH). Let $H : \mathcal{X} \to \mathcal{Y}$ be some collision resistant hash function, *i.e.*, it is computationally infeasible to find data $x_1 \neq x_2$ such that $H(x_1) = H(x_2)$, where x_1 and x_2 are data. The hash is additively homomorphic for any $x_1, \ldots, x_k$, that is, $H(\sum_{i\in[k]} x_i) = \sum_{i\in[k]} H(x_i)$. The hash can be instantiated from the lattice problem Ring-SIS [11].

Particularly, let R be the cyclotomic ring with dimension N, where N is a power of 2, i.e., $R = \mathbb{Z}[x]/(x^N + 1)$, and $R_Q = R/QR$ for some modulus Q. Then we construct the AHH process by defining a function H which takes the input with domain $\mathcal{X} = \{x \in \mathbb{Z}^{N\ell} \subset R^\ell : \|x\|_\infty < \beta\}$ for some $\beta \in \mathbb{Z}$; and output with domain $\mathcal{Y} = R_Q^k$. The hash H description is a ring matrix $\mathbf{A} \in R_Q^{k\times\ell}$. On input $x \in \mathcal{X}$, where x can be interpret as a ring vector in R^ℓ, $H(x)$ outputs $h = \mathbf{A}\cdot x \mod Q$. The output h is the hash.

Verifiable Aggregation Protocol (VAP). By using AHH, we can design a verifiable aggregation protocol against a malicious FL moderator. Consider the scenario that n honest clients $C_1, \ldots, C_n$ who hold (non-private) inputs $x_1, \ldots, x_n$ cooperate with the moderator M who computes the aggregation of all the inputs in a verifiable way. We next describe the proposed verifiable aggregation protocol.

- Each client C_i first broadcasts data hash $h_i = H(x_i)$ to all clients and the moderator, then, sends data x_i to the moderator M.
- The moderator broadcasts the aggregated result $z = \sum_{i\in[n]} x_i$ to all clients.
- Each client checks $\sum_{i\in[n]} h_i = H(z)$ to determine whether z is the correct aggregated result.

Coin Tossing. In many distributed protocols, it is important for parties to generate common random strings where no adversarial parties can bias the outcome. A coin tossing protocol [5] can achieve this goal. In practice, one can use the blockchain as a source of a random string, and each party just agrees on reading at a particular location on the blockchain.

Secret Sharing. An (n, t) secret sharing scheme consists of two algorithms, Share and Recon, where Share(x) distributes the input x into n shares such that only when t of them are collected can Recon() recovers the input x. Otherwise, x is information-theoretically hidden. Shamir's secret sharing scheme [12] achieves the goal based on polynomial evaluation and interpolation.

Table 1: Our protocols for handling different FL scenarios.

	Non-private Aggr.	Private Aggr.
Single M	SVM	SSVM
Multiple Ms	SVM (Extended)	MSVM

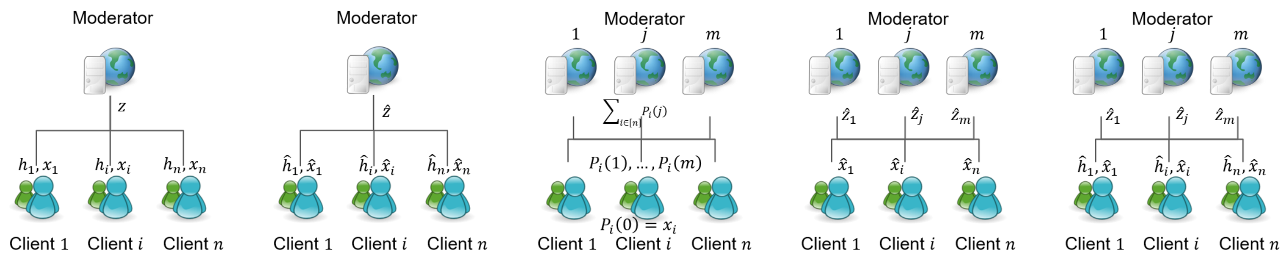
Threshold Additive Homomorphic Encryption (TAHE). A TAHE scheme consists KeyGen protocol, Enc(), and Dec() algorithms. The KeyGen protocol generates a common public-key pk and shares of the corresponding secret-key sk to every party. Enc(x, pk) encrypts x to the ciphertext $\hat{x}$. Dec$(\hat{x}, \mathsf{sk})$ decrypts $\hat{x}$ to obtain x, where the functionality works only when every party holding the share of sk participates. This primitive can be implemented from some protocols in prior works, such as [8].

3 Trustworthy Federated Learning

We present multiple protocols that are suitable for various scenarios with different security guarantees, using the tools described in Section 2. Particularly, we consider two important design demands—(1) whether the clients' gradients need to be kept private, and (2) whether more moderators are available during the FL, as summarized in Table 1. Based on the considerations, we design different protocols.

Several critical security requirements must be satisfied for trustworthy FL, namely reliability and robustness for moderators and privacy for clients, with the following goals: (1) reliable aggregated result: how to verify the correctness of the aggregated result at the client, which is related to the concept of *verifiable computing*; (2) client privacy: how to protect each client's gradients, which is related to *data encryption*; and (3) robust moderators: how to proceed with the computation when a moderator drops out or fails during training, which is related to the concept of *MPC*. Specifically, we propose protocols to address the following three scenarios as follows:

- **A Single Verifiable Moderator (SVM) in** §**3.1**. Our goal in SVM is to check the correctness of the aggregated result for a moderator. Here, we consider a simpler case where the clients' gradients are non-private. This method can be extended to the setting with multiple moderators, as clients simply run the single moderator protocol with each of the moderators. As long as one of the moderators does not drop out during FL training, the clients can receive the expected aggregation outcomes, implying a robust FL for non-private gradients.
- **A Single Secure and Verifiable Moderator (SSVM) in** §**3.2**. This scenario is similar to SVM, except the clients'



(a) SVM (b) SSVM (c) MSVM-2/3 (d) MSVM-half (e) MSVM-one

Figure 1: Five investigated scenarios for achieving secured and verifiable federated learning. We show the data transmitted between the moderators and the clients to highlight the difference.

gradients are private and should be protected, for achieving a verifiable and private aggregation.

- **Multiple Secure and Verifiable Moderators (MSVM) in** §3.3. MSVM focuses on the robust moderators, where the aggregation protocol cannot be disrupted by failures from the moderator. To achieve this goal, we decentralize a single moderator into multiple moderators so that they can run some distributed protocol to prevent the disruption caused by moderator failure.

Figure 1 summarizes the transmitted data between the moderator(s) and the clients of all scenarios.

3.1 A Single Verifiable Moderator

Scenario. Consider the case where there is only one (perhaps malicious) moderator and n honest clients $\{C_i\}_{i\in[n]}$ who hold **non-private** inputs $\{x_i\}_{i\in[n]}$. The clients want to perform a verifiable FL training aggregation update via the moderator to obtain the verified aggregated result z.

Solution. This can be achieved simply by using VAP, where C_i uses the quantized gradients as the input x_i in the protocol. The gradient quantization is used to fulfill the basic requirement $x_i \in \mathbb{Z}$. Specifically, at the beginning of federated training, the moderator generates an AHH function H and broadcasts H to all clients. For each training iteration, the clients negotiate a zero-point and a scale to quantize the gradients to certain bits to generate the input x_i. Then, each client verifies the aggregated result z transmitted from the moderator with H and VAP.

3.2 A Single Secure and Verifiable Moderator

Scenario. We extend the SVM scenario with **private** inputs $\{\hat{x}_i\}_{i\in[n]}$ held on clients. The clients want to perform a verifiable FL training via the moderator, without revealing their private inputs.

Solution. At the beginning of training, the clients first run the KeyGen protocol from a TAHE, and obtain a common public pk and individual shares of secret keys sk_i. Then the clients and the moderator run VAP, where each client C_i transmits input $\hat{x}_i \leftarrow \mathsf{Enc}(x_i, \mathsf{pk})$ to the moderator, calculates the hash with $\hat{x}_i$, and then receives $\hat{z}$ from the moderator. The clients run the decode protocol to obtain $\mathsf{Dec}(\hat{z}, \mathsf{sk}_i)$ if the consistency check passes, namely, $H(\hat{z}) = \sum_{i\in[n]} \hat{h}_i$. Otherwise, the clients abort by catching the cheating behavior from the moderator.

3.3 Multiple Secure and Verifiable Moderators

In SVM and SSVM, we assume that a single moderator will always complete the protocol. We relax this condition by considering how to design a robust protocol against possible moderator failure (either maliciously or by random faults). To achieve robustness, we introduce *redundant computation* into the federated learning. Namely, we decentralize the single moderator into m moderators, *e.g.*, $\{M_j\}_{j\in[m]}$. As in the prior sections, we assume that the clients are honest. In the following, we present three protocols against different corrupted patterns of the moderators.

3.3.1 At Least $\frac{2}{3}m$ Honest Moderators (MSVM-2/3)

Scenario. Assuming that at least $\frac{2}{3}m$ honest moderators will complete the protocol, and the other $\frac{1}{3}m$ moderators might be either corrupted or dropped out at any moment.

Solution. We can run the classic BGW protocol [3], using Shamir's secret sharing. Specifically,

- Each client C_i generates a polynomial P_i with degree $\frac{2}{3}m - 1$ and $x_i = P_i(0)$. Then, C_i sends $P_i(j)$ to moderator M_j by $\mathsf{Share}(x_i)$.
- Each M_j sends $p_j = \sum_{i\in[n]} P_i(j)$ to all clients.
- Each client checks the receives $(p_1, \ldots, p_m)$ to reconstruct the polynomial P by $\mathsf{Recon}(p_1, \ldots, p_m)$. By the homomorphic and robustness properties of the Shamir's scheme, the aggregated result z can be calculated with $z = P(0)$, as long as $2/3$ receives are computed correctly.

We note that this protocol is information-theoretically secure, and thus no cryptographic tool is needed.

3.3.2 More than $\frac{1}{2}m$ Honest Moderators (MSVM-half)

Scenario. We further relax the $\frac{2}{3}$ condition to $\frac{1}{2}$. We assume the existence of $\frac{1}{2}m + 1$ honest moderators.

Solution. We solve the scenario by the TAHE scheme. Specifically,

- The clients first run the KeyGen protocol and obtain a common public key pk and individual shares of the secret key sk_i.
- Each client C_i sends $\hat{x}_i = \mathsf{Enc}(x_i, \mathsf{pk})$ to all moderators.
- Each M_j broadcasts $\hat{z}_j = \sum_{i \in [n]} \hat{x}_i$.
- Each client receives $(\hat{z}_1, ..., \hat{z}_m)$ and checks majority consistency of $\hat{z}$. All clients run the protocol $\mathsf{Dec}(\hat{z}, \mathsf{sk}_i)$ to obtain the aggregated result z.

From the assumption on an honest majority of the servers, the majority vote suffices to guarantee the correct sum of the ciphertexts. Thus, there is no need for VAP.

3.3.3 At Least One Honest Moderator (MSVM-one)

Scenario. Finally, we consider the worst case where only one honest moderator can be guaranteed.

Solution. We propose a solution combined with verifiable computing and the TAHE protocol. The solution is similar to that in Section 3.3.2, except that the clients run the VAP with each moderator M_j. Specifically,

- Each client C_i sends $\hat{x}_i = \mathsf{Enc}(x_i, \mathsf{pk})$ to all moderators and $\hat{h}_i = H(\hat{x}_i)$ to all clients.
- Each M_j broadcasts $\hat{z}_j = \sum_{i \in [n]} \hat{x}_i$.
- Each client receives $(\hat{z}_1, \ldots, \hat{z}_m)$ and performs hash consistency check if $H(\hat{z}_j) = \sum_{i \in [n]} \hat{h}_i$ is true to every j, where $\hat{z}$ is the first ciphertext passed the hash consistency check. All clients run the protocol $\mathsf{Dec}(\hat{z})$ to obtain the aggregated result z.

4 Experiment Results

We implement and evaluate all proposed protocols and compare them with several baseline methods including (1) centralized training (Cen), (2) quantized centralized training (Cen (Q)), (3) federated training (FL), and (4) quantized federated training (FL (Q)). We report the evaluation using two performance metrics: test accuracy and the time for a training iteration. All the scenarios and cryptographic tools are implemented by Python3.8.10+cu11.5 on a server with two Intel 6238T CPU @ 1.90GHz, 16GB RAM, and a Tesla T4 GPU. We select a common dataset, CIFAR-10, to test our implementation, which is easy to adapt to other datasets. We use the original training and test splits to train and validate the neural network, respectively, for a four-client FL. Specifically, we use the Dirichlet Distribution with the skewness parameter 0.1 to distribute the samples to the clients. Figure 2 shows the detail structure of client model, which contains 1,250,858 trainable parameters.

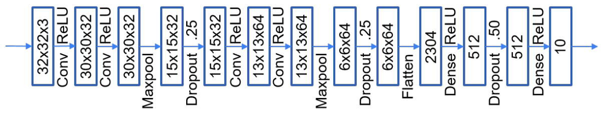


Figure 2: The detail structure of the client model. The number in the box shows the size of the tensor.

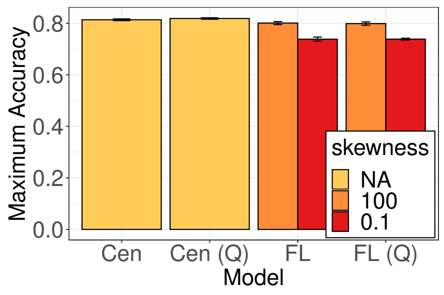


Figure 3: Max test accuracy of the neural network models.

Implementation Details. For the Cen (Q) and FL (Q), we quantize the model weights from `float32` to `int8` after the client finishes the training at each iteration. For FL (Q), we introduce a process to exchange the maximum and minimum values of the client weights to negotiate a range for weight quantization. For the followed experiments, we set the number of clients to four $n = 4$ and the number of moderators to three $m = 3$. We use common parameters of the cryptographic tools. For the additively homomorphic encryption scheme, we implement the BFV [1] scheme with ring dimension $N = 2048$, plaintext modulus $p = 524287$, and ciphertext modulus $Q = 2^{56}$. For AHH, we use the following parameter sets: for non-encrypted values, $N = 1024$, $k = 2$, $\ell = 1222$, $\beta = 2^8$, $n = 2^{10}$, and $Q \approx 2^{62}$; for encrypted values (or un-encrypted bigger values), $N = 4096$, $k = 2$, $\ell = 611$, $\beta = 2^{40}$, $n = 2^8$, and $Q \approx 2^{62}$. These schemes are at least 128 bit-security. We adopt the `Numpy.polymul` function to calculate the polynomial multiplication, which is heavily used in the AHH and encryption/decryption. The time complexity of the current implementation is $O(N^2)$, where N is the ring dimension, and can be improved to $O(N \log N)$ by the Fast Fourier Transform.

Testing Accuracy. Figure 3 shows the average maximum test accuracy of the investigated methods from five separated experiments with different seeds. The interval on each bar indicates the standard deviation of the five experiments. Overall, the maximum accuracy is stable among experiments. The experiment results follow our expectations. We can observe that (1) the weight quantization does not largely impact the accuracy; (2) the accuracy of the FL algorithms is slightly lower than the centralized training;

and (3) the level of data skewness does impact the accuracy and the parameter value and the accuracy have a positive correlation. We note that no weight bias is introduced by the proposed protocols. Hence, the test accuracies are highly similar to the accuracy of the FL (Q) experiment with skewness parameter 0.1. Specifically, the testing accuracy of SVM, SSVM, MSVM-half, and MSVM-one are 0.742 (0.009), 0.741 (0.009), 0.736 (0.002), and 0.734 (0.002), respectively. The values in the parentheses are the standard deviation of the experiments unless otherwise specified. We did not measure the accuracy of the MSVM-2/3 protocol because of the unreasonable long training time. Details are discussed in the next section.

Training Time. For the MSVM-2/3 protocol, according to the $(m,\ 2/3m)$-Shamir's secret sharing, a gradient generates a polynomial P with degree $2/3m-1$. We extend the degree of polynomial from 1 to 3 because the one-degree polynomial is relatively small due to $m=3$. Then, we measure the time for generating shares and reconstructing P for one gradient, which takes 0.00032 (0.00010) and 0.34 (0.006) second, respectively. In our FL setting, each client transmits $1,250,858$ gradients. A training iteration takes 4.92 days even if clients run parallel, which is impossible to wait for the model converge. Therefore, we report the theoretical time for a training iteration.

Table 2 reports the time and the time overhead compared to the centralized training for each procedure in the protocol. The dash symbol denotes that the protocol does not include the procedure. We omit the standard deviation to fulfill the space limitation because the values are usually very small. The train time includes the time for performing the forward and backward processes to update the neural network model. The consistency (Con.) check time includes the time for hash consistency check, majority consistency check, and the equality check for the aggregation. Generally, the time usages for the four baselines are highly similar. The train time of the protocols is slightly different due to the computation variance although all experiments were conducted on the same device. Weight quantization and dequantization only take a few milliseconds without large accuracy degradation, which can be considered in the security protocol design. The verifiable computing process (AHH and Con. check) increases about four seconds for an iteration, which increases about 30% per iteration training time compared to the centralized training. If the weights are encrypted, the verifiable process takes additional time according to the increased size of the ciphertext compared to the plaintext. Overall, enabling the homomorphic addition to secure weights largely increases the training time. The time can be reduced by the hardware support and the better implementation. The aggregation time increases in the multiple moderator scenarios but the increment is relatively smaller than the time for the gradient protection. The hash consistency check greatly increases in the MSVM-one experiments, which takes about 8.5 seconds (2 checks), because we random the honest moderator during training. The average time for checking one $\hat{z}$ and the corresponding hash takes about 4.2 seconds. The hash consistency check of the MSVM-half experiment is small because checking the majority consistency of $\hat{z}$ is rapid. Overall, protecting gradients greatly increases the training time. The time of the verifiable computing only depends on the size of the aggregation inputs. Overall, the time overhead is reasonable, which shows the practicality for verifying the correctness of the moderators.

5 Related Work

Federated Learning (FL) has been extensively studied [10] and is still in active development. Most existing FL systems operated with a single centralized moderator. We survey a new FL that works with more than one moderator. The multi-center FL [17] learns multiple global models and simultaneously derives the optimal matching between users and centers using a multi-center aggregation mechanism. We note that our proposed multiple decentralized moderators can achieve a similar goal but we focus more on security and trustworthiness. The central-server-free FL of [9] builds upon the framework and theoretical guarantees in the generic social network scenario, where the trust is unidirectional. Their analysis shows results on how users can benefit from communication with trusted users in the FL scenario. In comparison, our proposed FedTrust does not rely on the trust of the FL client in the moderator, while the FL can still perform securely and robustly.

Secure Aggregation. SecAgg [4] is a failure-robust FL secure aggregation protocol based on pairwise additive masking and Shamir's secret sharing, and with significant computation cost. SecAgg+ [2] reduces the communication costs by using a sparse communication graph with a logarithmic degree. By using a probabilistic argument, SecAgg+ [2] proves that their protocol achieves strong guarantees for privacy and resilience against user dropouts. The hybrid method of [15] is based on threshold homomorphic encryption, while HybridAlpha [18] is based on functional encryption. Both above methods are composed of differential privacy and Secure Multiparty Computation; they rely on a trusted party for key distribution. TurboAgg [13] achieves a secure aggregation overhead of $O(n \log n)$ as opposed to $O(n^2)$ by adding aggregation redundancy via Lagrange coding and can handle a user dropout rate of 50%. The method of [6] is based on designing the topology of secret sharing nodes as a sparse random graph for efficient secure aggregation. The LightSecAgg [14] changes the design from the random-seed reconstruction of the dropped users to the one-shot aggregation-mask reconstruction of

Table 2: Average computational time and overhead (per iteration in seconds) of training the network of 1,250,858 parameters.

Exp	Train	Aggregate	Quant.	Dequant.	AHH	Con. Check	Encrypt	Decrypt	Overhead
Cen	12.584	–	–	–	–	–	–	–	0%
Cen (Q)	12.530	–	0.004	0.002	–	–	–	–	-0.3%
FL	12.245	0.004	–	–	–	–	–	–	-2.7%
FL (Q)	12.215	0.002	0.003	0.004	–	–	–	–	-2.9%
SVM	11.995	0.001	0.003	0.004	2.166	2.165	–	–	29.8%
SSVM	13.141	0.027	0.002	0.005	4.360	4.362	4.216	2.094	124.2%
MSVM-half	13.128	0.078	0.002	0.005	–	0.057	4.217	2.105	55.7%
MSVM-one	13.036	0.077	0.002	0.017	4.356	8.465	4.211	2.105	156.4%

the active users via mask encoding/decoding. This method achieves the same privacy and dropout-resiliency guarantees while significantly reducing the overhead. To the best of our knowledge, all existing secure aggregation works are single-sided and focus on handling unreliable client dropouts. FedTrust is the first study on relaxing the trust of moderators from the clients and handling moderator robustness and security with verifiable computation.

6 Conclusion

This paper aims to mitigate the Federated Learning assumption where the aggregation relies on a trustworthy moderator. We investigate multiple scenarios, from a single and verifiable moderator to multiple secure and verifiable moderators, comprehensively exploring the implicit issues between clients and moderators. Specifically, we proposed different protocols for various scenarios using cryptographic tools related to verifiable computing, data encryption/decryption, and multi-party computation. Experiment results show the efficacy of the proposed protocols on testing accuracy and training time. With increasing demands of the data size for heterogeneous multimedia applications, we expect the proposed protocols can be a start to improve the robustness and privacy levels of FL.

References

[1] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

[2] J. Bell, K. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova. Secure single-server aggregation with (poly)logarithmic overhead. In *ACM SIGSAC CCS*, 2020.

[3] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC*, pages 1–10. ACM, 1988.

[4] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In *ACM SIGSAC*, 2017.

[5] R. Canetti. Universally composable security. *J. ACM*, 67(5):28:1–28:94, 2020.

[6] B. Choi, J. Sohn, D. Han, and J. Moon. Communication-computation efficient secure aggregation for federated learning. In *arXiv:2012.05433*, 2021.

[7] S. Gordon, J. Katz, F. Liu, E. Shi, and H. Zhou. Multi-client verifiable computation with stronger security guarantees. In *TCC LNCS 9015*, pages 144–168, 2015.

[8] S. Gordon, F. Liu, and E. Shi. Constant-round MPC with fairness and guarantee of output delivery. In *CRYPTO*, volume 9216 of *LNCS*, pages 63–82. Springer, 2015.

[9] C. He, C. Tan, H. Tang, S. Qiu, and J. Liu. Central server free federated learning over single-sided trust social networks. In *arXiv 1910.04956*, 2019.

[10] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawit, Z. Charles, G. Cormode, and *et al.* *Advances and Open Problems in Federated Learning*. now publishers inc., 2021.

[11] C. Peikert. A decade of lattice cryptography. *Found. Trends Theor. Comput. Sci.*, 10(4):283–424, 2016.

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

[13] J. So, B. Guler, and S. Avestimehr. Turbo-Aggregate: Breaking the quadratic aggregation barrier in secure federated learning. *IEEE JSAIT*, 2:479–489, 2021.

[14] J. So, C. He, C. Yang, S. Li, Q. Yu, R. Ali, B. Guler, and S. Avestimehr. LightSecAgg: a lightweight and versatile design for secure aggregation in federated learning. In *MLSys*, 2022.

[15] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou. A hybrid approach to privacy-preserving federated learning. In *arXiv:1812.03224*, 2019.

[16] O. Wahab, A. Mourad, H. Otrok, and T. Taleb. Federated machine learning: Survey, multi-level classification, desirable criteria and future directions in communication and networking systems. *IEEE Commun. Surv. Tutor.*, 23(2):1342–1397, 2021.

[17] M. Xie, G. Long, T. Shen, T. Zhou, X. Wang, J. Jiang, and C. Zhang. Multi-center federated learning. In *arXiv:2005.01026*, 2021.

[18] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, and H. Ludwig. HybridAlpha: An efficient approach for privacy-preserving federated learning. In *ACM Workshop on Artificial Intelligence and Security*, pages 13–23, 2019.

[19] S. Zhou, M. Liao, B. Qiao, and X. Yang. A survey of security aggregation. In *ICACT*, pages 334–340, 2022.