future internet

MDPI

Article

Secure Data Sharing in Federated Learning through Blockchain-Based Aggregation

Bowen Liu and Qiang Tang *

Luxembourg Institute of Science and Technology (LIST), 5, Avenue des Hauts-Fourneaux, L-4362 Esch-sur-Alzette, Luxembourg
* Correspondence: qiang.tang@list.lu

Abstract: In this paper, we explore the realm of federated learning (FL), a distributed machine learning (ML) paradigm, and propose a novel approach that leverages the robustness of blockchain technology. FL, a concept introduced by Google in 2016, allows multiple entities to collaboratively train an ML model without the need to expose their raw data. However, it faces several challenges, such as privacy concerns and malicious attacks (e.g., data poisoning attacks). Our paper examines the existing EIFFeL framework, a protocol for decentralized real-time messaging in continuous integration and delivery pipelines, and introduces an enhanced scheme that leverages the trustworthy nature of blockchain technology. Our scheme eliminates the need for a central server and any other third party, such as a public bulletin board, thereby mitigating the risks associated with the compromise of such third parties.

Keywords: federated learning; privacy-preserving; blockchain

Citation: Liu, B.; Tang, Q. Secure Data Sharing in Federated Learning through Blockchain-Based Aggregation. *Future Internet* **2024**, *16*, 133. https://doi.org/10.3390/fi16040133

Academic Editors: Qiang Duan, Zhihui Lu and Paolo Bellavista

Received: 19 February 2024
Revised: 4 April 2024
Accepted: 11 April 2024
Published: 15 April 2024

1. Introduction

With the advancement of big data and artificial intelligence (AI) technologies, the challenge of securely and reliably training machine learning (ML) models on distributed and heterogeneous data sources without compromising privacy and integrity has become increasingly prominent. The term federated learning (FL) was introduced by Google in 2016, at a time when the use and misuse of personal data were gaining global attention. The Cambridge Analytica scandal awakened Facebook users and those of similar platforms to the dangers of sharing personal information online [1]. It also sparked a wider debate on the pervasive tracking of people on the internet, often without their consent. In response, many countries and regions have passed or proposed data privacy laws, such as the General Data Protection Regulation (GDPR) in Europe [2].

FL is a distributed ML paradigm that enables multiple entities to collaboratively train a model from their local data, without exposing their raw data to each other or a central server [3]. This approach stands in contrast to traditional centralized ML techniques, where local datasets are merged into one location. Therefore, FL has the potential to address the prevalent limitations and challenges in the traditional approach, particularly the critical issues of data privacy. The main application of FL is to train models on data that are sensitive, distributed, or heterogeneous, such as personal data on mobile phones or data from different organizations. For example, FL can be used to improve spam filters and recommendation tools without accessing users' emails or preferences. It can also be used to leverage data from sensors and smart devices for various tasks, such as anomaly detection, predictive maintenance, and optimization.

Despite the promises of FL, two main risks persist in the distributed and decentralized nature of the learning process, namely the privacy challenge and model quality. Privacy concerns arise from adversaries attempting to identify sensitive patterns in local updates, potentially compromising the global model [4–6]. Lyu et al. [7] provide a general discussion

about privacy and robustness attacks against FL and provide some direction for mitigating them. The German BSI published a detailed report on specific privacy risks, such as membership inference attacks and model inversion attacks [8]. Simultaneously, in terms of model quality, integrity challenges stem from participants acting maliciously by injecting poisoned or malformed inputs during the FL process, where even a single malicious input can have a significant impact [9,10]. Apart from these risks, there are also other related concerns for FL in practice, and a comprehensive survey on them is provided in [11]. To mitigate these risks, various solutions have been proposed. For example, with secure aggregation, instead of sharing the plaintext local update, only masked updates are transmitted, so that the server can aggregate the global model correctly without the knowledge of individual updates [12,13]. In addition, differential privacy is also widely used, adding noise to the exchanged values to provide additional robustness [14,15]. In the literature, one notable contribution to addressing these challenges is the EIFFeL framework [16]. By leveraging a public bulletin board, EIFFeL aims to preserve the privacy of input data from different clients as well as guarantee the integrity of the inputs from these clients (e.g., the value of inputs should be in proper ranges).

Contribution and Organization

As privacy and integrity are two key challenges hindering the widespread adoption of FL, this paper is dedicated to investigating a rigorous solution to address these challenges. Referring to [11], FL can be categorized into two categories: cross-silo FL and cross-device FL. In this paper, we focus on the cross-silo setting, wherein model training occurs on siloed data belonging to several clients from different organizations. These clients can communicate with third-party servers in a synchronous manner.

Our first contribution involves investigating the architecture and workflow of the EIFFeL framework along with analyzing its security guarantees. To this end, we identify specific areas where the EIFFeL framework exhibits potential risks, such as the compromise of the central server and/or the public bulletin board. We also highlight other efficiency concerns.

Our second contribution involves adapting the EIFFeL framework to leverage the security guarantees of blockchain platforms. To this end, we adapt the operations in the EIFFeL framework so that the functions of both the central server and the public bulletin board can be replaced by a blockchain platform. To address the challenge posed by blockchain's limitation to only perform deterministic operations and the requirement that no direct communication among clients is necessary, we integrate the Burmester-Desmedt group key exchange protocol into our solution, to make sure that the clients can jointly compute the final parameter update without breaching its confidentiality.

Our last contribution involves implementing our solution and shedding some light on the computational complexities. Note that the proposed solution is blockchain platform agnostic, so we omit the implementation details related to the blockchain part. This is earmarked for future work.

It is worth noting that privacy protection for FL is a very challenging task. Even with secure aggregation, some privacy risks may still remain [11]. To address this, differential privacy, particularly local differential privacy, can be applied. Since such measures are used to secure aggregation and can be applied independently, we leave this out in this paper.

The paper consists of the following sections: we present the background and motivation of FL and blockchain in Section 2. In Section 3, we review the design and analysis of the EIFFeL framework. Following this, an enhanced scheme is proposed; its security analysis, along with a demonstration of performance, is presented in Section 4. Finally, we summarize our work in Section 5.

2. Preliminary

2.1. Federated Learning Approach

FL represents a decentralized paradigm that is reshaping traditional ML methods. In this collaborative approach, multiple clients contribute to training a shared model without divulging raw data. Assuming a central server and n clients $\mathcal{C}_i$ $(1 \leq i \leq n)$, a high-level overview of the FL workflow is summarized in Figure 1; we refer to the relevant handbook for more details, such as [3].

1. Initial phase:
   (a). The central server defines an initial model, $\boldsymbol{m}_{init}$, which can be chosen based on prior knowledge or domain expertise.
   (b). Each client, $\mathcal{C}_i$, synchronizes the initial model, $\boldsymbol{m}_{init}$, from the server, and sets its local model to be $\boldsymbol{m}_i = \boldsymbol{m}_{init}$.
2. The following training steps are iterated for a certain number of iterations until a satisfactory model is achieved. Here, each iteration is called an **Epoch**.
   (a). **Local model training**: Each client, $\mathcal{C}_i$, trains its local model using its local dataset, $\boldsymbol{D}_i$, and obtains a new local model with new parameters, $\boldsymbol{u}_i$.
   (b). **Model parameter sharing**: Each client, $\mathcal{C}_i$, sends the newly generated model parameters, $\boldsymbol{u}_i$, to the central server.
   (c). **Model parameter aggregation**: The central server aggregates the model updates received from the clients to produce an aggregate update, $\mathcal{U}$, and sends it back to each client.
   (d). **Model update**: Each client, $\mathcal{C}_i$, resets its local model with the parameters, $\mathcal{U}$.

Figure 1. Federated learning approach.

As we mentioned in Section 1, while FL enhances privacy by not sharing raw data with the central server or other devices, a crucial concern arises: the trustworthiness of the central server. Two major challenges need to be considered: privacy challenge and integrity challenge, as explained in Section 3.

2.2. Blockchain Overview

A blockchain is a chain of blocks; each block contains a list of transactions. These blocks are linked and secured through cryptographic hashes, forming a continuous, unalterable chain. The decentralized nature of a blockchain means that no single entity has control over the entire network, mitigating the risk of a central point of failure. Transactions in a blockchain are grouped into blocks, and each block includes a reference to the previous block, creating a chronological chain of events. Miners, i.e., participants in the network with computational power, play a crucial role in validating and adding transactions to the blockchain. This sequential structure ensures the integrity of the data, as altering information in one block would require changing all subsequent blocks, which is an impractical and computationally infeasible task. For a more comprehensive review, we refer to comprehensive references, such as [17–19]. Irrespective of the variations in blockchain structures, the following properties are anticipated:

- Democracy and decentralized control: In systems utilizing proof of work (PoW) as the consensus mechanism in permissionless scenarios, everyone has the potential to act as a miner, possessing equal privileges to generate and approve blocks for the blockchain. Although variations may exist in different cases, the overarching principle remains: blockchain technology eliminates the need for a singular, fully trusted entity, thereby averting the vulnerability of a single point of failure.
- Integrity and immutability: In the absence of an attacker or a coalition of attackers dominating the consensus process, such as when more than 51% of the computing power in the Bitcoin blockchain is handled by semi-honest miners, it becomes infeasible to modify agreed-upon blocks in the consensus.

– Consistency: Despite potential attacks from robust adversaries, the chain upholds a singular and consistent perspective, as outlined by the aforementioned assumptions. However, it is essential to note that deviations from predefined rules by nodes can lead to the generation of forks, resulting in different perspectives among participants, as observed in scenarios like Ethereum.

These properties enhance auditability and transparency and improve the overall trustworthiness of the system. Some have conceptualized blockchain as a social trust machine (https://www.economist.com/leaders/2015/10/31/the-trust-machine (accessed on 1 April 2024)). The trust users place in blockchain systems predominantly stems from the cryptographic viewpoint that the majority of miners will act as semi-honest participants. The semi-honest assumption essentially asserts that these miners will respect the predetermined protocols, carrying out actions exactly as specified and programmed in the blockchain software. In particular, the assumption rules out the possibility of collusion among miners to disrupt the regular operations of the blockchain.

There is a vast body of literature on utilizing blockchain to address security issues in FL, more information can be found in recent survey papers, such as [20,21].

3. EIFFeL Framework and Security Analysis

The EIFFeL framework [16] presents an innovative FL approach that effectively tackles the challenges of data privacy and integrity. Moving beyond the conventional server–client architecture, as outlined in Section 2.1, EIFFeL incorporates a public bulletin board, which broadcasts intermediary information for secure aggregation and integrity checks. This unique architecture positions all clients as verifiers for each other, with the server playing a collective role. In more detail, secure aggregation enables each client to conceal its parameter update within the aggregated value, by leveraging secret sharing and encryption techniques. The integrity property is ensured through non-interactive proof techniques and the adoption of a public bulletin board. In summary, EIFFeL is designed to tolerate multiple malicious client interventions while ensuring the proper aggregation of model parameters.

3.1. Threat Model

In EIFFeL, it assumes that all honest clients correctly follow the protocol and have properly structured inputs. Two types of malicious adversaries are considered:

– Malicious clients: Multiple malicious clients can arbitrarily deviate from the protocol. They may (1) compromise the aggregate by submitting malformed updates; (2) cause the honest clients to complete an integrity check; (3) violate the privacy of honest clients, which may collude with the server.
– Malicious server: It aims to violate the privacy of clients by trying to recover their raw updates. A malicious server may (1) mark the inputs from honest clients as invalid; (2) mark the inputs from malicious clients as valid, so as to decide which one will be aggregated.

3.2. Scheme Architecture and Workflow

As mentioned previously, EIFFeL consists of a public bulletin board $\mathcal{B}$, a single server $\mathcal{S}$, and n clients, $\mathcal{C}_i$ $(1 \leq i \leq n)$. It is designed to tolerate a limited number of malicious $m < \lfloor \frac{n}{3} \rfloor$. A high-level sketch of the interactions among a client and other entities is shown in Figure 2 while a detailed description of the EIFFeL scheme is shown in Figures 3 and 4. Note that, for simplicity, we only show one client in the diagram. The numbering of the iteration steps corresponds to the description in Figure 4. In order to ensure (1) privacy for all honest clients, and (2) input integrity, where the server is motivated to verify if all individual updates are well-formed, all clients function as verifiers for each other, while the server also participates collectively in this role. Verification remains achievable even in the presence of m malicious clients.

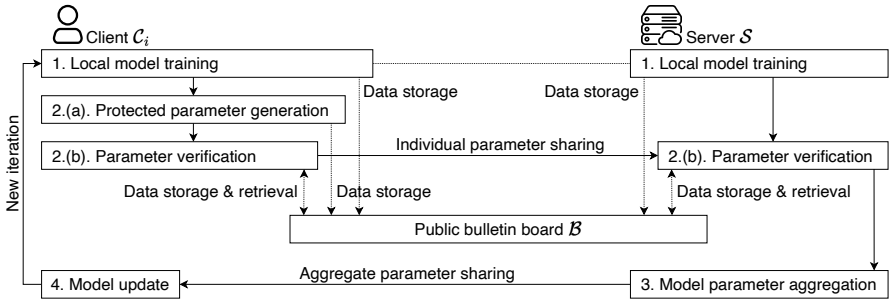

Figure 2. System Architecture of the EIFFeL Scheme.

To achieve the designated security guarantees, EIFFeL relies on several cryptographic building blocks (we omit the detailed definition of algorithms; refer to [16] for the full definitions), with each serving a specific purpose in enhancing the scheme's security:

- Shamir's t-out-of-n secret sharing scheme [22]: This cryptographic method facilitates the distribution of a secret among n participants, and it requires at least t participants (the threshold) to collectively reconstruct the original secret. Two algorithms are defined, $SS.share(\cdot)$ is for generating secret sharing and $SS.recon(\cdot)$ is for reconstructing the secret. Assuming m malicious clients, t is set as $m+1$ in EIFFeL.
- Reed–Solomon error correcting code [23]: It is an error-correcting code used in digital communication and data storage. Reed–Solomon codes add redundant symbols to the original data, allowing the receiver to detect and correct errors during transmission. EIFFeL uses the $[n, m+1, n-m]$ Reed–Solomon error correcting code, where any set of shares containing $m < \lfloor \frac{n}{3} \rfloor$ malicious shares can be used to recover the secret with $robustRecon(\cdot)$.
- Key agreement protocol: It enables two or more parties to agree upon a shared secret key. It involves three algorithms: $KA.param(\cdot)$ is used to generate the parameters, $KA.gen(\cdot)$ is used to generate a public/private key pair, and $KA.agree(\cdot)$ is used to agree on a common secret key.
- Authenticated encryption: A cryptographic process that combines encryption and message authentication to guarantee the integrity and confidentiality of transmitted data. It includes the algorithms of key generation $AE.gen(\cdot)$, encryption $AE.enc(\cdot)$, and decryption $AE.dec(\cdot)$.
- Secret-shared non-interactive proofs (SNIPs) [24]: These are cryptographic protocols that allow multiple parties to jointly prove the truth of a statement without revealing their individual inputs. These proofs are constructed in such a way that the validity of the statement can be verified without requiring interaction between the parties. A public validation predicate $Valid(\cdot)$ is defined to conduct the integrity check.

The EIFFeL framework includes a setup phase and a model training phase. In the model training phase, each training epoch consists of four primary steps, as summarized in Figure 1: (1) local model training, (2) model parameter sharing, (3) model parameter aggregation, and (4) model update. We briefly recap the set and training phases in Figure 3 and Figure 4, respectively, which visually encapsulate the essence of its mechanism.

- Setup:
  - All parties are given the security parameter, k, the number of clients, n, the threshold for malicious clients, m, and a field $\mathbb{F}$ for secret sharing usage. All clients honestly generate $pp \leftarrow KA.gen(k)$, and the server, $\mathcal{S}$, initializes the malicious client list, $\mathcal{C}^* = \emptyset$, and lists $Flag[i] = \emptyset$ for clients that have flagged the client, $\mathcal{C}_i$, as malicious.

Figure 3. Setup of EIFFeL scheme.

1. **Local model training**:
   Each client $\mathcal{C}_i$:
   – Generates its key pair $(pk_i, sk_i) \xleftarrow{\$} KA.gen(pp)$ and announces the public key on the public bulletin board $\mathcal{B}$.
   Server $\mathcal{S}$:
   – Publishes the validation predicate $Valid(\cdot)$ on the public bulletin board $\mathcal{B}$.
2. **Model parameter sharing**: The following steps are performed:
   (a). Each client, $\mathcal{C}_i$, generates its protected parameters as follows:
      i. Establishes $n-1$ common secret keys $sk_{i,j} \leftarrow KA.agree(pk_j, sk_i)$ with each of the other clients.
      ii. Generates a proof $\boldsymbol{\pi}_i = (\boldsymbol{h}_i, (\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i)), \boldsymbol{h}_i \in \mathbb{F}[X], (\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i) \in \mathbb{F}^3, \boldsymbol{a}_i \cdot \boldsymbol{b}_i = \boldsymbol{c}_i$ for the computation $Valid(\boldsymbol{u}_i) = 1$, where $\boldsymbol{u}_i \in \mathbb{F}^d$ is its local update with d dimensions.
      iii. Creates shares of its update $\boldsymbol{u}_i$ for all clients, denoted as $\{(1, \boldsymbol{u}_{i,1}], \cdots, (n, \boldsymbol{u}_{i,n}), \Psi_{\boldsymbol{u}_i}\} \leftarrow SS.share(u, [n], m+1)$.
      iv. Creates shares of its proof $\boldsymbol{\pi}_i$ for other clients: $\{(1, \boldsymbol{h}_{i,1}], \cdots, (n, \boldsymbol{h}_{i,n}), \Psi_{\boldsymbol{h}_i}\} \leftarrow SS.share(\boldsymbol{h}_i, [n] \setminus i, m+1)$, $\{(1, \boldsymbol{a}_{i,1}], \cdots, (n, \boldsymbol{a}_{i,n}), \Psi_{\boldsymbol{a}_i}\} \leftarrow SS.share(\boldsymbol{a}_i, [n] \setminus i, m+1)$, $\{(1, \boldsymbol{b}_{i,1}], \cdots, (n, \boldsymbol{b}_{i,n}), \Psi_{\boldsymbol{b}_i}\} \leftarrow SS.share(\boldsymbol{b}_i, [n] \setminus i, m+1)$, $\{(1, \boldsymbol{c}_{i,1}], \cdots, (n, \boldsymbol{c}_{i,n}), \Psi_{\boldsymbol{c}_i}\} \leftarrow SS.share(\boldsymbol{c}_i, [n] \setminus i, m+1)$.
      v. Publishes the encrypted proof strings, $\forall \mathcal{C}_j \in \mathcal{C}_{\setminus i}, \overline{(j, \boldsymbol{u}_{j,1})||(j, \boldsymbol{\pi}_{j,1})} \leftarrow AE.enc_{sk_{i,j}}((j, \boldsymbol{u}_{i,j})||(j, \boldsymbol{\pi}_{i,j}))$, $\boldsymbol{\pi}_{i,j} = \boldsymbol{h}_{i,j}||\boldsymbol{a}_{i,j}||\boldsymbol{b}_{i,j}||\boldsymbol{c}_{i,j}$ on the public bulletin board $\mathcal{B}$ with its check strings $(\Psi_{\boldsymbol{u}_i}, \Psi_{\boldsymbol{\pi}_i})$.
   (b). The protected parameters are verified as follows:
      (i) Verifying validity of secret shares:
      Each client $\mathcal{C}_i$:
         (A). Retrieves and decrypts the shares pertaining to it $(i, \boldsymbol{u}_{j,i})||(i, \boldsymbol{\pi}_{j,i}) \leftarrow AE.dec_{pk_{i,j}}(\overline{(i, \boldsymbol{u}_{j,i})||(i, \boldsymbol{\pi}_{j,i})})$.
         (B). Verifies the shares $\boldsymbol{u}_{j,i}(\boldsymbol{\pi}_{j,i})$ using check string $\Psi_{\boldsymbol{u}_i}(\Psi_{\boldsymbol{\pi}_i})$.
         (C). If any share fails to be decrypted or verified, flag its creator on $\mathcal{B}$.
      Server $\mathcal{S}$:
         (A). Upon receiving a report (e.g., $\mathcal{C}_i$ flags $\mathcal{C}_j$), updates $Flag[j] = Flag[j] \cup \mathcal{C}_i$.
         (B). Updates $\mathcal{C}^*$ as follows:
            – If $|Flag[i]| > m$, $\mathcal{C}_i$ is marked as malicious: $\mathcal{C}^* = \mathcal{C}^* \cup \mathcal{C}_i$.
            – If any client, $\mathcal{C}_i$, reported more than m clients, it is considered as malicious: $\mathcal{C}^* = \mathcal{C}^* \cup \mathcal{C}_i$.
            – For a client $\mathcal{C}_i$ that has been reported, but with $|Flag[i]| \leq m$, server $\mathcal{S}$ intervenes for further verification. $\mathcal{S}$ requests the shares (in clear) from $\mathcal{C}_i$ for the clients who flagged it (e.g., $((j, \boldsymbol{u}_{i,j})||(j, \boldsymbol{\pi}_{i,j}))$ generated by $\mathcal{C}_i$ for $\mathcal{C}_j$), and verifies the contents with the relevant check string (e.g., $(\Psi_{\boldsymbol{u}_i}, \Psi_{\boldsymbol{\pi}_i})$). If the verification fails, $\mathcal{C}_i$ is marked as malicious; otherwise, $\mathcal{C}_j$ is instructed to use the released share for its computations.
         (C). Announces the updated $\mathcal{C}^*$ on the public bulletin board $\mathcal{B}$.
      (ii) Generation of proof summaries by the clients:
      Server $\mathcal{S}$:
         – Announces a random number, $r \in \mathbb{F}$, on the public bulletin board, $\mathcal{B}$.
      Each client $\mathcal{C}_i$:
         – Generates a summary, $\sigma_{j,i}$, of the proof string, $\boldsymbol{\pi}_{j,i}$, based on r and SNIP, $\forall \mathcal{C}_j \notin \mathcal{C}^*$, and publishes it on the public bulletin, $\mathcal{B}$.
      (iii) Verification of proof summaries by the server:
      Server $\mathcal{S}$:
         – Collects and verifies all proof summaries from $\mathcal{C} \setminus \mathcal{C}^*$ with $robustRecon(\cdot)$, and updates $\mathcal{C}^*$ based on the result.
      (iv) Each client $\mathcal{C}_i$:
         – If $\mathcal{C}_i \in \mathcal{C}^*$, it can initiate a dispute by transmitting the transcript of the reconstruction of σ_i. If any successful dispute occurs, all clients abort the protocol, since the server, $\mathcal{S}$, is considered malicious by withholding the valid updates.
         – Otherwise, it sends the aggregate $\mathcal{U}_i = \sum_{\mathcal{C}_j \notin \mathcal{C}^*} u_{j,i}$ to the server $\mathcal{S}$.
3. **Model parameter aggregation**: Server $\mathcal{S}$ recovers the final aggregate, $\mathcal{U} \leftarrow robustRecon(\{i, \mathcal{U}_i\})_{\mathcal{C}_j \notin \mathcal{C}^*}$, and sends it to clients.
4. **Model update**: Each client, $\mathcal{C}_i$, resets its local model with the parameters, $\mathcal{U}$.

Figure 4. One iteration/epoch of the EIFFeL scheme.

3.3. Analysis of the EIFFeL Framework

In [16], the authors performed an analysis of the EIFFeL framework and showed that the framework achieves the pre-defined privacy and integrity properties. In the following, we analyze the assumptions made in their analysis and also demonstrate several observations on the design of the framework.

First of all, the existence of the required public bulletin board and the associated assumptions is very tricky. To guarantee the security of EIFFeL, the public bulletin board should offer the following guarantees:

- It should be corrupt-resistant against all entities, including the server, clients, and other attackers. The data should not be changed, deleted, or manipulated by malicious entities in any manner.
- It should be able to validate the identities of clients (and the server) according to some public key infrastructure (PKI). By default, some PKI information should be stored on the bulletin board so that it can validate the identity claims of its users.
- It should be able to establish a secure channel with its users so that the integrity and authenticity of the users' data can be guaranteed during the transmission.

Public bulletin boards have been used in many scenarios [25], and they can certainly play an important role in FL. However, how to achieve the last two guarantees is not trivial. In particular, the involvement of a PKI means that there will be another fully trusted third party. The trust relationship among all the entities in the solution needs to be clarified in order to guarantee that all threats are properly present.

Secondly, there is some ambiguity in remark 1 of reference [16]. It states that EIFFeL prevents the server from "Mark the input of an honest client as invalid and include it in the final aggregate". In fact, the server can manipulate the final aggregate without being noticed, in step 3 of the model parameter aggregation. This is equivalent to invalidating the input of honest clients, as the final aggregate no longer appears to be an aggregation of inputs from all honest clients. This implies that EIFFeL cannot guarantee the integrity of the aggregate in the presence of a malicious server. Note that this observation does not conflict with the desired security properties, i.e., against the server, only privacy is expected. In addition, if the server deviates from the protocol, Lemma 4 from [16] will not hold. This lemma states that "The final aggregate must contain the updates of all honest clients or the protocol is aborted". In fact, the server can maliciously change the aggregate without being detected, as we said before. Although this observation does not show a security flaw in the EIFFeL framework, we regard it as a drawback, and ideally, we should avoid such potential "attacks". This is also motivated by some recent findings (e.g., [26]), which show that failure of integrity (e.g., data poisoning) can lead to privacy breaches.

Thirdly, there are two inefficient designs in the solution. One is about the parameter generation in Step 1 and the key agreement at the beginning of Step 2. During the FL process, all four steps will be iterated hundreds of times in order to reach convergence. According to the description, the key materials need to be repeated in each iteration. This unnecessarily increases the complexity of the solution. These parameters can be set up in the setup phase so that the key materials can be used in all the training iterations (or, epochs). The other involves the usage of authenticated encryption. In the solution, it is assumed that the public bulletin board does not allow any manipulation of the stored data; therefore, the encrypted data in Step 2 will not be manipulated regardless of whether the encryption is authenticated or not. Therefore, standard symmetric encryption will suffice.

Fourthly, the result of the server's operation in Step 2 is confusing. For client $\mathcal{C}_i$, which has been reported but $|Flag[i]| \leq m$, server $\mathcal{S}$ intervenes for further verification. $\mathcal{S}$ requests the shares (in clear) from $\mathcal{C}_i$ for the clients who flagged it (e.g., $((j, \boldsymbol{u}_{i,j})||(j, \boldsymbol{\pi}_{i,j}))$ generated by $\mathcal{C}_i$ for $\mathcal{C}_j$), and verifies the contents with the relevant check string (e.g., $(\Psi_{\boldsymbol{u}_i}, \Psi_{\boldsymbol{\pi}_i})$). If the verification fails, $\mathcal{C}_i$ is marked as malicious; otherwise, $\mathcal{C}_j$ is instructed to use the released share for its computations. The server's operation occurs because either $\mathcal{C}_i$ is maliciously flagged $\mathcal{C}_j$ or $\mathcal{C}_j$ is sent the wrong ciphertext to $\mathcal{C}_i$, on purpose (note that $\mathcal{C}_j$ can disclose the right plaintext to make the server's verification pass). However, neither party will be determined as malicious according to the protocol. It is unclear why this is the case.

4. The Enhanced Scheme

In this section, we describe the enhanced scheme that eliminates the need for any third-party server or bulletin board by integrating blockchain technology. It is worth emphasizing that this scheme aims to address the existing privacy and security challenges in the EIFFeL framework, particularly those related to the third-party bulletin board. Certainly, in our design, we attempt to minimize the complexity overhead for all the involved entities.

Our scheme eliminates the need for a central server and any other third party, such as a public bulletin board, thereby mitigating the risks associated with the compromise of such third parties.

Similar to EIFFeL, the enhanced scheme assumes a blockchain platform and multiple (n) clients, and it relies on the same building blocks. In each iteration/epoch, the interactions are shown in Figure 5, where the setup phase is shown in Figure 6 and the numbering of steps corresponds to the description in Figure 7. The overall design of the enhanced scheme is agnostic to any blockchain platform; however, the interaction details between clients and the blockchain (i.e., the smart contracts) may differ in the implementation. To keep the generality and simplicity, we skip the details in our description. The design of smart contracts for the proposed scheme is quite straightforward based on the following fact: when invoking a smart contract, the requester (i.e., a client) can pass the location of necessary input parameters (i.e., where these data are located on the blockchain) so that the smart contract can fetch the data and perform the desired operations in a deterministic manner.

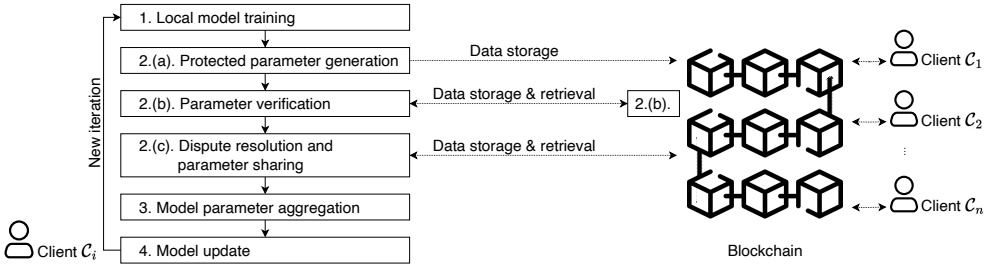

Figure 5. System architecture of enhanced scheme.

To start FL training, the setup phase is depicted in Figure 6, and the training epoch, shown in Figure 7, will be iterated until a satisfactory global model is achieved. By its nature, information stored on a blockchain is not supposed to be updated. In our description in Figure 7, if a piece of information is stored in the blockchain, then it means that a transaction is made to the blockchain to store the information. When we say that a piece of information is updated (e.g., $\mathcal{C}^*$), we simply mean that a new version of this information is stored on the blockchain, and all follow-up computations should be based on this new version of information.

– Setup:
  - Prepare all the necessary smart contracts for the chosen **blockchain**.
  - Generate a security parameter, k, the number of clients, n, the threshold for malicious clients, m, and a filed $\mathbb{F}$ for secret sharing usage. Initialize the malicious client list, $\mathcal{C}^* = \emptyset$, and lists $Flag[i] = \emptyset$ of clients that have flagged the client, $\mathcal{C}_i$, as malicious. A validation predicate, $Valid(\cdot)$, is chosen to guarantee integrity, and a collision-resistant hash function H is chosen. All the parameters are stored on the **blockchain**.
  - A group, $\mathbb{G}$, and its generator, g, are generated for the Burmester-Desmedt group key agreement protocol (Burmester et al., 2005); $\mathbb{G}$ and g are stored on the **blockchain**.
  - Generate $pp \leftarrow KA.gen(k)$ for a two-party key agreement scheme, KA. Based on pp, each client, $\mathcal{C}_i$, generates its key pair $(pk_i, sk_i) \xleftarrow{\$} KA.gen(pp)$ and stores the public key on the **blockchain**. Note that pp is also stored on the **blockchain**. Choose a standard symmetric key encryption scheme, SE (e.g., AES in the counter mode), and store its information on the **blockchain**.
  - Each client, $\mathcal{C}_i$, establishes $n-1$ common secret keys $sk_{i,j} \leftarrow KA.agree(pk_j, sk_i)$ with every other client. It stores the hash values $\mathsf{H}(sk_{i,j}||i||j)$ $(1 \leq j \neq i \leq n)$ on the **blockchain**.
  - All clients check the hash values of secret keys generated by others and resolve any mistakes, if there are any. Note that $sk_{i,j}$ can be computed by both clients, $\mathcal{C}_i$, and $\mathcal{C}_j$, so that they can check each other's hash values.

Figure 6. Setup of the Enhanced scheme [27].

1. **Local model training**:
   Each client $\mathcal{C}_i$:
   - Generates its local update, $\boldsymbol{u}_i \in \mathbb{F}^d$, which is a value of d dimensions.
2. **Model parameter sharing**: The following steps are performed:
   (a). Each client, $\mathcal{C}_i$, generates its protected parameters as follows:
      i. Generates a proof $\boldsymbol{\pi}_i = (\boldsymbol{h}_i, (\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i)), \boldsymbol{h}_i \in \mathbb{F}[X], (\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i) \in \mathbb{F}^3, \boldsymbol{a}_i \cdot \boldsymbol{b}_i = \boldsymbol{c}_i$ for the computation $Valid(\boldsymbol{u}_i) = 1$.
      ii. Creates shares of its update $\boldsymbol{u}_i$ for all clients, denoted as $\{(1, \boldsymbol{u}_{i,1}], \cdots, (n, \boldsymbol{u}_{i,n}), \Psi_{\boldsymbol{u}_i}\} \leftarrow SS.share(u, [n], m+1)$.
      iii. Creates shares of its proof $\boldsymbol{\pi}_i$ for other clients: $\{(1, \boldsymbol{h}_{i,1}], \cdots, (n, \boldsymbol{h}_{i,n}), \Psi_{\boldsymbol{h}_i}\} \leftarrow SS.share(\boldsymbol{h}_i, [n] \setminus i, m+1)$, $\{(1, \boldsymbol{a}_{i,1}], \cdots, (n, \boldsymbol{a}_{i,n}), \Psi_{\boldsymbol{a}_i}\} \leftarrow SS.share(\boldsymbol{a}_i, [n] \setminus i, m+1)$, $\{(1, \boldsymbol{b}_{i,1}], \cdots, (n, \boldsymbol{b}_{i,n}), \Psi_{\boldsymbol{b}_i}\} \leftarrow SS.share(\boldsymbol{b}_i, [n] \setminus i, m+1)$, $\{(1, \boldsymbol{c}_{i,1}], \cdots, (n, \boldsymbol{c}_{i,n}), \Psi_{\boldsymbol{c}_i}\} \leftarrow SS.share(\boldsymbol{c}_i, [n] \setminus i, m+1)$.
      iv. Publishes the encrypted proof strings $\forall \mathcal{C}_j \in \mathcal{C}_{\setminus i}, \overline{(j, \boldsymbol{u}_{j,1})||(j, \boldsymbol{\pi}_{j,1})} \leftarrow SE.enc_{sk_{i,j}}((j, \boldsymbol{u}_{i,j})||(j, \boldsymbol{\pi}_{i,j}))$, $\boldsymbol{\pi}_{i,j} = \boldsymbol{h}_{i,j}||\boldsymbol{a}_{i,j}||\boldsymbol{b}_{i,j}||\boldsymbol{c}_{i,j}$ with its check strings $(\Psi_{\boldsymbol{u}_i}, \Psi_{\boldsymbol{\pi}_i})$ on the **blockchain**.

   (b). The protected parameters are verified as follows:
      i. Verifies validity of secret shares:
      Each client $\mathcal{C}_i$:
      (A). Retrieves and decrypts the shares pertaining to it: $(i, \boldsymbol{u}_{j,i})||(i, \boldsymbol{\pi}_{j,i}) \leftarrow SE.dec_{pk_{i,j}}(\overline{(i, \boldsymbol{u}_{j,i})||} \overline{(i, \boldsymbol{\pi}_{j,i})})$.
      (B). Verifies the shares $\boldsymbol{u}_{j,i}(\boldsymbol{\pi}_{j,i})$ using the check string $\Psi_{\boldsymbol{u}_i}(\Psi_{\boldsymbol{\pi}_i})$.
      (C). If any share fails to be decrypted or verified, flag its creator on the **blockchain**.
      Blockchain (via smart contract):
      (A). Upon receiving a report (e.g., $\mathcal{C}_i$ flags $\mathcal{C}_j$), updates $Flag[j] = Flag[j] \cup \mathcal{C}_i$.
      (B). Updates $\mathcal{C}^*$ as follows:
      - If $|Flag[i]| > m$, $\mathcal{C}_i$ is marked as malicious: $\mathcal{C}^* = \mathcal{C}^* \cup \mathcal{C}_i$.
      - If any client, $\mathcal{C}_i$, is reported as more than m clients, it is considered malicious: $\mathcal{C}^* = \mathcal{C}^* \cup \mathcal{C}_i$.
      - For every client $\mathcal{C}_j$ that has been reported, but $|Flag[j]| \leq m$, suppose that $\mathcal{C}_i$ has flagged $\mathcal{C}_j$; perform the following steps:
        (I). Asks $\mathcal{C}_i$ and $\mathcal{C}_j$ to publish $sk_{i,j}$ and $sk_{j,i}$ on the **blockchain**.
        (II). Compares these keys to the hash values from the setup phase; if some of them do not match, then flag the owner and update $\mathcal{C}^*$. Note that these keys should be equal and their hash values should be equal to that stored in the setup phase.
        (III). Uses $sk_{i,j}$ to decrypt the encrypted share from $\mathcal{C}_j$ to $\mathcal{C}_i$, and verifies the result. If everything passes, update $\mathcal{C}^* = \mathcal{C}^* \cup \mathcal{C}_i$; otherwise, update $\mathcal{C}^* = \mathcal{C}^* \cup \mathcal{C}_j$.
      ii. Generation of proof summaries:
      Blockchain (via smart contract):
      - Hash the current data on **the blockchain** to generate a random number $r \in \mathbb{F}$.
      Each client $\mathcal{C}_i$:
      - Generates a summary $\sigma_{j,i}$ of the proof string $\boldsymbol{\pi}_{j,i}$ based on r and SNIP, $\forall \mathcal{C}_j \notin \mathcal{C}^*$, and publishes it on the **blockchain**.
      iii. Verification of proof summaries:
      Blockchain (via smart contract):
      - Collects and verifies all proof summaries from $\mathcal{C} \setminus \mathcal{C}^*$ with $robustRecon(\cdot)$, and updates $\mathcal{C}^*$ based on the result.

   (c). Each client $\mathcal{C}_i$:
      i. If $\mathcal{C}_i \in \mathcal{C}^*$, it can raise a dispute. If any successful dispute occurs, all clients abort the protocol.
      ii. Selects a random number, r_i, and generates the Burmester-Desmedt group key agreement material g^{r_i}, publishes g^{r_i} on the **blockchain**.
      iii. Generates $tk_i = (\frac{g^{r_{i+1}}}{g^{r_{i-1}}})^{r_i}$ and publishes it on the **blockchain**.
      iv. Computes the Burmester-Desmedt group key as $ek_i = (tk_{i-1})^{l \cdot r_i} \cdot (tk_i)^{n-1} \cdots tk_{i-2}$, supposing l clients are participating at this stage; publishes $\mathsf{H}(ek_i||i)$ on the **blockchain**.
      v. Checks that all other clients possess the same ephemeral key as ek_i.
      vi. If so, publishes the aggregate $SE.enc_{ek_i}(\mathcal{U}_i)$, where $\mathcal{U}_i = \sum_{C_j \notin \mathcal{C}^*} u_{j,i}$, on the **blockchain**.
3. **Model parameter aggregation**: Each client, $\mathcal{C}_i$, recovers the final aggregate $\mathcal{U} \leftarrow robustRecon(\{x, \mathcal{U}_x)\}_{C_x \notin \mathcal{C}^*}$, after obtaining all the $\mathcal{U}_x$ after decryption.
4. **Model update**: Each client, $\mathcal{C}_i$, resets its local model with the parameters, $\mathcal{U}$.

Figure 7. Training procedures in one iteration/epoch.

4.1. Security Analysis

Similar to other setup phases in cryptosystems, we assume this phase is performed in a trusted manner. Depending on the precise FL scenario, some trusted entity might be needed to supervise and help carry out this phase. The chosen blockchain platform

should be tamper-resistant and will not collude with any entity in the FL system (i.e., none of the clients should be able to collude with the blockchain). For simplicity, we assume there is a secure channel between the blockchain and any client, in the sense that the client will be authenticated for its access to the blockchain, and the communication is protected regarding its integrity. Similar to EIFFeL, we assume that, at most, $m < \lfloor \frac{n}{3} \rfloor$ malicious users are tolerated, and the threshold of Shamir's t-out-of-n secret sharing scheme is also set as $t = m + 1$. We omit attacks with the single purpose of denial of service (DoS). In practice, such attacks can be prevented by additional security mechanisms.

In comparison to the EIFFeL scheme, the major philosophy of the enhanced scheme is to use the blockchain platform to replace both the public bulletin board and the server. To this end, all the actions performed by the blockchain platform are deterministic. Apart from the new setup phase, the following major changes have been made: (1) regarding the fourth comment in Section 3.3, a new verification procedure has been proposed for the blockchain platform to trace the malicious client. (2) Leveraging the blockchain platform, the clients execute a Burmester-Desmedt group key agreement protocol [27] to generate a shared group key. The validity of the key is checked as well. (3) Instead of asking a third party to compute the aggregate, the clients store the encrypted shares under the group key on the blockchain platform, so that they can recover the aggregate locally. In contrast to the EIFFeL scheme, the need for a dedicated server has been eliminated. As a result, the risks following the compromise of the server are also eliminated. By asking the clients to share the credentials and intermediary results on the blockchain platform, the clients can check by themselves the validity of the credentials and results from other users. This is facilitated by making the validation operations deterministic.

It is worth stressing that the enhanced scheme preserves all the privacy and integrity properties of the EIFFeL scheme due to the fact that all the security and integrity mechanisms have been kept in the new scheme.

4.2. Performance Analysis

We implemented the enhanced scheme in Python with several existing libraries, such as NumPy [28] and Cryptography [29] (the source code of the enhanced scheme instantiation is available at https://github.com/MoienBowen/Blockchain-Federated-Learning (accessed on 1 April 2024)). In the experiment, we used a PC as the simulation platform, with an Intel® Core™i7-4770 CPU @ 3.4 GHz processor with 16 GB RAM.

Dataset. Given our primary focus on 5G security, we selected a dataset tailored to this domain: the 5G-NIDD dataset [30], a robust compilation of network intrusion data specifically designed for 5G wireless networks, containing 1,215,890 records. This dataset was meticulously generated on an operational 5G testbed, part of the 5G test network (5GTN) at the University of Oulu, Finland. It features an extensive range of simulated attack scenarios, offering a valuable dataset for AI/ML model training. Detailed categories and their respective distributions are presented in Table 1.

Setup. Our parameter selection was guided by our goal of 128-bit security. For example, we used the Secp256r1 curve for the Burmester-Desmedt group key agreement protocol [27], AES-CTR for encryption, SHA-256 for hashing, and a 256-bit prime field as $\mathbb{F}$. Moreover, we considered that there were $n = 50, 100, 150$, and 200 clients, tolerating up to $m = \frac{n}{10}$ malicious ones, with $d = 1000$ as the size of the update gradient vector. Each client owned 24,053, 12,156, 8100, and 6075 records, respectively (each category of the dataset was divided equally and randomly into n copies). We omitted the consensus time of the blockchain but this did not affect the representation of the protocol performance.

Table 1. The 5G-NIDD dataset attack categories and distribution.

Category	Number of Records
Benign	477,737
ICMPFlood	1155
HTTPFlood	140,812
SlowrateDos	73,124
SYNFlood	9721
SYNScan	20,043
TCPConnectScan	20,052
UDPFlood	457,340
UDPScan	15,906

Results. In Table 2, we emphasize the runtime of Step 2—model parameter sharing and Step 3—model parameter aggregation, as the performance mainly depends on the number of clients. The values represent the average execution times of single iterations/epochs over 10 runs.

Table 2. Runtime of proposed scheme in ms.

Step	Participant	Runtime (ms)			
		n = 50	100	150	200
2.(a).	Client	491.37	974.21	1477.88	1990.13
	Blockchain	-	-	-	-
2.(b).	Client	1073.43	2106.11	3204.75	4283.49
	Blockchain	995.14	1997.64	3091.71	4107.93
2.(c).	Client	2079.58	4215.38	6314.49	8284.27
	Blockchain	-	-	-	-
3.	Client	171.21	354.14	498.87	637.61
	Blockchain	-	-	-	-

Regarding the blockchain in Step 2, we calculate the computational time on the PC we use. In practice, the computational time will be based on the mining nodes. Moreover, the complexity will also come from the consensus protocol used in the blockchain platform. Obtaining precise running time statistics on chosen blockchain platforms will be looked at in future work.

5. Conclusions

This paper culminates with an in-depth discussion of the enhanced scheme for FL, which harnesses the power of blockchain technology. The scheme eliminates the need for a dedicated server and a public bulletin board in the EIFFeL scheme, thereby reducing the risks associated with the compromise of these entities. The clients share credentials and intermediary results on the blockchain platform, allowing them to verify the validity of credentials and results from other users. This paper provides a performance analysis of the enhanced scheme, demonstrating its efficiency and effectiveness in the realm of FL. This paper shows that blockchain technology can enhance the privacy and integrity of FL and open up new possibilities for collaborative ML in various domains.

There are several directions for future work. Here, we only mention two examples. One is to fully implement the proposed scheme, particularly by selecting a blockchain platform, and encoding the desired operations into smart contracts. This will help us understand the overall complexity of the proposed scheme. The other one is to incorporate the concept of differential privacy to further reduce the privacy risks. To this end, it

is interesting to see how privacy enhancement could affect accuracy and other metrics. This will help us understand the precise trade-offs between privacy protection and the performance of FL.

Author Contributions: Conceptualization, Q.T.; formal analysis, B.L. and Q.T.; funding acquisition, B.L. and Q.T.; investigation, B.L. and Q.T.; methodology, B.L. and Q.T.; project administration, Q.T.; software, B.L.; supervision, Q.T.; validation, B.L. and Q.T.; writing—original draft, B.L. and Q.T.; writing—review and editing, B.L. and Q.T. All authors have read and agreed to the published version of the manuscript.

Funding: Bowen Liu and Qiang Tang are supported by the 5G-INSIGHT bi-lateral project (ANR-20-CE25-0015-16), funded by the Luxembourg National Research Fund (FNR) and the French National Research Agency (ANR).

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Davies, H. Ted Cruz Using Firm That Harvested Data on Millions of Unwitting Facebook Users. Available online: https://www.theguardian.com/us-news/2015/dec/11/senator-ted-cruz-president-campaign-facebook-user-data (accessed on 4 January 2024).
2. European Parliament; Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council. Available online: https://data.europa.eu/eli/reg/2016/679/oj (accessed on 4 May 2016).
3. Krishnan, S.; Anand, A.J.; Srinivasan, R.; Kavitha, R.; Suresh, S. *Federated Learning*; CRC Press: Boca Raton, FL, USA, 2024.
4. Boenisch, F.; Dziedzic, A.; Schuster, R.; Shamsabadi, A.S.; Shumailov, I.; Papernot, N. Reconstructing Individual Data Points in Federated Learning Hardened with Differential Privacy and Secure Aggregation. In Proceedings of the 2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P), Delft, The Netherlands, 3–7 July 2023; IEEE Computer Society: Piscataway, NJ, USA, 2023; pp. 241–257.
5. Melis, L.; Song, C.; De Cristofaro, E.; Shmatikov, V. Exploiting unintended feature leakage in collaborative learning. In Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 691–706.
6. Yin, H.; Mallya, A.; Vahdat, A.; Alvarez, J.M.; Kautz, J.; Molchanov, P. See through gradients: Image batch recovery via gradinversion. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2021; pp. 16337–16346.
7. Lyu, L.; Yu, H.; Ma, X.; Chen, C.; Sun, L.; Zhao, J.; Yang, Q.; Yu, P.S. Privacy and Robustness in Federated Learning: Attacks and Defenses. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, 1–21. [CrossRef] [PubMed]
8. Adilova, L.; Böttinger, K.; Danos, V.; Jacob, S.; Langer, F.; Markert, T.; Poretschkin, M.; Rosenzweig, J.; Schulze, J.P.; Sperl, P. Security of AI-Systems: Fundamentals. Available online: https://doi.org/10.24406/publica-1503 (accessed on 15 March 2024).
9. Blanchard, P.; El Mhamdi, E.M.; Guerraoui, R.; Stainer, J. Machine learning with adversaries: Byzantine tolerant gradient descent. *Adv. Neural Inf. Process. Syst.* **2017**, *30*.
10. Fang, M.; Cao, X.; Jia, J.; Gong, N. Local model poisoning attacks to {Byzantine-Robust} federated learning. In Proceedings of the 29th USENIX security symposium (USENIX Security 20), Boston, MA, USA, 12–14 August 2020; pp. 1605–1622.
11. Kairouz, P.; McMahan, H.B.; Avent, B.; Bellet, A.; Bennis, M.; Bhagoji, A.N.; Bonawitz, K.; Charles, Z.; Cormode, G.; Cummings, R.; et al. Advances and Open Problems in Federated Learning. *Found. Trends Mach. Learn.* **2021**, *14*, 1–210. [CrossRef]
12. Bell, J.H.; Bonawitz, K.A.; Gascón, A.; Lepoint, T.; Raykova, M. Secure single-server aggregation with (poly) logarithmic overhead. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual, 9–13 November 2020; pp. 1253–1269.
13. Bonawitz, K.; Ivanov, V.; Kreuter, B.; Marcedone, A.; McMahan, H.B.; Patel, S.; Ramage, D.; Segal, A.; Seth, K. Practical secure aggregation for privacy-preserving machine learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1175–1191.
14. Kairouz, P.; Liu, Z.; Steinke, T. The distributed discrete gaussian mechanism for federated learning with secure aggregation. In Proceedings of the International Conference on Machine Learning, PMLR, Virtual, 18–24 July 2021; pp. 5201–5212.
15. Liu, B.; Pejó, B.; Tang, Q. Privacy-Preserving Federated Singular Value Decomposition. *Appl. Sci.* **2023**, *13*, 7373. [CrossRef]
16. Roy Chowdhury, A.; Guo, C.; Jha, S.; van der Maaten, L. Eiffel: Ensuring integrity for federated learning. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, Los Angeles, CA, USA, 7–11 November 2022; pp. 2535–2549.
17. Diedrich, H. *Ethereum: Blockchains, Digital Assets, Smart Contracts, Decentralized Autonomous Organizations*; Wildfire Publishing: Sydney, Australia, 2016.

18. Narayanan, A.; Bonneau, J.; Felten, E.; Miller, A.; Goldfeder, S. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*; Princeton University Press: Princeton, NJ, USA, 2016.
19. Swan, M. *Blockchain: Blueprint for a New Economy*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2015.
20. Qammar, A.; Karim, A.; Ning, H.; Ding, J. Securing federated learning with blockchain: A systematic literature review. *Artif. Intell. Rev.* **2023**, *56*, 3951–3985. [CrossRef] [PubMed]
21. Yu, F.; Lin, H.; Wang, X.; Yassine, A.; Hossain, M.S. Blockchain-empowered secure federated learning system: Architecture and applications. *Comput. Commun.* **2022**, *196*, 55–65. [CrossRef]
22. Shamir, A. How to share a secret. *Commun. ACM* **1979**, *22*, 612–613. [CrossRef]
23. Lin, S.; Costello, D.J. *Error Control Coding: Fundamentals and Applications*; Pearson/Prentice Hall: Upper Saddle River, NJ, USA, 2004.
24. Corrigan-Gibbs, H.; Boneh, D. Prio: Private, robust, and scalable computation of aggregate statistics. In Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), Boston, MA, USA, 27–29 March 2017; pp. 259–282.
25. Suwito, M.H.; Tama, B.A.; Santoso, B.; Dutta, S.; Tan, H.; Ueshige, Y.; Sakurai, K. A Systematic Study of Bulletin Board and Its Application. In Proceedings of the ASIA CCS '22: ACM Asia Conference on Computer and Communications Security, Nagasaki, Japan, 30 May–3 June 2022; Suga, Y., Sakurai, K., Ding, X., Sako, K., Eds.; ACM: New York, NY, USA, 2022; pp. 1213–1215.
26. Tramèr, F.; Shokri, R.; Joaquin, A.S.; Le, H.; Jagielski, M.; Hong, S.; Carlini, N. Truth Serum: Poisoning Machine Learning Models to Reveal Their Secrets. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, 7–11 November 2022; Yin, H., Stavrou, A., Cremers, C., Shi, E., Eds.; ACM: New York, NY, USA, 2022; pp. 2779–2792.
27. Burmester, M.; Desmedt, Y. A secure and scalable Group Key Exchange system. *Inf. Process. Lett.* **2005**, *94*, 137–143. [CrossRef]
28. Python Cryptographic Authority. Python Library NumPy. Available online: https://numpy.org/ (accessed on 13 February 2024).
29. Oliphant, T.; Contributors Community. Python Library Cryptography. Available online: https://cryptography.io/en/latest/ (accessed on 13 February 2024).
30. Samarakoon, S.; Siriwardhana, Y.; Porambage, P.; Liyanage, M.; Chang, S.Y.; Kim, J.; Kim, J.; Ylianttila, M. 5G-NIDD: A Comprehensive Network Intrusion Detection Dataset Generated over 5G Wireless Network. *arXiv* **2022**, arXiv:2212.01298.