Efficient Verifiable Protocol for Privacy-Preserving Aggregation in Federated Learning

Tamer Eltaras, Farida Sabry, *Member, IEEE*, Wadha Labda, Khawla Alzoubi, and Qutaibah Malluhi, *Member, IEEE*

***Abstract*— Federated learning has gained extensive interest in recent years owing to its ability to update model parameters without obtaining raw data from users, which makes it a viable privacy-preserving machine learning model for collaborative distributed learning among various devices. However, due to the fact that adversaries can track and deduce private information about users from shared gradients, federated learning is vulnerable to numerous security and privacy threats. In this work, a communication-efficient protocol for secure aggregation of model parameters in a federated learning setting is proposed where training is done on user devices while the aggregated trained model could be constructed on the server side without revealing the raw data of users. The proposed protocol is robust against users' dropouts, and it enables each user to independently validate the aggregated result supplied by the server. The suggested protocol is secure in an honest-but-curious environment, and privacy is maintained even if the majority of parties are in collusion. A practical scenario for the proposed setting is discussed. Additionally, a simulation of the protocol is evaluated, and results demonstrate that it outperforms one of the state-of-art protocols, especially when the number of dropouts increases.**

***Index Terms*— Federated learning, privacy-preserving, secure aggregation, verifiable aggregation, cloud computing.**

I. Introduction

FEDERATED learning has been actively researched in the last five years [1], [2] as a collaborative way to perform machine learning tasks between many clients, possibly mobile devices, without the data leaving the clients to preserve their privacy. The service provider in this setting just orchestrates many clients, receives local models' parameters, then updates, and ensures the validity of the global aggregated model whether it is a deep-learning model [1], a tree-based model [3], [4] or any other model type.

Federated learning (FL) faces many challenges [2], such as devices' heterogeneity, limited resources, availability, and communication overhead. Besides that, federated learning faces some security and privacy threats from external malicious actors that might do model update poisoning or traditional data poisoning. A wide range of different threats and attacks are reviewed in [5] and [6]. Mitigating these kinds of poisoning attacks is challenging to distinguish between honest and malicious updates [7], [8]. An adversarial server also represents a threat to a federated learning setting as a curious actor at the server side might do reverse engineering for the local model's parameters received at the server to reveal private data.

In order to do this aggregation in a secure manner, researchers studied numerous possibilities [9], including using homomorphic encryption, differential privacy, secure multi-party computation protocols [10], and trusted execution environments [2].

Each of these techniques has its own pros and cons. While homomorphic encryption provides a private solution by aggregating encrypted data from clients, prohibiting the server from reverse-engineering the model's weights or discovering training data, it is computationally expensive and impractical for the majority of applications [11].

It also does not consider the problem of dropouts when some devices drop out of the network due to connectivity problems or battery power outages. Using differential privacy alone doesn't consider the dropouts as well and faces the privacy-utility trade-off challenge but can operate in the shuffled model where a trusted third party shuffles the noisy client updates before forwarding them to the server [12].

Research in secure multi-party computation protocols and their applications for have started a long-time ago [10], [13]. However, state-of-the-art secure multi-party computation protocols based on secret sharing for federated learning [14], [15], [16] still faces challenges with the communication overhead incurred in the aggregation and verification process. They also cannot accommodate the cross-silo setting where client devices do the local training, but these clients belong to different organizations with the requirement to ensure their clients' privacy. Examples of this scenario include healthcare organizations, different banks, or multiple operating vehicular ad hoc networks (VANETs) [17] scenarios. The proposed protocol addresses these two issues for secure multiparty computation of the gradient vector. In the next Section, the contributions of our research work are emphasized.

Manuscript received 5 July 2022; revised 21 November 2022 and 10 March 2023; accepted 18 April 2023. Date of publication 8 May 2023; date of current version 15 May 2023. This work was supported in part by the Qatar National Research Fund (QNRF) under Project ECRA 01-006-1-001 and in part by the Qatar National Library. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Edgar Weippl. *(Corresponding author: Tamer Eltaras.)*

Tamer Eltaras, Farida Sabry, Wadha Labda, and Qutaibah Malluhi are with the Computer Science and Engineering Department, Qatar University, Doha, Qatar (e-mail: tamer.taras@qu.edu.qa).

Khawla Alzoubi is with the Engineering Technology Department, Community College, Qatar University, Doha, Qatar.

Digital Object Identifier 10.1109/TIFS.2023.3273914

A. Our Contributions

Our protocol primarily addresses the fundamental security challenges in federated learning: the confidentiality of local gradients and the verifiability of aggregation through incorporating auxiliary nodes that represent organizations such as hospitals, banks, or VANETs. These auxiliary nodes participate in the protocol to help in keeping the client nodes' data private; further explanation will be provided in section V. The contributions of this research work to the area of privacy-preserving and verifiable aggregation depending on secure multi-party computations can be summarized as follows:

- An efficient, verifiable privacy-preserving aggregation algorithm is proposed. It relies on lightweight primitives.
- To ensure the confidentiality of the user's local gradients, a single-masking protocol is used in our scheme instead of a double-masking protocol used in most literature work.
- For the verification of the aggregated result, we use the concept of double aggregation, which is very lightweight in computation compared to cryptographic primitives used in most of the research work in the literature.
- The algorithm is compared in terms of communication, computation, and storage complexities to existing algorithms.
- Performance evaluation and analysis of the proposed algorithm with changing the number of auxiliary nodes and dropout percentage is presented.

We are mainly focused on the setting where several organizations are cooperating, and each organization has several users. The organizations need to collaborate to train a global model on all user's private data without violating the privacy of the users' data either by the organization they are affiliated with or by other organizations. Our scheme allows each organization to participate in the protocol as an auxiliary node, which lets each organization guarantee its users' privacy.

B. Organization

The rest of the paper is organized as follows. Section II presents related work in the literature. In Section III the background needed for the approach in the paper is briefly discussed. Section IV presents an application scenario for the proposed protocol in healthcare domain and illustrates the system architecture of the proposed scheme and its threat model. Section V explains the protocol steps and handling of dropouts. It also addresses the verifiability of the aggregated result. Security analysis for the protocol is presented in section VI. Evaluation of the performance of the proposed algorithm is discussed in section VII. Finally, the paper is concluded in section VIII. To make the paper easy to follow, we summarize the mathematical symbols and notations used in the paper in Table I.

TABLE I
MATHEMATICAL NOTATIONS AND THEIR DESCRIPTIONS

Notation	Description
N	set of the user's nodes.
M	set of the auxiliary nodes.
U_i	set of online users participating in the training round i
pk_n	public key for node n
sk_n	secret key for node n
$s_{m,n}$	the agreed key shared between user n and auxiliary node m
P_m	sum of random variables at auxiliary node m
x_n	private model parameters vector to share at node n
V	length of parameters vector x_n
α_m	a random value generated by the auxiliary node m
α	a universal key shared by all the users
$K_{n,m}$	a shared key between user n and auxiliary node m
K_m	the sum of the shared keys held by auxiliary node m
K	the sum of all the keys held by the auxiliary nodes
MAC	message authentication code
X	final aggregated vector
R	range from which input vector values are sampled

II. RELATED WORK

Our research contributes to two areas; secure aggregation and verification of server work in federated learning. In this section, we briefly review recent related research work in these two areas.

A. Secure Aggregation

Secure aggregation in federated learning (FL) refers to the aggregate computation of the sum of local models' parameters updates in a secure way without learning any information about the personal private data that produced these parameters. This has been done in the literature in various ways that differ in terms of computation complexity, communication latency, and how they deal with the problem of dropout nodes which is a common problem in federated learning settings. This research area has been actively researched in the last five years. In this section, only a limited number of examples of the research work using differential privacy, homomorphic encryption, secret sharing, and other secure multi-party computation techniques are reviewed.

1) Using Differential Privacy (DP): The authors in [18] proposed using a local differential privacy mechanism to update the local weights of a deep neural network adapting to the varying ranges of weights at different layers. They used parameter shuffling aggregation to bypass the curse of dimensionality to avoid privacy budget explosion. In a similar way, the authors in [19] used local differential privacy to add noise to the local models' parameters before aggregation. They analyzed the compromise between convergence performance and privacy protection levels. They showed that increasing the number of users participating in FL can increase the model convergence and emphasized the trade-off between the model convergence and the privacy-protection level. Applying local differential privacy at the local models has the advantage of less communication time needed as only the differentially private local model parameters are exchanged. However, this approach requires a large number of participating users, and it isn't evaluated for the dropouts effect. To alleviate some of these problems, Kairouz et al. [20] proposed adding discrete Gaussian noise before performing secure aggregation and after discretization of the user model updates. The authors in [21] depended on a distributed Laplace perturbation mechanism which is more efficient in terms of noise generation time. A problem with the approach in [20] is that privacy guarantees degrade as the dropped-out users increase. In [22], the

authors combine the addition of Gaussian noise with a learning with errors (LWE)-based masking protocol that substantially reduces the communication complexity required to add large vectors. The authors in [23] also achieved low communication overhead with a training mechanism that requires flexible participation of clients. In [19], the authors used differential privacy to protect privacy by adding artificial noise to parameters at the client's side before aggregation. The study explored the relationship between convergence performance and levels of privacy protection. In [24], a comparison was made between FL and local differential privacy in terms of efficiency and privacy loss. However, the performance of applying local differential privacy to FL was not investigated. The work in [25] introduced a local differential privacy FL framework for industrial-grade text mining, demonstrating that it could provide data privacy and model accuracy. In [26], the authors describe a hybrid approach that combines differential privacy and SMC to achieve a balance between accuracy and vulnerability to inference attacks. The goal is to address the potential for low accuracy when using differential privacy and the vulnerability to inference associated with SMC.

2) Using Homormorphic Encryption: Homomorphic encryption (HE) has been actively researched for use in multiparty computation for deep learning tasks and then in federated learning [27], [28], [29], [30], [31], especially after succeeding in supporting approximate arithmetic over encrypted data [32], which means users can send their gradients encrypted to be added while keeping it private.

Phong et al. [33] used additively homomorphic encryption in asynchronous stochastic gradient descent training for a neural network. Truex et al. [27] combined additively homomorphic encryption (HE) with DP, but their approach cannot handle client dropouts. Using HE results in a significant runtime overhead which can be seen as impractical for real-world FL. Using a batch encryption technique, BatchCrypt [28] reduces the encryption and transmission overhead of HE-based aggregation and only requires a single round of communication. To safeguard model parameters, The authors in [29] proposed using (HE) approach that can directly execute arithmetic operations on ciphertexts without decryption. Based on a lightweight symmetric homomorphic encryption, the authors in [30] proposed an efficient and verifiable cipher-based matrix multiplication algorithm to ensure training security in a completely decentralized framework. In [34], the authors proposed a federated learning approach that prioritizes privacy using a multi-key homomorphic encryption protocol. The approach encrypts model updates with an aggregated public key before aggregating them on the server. Decryption requires collaboration from all participating devices, preventing unauthorized access to the participants' data. The authors of [35] combined ternary gradients federated learning with secret sharing and homomorphic encryption techniques to develop privacy-preserving protocols to protect against semi-honest adversaries. However, the computational burden of HE renders it inapplicable for real-world training with FL and negatively affects scalability.

3) Using Secret Sharing: Bonawitz et al. [36] presented FL's secure aggregation. Their protocol can withstand client dropouts. To prevent access to local models, they employed blinding with random values, Shamir's Secret Sharing (SSS), and symmetric encryption. However, their aggregation needs at least four communication cycles every iteration between each client and the aggregator. This imposes a severe burden on clients with limited resources and WAN connections. VerifyNet [15], and VeriFL [14] modified the protocol of Bonawitz et al. [36]. Authors in [15] added verifiability on top of the protocol in [36] to guarantee the correctness of the aggregation, and in [14], the authors reported optimization of the communication and computation overhead in case of a large number of dropouts as it is always the case in a federated learning setting. However, these protocols rely on a trusted party to generate public/private key pairs for all clients. SAFE-Learn [37] introduced a generic design for efficient private aggregation for FL to overcome the aforementioned problems since their proposed protocol needs only two communication rounds in each iteration, it does not rely on expensive cryptographic primitives on client devices, and there is no need to trust a third party. The authors in [38] mixed masking using random keys while supporting quantization-based model compression to boost communication efficiency. They relied on hardware-assisted trusted execution environments (TEE) for verification which requires extra costs. Blockchain technology can be used to secure federated learning and introduce device and model trust as demonstrated in [39] and [40]. In [41], the authors proposed a secure aggregation protocol that is robust to client dropouts using a novel multi-secret sharing scheme based on Fast Fourier Transform (FFT). A new framework for secure aggregation was introduced in [42], which uses a multi-group circular strategy and additive secret sharing for model aggregation.

Our proposed protocol belongs to this category, but it uses lightweight primitives and single masking protocol, as will be discussed in detail in section V, without depending on TEE as in [38], or the need to trust a third party as in [14] and [36], or the use of expensive cryptographic primitives as in [36].

B. Verification

As the service provider may return incorrect results to the users either deliberately or due to unexpected situations, it is recommended that client devices have the ability to verify the aggregated model parameters sent by the service provider. The authors of VerifyNet [15] proposed that the server the aggregated result together with a proof to each client device. They utilized homomorphic hash function and pseudorandom generation to provide verifiability for each user. Modifications to this technique were done in [14] and [16] to decrease communication overhead and computational complexity, respectively. However, these techniques [14], [15] were analyzed in a recent publication [43], and it was pointed out that they still face some security vulnerabilities if the server colludes with a malicious user. In [43], the authors used linear homomorphic hash and digital signature for achieving traceable verification for the aggregation results and identifying the epoch at which the results went wrong but at the cost of increasing communication overhead. Luo et al. [44] used a basic signature method for the problem of verification

where each client only needs to verify an aggregated signature which is independent of the number of clients. Each client then unmasks the aggregated gradient, updates the parameters of its local model, and proceeds to the next iteration. It was claimed in [30] that integrity verification is guaranteed for every model training step using their aggregation method. Differently, SafetyNets [45] used interactive proof techniques to verify the accuracy of the aggregated result supplied by the server. In work [46], a verifiable system is offered to perform verification, similar to [38], based on trusted hardware such as SGX, TrustZone, and Sanctum. However, these techniques provide a limited number of activation functions or demand additional hardware.

III. Preliminaries

To make the article easy to follow, we explain some cryptographic primitives used in our approach, which should facilitate understanding the proposed protocol.

A. Key Agreement

A key agreement algorithm allows any party u to combine their private key s_u^{SK} with the public key s_v^{PK} for any party v to obtain private shared key $s_{u,v}$ between u and v. We use the Diffie-Hellman key agreement in our protocol to generate the shared key (seed) between each user and each auxiliary node. Specifically, given a group G with prime order q, where g is the generator of group G, each user can agree with each auxiliary node on a secret share as follows:

- Each user chooses a secret key U_{SK} and generate its public key as $g^{U_{SK}} modp$ and shares the public key with the server.
- Each auxiliary node chooses a secret key A_{SK} and generates its public key as $g^{A_{SK}} modp$ and share the public key with the server.
- The server broadcasts the public keys to the parties.
- The shared key is now computed as $key = (g^{A_{SK}})^{U_{SK}} modp = (g^{U_{SK}})^{A_{SK}} modp$

B. Symmetric Encryption

Symmetric encryption is the traditional algorithm that uses only one key for encryption and decryption. Given the key SK and the information x to be encrypted, the encrypted information is obtained by the algorithm $\textbf{AE.enc}(SK, x) \rightarrow \hat{X}$. The ciphertext $\hat{X}$ can be decrypted by the algorithm $\textbf{AE.dec}(SK, \hat{X}) \rightarrow x$. In our model, we use symmetric encryption to communicate the messages between auxiliary nodes and users through the server without the server violating the confidentiality of the messages. We rely on this technique to avoid making private channels between each user and each auxiliary node to exchange the messages. We encrypt the messages, send them to the server, and broadcast them to the users.

C. Pseudo-Random Generator

We employ a secure Pseudo random generator **PRG** that takes a seed and produces a random number as an output. The **PRG** has to preserve two properties:

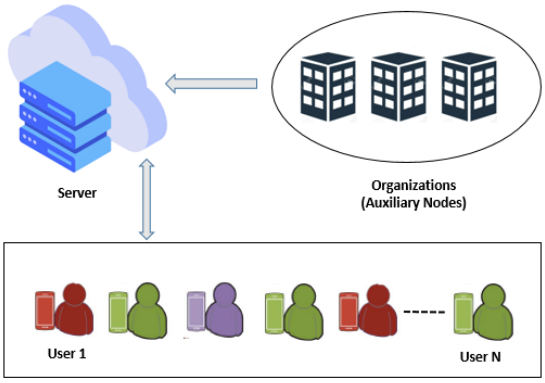


Fig. 1. System architecture.

- The output must be computationally indistinguishable from a uniform element sampled from the output space as long as the seed is hidden from the distinguisher.
- The same exact output is generated using the same seed.

IV. System Architecture

A. General System Architecture and Threat Model

As shown in Figure 1, our system model consists of three entities, auxiliary nodes, users, and the server.

- Auxiliary Nodes: These nodes are a set of nodes that can't all collude together and are keen on the privacy of the data of the users so that the server can't reveal the machine learning local model parameters and analyze them to learn about the users' training data. These auxiliary nodes can represent organizations such as hospitals or medical entities running research in the healthcare domain or banks in the banking domain. They are robust and don't participate in any training process. Their main job is to agree with the users on shared keys used as seeds for generating the random numbers used in masking the gradients. In each protocol round, each auxiliary node agrees with each user in the system on a distinct secret random key. This key agreement would typically happen without directly communicating with users. Each auxiliary node would then compute the sum of all the random numbers at its disposal and send the summation to the server.
- User: Each user sends its local gradients to be aggregated securely at the server without revealing these gradients, as they can be used in a reverse engineering setting to trace back the users' data. The users will use the random numbers shared with the auxiliary nodes to mask their private gradients. Finally, each client verifies that the server has computed the correct aggregation.
- Server: The cloud server aggregates the masked gradients uploaded by all online users and the summation of all the random numbers uploaded by the auxiliary nodes. As a result, the server will aggregate all the local gradients without revealing each user's gradient.

The following are the assumptions in our threat model:

- All participants will follow the protocol steps, but they may try to infer other users' private data.

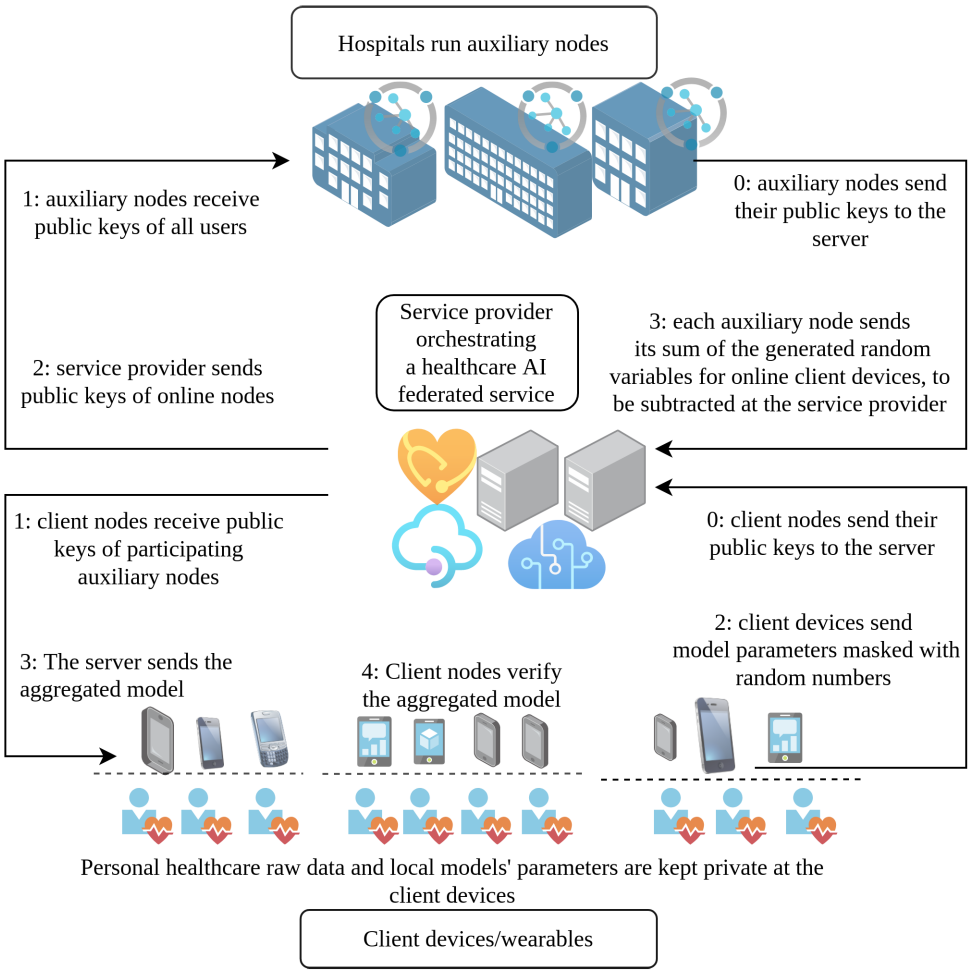


Fig. 2. Aggregation protocol steps and actors in healthcare application scenario: Setup phase (steps 0 and 1), Learning phase (steps 2 and 3), and Verification (step 4).

- The server could collude with up to $|N| - 2$ users.
- The server could collude with up to $|M| - 1$ auxiliary nodes.
- The server could return a modified version of the aggregated result to the users.

The protocol aims to protect the confidentiality of users' local gradients while enabling each user to verify the aggregated result returned by the server. Using our protocol, for the server to violate a user's privacy, it has to collude with all the auxiliary nodes or collude with $|N| - 1$ users where $|N|$ is the number of the users.

B. Healthcare Application Scenario

In the healthcare domain, AI applications that are provided by different service providers are facing challenges and receiving critiques on the privacy of patients' or users' data [47], which is stored and kept at the service provider to learn from. Federated learning [48] was proposed to solve this problem to help learn a model from users' data without the data leaving the users' devices to assure its privacy. Since then, research has been going to improve issues with the security of the federated learning approach, such as the problem discussed in this paper of having a server that will aggregate the local model updates and send an updated global model but may be curious to analyze the gradients and parameters received from the clients to learn more about their personal data. The proposed scheme presented in section V sends these gradients with certain random perturbations to the service provider with the help of auxiliary nodes as will be specified. Additionally, verifying the aggregated model by the server is a challenge as the server may return incorrect results to reduce its costs or to have an edge over other competitors.

An example scenario in the healthcare domain where the proposed scheme can fit is shown in Figure 2. An application service provider for an AI application/study, in the middle of the figure, orchestrates the federated learning process for learning from private health data such as different biosignals, personal attributes, and possibly medications at the client devices. In this paper, we will use the words; users and client devices interchangeably. All participants (service providers, hospitals engaged in the study/application, and users through their client devices, "cellular phones/wearable devices") follow the protocol correctly. However, there are some threats that the proposed protocol can handle, such as possible reverse engineering of the models' updates at the service provider to infer users' private data, the service provider may provide

wrong information about dropout users, and it may attempt to manipulate the aggregated result.

In this scenario, it is assumed that some hospitals can collude with each other, e.g., they could belong to the same entity (university/research institute/project), but collusion cannot involve all. Similarly, the service provider may collude with some hospitals participating in a funded research project, for example, and with some users/client devices but not with all.

Each hospital will run one or more auxiliary nodes to exchange the seeds and generate random numbers for all the client devices through the server. At step 0, the auxiliary nodes, as well as the client devices, send their public keys to the service provider as shown in Figure 2 with the step numbers for the messages exchanged following the round numbers in which the message is sent in the proposed protocol; Protocol I. The service provider, in turn, broadcasts the keys of the clients to all auxiliary nodes and all keys of auxiliary nodes to all the clients in step 1. The client devices mask the model parameters by adding random numbers to them to keep them private. The masked models' parameters are sent to the service provider in round 2. The service provider, in turn, requests auxiliary nodes to send the sum of random numbers for participating client devices. Each auxiliary node will then send the sum of random numbers which it generated to the service provider. The service provider aggregates the resulting model parameters by adding all the data received from client devices and subtracting all the numbers received from the auxiliary nodes (unmasking) in round 3. The client devices will then verify the aggregated results in round 4.

V. Proposed Scheme

In this section, we present the technical details of the proposed protocol. From a high-level view, the protocol aims to solve three problems that exist in the federated learning process by:

- Protecting the user's privacy that may be leaked from the user's local gradients.
- Eliminating the effect of the dropout of users during the training process.
- Enabling users to verify the result computed by the server.

The process starts after each user trains the model locally by their private dataset. Each user has to upload its local gradients to be aggregated by the server. The aggregation has to be done in a secure and private manner such that the local gradients can't be revealed to any party, even to the aggregator. In this protocol, the idea of masking to hide the local gradients of the users is adopted. Each user will add a set of random numbers to its gradients before sending them to the server. For generating and handling these random numbers, the protocol relies on a setting where a set of nodes called auxiliary nodes are used. These auxiliary nodes have two main jobs:

- Each auxiliary node agrees with each user on a shared key which will be used as a seed for generating synchronized random numbers. Therefore, starting with this seed, in the same round, both parties generate the same random values.
- Each auxiliary node helps the server aggregate the gradients by providing the server with the required masks to cancel all the random values that have been added to the gradients.

The complete protocol steps are listed in Protocol I, and an illustration of the data kept at each participating party and how it is aggregated at the server is shown in Figure 3. In the beginning, each auxiliary node and each user generate three key pairs $\{(pk_n^1, sk_n^1), (pk_n^2, sk_n^2), (pk_n^3, sk_n^3)\}$ and $\{(pk_m^1, sk_m^1), (pk_m^2, sk_m^2), (pk_m^3, sk_m^3)\}$respectively and send their public keys to the server. Each user can agree with each auxiliary node on three shared keys by these keys. The first key is used in encryption and decryption, the second is used as a common seed for generating synchronized random numbers, and the third is used in the process of verification. Each user uses the second key as a seed to generate random values to mask its local gradients. Each auxiliary node has only a part of the mask that each user adds to its local gradients. The set of keys can be viewed as a matrix where each auxiliary node creates a column in this matrix, and each user takes a row from that matrix. Therefore, none of the auxiliary nodes can reveal the mask of any of the users. Each user uploads its masked gradients to the server and each auxiliary node uploads the summation of its generated random numbers. By aggregating all these values at the server, all the masks will be canceled, and the server will be able to get the right aggregated result of the actual local gradients of the users.

A. Protecting the User's Local Gradients

Assume that the number of the users is $|N|$, the number of the auxiliary nodes is $|M|$, and the number of online users that participate in the current round is $|U|$ where each user $n \in U$ has a unique ID known to both the server and the auxiliary nodes. Each user $n \in U$ holds a private gradient x_n and needs to hide it from all other parties. Each auxiliary node $m \in M$ will agree on a shared key with each user $s_{n,m}$ (the number $s_{n,m}$ is the shared key between the auxiliary node number m and the client number n). By these seeds, in every round i each client n and each auxiliary node m generate an agreed random number denoted as $\boldsymbol{PRG}(s_{n,m}, round(i))$. Hence, each user can encrypt their local gradient as follows.

$$\hat{x_n} = x_n + \sum_{m=1}^{M} \boldsymbol{PRG}(s_{n,m}) \tag{1}$$

Also, each auxiliary node sum all the random numbers at its disposal as follows.

$$P_m = \sum_{n=1}^{N} \boldsymbol{PRG}(s_{n,m}) \tag{2}$$

Then, each user submits their encrypted gradient $\hat{x_n}$ to the server, and each auxiliary node submit the sum of its random numbers P_m to the server. The server can aggregate all the local gradients $X = \sum_{n=1}^{N} x_n$ as follows.

$$X = \sum_{n=1}^{N} x_n = \sum_{n=1}^{N} \hat{x_n} - \sum_{m=1}^{M} P_m \tag{3}$$

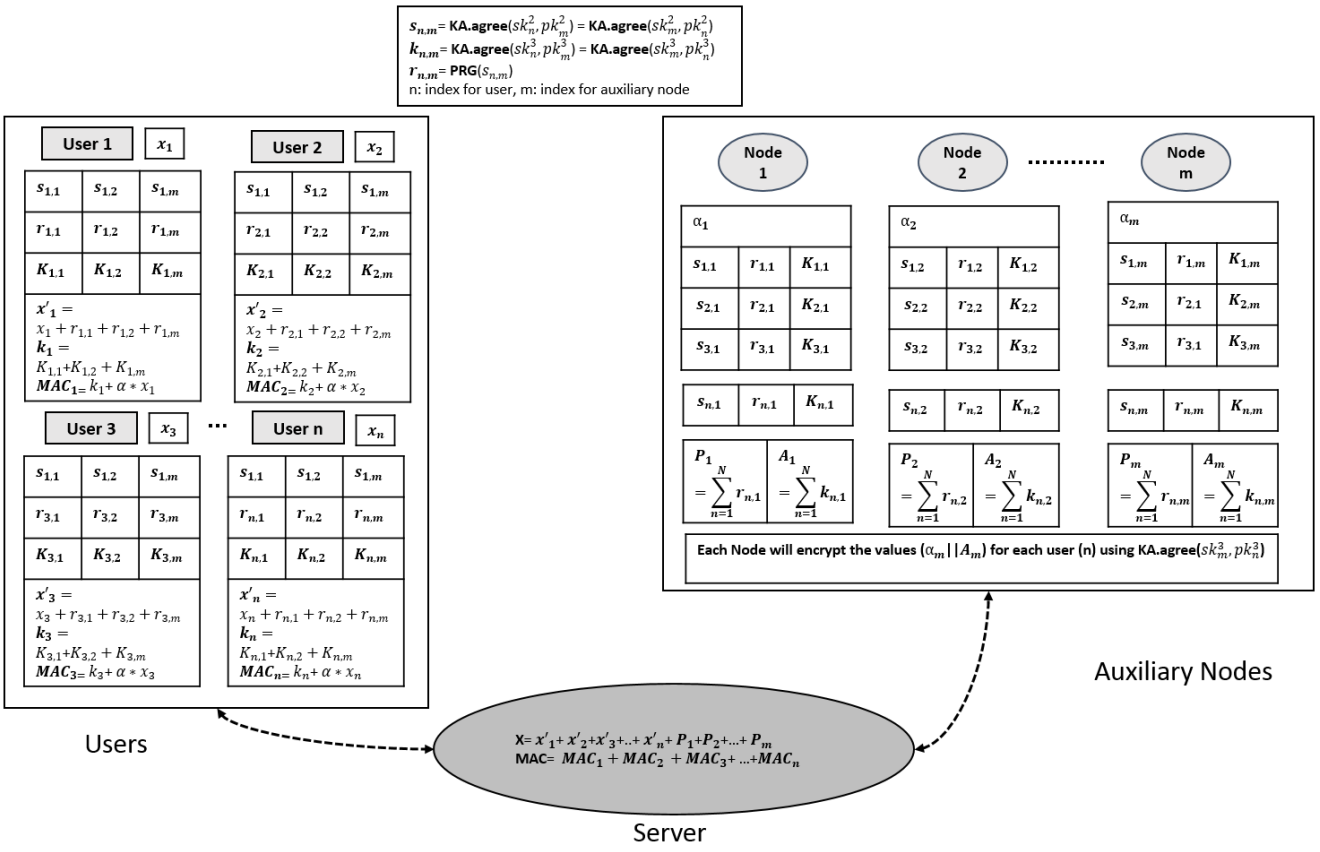


Fig. 3. Illustration for the data view at participating parties in the protocol and its aggregation at the server.

In equation (3) all the random numbers cancel each other, and the gradients will be aggregated. To illustrate this equation, we can think that we have a matrix of random numbers, where each auxiliary node holds a column in the matrix, and each row is shared with a specific user. So the summation of the columns $\sum_{m=1}^{M} P_m$ cancel the summation of the rows $\sum_{m=1}^{M}\sum_{n=1}^{N} \boldsymbol{PRG}(s_{n,m})$, and only the gradients $\sum_{n=1}^{N} x_n$ remain.

B. Handling the Dropouts

Our protocol will handle the dropouts by default. Firstly, the server will receive the gradients from the users. Then, after some specific time, the server will ask the auxiliary nodes to upload the randomness corresponding to users who send their gradients. Even if some users upload their gradients late, the server will not reveal their private data as the users' private data is still masked with the randomness values, but those users will be excluded from participating in this round.

C. Verifiability

To enable each user to verify the result returned by the server, we rely on the concept of double aggregation. The first aggregation is used to compute the aggregated gradient, whereas the second is used to demonstrate the correctness of the first.

Each user will agree with each auxiliary node on a shared key $K_{n,m} \leftarrow \textbf{KA.agree}(sk_n^3, pk_m^3)$ and compute the summation of these keys to generate their key denoted as $K_n \leftarrow \sum_{m\in M} k_{n,m}$. Also, each auxiliary generates a shared key with each user $K_{n,m} \leftarrow \textbf{KA.agree}(sk_m^3, pk_n^3)$ and compute the summation of these keys to generate their key denoted as $K_m \leftarrow \sum_{n\in n} k_{n,m}$. It is obvious that the summation of $\sum_{n=1}^{N} K_n \rightarrow \boldsymbol{K}$ is equal to the summation of $\sum_{m=1}^{M} k_m \rightarrow \boldsymbol{K}$. Each auxiliary node has to share the value K_m with all the users. Each auxiliary node will also sample a random number α_m and share it with all users. The summation of $\sum_{m=1}^{M} \alpha_m \rightarrow \boldsymbol{\alpha}$ will be used as a universal key. The way of sharing both α_m and K_m is through the server, as there are no channels between users and the auxiliary nodes. Each auxiliary node will encrypt the values α_m and K_m for each user using a shared agreed key $ct_{n,m} \leftarrow \textbf{AE.enc(KA.agree}(sk_m^1, pk_n^1), \alpha_m||K_m)$ then send the value to the server. The server will forward the cipher texts to corresponding users. Each user will decrypt the received ciphertexts as $\alpha_m||K_m \leftarrow \textbf{AE.dec(KA.agree}(sk_n^1, pk_m^1), ct_{n,m})$.

So each user computes K and α as follows:

$$\boldsymbol{\alpha} = \sum_{m=1}^{M} \alpha_m, \tag{4}$$

and

$$\boldsymbol{K} = \sum_{m=1}^{M} K_m \tag{5}$$

Then, each user computes its mask **MAC** as follow:

$$MAC_n = K_n + \alpha \times x_n \tag{6}$$

Besides the final result X, The server aggregate all the **MAC**s and sends the result $MAC = \sum_{n=1}^{N} MAC_n$ to each user.

So each user n can validate the result by verifying the below equation:

$$MAC - K - \alpha \times X = 0 \tag{7}$$

Each user can easily verify equation (7) as they receive the universal key α and the final key K from the auxiliary nodes and receive the result X and the key MAC from the server.

VI. Security Analysis

In this section, the security of the proposed protocol is analyzed to show how it preserves the privacy of each user's local gradients. The protocol security is ensured in the presence of at least one auxiliary node and two honest clients. As shown above, each user masks its local gradient as

$$\hat{x_n} = x_n + \sum_{m=1}^{M} \boldsymbol{PRG}(s_{n,m}) \tag{8}$$

The security of the above equation relies on a lemma that says if uniformly random numbers are added to users' values, the resulting values look uniformly random.

To prove that $\hat{x_n}$ provides a sufficient level of security, we first introduce some notations. We will consider that a server S interacts with a set of U of n users, and the underlying cryptographic primitives are instantiated with the security parameter k. We will use the symbol U_i to denote for the set of users that successfully upload their local gradients in round $i-1$, such that $U_4 \subseteq U_3 \subseteq U_2 \subseteq U_1 \subseteq U$ as the users may drop out at any point during the execution.

Given a subset $W \subseteq U \cup S$ of parties, the joint view of all parties in W can be seen as a random variable $\mathbf{REAL}_W^{u,k}(x_U, U_1, U_2, U_3, U_4)$, where k indicate the security parameter used in the protocol. The view of a party during the protocol execution consists of its input, randomness, and all messages this party received from other parties. Once the party aborts and stops receiving messages, its view remains with the last message received. Here we are going to discuss the security of our protocol under collusion between the parties. First, we introduce a theorem that shows that any collusion between the users and the auxiliary nodes will not violate the privacy of other users' private data.

Theorem 1 (Defense Against Joint Attacks From Multiple Users and Multiple Auxiliary Nodes): For all $k, W \subseteq U, x_U$, and $U_4 \subseteq U_3 \subseteq U_2 \subseteq U_1 \subseteq U$ there is a Probabilistic-Polynomial-Time (PPT) simulator SIM whose output is indistinguishable from the output of $\mathbf{REAL}_W^{u,k}$.

$$\mathbf{REAL}_W^{u,k}(x_U, U_1, U_2, U_3, U_4) \equiv \mathbf{SIM}_W^{u,k}(x_U, U_1, U_2, U_3, U_4)$$

Proof: The joint view of the parties in W does not depend on the user's inputs that are not in W because the server's view is removed (users do not share their inputs with auxiliary nodes). One way to achieve a perfect simulation is to let the simulator run the honest but curious users on their genuine inputs and the rest on fake input. We emphasize that the simulated view of users in W is identical to the output of the real view. For the honest users (not in W), the simulator uses random values instead of the gradients to compute the masked value $\hat{x_n}$. The parties in set W will not be able to identify which values have been used by other parties. The server just sends the list of the online user's participating in the round of masking to the auxiliary nodes and the final aggregation to the users. Therefore, the simulated view of parties in W is indistinguishable from the output of real view $\mathbf{REAL}_W^{u,k}$. □

Theorem 2 (Defense Against Joint Attacks From the Cloud Server, Multiple Users, and Multiple Auxiliary Nodes): For all $k, W \subseteq U \cup S, x_U$, and $U_4 \subseteq U_3 \subseteq U_2 \subseteq U_1 \subseteq U$ there is a PPT simulator SIM that its output is indistinguishable from the output of $\mathbf{REAL}_W^{u,k}$.

$$\mathbf{REAL}_W^{u,k}(x_U, U_1, U_2, U_3, U_4) \equiv \mathbf{SIM}_W^{u,k}(x_U, U_1, U_2, U_3, U_4)$$

Proof: To prove the above theorem, we use a standard hybrid argument. The idea behind this approach is to start from the actual view and then execute a series of secure modifications in the condition that any two subsequent variables are computationally indistinguishable, which ultimately makes a simulated view indistinguishable from the real view.

Hyb0 This random variable represents the joint view of the parties in W in a real execution of the protocol.

Hyb1 In this hybrid, we fix a specific user $\acute{n} \in \{U_3 \setminus W\}$. For the honest auxiliary nodes $m \in \{U_3 \setminus W\}$, the simulator replaces the operation of generating the shared key between each auxiliary node $m \in \{U_3 \setminus W\}$ and $\acute{n}$ with a uniform random number.
Specifically, a random value $\acute{r}_{\acute{n},m}$ is selected for auxiliary node $m \in \{U_3 \setminus W\}$ and $\acute{n}$. Instead of sending

$$\hat{x_n} = x_n + \sum_{m=1}^{M} \boldsymbol{PRG}(s_{n,m}) \tag{9}$$

The simulator submits

$$\hat{x_n} = x_n + \sum_{m \in W} \boldsymbol{PRG}(s_{n,m}) + \sum_{m \in U_2 \setminus W} \boldsymbol{PRG}(r_{\acute{n},m}) \tag{10}$$

The DDH assumption ensures that this hybrid is computationally indistinguishable from the first one.

Hyb2 In this hybrid for the same specific user $\acute{n} \in \{U_3 \setminus W\}$ instead of using $\boldsymbol{PRG}(r_{\acute{n},m})$ the simulator uses uniformly random number r_m with appropriate size to replace it.
Note that the only change in this hybrid is to substitute the output of a $\boldsymbol{PRG}$ with a uniformly random value. Therefore, depending on the security of the $\boldsymbol{PRG}$, we can argue that this hybrid is computationally distinguishable from the previous one.

Hyb3 In this hybrid, for each user $n \in U_2 \setminus W$ instead of sending their gradient x_n they use a random value selected by the simulator R_n, conditioned on $\sum_{n \in U_3 \setminus W} R_n = \sum_{n \in U_3 \setminus W} x_n = z$, so instead of

TABLE II
COMPARISON OF COMPUTATION, COMMUNICATION, AND STORAGE COMPLEXITY OF THE PROTOCOL AT THE CLIENT SIDE

	PSA [37]	Verifynet [15]	Our model
Computation	$O(N^2+VN)$	$O(N^2+VN)$	$O(M+V)$
Communication	$O(N+V)$	$O(N+V)$	$O(M+V)$
Storage	$O(N+V)$	$O(N+V)$	$O(M+V)$

TABLE III
COMPARISON OF COMPUTATION, COMMUNICATION, AND STORAGE COMPLEXITY OF THE PROTOCOL AT THE SERVER SIDE

	PSA [37]	Verifynet [15]	Our model
Computation	$O(VN^2)$	$O(VN^2)$	$O(NV+M)$
Communication	$O(N^2+NV)$	$O(N^2+NV)$	$O(NV+M+N)$
Storage	$O(N^2+V)$	$O(N^2+V)$	$O(N+M+V)$

sending

$$\hat{x_n} = x_n + \sum_{m=1}^{M} \boldsymbol{PRG}(s_{n,m}) \tag{11}$$

SIM sends

$$\hat{x_n} = R_n + \sum_{m=1}^{M} \boldsymbol{PRG}(s_{n,m}) \tag{12}$$

Therefore, the simulator has already completed the proof since $\boldsymbol{SIM}$ Simulates $\boldsymbol{REAL}$ without knowing x_n for all the users $n \in U_3 \setminus W$ and the output of the $\boldsymbol{SIM}$ is computationally indistinguishable from the output of $\boldsymbol{REAL}$.

□

VII. PERFORMANCE EVALUATION

In this section, we compare our proposed protocol with two well-known secure aggregation protocols used in [36] and [15]. While several other secure aggregation protocols exist in the literature, we specifically chose to compare our protocol with these two because they are the most relevant state-of-the-art protocols. One key reason for this selection is that they employ the same technology as our proposed protocol for secure aggregation. This allows for meaningful and useful comparisons of performance. Other protocols that adopt a different approach to secure aggregation may have different performance parameters, making it difficult to compare them in a meaningful manner. Both of the protocols we chose to compare against, as well as our proposed protocol, address the challenge of user dropouts, which is a significant issue in federated learning. Furthermore, these two protocols are widely used as benchmarks in the literature.

All complexity calculations presented below assume a single server, M auxiliary nodes, and N users, where each user has a model parameters vector of size V. The cost of the public key infrastructure and all signatures are ignored as they do not change any of the asymptotic complexities depending on M, N, and V. The results of comparing our protocol to the two aforementioned protocols in terms of computation, communication, and storage, on both the client and server sides, are reported in Table II and Table III respectively.

A. Performance Analysis for User/Client devices

1) Computation Cost: Computation cost at the user side is mainly attributed to generating the random values with each auxiliary node in $O(M)$ complexity, masking the parameters vector with the random values at a computation cost of $O(V)$, computing the values α and K to be used in validation in $O(M)$ complexity and verification that takes $O(V)$. Therefore, the overall computation cost at the client is $O(M+V)$.

2) Communication Cost: The communication cost for each user is attributed to: (1) sending its 3 public keys to the server and receiving $3M$ public keys and M encrypted secret shares received from the server at a communication cost of $O(M)$, (2) sending a masked parameters vector of size V to the server, and sending to the server the secret share of size V at a communication cost of $O(V)$. Thus, the total communication complexity is $O(M+V)$.

3) Storage Cost: Beside the user's own data stored for its keys generation, the user must store $3M$ keys corresponding to auxiliary nodes, the values of α and K, and the data vector for the model parameters (which it can mask in-place), which has a size of V. The overall storage cost at the client sums up to $O(M+V)$.

B. Performance Analysis for Auxiliary Nodes

1) Computation Cost: The computation cost at an auxiliary node is mainly attributed to generating the random values with $O(N)$ complexity, generating the $K_{n,m}$ values with $O(N)$ complexity as well, and calculating P_m for the list of online users, which is $O(N)$ making the total complexity $O(N)$.

2) Communication Cost: The communication cost at each auxiliary node is attributed to (1) sending its 3 public keys to the server, (2) receiving $3N$ public keys and sending N encrypted secret values to the server, (3) receiving a list of online users from the server, and (4) sending the computed sum of the random vectors at an auxiliary node. Thus, the overall communication cost at the auxiliary nodes is $O(N)$.

3) Storage Cost: Besides the auxiliary node's own data stored for its keys generation, it has to store the users' public keys received from the server ($3N$), $2N$ values for the shared keys $K_{n,m}$ and α_m, and the list of online users(N). Therefore, the total storage cost is $O(N)$.

Protocol I: Verifiable Secure Aggregation Protocol Using Auxiliary Nodes

– **Setup**
- All parties agree on the security parameter λ, and honestly generated public parameter $pp \leftarrow KA.gen(\lambda)$.
- All users have a private authenticated channel with the server.
- All auxiliary nodes have a private authenticated channel with the server.

– **Round 0** (Keys Advertising) **User n**:
- Generate 3 key pairs $(pk_n^1, sk_n^1) \leftarrow$ **KA.gen**(pp),$(pk_n^2, sk_n^2) \leftarrow$ **KA.gen**(pp),$(pk_n^3, sk_n^3) \leftarrow$ **KA.gen**(pp).
- Send the public keys $(pk_n^1, pk_n^2, pk_n^3)$to the server.

Auxiliary node m:
- Generate 3 key pairs $(pk_m^1, sk_m^1) \leftarrow$ **KA.gen**(pp),$(pk_m^2, sk_m^2) \leftarrow$ **KA.gen**(pp),$(pk_m^3, sk_m^3) \leftarrow$ **KA.gen**(pp).
- Send the public keys $(pk_m^1, pk_m^2, pk_m^3)$to the server.

Server Side:
- Collect number of keys $\leq |U_1|$ sent by each user, where U_1 denotes the set of users in this round.
- Collect the keys from all auxiliary nodes $|M|$, where M is set of all auxiliary nodes.
- Broadcast to all the users in U_1 the list of $\{(pk_m^1, pk_m^2, pk_m^3)\}_{m\in M}$.
- Broadcast to all the auxiliary nodes in M the list of $\{(pk_n^1, pk_n^2, pk_n^3)\}_{n\in U_1}$.

– **Round 1** (Key Sharing)
Auxiliary node m:
- Receive the list of keys $(pk_n^1, pk_n^2, pk_n^3)_{n\in U_1}$ from the cloud server.
- For each user $n \in U_1$, compute $K_{n,m} \leftarrow$ **KA.agree**(sk_m^3, pk_n^3).
- Compute $K_m \leftarrow \sum_{n\in U_1} K_{n,m}$.
- Sample a random element $\alpha_m \leftarrow \mathbb{F}$.
- For each user $n \in U_1$, compute $ct_{n,m} \leftarrow$ **AE.enc(KA.agree**$(sk_m^1, pk_n^1), \alpha_m||K_m)$.
- Send all the cipher-texts $\{ct_{n,m}\}_{n\in U_1}$ to the server

User n:
- Receive the list of keys $(pk_m^1, pk_m^2, pk_m^3)_{m\in M}$ from the cloud server.
- Receive the set of cipher-texts $\{ct_{n,m}\}_{m\in M}$.
- For each user $n \in U_1$, compute $\alpha||K \leftarrow \sum_{m\in M}$**AE.dec(KA.agree**$(sk_n^1, pk_m^1), ct_{n,m})$.
- For each user $n \in U_1$, compute $K_n \leftarrow \sum_{m\in M}$**KA.agree**$(sk_n^1, pk_m^1)$.

Server Side:
- Collect cipher-texts$\{ct_{n,m}\}_{n\in U_1}$ sent by each auxiliary node.
- Send a set of cipher-texts $\{ct_{n,m}\}_{m\in M}$ generated by m ∈ M $\{ct_{n,m}\}_{m\in M}$ to each user $n \in U_1$

– **Round 2** (Masking Input) **User n**:
- Calculate the shared key with every auxiliary node m as $s_{n,m} \leftarrow KA.agree(sk_n^2, pk_m^2)$ and expand this value using a **PRG** and calculate the masked input vector $\hat{x_n} \leftarrow x_n + \sum_{m\in M}$ **PRG**$(s_{n,m})$ (mod R).
- Calculate the MAC of the input vector as $\text{MAC}_n = K_n + \alpha \times x_n$ (mod R).
- Send to the server the masked parameters vector x_n and the MAC vector MAC_n.

Server Side:
- Receive messages (masked parameters vectors x_n and MAC_n) from the online users (represented as $U_2 \subseteq U_1$).
- Broadcast the list of U_2 to each auxiliary node $\in M$.

– **Round 3** (Unmasking Input)
Auxiliary node m:
- Receive the list of U_2 that represent the online users.
- Calculate the shared key with each user $n \in U_2$ $s_{n,m} \leftarrow KA.agree(sk_m^2, pk_n^2)$, and expand this value using a **PRG** into a random vector $P_{n,m} \leftarrow$ **PRG**$(s_{n,m})$.
- Calculate the $P_m \leftarrow \sum_{n\in U_2} P_{n,m}$
- Send the value P_m to the server.

Server Side:
- Receive the values P_m from the auxiliary nodes.
- Calculate the aggregated gradients for all users $n \in U_2$ as $X = \sum_{n\in U_2} \hat{x_n} - \sum_{m\in M} P_m$ (mod R).
- Calculate the aggregated MAC for all users $n \in U_2$ as $MAC = \sum_{n\in U_2} MAC_n$.
- Broadcast (X, MAC) to each user $n \in U_3$

– **Round 4** (Verification)
User n:
- Receive the pair (X,MAC) and compute $MAC' = K + \alpha X$
- Accept X if $MAC' = MAC$.

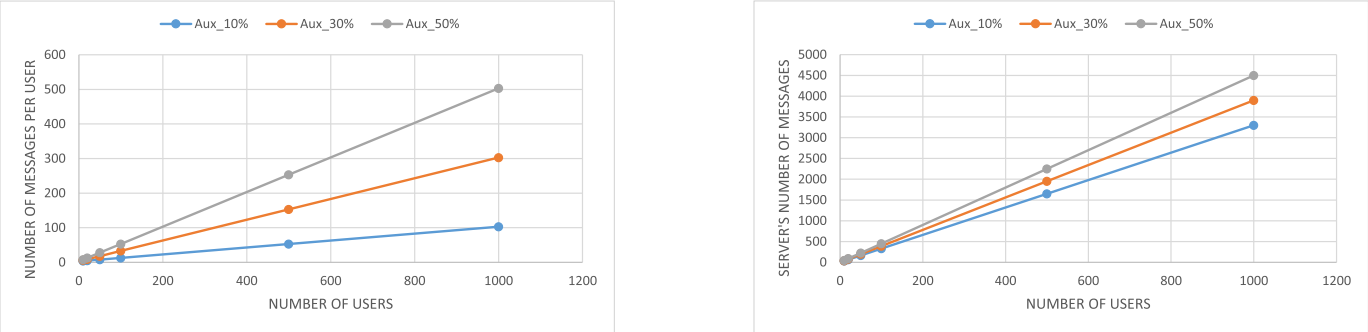


Fig. 4. Keys exchange costs per user and server as the number of users increases, with the number of auxiliary nodes varying by 10%, 30%, and 50% of the number of users.

C. Performance Analysis for the Server

1) Computation Cost: The server's computation cost is attributed to (1) aggregation of the local gradients, which involves adding the masked vectors received from the users and subtracting the random vectors received from the auxiliary nodes with the overall complexity of $O(NV + M)$ and (2) calculation of the aggregated MAC for online users, which takes $O(N)$ time in the worst case when all users are online with no dropouts. Thus, the overall computation cost becomes of complexity $O(NV + M)$.

2) Communication Cost: The server acts as the interface between the users and auxiliary nodes. It participates in all the communication between users and auxiliary nodes. In round 0, it collects the keys from users and auxiliary nodes and broadcasts the user's keys to all auxiliary nodes and auxiliary nodes' keys to all users with communication overhead of complexity $O(N + M)$. In round 1, the server collects the ciphertexts for the encrypted values sent by the auxiliary nodes and forwards them to each user, which is $O(M + N)$. In round 2, the server receives masked parameters vector of length (V) and MAC vector of length (V) from each online user with a maximum of N users in case of no dropouts with overall complexity $O(NV)$. In round 3, the server receives the lists of randoms from auxiliary nodes $O(M)$ and broadcasts the aggregated parameters vector (V) and aggregated MAC (V) to online users (maximum N) with a communication complexity of $O(NV)$. Therefore, the total communication complexity sums up to $O(NV + M + N)$.

3) Storage Cost: The server has to store the public keys of the users $(3N)$ and the public keys of the auxiliary nodes $(3M)$ in the first round. It stores the auxiliary nodes' ciphertexts (M) in the second round. The list of online users (N in case of no dropouts) is stored in the third round. In the last round, the server stores the aggregated parameters vector (V) and the aggregated MAC vector (V) for a total storage complexity $O(N + M + V)$.

D. Prototype Implementation and Setup

We developed a Python prototype on a desktop machine with a 2.60 GHz Intel Core i7-6700HQ CPU and 7.5 GB RAM. The prototype included the following cryptographic primitives:

- For the key agreement, we used the elliptic-curve Diffie-Hellman protocol.
- For Secret Sharing, we used t-out-of-n Shamir secret sharing.
- For encryption, we used Advanced Encryption Standard (AES) with a 128-bit key in counter mode.
- For the pseudorandom number generator, we used SHA-256.

To evaluate the performance, we evaluated the execution time for four phases: key sharing, Masking gradients, aggregation, and validation.

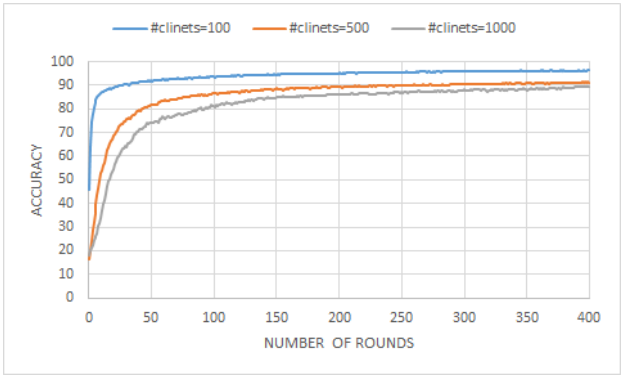
E. Experimental Results

To conduct our experiments, we used randomly generated 10K-entry vectors with 64-bit entries as the users' local gradients. We varied the number of users and user dropout ratio to acquire a broad understanding of how the two factors affect the performance of the four phases.

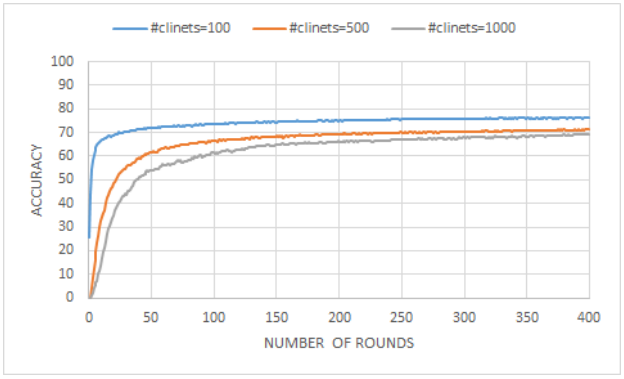
Table IV compares the overall performances of the PSA model, the Verifynet model, and our model with varying user numbers and dropout ratios. During the **sharing key** phase, our model demonstrates a lower cost than PSA and Verifynet as in our model, only the auxiliary nodes have to share the values of α_M and k_M to users, whereas, in the other two models, each user must make shares of both its secret key and its private value and share them with every other user. During the **masking input** phase, we did not observe a significant performance gap between PSA and our model. VerifyNet, on the other hand, incurs enormous overhead, mostly because of its extensive use of group operations to achieve bilinear pairing. During the phase of **unmasking input and aggregation**, when there are no dropouts, the costs of PSA and our model are comparable. Nevertheless, when dropouts occur, the computation cost in PSA increases exponentially while our model maintains a constant computation cost. This is expected since, in PSA, for each dropped user, the server must remove that user's pairwise masks for each surviving client. This requires the server to talk to a certain number of clients to get this dropped user's secret key and then figure out all the masks. However, in our model, only the auxiliary node excludes dropped users from the computation. VerifyNet's costs are much higher than PSA and our model for the same reason mentioned for the masking input phase. Lastly, during

TABLE IV
COMPARATIVE ANALYSIS RESULT

	PSA [37]		Verifynet [15]		Our Model	
	n= 500	n= 1000	n= 500	n= 1000	n= 500	n= 1000
	0% dropouts					
Sharing Key	3,045 ms	11,571 ms	3,100 ms	11,500 ms	524 ms	1500 ms
Masking Input	13,728 ms	27,350 ms	815,401 ms	828,691 ms	15,401 ms	28,691 ms
Unmasking Input and aggregation	121 ms	311 ms	954,540 ms	1,685,903 ms	454 ms	903 ms
Verification	N/A	N/A	1,355,426 ms	1,324,508 ms	125 ms	225ms
	10% dropouts					
Sharing Key	2,871 ms	11,728 ms	2,590 ms	11,550 ms	450 ms	1345 ms
Masking Input	13,353 ms	25,158 ms	714,434 ms	825,644 ms	14,434 ms	25,644 ms
Unmasking Input and aggregation	625,795 ms	2,455,050 ms	1,395,243 ms	1,340,564 ms	395 ms	697 ms
Verification	N/A	N/A	1,367,280 ms	1,290,428 ms	176 ms	240 ms
	20% dropouts					
Sharing Key	3,204 ms	14,193 ms	4,125 ms	15,326 ms	522 ms	1226 ms
Masking Input	14,184 ms	33,257 ms	913,422 ms	923,764 ms	13,422 ms	23,764 ms
Unmasking Input and aggregation	1,094,653 ms	4,457,614 ms	3,450,296 ms	3,252,794 ms	394 ms	794 ms
Verification	N/A	N/A	1,367,215 ms	1,401,329 ms	180 ms	259ms

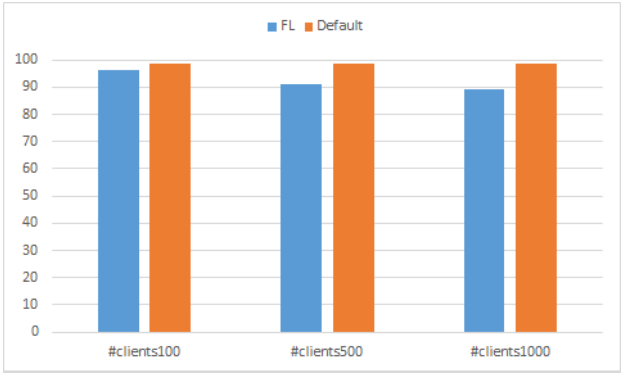


(a) MNIST

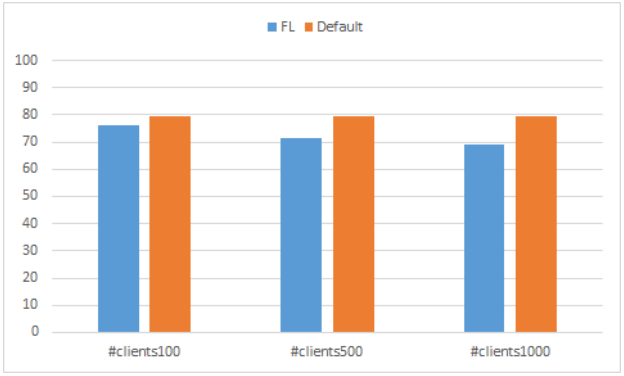


(b) CIFAR100

Fig. 5. Accuracy versus the number of iterations for different numbers of clients.



(a) MNIST- FL vs default



(b) CIFAR100- FL vs default

Fig. 6. Comparison of the accuracy of the FL model versus the default model.

the **verification** phase, our model is considerably lighter than Verifynet since it predominantly leverages computationally lightweight PRG operations. Verifynet, on the other hand, employs computationally intensive cryptographic operations such as bilinear pairing and Homomorphic Hash Functions, which are far more expensive than PRG.

Figure 4 illustrates the impact of increasing the number of users on the number of messages associated with the exchange of keys between users and the server for various auxiliary node ratios. As seen, as the number of users increases, there is a slight change in the number of messages, and the lowest use of auxiliary nodes shows the less number of messages.

F. Accuracy

We evaluated the performance of two different neural network architectures on two popular datasets - MNIST and CIFAR100:

- MNIST [49] is a dataset consisting of grayscale images of handwritten digits from 0 to 9, comprising 60,000 training and 10,000 testing images, each of size $28 \times 28 \times 1$, and labeled into one of the ten classes.
- CIFAR100 [50] is a dataset that contains RGB images of 100 classes, with 500 training images and 100 testing images per class, each of size $32 \times 32 \times 3$.

For the MNIST dataset, we used a three-layer network with two hidden, fully connected layers of 256 neurons and rectified linear units. The output layer was fully connected with 10 output neurons and utilized softmax activation. We used the stochastic gradient descent optimizer with a learning rate of 0.001. For the CIFAR100 dataset, we employed a convolutional neural network (CNN) with seven convolutional layers, each consisting of 3×3 filters and a stride of 1. Each convolutional layer was followed by rectified linear units and 2×2 max pooling with a stride of 2. The fully connected layer used softmax activation. We used the Adam optimizer with a learning rate of 0.001.

We evaluated the accuracy of Federated Learning (FL) models against a default model. The default model utilized ResNet as a pre-trained model and solely trained the fully-connected layer while maintaining the convolutional layer parameters constant.

Fig 5 shows the classification accuracy and number of rounds for different numbers of clients. The figure illustrates that as the number of clients increases, more rounds are needed to reach a specific accuracy. Fig 6 compares the accuracy of the FL model with that of the default model. The results show that the FL model achieves comparable accuracy, albeit slightly lower than the accuracy attained by the default model.

VIII. Conclusion

In this research, we propose a secure aggregation protocol that can be employed in a federated learning setting. The protocol depends on the use of auxiliary nodes that cannot all practically collude together. At the same time, the use of auxiliary nodes reduces the communication and computation costs as well as the storage cost on low-resource devices for clients and the service provider as well. These auxiliary nodes can represent hospitals in a healthcare scenario, banks in a financial or banking application, etc. The analysis of the protocol showed reduction in the computation, communication and storage cost compared to state-of-art protocols at the client nodes. In the proposed protocol, the cost of computation, communication, and storage on the low-resource devices (e.g., mobile phones/wearable devices) of client nodes depends only on the number of auxiliary nodes and the length of the weight parameters vector and is independent of the number of users joining or leaving the training process at each round. Additionally, the verification step in the proposed protocol relies on lightweight computations, which reduces power consumption on client devices.

Acknowledgment

Contents of the research are solely the responsibility of the authors and do not necessarily represent the official views of the Qatar National Research Fund (QNRF).

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, in Proceedings of Machine Learning Research, vol. 54, A. Singh and J. Zhu, Eds., Apr. 2017, pp. 1273–1282. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a.html

[2] P. Kairouz et al., "Advances and open problems in federated learning," *Found. Trends Mach. Learn.*, vol. 14, nos. 1–2, pp. 1–210, 2019, doi: 10.1561/2200000083.

[3] Y. Wu, S. Cai, X. Xiao, G. Chen, and B. C. Ooi, "Privacy preserving vertical federated learning for tree-based models," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 2090–2103, Aug. 2020, doi: 10.14778/3407790.3407811.

[4] Q. Li, Z. Wen, and B. He, "Practical federated gradient boosting decision trees," in *Proc. 34th AAAI Conf. Artif. Intell.*, New York, NY, USA, Feb. 2020, pp. 4642–4649. [Online]. Available: https://aaai.org/ojs/index.php/AAAI/article/view/5895

[5] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, "A survey on security and privacy of federated learning," *Future Gener. Comput. Syst.*, vol. 115, pp. 619–640, Feb. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X20329848

[6] N. Bouacida and P. Mohapatra, "Vulnerabilities in federated learning," *IEEE Access*, vol. 9, pp. 63229–63249, 2021.

[7] C. Fung, C. J. M. Yoon, and I. Beschastnikh, "Mitigating sybils in federated learning poisoning," 2018, *arXiv:1808.04866*.

[8] Y. Khazbak, T. Tan, and G. Cao, "MLGuard: Mitigating poisoning attacks in privacy preserving distributed collaborative learning," in *Proc. 29th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2020, pp. 1–6.

[9] Y. Zhang and H. Yu, "Towards verifiable federated learning," in *Proc. 31st Int. Joint Conf. Artif. Intell.*, Jul. 2022, pp. 5686–5693, doi: 10.24963/ijcai.2022/792.

[10] M. Atallah, K. Pantazopoulos, and E. Spafford, "Secure outsourcing of some computations," Purdue Univ., West Lafayette, IN, USA, Tech. Rep. 96-074, 1998. [Online]. Available: http://docs.lib.purdue.edu/cstech/1328

[11] Z. Liu, J. Guo, W. Yang, J. Fan, K.-Y. Lam, and J. Zhao, "Privacy-preserving aggregation in federated learning: A survey," *IEEE Trans. Big Data*, early access, Jul. 15, 2022, doi: 10.1109/TBDATA.2022.3190835.

[12] A. Girgis, D. Data, S. Diggavi, P. Kairouz, and A. T. Suresh, "Shuffled model of differential privacy in federated learning," in *Proc. 24th Int. Conf. Artif. Intell. Statist.*, in Proceedings of Machine Learning Research, vol. 130, A. Banerjee and K. Fukumizu, Eds., Apr. 2021, pp. 2521–2529. [Online]. Available: https://proceedings.mlr.press/v130/girgis21a.html

[13] M. Nassar, A. Erradi, F. Sabry, and Q. M. Malluhi, "A model driven framework for secure outsourcing of computation to the cloud," in *Proc. IEEE 7th Int. Conf. Cloud Comput.*, Jun. 2014, pp. 968–969.

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

[15] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "VerifyNet: Secure and verifiable federated learning," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 911–926, 2020.

[16] C. Hahn, H. Kim, M. Kim, and J. Hur, "VerSA: Verifiable secure aggregation for cross-device federated learning," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 1, pp. 36–52, Jan. 2023.

[17] Y. Yang, L. Zhang, Y. Zhao, K.-K.-R. Choo, and Y. Zhang, "Privacy-preserving aggregation-authentication scheme for safety warning system in fog-cloud based VANET," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 317–331, 2022.

[18] L. Sun, J. Qian, and X. Chen, "LDP-FL: Practical private aggregation in federated learning with local differential privacy," 2020, *arXiv:2007.15789*.

[19] K. Wei et al., "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3454–3469, 2020.

[20] P. Kairouz, Z. Liu, and T. Steinke, "The distributed discrete Gaussian mechanism for federated learning with secure aggregation," 2021, *arXiv:2102.06387*.

[21] S. Goryczka, L. Xiong, and V. Sunderam, "Secure multiparty aggregation with differential privacy: A comparative study," in *Proc. Joint EDBT/ICDT Workshops*, New York, NY, USA, Mar. 2013, pp. 155–163, doi: 10.1145/2457317.2457343.

[22] T. Stevens, C. Skalka, C. Vincent, J. Ring, S. Clark, and J. Near, "Efficient differentially private secure aggregation for federated learning via hardness of learning with errors," 2021, *arXiv:2112.06872*.

[23] H. Zhou, G. Yang, H. Dai, and G. Liu, "PFLF: Privacy-preserving federated learning framework for edge computing," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 1905–1918, 2022.

[24] H. Zheng, H. Hu, and Z. Han, "Preserving user privacy for machine learning: Local differential privacy or federated machine learning?" *IEEE Intell. Syst.*, vol. 35, no. 4, pp. 5–14, Jul. 2020.

[25] Y. Wang, Y. Tong, and D. Shi, "Federated latent Dirichlet allocation: A local differential privacy based framework," in *Proc. AAAI Conf. Artif. Intell.*, 2020, vol. 34, no. 4, pp. 6283–6290.

[26] S. Truex et al., "A hybrid approach to privacy-preserving federated learning," in *Proc. 12th ACM Workshop Artif. Intell. Secur.*, Nov. 2019, pp. 1–11.

[27] S. Truex et al., "A hybrid approach to privacy-preserving federated learning," 2018, *arXiv:1812.03224*.

[28] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, *BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning*. Berkeley, CA, USA: USENIX Association, 2020.

[29] J. Park and H. Lim, "Privacy-preserving federated learning using homomorphic encryption," *Appl. Sci.*, vol. 12, no. 2, p. 734, Jan. 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/2/734

[30] J. Zhao, H. Zhu, F. Wang, R. Lu, Z. Liu, and H. Li, "PVD-FL: A privacy-preserving and verifiable decentralized federated learning framework," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 2059–2073, 2022.

[31] Z. Ma, J. Ma, Y. Miao, Y. Li, and R. H. Deng, "ShieldFL: Mitigating model poisoning attacks in privacy-preserving federated learning," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 1639–1654, 2022.

[32] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology—ASIACRYPT 2017*, T. Takagi and T. Peyrin, Eds. Cham, Switzerland: Springer, 2017, pp. 409–437.

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

[34] J. Ma, S. Naas, S. Sigg, and X. Lyu, "Privacy-preserving federated learning based on multi-key homomorphic encryption," *Int. J. Intell. Syst.*, vol. 37, no. 9, pp. 5880–5901, Sep. 2022.

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

[36] K. Bonawitz et al., "Practical secure aggregation for federated learning on user-held data," 2016, *arXiv:1611.04482*.

[37] H. Fereidooni et al., "SAFELearn: Secure aggregation for private federated learning," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2021, pp. 56–62.

[38] Y. Zheng, S. Lai, Y. Liu, X. Yuan, X. Yi, and C. Wang, "Aggregation service for federated learning: An efficient, secure, and more resilient realization," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 2, pp. 988–1001, Mar. 2023.

[39] Y. Miao, Z. Liu, H. Li, K.-K.-R. Choo, and R. H. Deng, "Privacy-preserving Byzantine-robust federated learning via blockchain systems," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 2848–2861, 2022.

[40] H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Blockchained on-device federated learning," *IEEE Commun. Lett.*, vol. 24, no. 6, pp. 1279–1283, Jun. 2020.

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

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

[43] Y. Ren, Y. Li, G. Feng, and X. Zhang, "Privacy-enhanced and verification-traceable aggregation for federated learning," *IEEE Internet Things J.*, vol. 9, no. 24, pp. 24933–24948, Dec. 2022.

[44] F. Luo, S. Al-Kuwari, and Y. Ding, "SVFL: Efficient secure aggregation and verification for cross-silo federated learning," *IEEE Trans. Mobile Comput.*, early access, Nov. 4, 2022, doi: 10.1109/TMC.2022.3219485.

[45] Z. Ghodsi, T. Gu, and S. Garg, "SafetyNets: Verifiable execution of deep neural networks on an untrusted cloud," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1–12.

[46] F. Tramèr and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," 2019, *arXiv:1806.03287*.

[47] F. Sabry, T. Eltaras, W. Labda, F. Hamza, K. Alzoubi, and Q. Malluhi, "Towards on-device dehydration monitoring using machine learning from wearable device's data," *Sensors*, vol. 22, no. 5, p. 1887, Feb. 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/5/1887

[48] J. Konečný, 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*.

[49] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[50] A. Krizhevsky et al., "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.