

AnoFel: Supporting Anonymity for Privacy-Preserving Federated Learning

Ghada Almashaqbeh
University of Connecticut
Storrs, Connecticut, USA
ghada@uconn.edu

Zahra Ghodsi
Purdue University
West Lafayette, Indiana, USA
zahra@purdue.edu

Abstract

Federated learning enables users to collaboratively train a machine learning model over their private datasets. Secure aggregation protocols are employed to mitigate information leakage about the local datasets from user-submitted model updates. This setup, however, still leaks the user *participation* in training, which can also be sensitive. Protecting user anonymity is even more challenging in dynamic environments where users may (re)join or leave the training process at any point of time.

This paper introduces AnoFel, the first framework to support private and anonymous dynamic participation in federated learning (FL). AnoFel leverages several cryptographic primitives, the concept of anonymity sets, differential privacy, and a public bulletin board to support anonymous user registration, as well as unlinkable and confidential model update submission. Our system allows dynamic participation, where users can join or leave at any time without needing any recovery protocol or interaction. To assess security, we formalize a notion for privacy and anonymity in FL, and formally prove that AnoFel satisfies this notion. To the best of our knowledge, our system is the first solution with provable anonymity guarantees.

To assess efficiency, we provide a concrete implementation of AnoFel, and conduct experiments showing its ability to support learning applications scaling to a large number of clients. For a TinyImageNet classification task with 512 clients, the client setup to join is less than 3 sec, and the client runtime for each training iteration takes a total of 8 sec, where the added overhead of AnoFel is 46% of the total runtime. We also compare our system with prior work and demonstrate its practicality. AnoFel client runtime is up to 5× faster than Truex et al. [73], despite the added anonymity guarantee and dynamic user joining in AnoFel. Compared to Bonawitz et al. [16], AnoFel is only 2× slower for added support for privacy in output, dynamic user joining, and anonymity.

Keywords

Anonymity, private federated learning, dynamic participation

1 Introduction

Privacy-preserving machine learning is a critical problem that has received huge interest from both academia and industry. Many crucial applications involve training ML models over highly sensitive user data, such as medical screening [35], credit risk assessment [33],

or autonomous vehicles [21]. Enabling such applications requires ML frameworks that preserve the privacy of users' datasets.


Federated learning (FL) aims to achieve this goal by offering a decentralized paradigm for model training. Participants, or clients, train the model locally over their datasets, and then share only the local gradients with the model owner, or the server. After aggregating updates from all clients, the server shares the updated model with these clients to start a new training iteration. This iterative process continues until the model converges.

However, individual model updates leak information about clients' private datasets [58, 63], and therefore aggregation should be done in a secure way: a server only sees the aggregated value rather than individual contributions from each client. A large body of work emerged building cryptographic protocols for secure aggregation to support private federated learning, e.g., [10, 16, 69, 71, 73, 79].

Anonymous client participation. A related question to preserving data privacy in federated learning is preserving client participation anonymity. That is, protecting client identity and breaking linkability with any information that could be deduced from training. Anonymity is critical for training models over sensitive data related to, e.g., user health, financial information, or other sensitive attributes such as ethnicity or sexual orientation. The mere knowledge that a user has participated in a training task could leak that they suffer from a particular disease or that they belong to a protected population group. Such leakage invades privacy, and may discourage participation.

As an example application, consider the users of smart watches or wearable fitness trackers. Roughly one-in-five U.S. adults say they regularly use such health monitoring devices [74], with researchers already using data from these apps for health research [76]. However, a majority of Americans surveyed either oppose sharing data from fitness trackers with medical researchers, or are unsure if such data sharing is an acceptable practice [74]. Using data from fitness trackers can help researchers monitor sleep in patients with neurocognitive disorders [8], or study the differences in cardiovascular health in sexual minority adults [19]. Participation in such studies however, reveals very sensitive information about users which they might not be willing to disclose. In this case, protecting the privacy of data (as is done in prior secure aggregation protocols) is not sufficient and does not hide the fact that a user participated in a study. This issue adds to already existing concerns among clients about the use of data, and might discourage user participation. Therefore, preserving participation anonymity is essential in such scenarios to ensure comprehensive user privacy.

Existing frameworks for private federated learning either don't support client participation anonymity, or suffer from security

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license visit <https://creativecommons.org/licenses/by/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Proceedings on Privacy Enhancing Technologies 2025(2), 88–106
© 2025 Copyright held by the owner/author(s).
<https://doi.org/10.56553/popets-2025-0051>

issues. Secure aggregation protocols [10, 16] require full identification of clients, through a public key infrastructure (PKI) or a trusted client registration process, to prevent Sybil attacks and impersonation. Even frameworks that deal with semi-honest adversaries [16, 69, 71, 73, 79] assign clients logical identities, where the mapping between the logical and real identities is known to the server or a trusted third party. On the other hand, techniques that anonymize datasets [52, 55, 72, 81] do not support participation anonymity, but rather hide sensitive information in the dataset before being used in training. At the same time, existing solutions for anonymous client participation have several limitations [23, 28, 40, 53, 83]: they either rely on fixed pseudonyms that are susceptible to traffic analysis attacks, assume a trusted party to mediate communication so this party is aware of the client participation behavior, or are vulnerable to man-in-the-middle attacks.

Dynamic settings. Allowing clients to (re)join or leave at any time is invaluable for training tasks targeting dynamic environments. A decentralized activity as federated learning may deal with heterogeneous settings involving weak clients who may use low-power devices or have unstable network connectivity. Even it could be the case that clients simply change their minds and abort the training protocol after it starts. The ability to support this dynamicity at low overhead promotes participation, but it is a more challenging setup for client anonymity.

Most privacy-preserving federated learning solutions do not support dynamic participation: usually clients must join at the onset of the training process during the setup phase, and stay until training concludes. Several solutions support client dropouts (at a relatively high overhead by employing interactive recovery protocols that reveal dropout identities) [16, 71, 73, 79] but not addition, or support both dropout and addition but at the expense of a constrained setup that places a cap on the number of clients who can participate [78]. To the best of our knowledge, supporting anonymity in a dynamic environment has been absent from the current state of the art.

An open question. Therefore, we ask the following question: *can we achieve private federated learning that supports users’ anonymity in both static and dynamic settings?*

1.1 Our Contributions

In this paper, we answer this question in the affirmative and present AnoFel, a system that fulfills the requirements above (Table 1).¹ In particular, we make the following contributions.

Formal security notion and analysis. Our goal is to build a provably-secure system, which requires a well-defined notion formally outlining the security and privacy guarantees that must be achieved. Towards this goal, we define a notion for private and anonymous federated learning (PAFL) that encompasses three properties: correctness, anonymity, and dataset privacy. Then, we formally prove the security of AnoFel based on this notion. To the best of our knowledge, we are the first to provide such formal definition covering anonymity, and the first to build a provably-secure client anonymity solution for private federated learning. We believe that our PAFL definition is of independent interest as

¹The work [79] optimizes the efficiency of [16] using the same architecture, thus it inherits the same features of [16] summarized in the table.

FL Protocol	Privacy in Computation	Privacy in Output	Dynamic Dropout	Dynamic Joining	Participation Anonymity
Ryffel et al. [69]	✓	✗	✗	✗	✗
Bonawitz et al. [16]	✓	✗	✓	✗	✗
Truex et al. [73]	✓	✓	✓	✗	✗
Xu et al. [78]	✓	✓	✓	✓*	✗
AnoFel (this work)	✓	✓	✓	✓	✓

Table 1: Comparison of FL protocols (representative works shown). Privacy in computation refers to protecting individual updates, whereas privacy in output refers to preventing privacy leakage from the output of aggregation (e.g., membership attacks). * Max number of clients is fixed at setup.

it is general enough to be used by other private and anonymous federated learning protocols.

System design. We instantiate AnoFel using several cryptographic primitives and privacy techniques to achieve participation anonymity for both static and dynamic settings. We observe that achieving full user participation anonymity requires protection not only during the training phase, but also during the registration phase when clients sign up for the training task. As such, our systems involves (1) an anonymous registration process guaranteeing that only legitimate clients with honestly-generated datasets can participate, and (2) an unlinkable model update submission that cannot be traced back to the originating client. We rely on anonymity sets and zero-knowledge proofs to achieve this, where a client proves owning a legitimate dataset (during registration), and being one of the registered clients (during model update submission), without revealing anything about their identity or registration information.

To support dynamic user participation and secure aggregation of model updates, our system employs threshold homomorphic encryption. Our insight is to split the roles of a model owner from the aggregators, and use a committee of aggregators to distribute trust. Aggregators receive encrypted model parameter updates (gradients) from users, and at the end of a training iteration, they (homomorphically) add these ciphertexts to produce a ciphertext of the aggregation. Afterwards, the aggregators decrypt the result allowing the model owner to access the aggregated plaintext updates so that a new training iteration can be started. This configuration of AnoFel removes clients’ involvement in the aggregation or decryption processes, and hence allows them to (re)join and leave training at any time without interrupting system operation. Furthermore, AnoFel employs a bulletin board to provide a persistent log accessible to all parties and facilitate indirect communication between them to reduce interaction. Another advantage of AnoFel, due to the use of the bulletin board, is that all clients receive the same initial model for any iteration, which addresses privacy attacks resulting from distributing different initial model parameters to clients [66]. To the best of our knowledge, AnoFel is the only private federated learning system that enjoys this advantage.

We observe that model (or training output) related attacks, such as membership, inference, and model inversion attacks, could compromise participation anonymity especially if the revealed data sample is known to belong to a particular client. To mitigate this issue, we employ differential privacy (DP). Unlike existing solutions which usually employ local DP at the client level, AnoFel employs DP at the aggregation level (without compromising privacy). In particular, clients perform local training and submit their encrypted

model updates that will be aggregated (by the aggregator committee) at the end of the iteration. However, before decrypting the aggregated updates, each aggregator will sample a noise value, encrypt it, and share it with the rest of the committee. We use ideas from [73] to reduce the noise amount from each aggregator by a factor of encryption threshold as we will describe later. All noise values are then added to the aggregated updates (using homomorphic add), after which decryption can take place. Our DP approach has several advantages; it supports dynamic settings and reduces the impact on accuracy. That is, regardless of the number of participating clients in an iteration, the appropriate noise level that satisfies the privacy leakage guarantees obtained by DP is always being added during the aggregation stage.

Implementation and evaluation. To show practicality, we implement AnoFel and empirically evaluate its performance covering different federated learning tasks. We demonstrate scalability of our system by testing scenarios that involve large numbers of clients and contemporary models—the benchmarked architectures are the largest studied in privacy-preserving federated learning literature [25, 54, 71, 73]. We show that AnoFel, with its supported features, incurs a reasonable added cost in runtime.

In a network of 16 clients, the client setup needed in AnoFel to join the training task takes less than 2 sec, and each training iteration for MNIST, CIFAR10, and TinyImageNet classification takes the client a total of 1.7, 3.1, and 7.8 sec, where the added overhead of AnoFel is 65%, 59%, and 48% of total runtime (including local training time) respectively. Compared to Truex et al. [73], AnoFel client runtime is up to 1.3×, 3.5×, and 5× faster across on MNIST, CIFAR10, and TinyImageNet benchmarks respectively. Despite the added anonymity guarantee in AnoFel and supporting dynamic participation, our system outperforms Truex et al. in client runtime due to employing DP at the aggregation level and implementing optimizations for parallelized client computation.

Bonawitz et al. [16] is only faster by 1.3× on MNIST, 1.8× on CIFAR10, and 2× on TinyImageNet for 512 clients due to client runtime growing quadratically with the number of clients, whereas AnoFel grows logarithmically. Additionally, their framework does not provide privacy in output, dynamic user joining, or participation anonymity.

Furthermore, we evaluate the accuracy of models trained with AnoFel for a range of number of clients and data distribution settings (IID and non-IID) and show that, compared to a non-DP baseline, the accuracy drop due to DP for the MNIST dataset is between 0.05% – 0.22%, for CIFAR10 is between 0.79% – 3.68%, and for TinyImageNet is between 8.11% – 12.27%.

2 A Security Notion for Private and Anonymous Federated Learning

In this section, we define a formal security notion for a private and anonymous federated learning scheme (PAFL). This notion, and its correctness and security properties, are inspired by [18, 31, 45, 70].

Notation. We use λ to denote the security parameter, α to denote correctness (or accuracy) loss parameter, γ to denote the privacy loss (or leakage) advantage of the adversary (α and γ are parameters tied to any non-cryptographic privacy technique used in the scheme, if any), $\text{negl}(\cdot)$ to denote negligible functions, and boldface letters to

represent vectors. The variable state represents the system state, which include the data recorded on the bulletin board (these posted by all parties, and the public parameters of the system building blocks). The notation $\mathcal{A}^{\mathcal{O}}$ means that an entity, in this case the adversary \mathcal{A} , has an oracle access to \mathcal{O} . Lastly, $\stackrel{\$}{\leftarrow}$ denotes drawn at random, and PPT is a shorthand for probabilistic polynomial time.

DEFINITION 1 ((α, γ)-PAFL SCHEME). Let Π be a protocol between a server S , set of aggregators \mathcal{AG} , and a set of clients \mathcal{C} such that each client $cl_i \in \mathcal{C}$ holds a private dataset D_i . Let \mathbf{M} be the initial model that S wants to train, \mathbf{M}_{Π} be the trained model produced by Π , and \mathbf{M}_{actual} be the model produced by training \mathbf{M} over the datasets in the clear (the datasets of the participating clients). Π is a private and anonymous federated learning (PAFL) scheme, parameterized by α and γ , if it satisfies the following properties for every \mathbf{M} :

- **α -Correctness:** The model trained by Π achieves an error bound α with high probability compared to the actual model. That is, for $\alpha \geq 0$ and an error function Er , with high probability we have $\text{Er}(\mathbf{M}_{actual}, \mathbf{M}_{\Pi}) \leq \alpha$.
- **Anonymity:** Any PPT adversary \mathcal{A} has a negligible advantage in winning the anonymity game AnonGame . Formally, for a security parameter λ , there exists a negligible function negl such that \mathcal{A} wins AnonGame with probability at most $\frac{1}{2} + \text{negl}(\lambda)$, where the probability is taken over all the randomness used by \mathcal{A} and Π .
- **γ -Dataset Privacy:** Any PPT adversary \mathcal{A} has a negligible additional advantage over γ in winning the dataset indistinguishability game DINDGame . Formally, for a security parameter λ and $\gamma \geq 0$, there exists a negligible function negl such that \mathcal{A} wins DINDGame with probability at most $\frac{1}{2} + \gamma + \text{negl}(\lambda)$, where the probability is taken over all randomness used by \mathcal{A} and Π .

Intuitively, a PAFL scheme is one that is correct and provides anonymity and dataset privacy for clients. Ideally, correctness guarantees that the outcome of a PAFL scheme (i.e., the final trained model) is identical to what will be produced by a training scheme that gets full access to the clients’ datasets. Anonymity means that no one can tell whether a client has registered or participated in any training iteration. In other words, submitted model updates, or any other information a client uses for registration, cannot be traced back to this client. Dataset privacy means that no additional information will be leaked about the private datasets of honest clients beyond any prior knowledge the adversary has.

To make our definition more general, we account for the use of non-cryptographic privacy techniques, such as differential privacy, that may result in accuracy and privacy loss. We do that by parameterizing our notion with α and γ that stand for correctness (or accuracy loss) and indistinguishability (or privacy loss) parameters, respectively. Having $\alpha = \gamma = 0$ reduces to the ideal case in which $\mathbf{M}_{actual} = \mathbf{M}_{\Pi}$, and an adversary has negligible advantage in breaking anonymity and dataset privacy. The bounds for α and γ are derived based on the non-cryptographic privacy techniques employed in the system. Furthermore, our notion accounts for dynamic client participation where not all clients may participate in every single training iteration. Thus, correctness is defined over \mathbf{M}_{actual} and \mathbf{M}_{Π} with respect to the same client participation pattern.

We define two security games to capture anonymity and dataset privacy, denoted as AnonGame and DINDGame, respectively, and the interfaces offered by a PAFL scheme. The goal is to protect the client’s anonymity and privacy such that no one can tell whether the client was involved in the training task, or reveal anything about their dataset. To capture that, we assume \mathcal{A} controls S and any subset of clients and aggregators given the conditions specified by the security games; that is, at least two clients in the system are honest, and the aggregator committee \mathcal{AG} can have at most $t - 1$ corrupted members, where t is the threshold required to correctly reveal the aggregated updates. All parties receive the security parameter λ , and are given oracle access to $\mathcal{O}_{\text{PAFL}}$. $\mathcal{O}_{\text{PAFL}}$ maintains the state of the system, including the set of registered clients and aggregators, and any additional information recorded on the board. $\mathcal{O}_{\text{PAFL}}$ supports the following query types:²

- (setup, 1^λ): takes the security parameter as input and sets up the system accordingly—creating the bulletin board, the public parameters needed by all parties/cryptographic building blocks, and the bounds/parameters needed by any additional non-cryptographic privacy techniques employed in the system. This command can be invoked only once.
- (register, p , aux): registers party p that could be a client or an aggregator committee. The field aux specifies the party type and its input: if p is a client, then aux will include its certified dataset and the certification information, while if p is an aggregator committee, aux will include, e.g., the committee’s public key. This command can be invoked anytime and as many times as desired.
- (train, cl, aux): instructs a (registered) client cl to train the model using its dataset and submit the model updates. The field aux defines the dataset that belongs to cl (the exact information is based on Π). This command can be invoked anytime and as many times as desired.
- (access): returns the updated model (after aggregating all submitted individual model updates received in an iteration). For any iteration, this command can be invoked only once and only at the end of that iteration.
- (corrupt, p , aux): allows \mathcal{A} to corrupt party p that could be a client or an aggregator. If p is a client, then aux will be the registration information of this client (e.g., in AnoFel, it is the dataset commitment as we will see later), while if p is an aggregator, aux will be the public key of that party. This command can be invoked at anytime allowing \mathcal{A} to corrupt up to $t - 1$ members in \mathcal{AG} , and up to $|\mathcal{C}| - 2$ clients.

Note that the notation cl only represents the type of a party to be a client, it does not contain its real identity.

Accordingly, the AnonGame proceeds as follows:

- (1) $b \xleftarrow{\$} \{0, 1\}$
- (2) state \leftarrow (setup, 1^λ)
- (3) $(cl_0, aux_0, cl_1, aux_1) \leftarrow \mathcal{A}^{\mathcal{O}_{\text{PAFL}}}(1^\lambda, \text{state})$
- (4) (train, cl_b, aux_b)
- (5) \mathcal{A} continues to have access to $\mathcal{O}_{\text{PAFL}}$
- (6) At the end, \mathcal{A} outputs b' , if $b' = b$ and:
 - (a) both cl_0 and cl_1 are honest,

²Note that aux is only a holder of auxiliary information that is parsed differently across queries (and across security games); it does not have a fixed meaning.

- (b) and there are at least two honest clients participated in every training iteration, and that these clients remained honest until the end of the game, then return 1 (meaning that \mathcal{A} won the AnonGame), otherwise, return 0.

Note that \mathcal{A} has access to the current state of the system at anytime, and can see the updated bulletin board after the execution of any command. Also, \mathcal{A} can access the updated model at the end of any iteration, and like everyone in the system, it can see all messages sent from the parties to the board (since all will be published on the board). However, if these messages are, e.g., encrypted, all what \mathcal{A} sees is the ciphertext but not the plaintext, and so on depending on the actual protocol instantiation. For the chosen clients, aux_i represents their registration information (which does not include the client identity or its actual dataset D_i).³

As shown, AnonGame captures the smallest anonymity set $u = 2$, which implies the case $u > 2$ in which there are u clients and the adversary tries to figure out which client among this set has participated in training. That is, for $u = 2$, the guarantee is that \mathcal{A} has negligible probability over a random guess success probability (i.e., $1/2$) to win. Having a larger anonymity set will not help \mathcal{A} since the random guess success probability will become smaller (i.e., $1/u$) and still \mathcal{A} ’s additional advantage is negligible.⁴ Furthermore, since \mathcal{A} can choose any two clients for the challenge, AnonGame also captures anonymity and unlinkability of registration. That is, although \mathcal{A} picks any client registration information (namely, aux_i) from the board, not winning the anonymity game means that \mathcal{A} cannot link the registration information to the client identity or model update submission. If the registration information can be linked to a model update submission, then \mathcal{A} can always win.

AnonGame game includes several conditions to rule out trivial adversary wins. The two clients that \mathcal{A} selects must be honest, otherwise, if any is corrupt, it will be trivial to tell which client was chosen. Furthermore, since aggregating model updates is simply computing the summation of these updates, if only cl_b participates in the iteration during which the challenge command is executed, it might be trivial for \mathcal{A} to win (same for any other iteration if only one honest client participates). This is because \mathcal{A} can access the updated model at the end of that iteration and can extract cl_b ’s individual updates. Thus, we add the condition that there must be at least two honest clients participated in any iteration, and these have to remain honest until the end of the game. Also, the aggregator committee \mathcal{AG} can have at most $t - 1$ corrupt parties, which is included in \mathcal{A} ’s capabilities as defined in the corrupt query stated earlier. Otherwise, if \mathcal{A} controls t members, it can access individual model updates at anytime, which may allow deducing information about the chosen client.⁵

The DINDGame proceeds as follows:

- (1) $b \xleftarrow{\$} \{0, 1\}$
- (2) state \leftarrow (setup, 1^λ)
- (3) $(D_0, aux_0, D_1, aux_1) \leftarrow \mathcal{A}^{\mathcal{O}_{\text{PAFL}}}(1^\lambda, \text{state})$

³If the adversary knows the dataset and identity of a client, then it knows that this client is part of the population, i.e., this client suffers from illness, for example; there is no point of hiding whether that client participated in training.

⁴Similar observation applies to DINDGame (which we define next).

⁵For correctness, we have $|\mathcal{AG}| \geq 2t - 1$ to guarantee that the aggregator committee has at least t honest members to allow revealing the aggregated updates correctly.

- (4) (register, $cl, (D_b, aux)$), (train, cl, aux)
- (5) \mathcal{A} continues to have access to \mathcal{O}_{PAFL}
- (6) \mathcal{A} outputs b' , if $b' = b$, and:
 - (a) cl is honest,
 - (b) and there is at least two honest clients participated in every training iteration, and that these clients remained honest until the end of the game, then return 1 (meaning that \mathcal{A} won the DINDGame), otherwise return 0.

This game follows the outline of AnonGame, but with a different construction of the challenge command to reflect dataset privacy. In particular, \mathcal{A} chooses two valid datasets (the field aux_i contains all information required to verify validity). The challenger chooses one of these datasets at random (based on the bit b), queries \mathcal{O}_{PAFL} to register a client using this dataset, and then instructs this client to train the model using the dataset D_b . Note that aux in line 4 is the registration information of the client constructed based on the Π scheme. \mathcal{A} continues to interact with the system, and can access the updated model at the end of any iteration. \mathcal{A} wins the game if it guesses correctly which of the datasets was chosen in the challenge where as before, conditions are added to rule out trivial adversary wins. Being unable to guess after seeing the outcome of the challenge train command, and even after accessing the aggregated model updates, means that a PAFL schemes does not reveal anything about the underlying datasets.

Note that the definition of DINDGame covers defending against membership inference attacks. That is, in the game \mathcal{A} chooses two datasets D_0 and D_1 among which the challenger will choose *one* at random and use it to register a client cl that will participate in training (so \mathcal{A} does not know whether D_0 or D_1 was selected). \mathcal{A} can access the trained model via \mathcal{O}_{PAFL} (using the query access) at the end of each iteration (including the last iteration when training concludes). Thus, in a protocol that does not protect against membership inference attacks, \mathcal{A} can simply take data points from D_0 and D_1 , and after accessing the trained model, it can perform a membership attack and figure out which dataset has been selected for training, thus always wins the game. On the other hand, a protocol that defends against membership attacks (like AnonGame), this strategy will not succeed (up to the privacy guarantees of the employed techniques) and will not help \mathcal{A} in winning DINDGame.

REMARK 1. *Our PAFL notion can be further generalized to have only the model owner S , i.e., no aggregators, so this party is the one who aggregates the individual model updates as well. Moreover, if preserving model privacy is required, then our notion can be extended with an additional property to reflect that. Since model privacy is outside the scope of this work, we did not include this property to keep the definition simple.*

REMARK 2. *It should be noted that our definition of anonymity (and so our scheme that satisfies this notion) does not leak negative information (i.e., a client has not participated in training). Both participation and the absence of participation are protected, i.e., identities of those who participate or do not participate are not revealed.*

REMARK 3. *We remark that anonymity at the network layer level (i.e., defending against leakage resulting from monitoring network traffic) is out of scope. In this work, we focus on anonymity at the*

application layer, which without it (even if we have anonymity at the network layer) client participation anonymity cannot be achieved. When employing a solution for network layer anonymity, all honest clients must use it during training (including the two clients selected in AnonGame). For example, say an anonymous communication protocol is used, a client participating in a training round will use this protocol to send her registration information and model updates, and a client who is not participating must mimic a similar behavior by sending dummy traffic. This is needed to avoid giving the adversary any advantage in the game based on absence of communication. Of course, employing such techniques would require modifying the protocol to, e.g., exclude dummy updates from aggregation so they will not invalidate the training output, and this exclusion must be done in a private way. We leave the integration of AnonGame with network layer anonymity solutions as a future work.

3 Building Blocks

In this section, we provide a brief background on the building blocks employed in AnonGame, covering all cryptographic primitives that we use and the technique of differential privacy.

Commitments. Cryptographic commitments allow a party to commit to a secret value she owns such that this commitment can be opened later. A commitment scheme consists of three PPT algorithms: Setup, Commit, and Open. On input the security parameter λ , Setup generates a set of public parameters pp . To commit to a value x , the committer invokes Commit with inputs pp, x , and randomness r to obtain a commitment c . Open(pp, c) opens a commitment by simply revealing x and r . Anyone can verify correctness of opening by computing $c' = \text{Commit}(pp, x, r)$ and check if $c = c'$.

A secure commitment scheme must satisfy: *hiding*, so that a commitment c does not reveal any information about x beyond any pre-knowledge the adversary has, and *binding*, so a commitment c to x cannot be opened to another value $x' \neq x$ (formal definitions can be found in [37]). These security properties enable a party to commit to their inputs (i.e., private datasets in our case), and publish the commitment publicly without exposing the private data.

Threshold homomorphic encryption. Homomorphic encryption allows computing over encrypted inputs and producing encrypted outputs. Such operations include homomorphic addition and multiplication. That is, let ct_1 be a ciphertext of x_1 , and ct_2 be a ciphertext of x_2 , then $ct_1 + ct_2$ produces a ciphertext of $x_1 + x_2$, and $ct_1 \cdot ct_2$ produces a ciphertext of $x_1 \cdot x_2$ (the exact implementation of the homomorphic '+' and '.' vary based on the encryption scheme). Some encryption schemes support only one of these operations, e.g., Paillier scheme [65] is only additively homomorphic. Supporting both addition and multiplication leads to fully homomorphic encryption [17, 61]. Since we focus on secure aggregation of model updates, we require only additive homomorphism.

A homomorphic encryption scheme is composed of three PPT algorithms: KeyGen that generates encryption/decryption keys (and any other public parameters pp), Encrypt that encrypts an input x to produce a ciphertext ct , and Decrypt that decrypts a ciphertext ct to get the plaintext x . Correctness states that Decrypt produces the original plaintext for any valid ciphertext produced by Encrypt, and that the homomorphic operations produce a correct ciphertext of the correct result. Security is based on the regular security

notion for encryption (i.e., semantic security), where we require indistinguishability against chosen-plaintext attacker (CPA).

The threshold capability is about who can decrypt the ciphertext. To distribute trust, instead of having the decryption key known to a single party, this key is shared among n parties. Thus, each of these parties can produce a partially decrypted ciphertext upon calling Decrypt, and constructing the plaintext requires at least t parties to decrypt. As such, in a threshold homomorphic encryption scheme, KeyGen will produce one public key (for encryption) and n shares of the secret key (for decryption), such that each party will obtain only her share (and will not see any of the others' shares). In AnoFel, we use the threshold Paillier encryption scheme [27].

Zero-knowledge proofs. A (non-interactive) zero-knowledge proof (ZKP) system allows a prover, who owns a private witness ω for a statement x in language \mathcal{L} , to convince a verifier that x is true without revealing anything about ω . A ZKP scheme is composed of three PPT algorithms: Setup, Prove, and Verify. On input a security parameter λ and a description of \mathcal{L} , Setup generates public parameters pp . To prove that $x \in \mathcal{L}$, the prover invokes Prove over pp , x , and a witness ω for x to obtain a proof π . To verify this proof, the verifier invokes Verify over pp , x , and π , and accepts only if Verify outputs 1. In general, all conditions needed to satisfy the NP relation of \mathcal{L} are represented as a circuit. A valid proof will be generated upon providing valid inputs that satisfy this circuit. Some of these inputs could be public—which the verifier will use in the verification process, while others are private—which constitute the witness ω that only the prover knows.

A secure ZKP system must satisfy completeness, soundness, and zero-knowledge. Completeness states that a proof generated in an honest way will be accepted by the verifier. Soundness ensures that if a verifier accepts a proof for a statement x then the prover knows a witness ω for x , i.e., the prover cannot convince the verifier with false statements. Finally, zero-knowledge ensures that a proof π does not reveal anything about the witness ω . Many ZKP systems add a succinctness property, meaning that the proof size is constant and verification time is linear in the input size and independent of the circuit size representing the NP relation. These are called zero-knowledge succinct non-interactive argument of knowledge (zk-SNARKs). Formal definitions of ZKP systems can be found in [15, 37]. In AnoFel, we use the proof system proposed in [38].

Differential privacy. Differential privacy (DP) is a technique usually used in federated learning to address training output-related attacks. Examples include membership and inference attacks, in which knowing a data point and the trained model enables an attacker to tell if this data point was used in training the model. DP guarantees that the inclusion of a single instance in the training datasets causes a statistically insignificant change to the training algorithm output (i.e., the trained model). Formally, DP is defined as follows [30] (where $\epsilon > 0$ and $0 < \delta < 1$):

DEFINITION 2 (DIFFERENTIAL PRIVACY). *A randomized mechanism \mathcal{K} provides (ϵ, δ) -differential privacy, if for any two datasets D_0 and D_1 that differ in a single entry, and for all $R \subseteq \text{Range}(\mathcal{K})$, we have: $\Pr[\mathcal{K}(D_0) \in R] \leq e^\epsilon \Pr[\mathcal{K}(D_1) \in R] + \delta$.*

We adopt the Gaussian DP mechanism as described in [75], but at the aggregator level rather than at the client level, and we apply an

optimization for the secure aggregation setting inspired by [45, 73]. That is, each aggregator in the committee \mathcal{AG} samples a noise from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$ divided by the threshold t . Then, after aggregation and before decryption, each aggregator adds the generated (encrypted) noise value to the aggregated updates. Thus, after decryption, the resulted aggregation contains the required noise level that satisfies the desired DP privacy guarantee.

During training, each client cl_i clips their model gradients \mathbf{g}_i to ensure that $\|\mathbf{g}_i\| < C$, where C is a clipping threshold for bounding the norm of gradients. Client cl_i then sets sensitivity $S_f = 2C/|D_i|$ where D_i is the i -th client's dataset, assuming gradients are shared after one epoch of local training. The noise scale is set as $\sigma \geq cTS_f/\epsilon$ where $c \geq \sqrt{2 \ln(1.25/\delta)}$ and T indicates exposures of local parameters (number of epochs or iterations), and satisfies (ϵ, δ) -DP for $\epsilon < 1$ [7, 30, 75].

The parameters of DP determines the accuracy bound for the trained model, and the success probability of the adversary in terms of the bounded privacy leaks. As for the error bound α , the Gaussian mechanism satisfies an absolute error bound $\alpha \leq O(R\sqrt{\log k})$, where k is the number of queries and $R := S_f\sqrt{k \log 1/\delta}/\epsilon$ [26, 29] (a query here indicates one training round, so k is the number of training iterations, and S_f is the query sensitivity, which is the sensitivity of the training function). The privacy leakage, or the adversary advantage γ for any strategy it may follow, also depends on the chosen values for δ and ϵ . Several works analyzed such bounds, for example, the work in [43] computed a bound for the success probability in membership attacks when employing DP, and in [64] DP bounds were computed for various adversary instances. Hence, γ resembles the union bound of the success probability of the attacker regardless of its strategy.

In describing our design, we use DP in a blackbox manner. That is, an aggregator in \mathcal{AG} invokes a function called DP.noise to sample the appropriate noise value. This makes our design modular as any secure DP mechanism, other than the one we employ, can be used.

4 AnoFel Design

AnoFel relies on four core ideas to achieve its goals: certification of clients' datasets to prevent impersonation, anonymity sets to support anonymous registration and model training, a designated aggregator committee to prevent accessing the individual model updates submitted by the clients, and a public bulletin board to facilitate indirect communication and logging. In this section, we present the design of AnoFel showing the concrete techniques behind each of these ideas and how they interact with each other.

4.1 System Model

As shown in Figure 1, for any learning task there is a model owner S , a model aggregator set (or committee) \mathcal{AG} of size n (we use a committee instead of a single aggregator to distribute trust and avoid single point of failure in the system), and a set of clients \mathcal{C} who want to participate in this learning task. During a training iteration, \mathcal{C} will train the initial model locally over their private datasets and publish encrypted updates. S has to wait until the iteration is finished, after which \mathcal{AG} will aggregate the updates and grant S access to the aggregated value. AnoFel can support any aggregation method based on summation or averaging (e.g., FedSGD [22] and

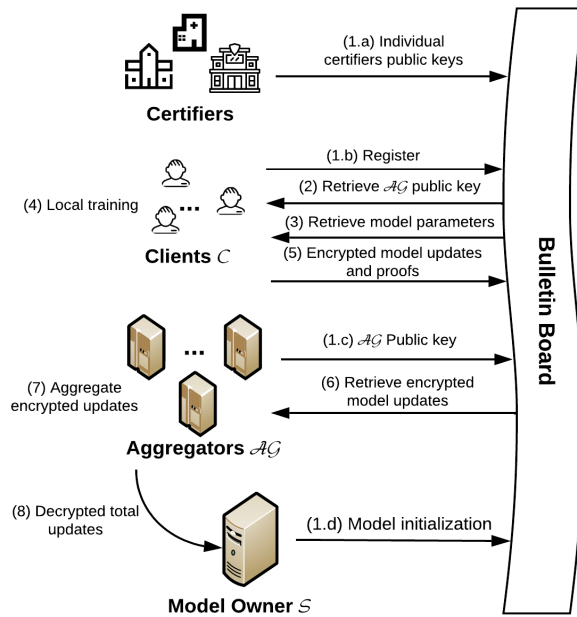


Figure 1: AnoFel system architecture. All steps (except 1.a - 1.c, and 2) will be repeated for each training iteration.

FedAVG [57]) given that the number of participants is public. All parties have access to a public bulletin board (that is instantiated in a decentralized way), allowing them to post and retrieve information about the training process. There could be several learning tasks going on in the system, each with its own S , \mathcal{AG} , and \mathcal{C} , and all are using the same bulletin board.⁶ In the rest of this section, we focus on one learning task to explain our protocol; several learning tasks will separately run the same protocol between the involved parties.

4.2 Threat Model

We assume a secure and immutable public bulletin board available to all parties, so it is an append-only, publicly-accessible log that accepts only authenticated information that complies with predefined correctness rules.⁷

We adopt the following adversary model (we deal with PPT adversaries). For clients, we assume them to be malicious during registration (a malicious party may behave arbitrarily), while we assume these clients to be semi-honest during training (a semi-honest party follows the protocol but may try to collect any additional information while executing the protocol). Thus, during registration, a client with an invalid (or poisoned) dataset may try to register, while during training, registered clients will use their valid (registered) datasets in training and submit valid updates. We assume the server S to be malicious, so it may try to impersonate clients in the registration phase, collude with a subset of the aggregators and/or clients during the training phase, or manipulate the model posted at the beginning of each iteration. For \mathcal{AG} , which is also semi-honest, we require its size $n \geq 2t - 1$, and we assume that at maximum $t - 1$

⁶In fact, it could be the case that the same parties are involved in several learning tasks, but they track each of these separately.

⁷This is instantiated in a decentralized way using a blockchain with miners verifying correctness (more details can be found in Appendix A).

members in \mathcal{AG} can be corrupt, where t is the threshold required for valid decryption. Lastly, we work in the random oracle model where hash functions are modeled as random oracles.

4.3 System Workflow

AnoFel achieves anonymity and privacy by combining a set of cryptographic primitives, such as threshold homomorphic encryption and zero-knowledge proofs (ZKPs), differential privacy (DP), and a public bulletin board. The latter is used to facilitate indirect communication between the parties and to create anonymity sets to disguise the participants. Our techniques of combining ZKPs and anonymity sets are inspired by recent advances in private and anonymous cryptocurrencies [18, 31, 70].

Each client must register before participation (step 1.b in Figure 1) by publishing on the board a commitment to the master public key and the dataset this client owns (commitments are never opened and don't leak any information). Similarly, the aggregators \mathcal{AG} must register by posting their public key on the board (step 1.c in Figure 1), which will be used by the clients to encrypt their model updates. The encrypted updates will be accompanied with a ZKP attesting that a client is a legitimate and registered data owner, but without revealing the public key or the dataset commitment of this client. Thus, anonymity is preserved against everyone (the server, aggregators, other clients, and any other party).

As shown in Figure 1 (steps 1.d and 3), at the beginning of each training iteration, the server publishes the initial model parameters on the bulletin board, which are retrieved by the clients to be used in the local training. The use of this immutable public board avoids direct communication between the server and clients; direct interaction would compromise anonymity. Moreover, it addresses privacy attacks resulting from distributing different initial model parameters to clients (this type of attacks has been recently demonstrated in [66]). In fact, to the best of our knowledge, AnoFel is the only private federated learning system that enjoys this advantage.

To prevent Sybil and data poisoning attacks, each dataset must be certified by its source, e.g., a hospital or a police station, and in the case of smart devices, by the manufacturer or the controller that collects authenticated data collected from these devices. Since exposing the certifier reveals the dataset type and impacts privacy, AnoFel creates an anonymity set for the certifiers by having all their public keys posted on the bulletin board (step 1.a in Figure 1). Thus, during registration, a client will provide a ZKP that its hidden (committed) dataset is signed by one of the certifiers, i.e., so this signature is valid under one of the registered certifiers' public keys, but without specifying which one.

Furthermore, we use DP to protect against training output-related attacks that may compromise anonymity. That is, if the adversary knows a data sample that belongs to a particular client, attacks such as membership attacks may reveal if this sample has been used in training, thus revealing that a client has participated. In DP, a noise value is added to the submitted encrypted updates before decryption. We do that at the aggregation phase without involving the clients. In particular, at the end of each iteration, each aggregator aggregates the submitted encrypted updates, then it encrypts a noise value (chosen based on the DP parameters employed) and combines it with the aggregated updates (using homomorphic

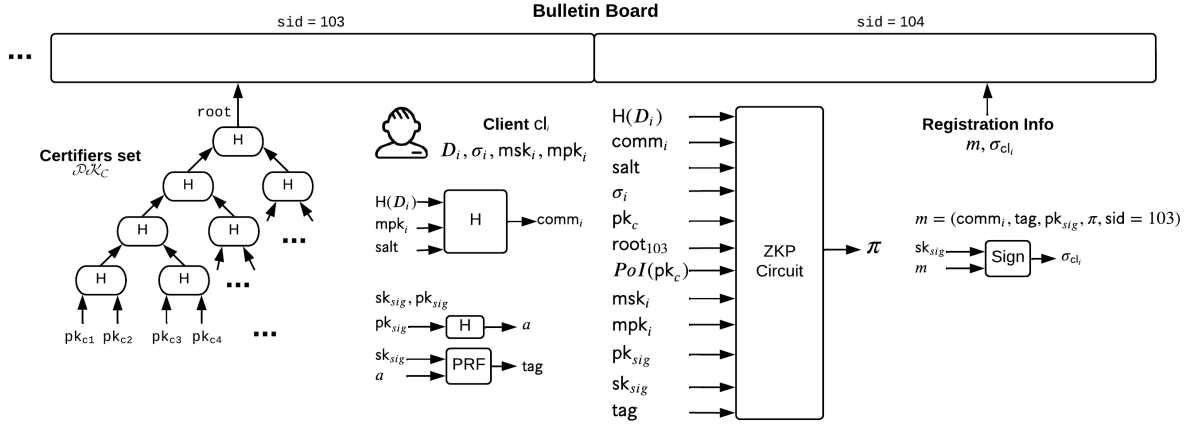


Figure 2: Client setup—anonymous registration process.

add). After that, each aggregator partially decrypts the resulted ciphertext and sends it to S . This approach supports dynamic client participation and reduces impacts on training accuracy; regardless of the number of clients participating in an iteration, the proper amount of noise is being added during the aggregation phase.⁸

Accordingly, AnoFel proceeds in three phases: setup, model training, and model access, as we discuss below.

4.3.1 Setup. The system setup includes creating the public bulletin board, generating all public parameters needed by the cryptographic primitives used, and configuring the DP parameters based on the desired accuracy and privacy levels. Then, the certifiers, aggregators, and each client run the setup process as follows.

Certifiers. Each certifier posts its public key on the board. Let \mathcal{PK}_C be the set of all certifiers’ public keys.

Aggregators. \mathcal{AG} run the setup of the threshold homomorphic encryption scheme, and generate a public key and shares of the secret key. They post the public key on the board, while each party keeps its secret key share. Note that a PKI is needed to ensure the real identities of the certifiers, aggregators, and model owner. Thus, posting the public key of any of these entities on the board requires a certificate (from a certificate authority in the PKI) to prove that indeed the party owns this key.

Clients. The client setup is more involved compared to the previous entities. This is a natural result of supporting anonymity. As shown in Figure 2, which is the detailed version of step 1.b in Figure 1, each client $cl_i \in \mathcal{C}$, with a dataset D_i and a master keypair (msk_i, mpk_i) ,⁹ obtains a certificate σ_i from its certifier. The certificate σ_i could be simply the certifier’s signature over $H(D_i) \parallel mpk_i$, where H is a collision resistant hash function. Then, cl_i commits to its dataset D_i (without revealing anything about it) as follows:

- Compute a commitment to D_i and mpk_i as (salt is a fresh random string in $\{0, 1\}^\lambda$): $comm_i = H(H(D_i) \parallel mpk_i \parallel salt)$.
- Generate a fresh digital signature keypair (pk_{sig}, sk_{sig}) , then compute $a = H(pk_{sig})$ and $tag = PRF_{msk_i}(a)$, where PRF is a pseudorandom function and tag serves as an authentication tag over the fresh key to bind it to the client’s master keypair. Similar to [70], we instantiate the PRF as $PRF_{msk_i}(a) = H(msk_i \parallel a)$.
- Generate a ZKP π to prove that the dataset is legit and owned by cl_i . In particular, this ZKP attests to the following statement (again without revealing anything about any of the private data that the client owns): given a commitment $comm_i$, a signature verification key pk_{sig} , a tag tag, and a bulletin board state index sid, client cl_i knows a dataset D_i , randomness salt, master keypair (mpk_i, msk_i) , a certifier’s key pk_c , a certificate σ_i issued by this certifier over D_i and mpk_i , and a signing key sk_{sig} , such that:
 - (1) $H(D_i)$, mpk_i , and salt are a valid opening for $comm_i$, i.e., $comm_i = H(H(D_i) \parallel mpk_i \parallel salt)$.¹⁰
 - (2) σ_i has been generated using pk_c over $H(D_i) \parallel mpk_i$, and that $pk_c \in \mathcal{PK}_C$ with respect to the set \mathcal{PK}_C registered on the board at state with index sid.
 - (3) The client owns mpk_i , i.e., she knows the msk_i corresponding to mpk_i .
 - (4) The tag over the fresh pk_{sig} is valid, i.e., compute $a = H(pk_{sig})$ and check that $tag = PRF_{msk_i}(a)$.
- Sign the proof and $comm_i$: set $m = (comm_i, tag, pk_{sig}, \pi, sid)$ and use sk_{sig} to sign m to obtain signature σ_{cl_i} .
- Post (m, σ_{cl_i}) on the bulletin board.

As for verifying the third condition, i.e., the client knows msk_i , it is simply done by computing the public key based on the input msk_i and checking that it is equal to mpk_i . Although mpk_i is not recorded explicitly on the bulletin board, it is bound to the client since it is part of the dataset commitment $comm_i$ certified by σ_i .

⁸Letting clients add noise to the updates before encryption impact accuracy since the aggregated updates will include excessive noise. Truex et al. [73] divide the noise scale by the number of clients so that, after aggregation, the desired noise level is satisfied. However, this approach does not work for dynamic setting as it requires fixing the number of clients at the onset of an iteration. If a client drops out, or is corrupt so S knows its added noise, the actual noise value will be smaller than desired.

⁹Clients need a PKI for their mpk_i , so a certifier can check that a client owns the presented mpk_i . However, to preserve anonymity, these keys are hidden in the training process, and a certifier cannot link an encrypted model update to the owner’s mpk_i .

¹⁰The commitment opening is a private input the client uses to generate the proof (and same applies to the training phase as we will see shortly). This opening is never revealed to the public; the board records only $comm_i$ and a ZKP on its well-formedness that does not leak anything about the private data used in the proof generation.

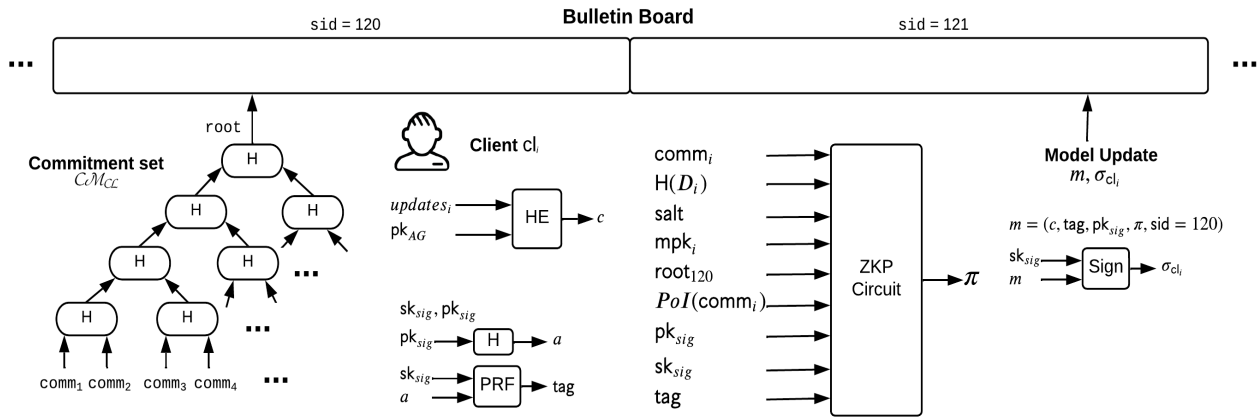


Figure 3: Model training—anonymous model update submission (HE is homomorphic encryption).

To allow efficient proof generation with respect to the anonymity set \mathcal{PK}_C , a Merkle tree is used to aggregate the key set \mathcal{PK}_C as shown in Figure 2. Proving that a key $pk_c \in \mathcal{PK}_C$ is done by showing a proof of inclusion (PoI) of that key in the tree. In other words, the circuit underlying the ZKP generation takes a membership path of the key and the tree root and verifies the correctness of that path. Thus, the cost will be logarithmic in the set size. The tree can be computed by the validators maintaining the board, with the root published on the board to allow anyone to use it when verifying the ZKP.

Note that a ZKP is generated with respect to a specific state of the anonymity set \mathcal{PK}_C . This state is the root of the Merkle tree of this set, which changes when a new certifier joins the system. Such change will invalidate all pending ZKPs, and thus, invalidate all pending client registrations tied to the older state. To mitigate this, a client should specify the state index sid based on which the ZKP (and hence, the Merkle tree) was generated. So if the board is a series of blocks as in Figure 2, sid is the block index containing the root used in the proof.

All the conditions that must be proved by a ZKP are modeled as an arithmetic circuit. The client has to present valid inputs that satisfy this circuit in order to generate a valid ZKP. Only registration with valid ZKPs are accepted, where \mathcal{CM}_{CL} denotes the set of all valid clients' commitments. Registration integrity is preserved due to the use of a secure digital signature scheme: if an adversary tampers with any of the information that a client submits— $comm_i$, tag , pk_{sig} , π , or sid —this will invalidate the signature σ_{cl_i} and will lead to rejecting the registration.

Note that clients can perform the setup at anytime, and once their registration information is posted on the board, they can participate in the model training immediately. Thus, AnoFel allows clients to join at anytime during the training process, and each client can perform the setup phase on their own.

4.3.2 Model Training. At the beginning of each training iteration, S posts the initial model parameters on the bulletin board (step 1.d in Figure 1). Each client cl_i retrieves them and trains the model locally over her dataset (step 3 and 4 in Figure 1, respectively). Then cl_i shares the model updates privately and anonymously without revealing anything about her private dataset or identity,

and without being linked back to this client's dataset commitment. Client cl_i does that as follows (see Figure 3, which is the detailed version of step 5 in Figure 1):

- Encrypt the model updates under \mathcal{AG} 's public key. This will produce a ciphertext c .
- Generate a fresh digital signature keypair (pk_{sig}, sk_{sig}) . Compute $a = H(pk_{sig})$ and $tag = PRF_{msk_i}(a)$.
- Produce a ZKP π (with respect to the current state of the board at index sid) attesting that: cl_i is a legitimate owner of a dataset, and that the fresh digital signature key was generated correctly. Thus, this ZKP proves the following statement: given a signature key pk_{sig} , and a tag tag , cl_i knows the opening of some commitment $comm \in \mathcal{CM}_{CL}$ (this proves legitimacy), and that tag was computed correctly over pk_{sig} as before.¹¹
- To preserve integrity, sign the proof, the ciphertext, and the auxiliary information. That is, set $m = (c, tag, pk_{sig}, sid, \pi)$ and sign m using sk_{sig} to produce a signature σ_{cl_i} .
- Post (m, σ_{cl_i}) on the bulletin board.

We use the Merkle tree technique to aggregate the commitment anonymity set \mathcal{CM}_{CL} . A client provides a PoI of its commitment in the Merkle tree computed over \mathcal{CM}_{CL} with respect to a specific state indexed by sid . The latter is needed since clients can join at anytime, so \mathcal{CM}_{CL} and its Merkle tree might change over time.

Accordingly, AnoFel naturally supports dynamic client participation. As mentioned before, a client who wants to join can do that immediately after finishing the setup. While (registered) clients who do not wish to participate in a training iteration simply do not send any updates. AnoFel does not need a recovery protocol to handle additions/dropouts since the setup of a client does not impact the setup of the system, \mathcal{AG} , or other clients. Also, the model updates submitted by any client do not impact the updates submitted by others. Moreover, any information needed to perform the setup is already on the board, so no interaction with other parties is needed.

¹¹Although we assume semi-honest clients during training, we still need the ZKP above to preserve integrity and ensure that only registered clients can participate. That is, a malicious adversary may impersonate a client during training (without registering), and may alter the submitted updates and sign them using her own pk_{sig} (thus we need to prove that pk_{sig} is honestly generated by a registered client).

Furthermore, submitting model updates is done in one shot; a client posts (m, σ_{cl_i}) on the board. Since we use non-interactive ZKPs, \mathcal{AG} , and any other party, can verify the proof on their own.

4.3.3 Model Access. At the beginning of each training iteration, each member in \mathcal{AG} samples a noise value by invoking DP.noise divided by t , encrypts it under the public key of \mathcal{AG} and posts the noise ciphertext on the board. At the end of the iteration, each member in \mathcal{AG} retrieves all client updates—those that are encrypted under \mathcal{AG} 's public key—from the board, and aggregate them using homomorphic add (steps 6 and 7 in Figure 1, respectively). Furthermore, each member retrieves the n encrypted noise values and add them to the aggregated encrypted updates (again, using homomorphic add). After that, each member decrypts the resulting ciphertext using its secret key share, producing a partial decryption that is sent to S (step 8 in Figure 1). Once S receives at least t responses, it will be able to construct the plaintext of the aggregated model updates and start a new training iteration.

Signaling the end of a training iteration relies on the board, where adding a future block with a specific index will signal the end. Thus, the system setup will determine the block index of when training starts, and the duration of each iteration (in terms of number of blocks). Since all parties have access to the board, they will be able to know when each iteration is over. Another approach is to simply have S post a message on the board to signal the iteration end.

Although AnoFel is a system for federated learning that involves several parties, it is not considered an interactive protocol. These parties do not communicate directly with each other—the bulletin board mediates this communication. When sending any message, the sender will post it on the bulletin board, and the intended recipient(s) will retrieve the message content from the board.

REMARK 4. *A malicious server may post an incorrectly updated model for a new training iteration. AnoFel can address this case; since the initial model is posted on the board, we can let the aggregators post the partially decrypted aggregated updates for each iteration on the board, so that clients can construct the plaintext of the aggregated updates to verify/produce the new model.*

4.4 Extensions

We discuss extensions to our system design to support stronger adversary model (malicious and semi-malicious ones) on both the client and aggregator side, how to instantiate the public bulletin board in a fully decentralized way, and how to optimize its storage cost. Due to space limitation, we discuss these in Appendix A.

4.5 Security of AnoFel

AnoFel realizes a correct and secure PAFL scheme based on Definition 1. In Appendix B, we formally prove the following theorem:

THEOREM 1. *The construction of AnoFel as described in Section 4 is a correct and secure PAFL scheme (cf. Definition 1).*

5 Performance Evaluation

In this section, we provide details on the implementation and performance evaluation of AnoFel, and benchmarks to measure its overhead compared to prior work.

# Clients	Setup		Training	
	Prove (s)	Verify (ms)	Prove (s)	Verify (ms)
16	1.96 ± 0.009	4.86 ± 0.014	0.80 ± 0.003	4.90 ± 0.042
32	2.01 ± 0.002	4.85 ± 0.019	0.86 ± 0.003	4.88 ± 0.008
64	2.09 ± 0.008	4.89 ± 0.038	0.96 ± 0.002	4.82 ± 0.028
128	2.15 ± 0.008	4.84 ± 0.035	1.04 ± 0.004	4.82 ± 0.036
256	2.21 ± 0.014	4.91 ± 0.024	1.14 ± 0.012	4.94 ± 0.062
512	2.27 ± 0.006	4.86 ± 0.020	1.25 ± 0.005	4.85 ± 0.009

Table 2: zk-SNARKs runtime for client setup and training.

5.1 Implementation

For hash functions, we use the Pedersen hash function [42], with an alternative implementation using Baby-Jubjub elliptic curve [11] and 4-bit windows [1], which requires less constraints per bit than the original implementation. For threshold additive-homomorphic encryption, we use the threshold version of Paillier encryption scheme [27] based on [5]. For digital signatures, we use EdDSA [12] over Baby-Jubjub elliptic curve based on [2]. For zero-knowledge proofs (ZKPs), we use Groth16 zk-SNARKS [38] implemented in libsnark [4]. We use PyTorch [67] to incorporate differential privacy, and implement the FedSGD [22] algorithm for federated learning.¹²

Dataset and Models. We evaluate the performance of AnoFel using three federated learning tasks. Our first benchmark is LeNet5 [51] architecture with 61.7K parameters trained on the MNIST [50] dataset. Our second benchmark is ResNet20 [41] architecture with 273K parameters trained over the CIFAR10 [49] dataset. Our third benchmark is SqueezeNet [39] with 832K parameters trained over TinyImageNet [80] dataset. This benchmark is the largest studied in private federated learning literature [25, 54, 73]. Since batch normalization is not compatible with DP [82], we replace all batch normalization layers with group normalization [77] in ResNet20 and SqueezeNet with negligible effect on accuracy.

Configuration. For our runtime experiments, we consider a network of $N = \{16, 32, 64, 128, 256, 512\}$ clients, and one committee consisting of 3 aggregators. Runtimes are benchmarked on an AMD Ryzen 5995WX CPU with 512GB of memory assuming 64 threads, and the mean of 5 runs and standard deviations are reported for each experiment. We present micro-benchmarks of AnoFel components as well as end-to-end benchmarks for a training iteration. We also evaluate AnoFel accuracy under IID and non-IID dataset settings. For IID, each user's data is sampled uniformly from the dataset. For non-IID, we use Dirichlet distribution with parameter $\alpha = 1$ following the implementation in [60]. The DP parameters are set as $\epsilon = 0.9$, $\delta = 10^{-5}$, and norm clipping threshold $C = 1$ for MNIST and $C = 2$ for CIFAR10 and TinyImageNet benchmarks.¹³

5.2 Results

Runtime Overhead. Table 2 shows the performance of ZKP for clients setup and training circuits. The prove and verify runtimes are reported for different numbers of clients participating in the

¹²We focus on the computational cost of privacy/anonymity guarantees rather than communication delays incurred by the board, so the board was simulated on the machine used in our benchmarks. In Appendix A, we discuss potential board instantiations, and provide overhead costs quoted from the literature to give a sense of the board communication delays.

¹³We will open source our code upon acceptance of the paper.

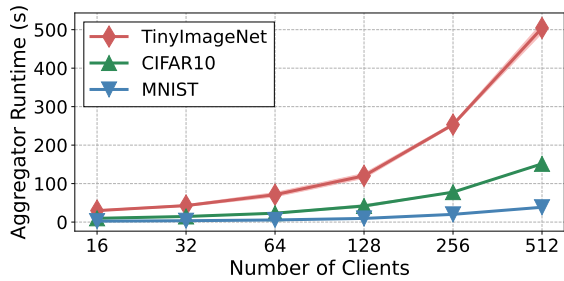


Figure 4: Aggregator computation time for a training iteration for MNIST on LeNet5, CIFAR10 on ResNet20, and TinyImageNet on SqueezeNet.

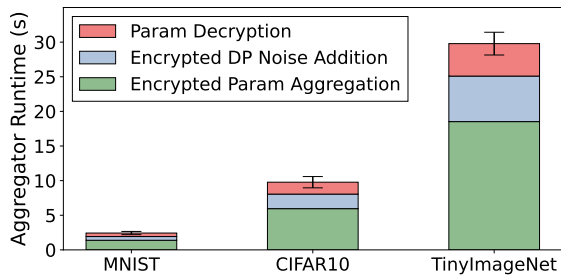


Figure 5: Breakdown of aggregator runtime for each training iteration for MNIST on LeNet5, CIFAR10 on ResNet20, and TinyImageNet on SqueezeNet.

learning task. The size of proof is a constant 1019 bits. The setup prove overhead is a one-time cost for clients, and the training prove overhead is incurred for each training iteration. As the results suggest, the prover runtime for both setup and training increase sub-linearly with the number of clients (due to the use of Merkle trees for anonymity sets as described in Section 4.3), remaining under 2.3 sec for all experiments. The verifier runtime remains constant under 5 msec. Later we will show that ZKP cost is a fraction of the total cost and adds little overhead to the overall runtime.

Next, we show results for an end-to-end training iteration. Figure 4 shows aggregators’ runtime during training for different numbers of clients. Aggregators aggregate the ciphertexts of the client updates using homomorphic addition. Then, they sample noise, encrypt it, and add it to the aggregated updates, followed by a partial decryption after which the result is sent to the model owner. Figure 5 depicts the breakdown of aggregators’ runtime for 16 clients. The cost of encrypting noise and final decryption depend only on the size of the model, while the cost of ciphertext aggregation depend on model size and number of clients. The aggregator runtime for MNIST, CIFAR10, and TinyImageNet benchmarks ranges from 2.4s to 38.5s, 9.8s to 151.4s, and 29.8s to 504.3s, respectively, for the reported number of clients.

The client’s runtime during a training round includes the cost of local model training, model update encryption, and a ZKP generation. The cost of encrypting model updates depends only on the size of the model, whereas the ZKP depends on the number of clients and grows logarithmically (again, due to anonymity set Merkle tree). Figure 6 presents the client runtime with detailed breakdown

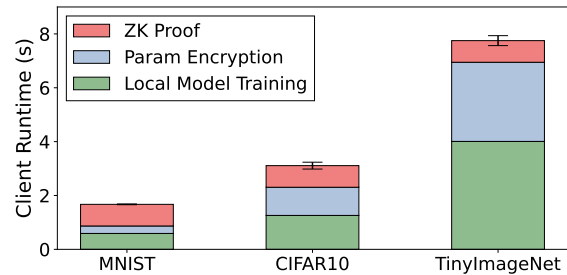


Figure 6: Breakdown of client runtime for each training iteration for MNIST on LeNet5, CIFAR10 on ResNet20, and TinyImageNet on SqueezeNet.

for a training iteration with $N = 16$ clients. The cost of generating keypairs, signatures, and hash computations are negligible, thus omitted from the plot. Local model training cost is measured for a training epoch over $\frac{D}{N}$ images (where D is dataset size and N is number of clients assuming IID distribution) on an NVIDIA RTX A6000 GPU, taking 0.59, 1.26, and 4.01 sec for MNIST, CIFAR10, and TinyImageNet benchmarks, respectively. The remaining protocol costs (encryption and ZKP) are measured using the CPU described in configuration. The overhead of ZKP are 48%, 25%, and 10% of total runtime, and for encryption they are 17%, 34%, and 38%, for MNIST, CIFAR10 and TinyImageNet benchmarks, respectively.¹⁴

Communication Overhead. The communication overhead of the parties during setup and training phases are as follows.

Clients: Client i ’s setup involves the certifier’s signature (96 B), and posting $m = (\text{comm}_i, \text{tag}, \text{pk}_{\text{sig}}, \pi, \text{sid})$ and its signature (360 B). Each training iteration involves obtaining model parameters (121 KB, 527 KB, and 1.6 MB of 16-bit updates for LeNet5, ResNet20, and SqueezeNet, respectively) and posting $m = (\text{c}, \text{AG}_{\text{pk}}, \text{tag}, \text{pk}_{\text{sig}}, \text{sid}, \pi)$ and its signature (9.2 MB, 41 MB, and 124 MB for LeNet5, ResNet20, and SqueezeNet, respectively).

Aggregators: During setup, aggregators post their (signed) public key (160 B). We refer to the size of encrypted model updates as \mathfrak{C} , which equals to 9.2 MB, 41 MB, and 124 MB for LeNet5, ResNet20, and SqueezeNet, respectively. Aggregators sample and post encrypted DP noise (same size as encrypted updates \mathfrak{C}) to the board. In each training iteration, aggregators retrieve encrypted updates submitted by clients (\mathfrak{C} per client), and n (aggregator committee size, where $n = 3$ in our implementation) encrypted DP noise values (total size $n \cdot \mathfrak{C}$). After homomorphic addition, each aggregator sends a partial decryption (of size \mathfrak{C}) to the model owner.

Model Owner: During each training iteration, model owner posts the model updates (121 KB, 534 KB, and 1.6 MB for Lenet5, ResNet20, and SqueezeNet, respectively), and obtains partial ciphertexts from aggregators (9.2 MB, 41 MB, and 124 MB for LeNet5, ResNet20, and SqueezeNet, respectively).

¹⁴The effect of dynamic participation can also be observed from our experiments. Clients perform the setup independently, and during training, only the ZKP depends on the number of clients. Figure 7 shows the effect of varying this number on client runtime. For aggregators, dynamicity impacts only the update aggregation step (since the number of ciphertexts to be added changes). That impact is depicted in Figure 4, showing the varying cost of aggregation with different numbers of clients.

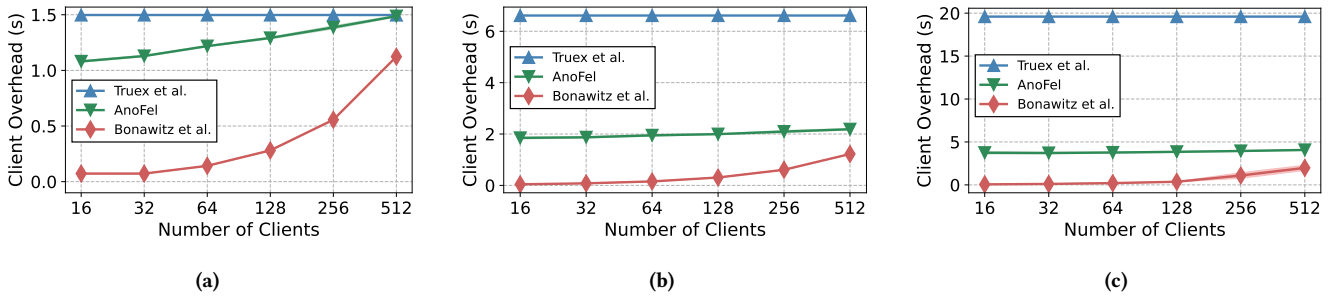


Figure 7: Comparing client computation time for one training round of AnoFel, Truex et al. [73], and Bonawitz et al. [16] for (a) MNIST on LeNet5, (b) CIFAR10 on ResNet20, and (c) TinyImageNet on SqueezeNet.

Comparison to Baseline. To better understand the performance of AnoFel, we provide comparison to prior work on privacy-preserving federated learning. We don’t know of any other framework that provides anonymity guarantees similar to AnoFel, and therefore we chose two recent systems for privacy-preserving federated learning, namely, Truex et al. [73] and Bonawitz et al. [16]. Truex et al. present a non-interactive protocol utilizing threshold homomorphic encryption for secure aggregation, and Bonawitz et al. develop an interactive protocol based on masking client updates.¹⁵

We benchmarked the client runtime for a training iteration in Bonawitz et al. protocol based on implementation found in [3] with fixes to allow more than 40 clients, and Truex et al. protocol using the implementation found in [6] for different number of clients. The results are shown in Figure 7. To simplify comparison, we only plot the overhead of each system, dropping the local model training cost which is the same for all frameworks.

When compared to Truex et al., AnoFel client runtime is up to 1.3 \times , 3.5 \times , and 5 \times faster on MNIST, CIFAR10, and TinyImageNet benchmarks, respectively. It is worth noting that Truex et al. client runtime is independent of the number of clients (model parameter encryption and DP noise addition depend only on the size of the model), whereas in AnoFel it grows logarithmically with the number of clients as can be observed in Figure 7. Nevertheless, in terms of client runtime, AnoFel outperforms Truex et al. for the ranges of the number of clients benchmarked. The reason for this speedup is two fold. First, the DP noise generation and addition in Truex et al. happens on the client side, whereas in AnoFel noise generation and (encrypted) addition is performed by the aggregator committee. Second, AnoFel implements optimizations in the parallel parameter encryption to reduce the context in processes, resulting in up to 1.8 \times reduction in encryption runtime compared to the parallel implementation in Truex et al.

Bonawitz et al. performs secure aggregation using an efficient protocol based on secret sharing. Compared to Bonawitz et al. protocol, AnoFel is at most 15 \times , 44 \times , and 62 \times slower on MNIST, CIFAR10, and TinyImageNet, respectively. However, the runtime gap between Bonawitz et al. and AnoFel reduces with larger numbers of clients. This is because the protocol of Bonawitz et al. requires clients to

generate pairwise masks between each other, thus making the client runtime grow quadratically with the number of clients. On the other hand, the client runtime in AnoFel grows only logarithmically with the number of clients as discussed before. For 512 participating clients, AnoFel is only slower by 1.3 \times on MNIST, 1.8 \times on CIFAR10, and 2 \times on TinyImageNet. Our results demonstrate the scalability of our framework; the cost of the additional features including participation anonymity and support for dynamic settings (not supported by prior work) is relatively low especially in large-scale scenarios.

Model Accuracy. We evaluate the model test accuracy in AnoFel incorporating DP and compare to a non-DP baseline (performing vanilla federated learning) in Figure 8. We assume number of clients $N = 256$ and non-IID data distribution. Accuracy evaluations for IID data and a range of number of clients is included in Appendix C. As shown in Figure 8, AnoFel achieves 98.99%, 85.18%, and 32.90% test accuracy on MNIST, CIFAR10, and TinyImageNet datasets, respectively. For the three datasets, the non-DP accuracy are 99.21%, 88.86%, and 44.05%, respectively. Across all benchmarks and client sizes (as depicted in Figure 9 in Appendix C), the accuracy drop due to DP for the MNIST dataset is between 0.05% – 0.22%, for CIFAR10 is between 0.79%–3.68%, and for TinyImageNet is between 8.11% – 12.27%. We note that Truex et al. accuracy matches AnoFel, whereas Bonawitz et al. has the same accuracy as the non-DP baseline. Specifically, we use a similar noise scaling idea as in Truex et al. (with the only difference being that in AnoFel noise is added at the aggregator level instead of the client level). Bonawitz et al. does not address membership attacks, hence, does not employ DP.

6 Related Work

Private federated learning. Bonawitz et al. [16] introduced one of the earliest schemes on private federated learning. Their scheme handles client dropouts (but not addition) using an interactive protocol, and does not support anonymity; each client is known by a logical identity, which in the malicious setting is tied to a public key (through a PKI) to prevent impersonation. A followup work [10] optimized the overhead of [16] in the semi-malicious model—the server is only trusted to handle client registration. This also violates anonymity since the server has full knowledge of the clients. The works [44, 71, 79] also targeted the efficiency of [16], and all inherit its interactivity issues and lack of anonymity.

Truex et al. [73] use homomorphic encryption (HE) and differential privacy to achieve secure aggregation. HE simplifies handling

¹⁵As shown in Table 1, Xu et al [78] is the closest to our work in terms of supported features (but it does not support anonymity). However, we were not able to find their implementation source code for evaluation. Furthermore, their protocol requires a trusted third party for decrypting aggregation results. Due to differing assumptions, it is challenging to provide a fair comparison between Xu and AnoFel/other work.

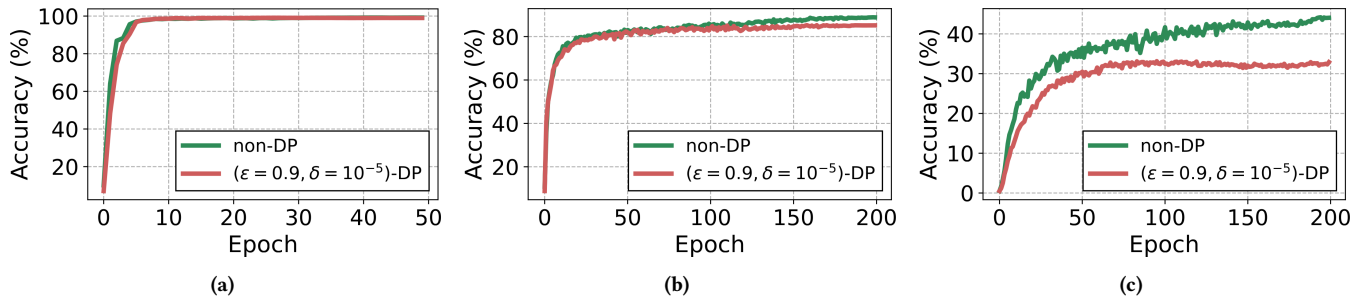


Figure 8: Accuracy of AnoFel per epoch compared with a non-DP baseline for non-IID data for (a) MNIST on LeNet5, (b) CIFAR10 on ResNet20, and (c) TinyImageNet on SqueezeNet.

dropouts, since a user’s update is independent of others’, but not addition as all users must be known during the setup phase to get shares of the decryption key. The proposed scheme relies on clients to decrypt the aggregated updates—which introduces excessive delays, and requires the server to know all clients and communicate with them directly—thus violating anonymity. Xu et al. [78] avoid the distributed decryption process, and allows for user additions (up to a maximum cap per iteration). However, this comes at the expense of introducing a trusted party to run the system setup and help in decrypting the aggregated model after knowing who participated in each iteration. Ryffel et al. [69] employ function secret sharing and assume a fixed client participation, with a (semi-honest) server that knows all clients and communicates with them directly. Thus, client anonymity is not supported. Mo et al. [59] use a trusted execution environment to achieve privacy. Beside not supporting anonymity, trusting a hardware is problematic due to the possibility of side-channel attacks. (An extensive survey on secure aggregation in federated learning can be found in [56].)

AnoFel addresses the limitations of prior work: it supports client anonymity, and does not involve them in the aggregation process—thus reducing overhead, and it supports dynamic participation without needing a recovery protocol or any trusted party. This is in addition to addressing recent attacks resulting from disseminating different initial models to clients.

Another line of prior work handles malicious clients during training using zero-knowledge proofs, based on norm checks [9, 54] and rank-based statistics [34]. We view these frameworks as complementary to our work in the sense that AnoFel can be similarly extended to support malicious clients during training.

Anonymity and Federated Learning. Several techniques were proposed to anonymize the dataset before being used in training, such as k -anonymity [24, 72], l -diversity [55], and t -closeness [52]. These techniques are considered complementary to AnoFel: they allow anonymizing a dataset, and our system guarantees client’s anonymity in the sense that no one will know if this client participated or which updates they have submitted.

The works [23, 28, 40, 53, 83] target the same anonymity notion as in AnoFel. Domingo et al. [28] utilize probabilistic multi-hop routes for model update submission, with the clients known by fixed pseudonyms instead of their real identities. However, such fixed pseudonyms provide only pseudoanonymity; several studies showed how network and traffic analysis can link these

pseudonyms back to their real identities [13, 48, 68]. Also, their anonymity guarantees is based on assuming that clients do not collude with the model owner, a strong assumption that AnoFel avoids. Li et al. [53] use interactive ZKPs to achieve client anonymity when submitting model updates. Their approach suffers from several security and technical issues: First, any party can generate a secret key and pass the proof challenge, not necessarily the intended client. Second, the proposed protocol requires some parameters to be made public, but no details on how to do this in an anonymous way. Third, no discussion on how to preserve message integrity, making the protocol vulnerable to man-in-the-middle attacks.

The scheme in [40] works at the physical layer; it randomly samples a subset of clients’ updates and aggregates their signals before submitting them to the server. This protocol assumes clients are trusted, and does not discuss how to preserve integrity of the communicated updates. Zhao et al. [83] introduce a trust assumption to achieve anonymity; a trusted proxy server mediates communication between clients and model owner. While Zhou et al. [84], in addition to assuming a trusted key generation center, require clients to interact with each other to establish a group key used for authenticating the submitted updates and supports only static settings. Lastly, Chen et al. [23] use a modified version of Tor to preserve anonymity; users authenticate each other and then negotiate symmetric keys to use for encryption. However, the negotiation and authentication processes are interactive, and the model owner records all clients’ (chosen) identities, thus anonymity is not guaranteed under these fixed identities. As a result, none of these systems supports anonymity in a provably secure way as AnoFel does.

7 Conclusion

In this paper, we presented AnoFel, the first provably secure framework for private and anonymous user participation in federated learning. AnoFel utilizes a public bulletin board, various cryptographic primitives, and differential privacy to support secure aggregation of model updates and client anonymity in both static and dynamic settings. We introduced the first formal security notion for private federated learning covering client anonymity, and we proved the privacy/anonymity guarantees of AnoFel based on this notion. We also demonstrated the efficiency and viability of AnoFel through a concrete implementation and extensive benchmarks covering large scale scenarios and comparisons to prior work.

Acknowledgments

The work of G.A. is supported by UConn's OVPR Research Excellence Program Award and in part by NSF Grant No. CNS-2226932.

References

- [1] 2022. 4-bit Window Pedersen Hash On The Baby Jubjub Elliptic Curve. https://iden3-docs.readthedocs.io/en/latest/_downloads/4b929e0f96aef77b75bb5cfc0f832151/Pedersen-Hash.pdf
- [2] 2022. EdDSA For Baby Jubjub Elliptic Curve with MiMC-7 Hash. https://iden3-docs.readthedocs.io/en/latest/_downloads/a04267077fb3fdbf2b608e014706e004/Ed-DSA.pdf
- [3] 2022. Google protocol implementation. <https://github.com/corentingiraud/federated-learning-secure-aggregation>
- [4] 2022. libsnark Library. <https://github.com/scipr-lab/libsnark>
- [5] 2022. Paillier scheme implementation. <https://github.com/meandmymind/hybrid-approach-to-ppfl>
- [6] 2022. Truex et al. protocol implementation. <https://github.com/lainisourgod/hybrid-approach-to-ppfl>
- [7] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 308–318.
- [8] Manan Ahuja, Shailee Siddhpuria, Christina Reppas-Rindlisbacher, Eric Wong, Jessica Gormley, Justin Lee, and Christopher Patterson. 2022. Sleep monitoring challenges in patients with neurocognitive disorders: A cross-sectional analysis of missing data from activity trackers. *Health Science Reports* (2022).
- [9] James Bell, Adrià Gascón, Tancrede Lepoint, Baiyu Li, Sarah Meiklejohn, Mariana Raykova, and Cathie Yun. 2023. {ACORN}: Input Validation for Secure Aggregation. In *32nd USENIX Security Symposium (USENIX Security 23)*. 4805–4822.
- [10] James Henry Bell, Kallista A Bonawitz, Adrià Gascón, Tancrede Lepoint, and Mariana Raykova. 2020. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 1253–1269.
- [11] Marta Bellés-Muñoz, Barry Whitehat, Jordi Baylina, Vanesa Daza, and Jose Luis Muñoz-Tapia. 2021. Twisted Edwards Elliptic Curves for Zero-Knowledge Circuits. *Mathematics* 9, 23 (2021), 3022.
- [12] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2012. High-speed high-security signatures. *Journal of cryptographic engineering* 2, 2 (2012), 77–89.
- [13] Alex Biryukov and Sergei Tikhomirov. 2019. Deanonymization and linkability of cryptocurrency transactions based on network analysis. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 172–184.
- [14] Ari Biswas and Graham Cormode. 2022. Verifiable Differential Privacy For When The Curious Become Dishonest. *arXiv preprint arXiv:2208.09011* (2022).
- [15] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Omer Paneth, and Rafail Ostrovsky. 2013. Succinct non-interactive arguments via linear interactive proofs. In *Theory of Cryptography Conference*. Springer, 315–333.
- [16] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1175–1191.
- [17] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)* 6, 3 (2014), 1–36.
- [18] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. 2020. Zether: Towards privacy in a smart contract world. In *International Conference on Financial Cryptography and Data Security*. Springer, 423–443.
- [19] Billy A Caceres, Yashika Sharma, Rohith Ravindranath, Ipek Ensari, Nicole Rosendale, Danny Doan, and Carl G Streed. 2023. Differences in Ideal Cardiovascular Health Between Sexual Minority and Heterosexual Adults. *JAMA cardiology* (2023), 335–346.
- [20] Miguel Castro, Barbara Liskov, et al. 1999. Practical byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.
- [21] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. 2015. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE international conference on computer vision*. 2722–2730.
- [22] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. 2016. Revisiting distributed synchronous SGD. *arXiv preprint arXiv:1604.00981* (2016).
- [23] Yijin Chen, Ye Su, Mingyue Zhang, Haoye Chai, Yunkai Wei, and Shui Yu. 2022. FedTor: An Anonymous Framework of Federated Learning in Internet of Things. *IEEE Internet of Things Journal* (2022).
- [24] Olivia Choudhury, Aris Gkoulalas-Divanis, Theodoros Salonidis, Issa Sylla, Yoonyoung Park, Grace Hsu, and Amar Das. 2020. Anonymizing data for privacy-preserving federated learning. *arXiv preprint arXiv:2002.09096* (2020).
- [25] Amrita Roy Chowdhury, Chuan Guo, Somesh Jha, and Laurens van der Maaten. 2021. EIFFeL: Ensuring Integrity for Federated Learning. *arXiv preprint arXiv:2112.12727* (2021).
- [26] Yuval Dagan and Gil Kur. 2022. A bounded-noise mechanism for differential privacy. In *Conference on Learning Theory*. PMLR, 625–661.
- [27] Ivan Damgård and Mads Jurik. 2001. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *International workshop on public key cryptography*. Springer, 119–136.
- [28] Josep Domingo-Ferrer, Alberto Blanco-Justicia, Jesús Manjón, and David Sánchez. 2021. Secure and Privacy-Preserving Federated Learning via Co-Utility. *IEEE Internet of Things Journal* (2021).
- [29] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. 2006. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology-EUROCRYPT 2006: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28-June 1, 2006. Proceedings 25*. Springer, 486–503.
- [30] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.
- [31] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. 2019. Quisquis: A new design for anonymous cryptocurrencies. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 649–678.
- [32] Jfrl Federated Rank Learning [n. d.]. <https://github.com/SPIN-UMass/FRL>
- [33] Jorge Galindo and Pablo Tamayo. 2000. Credit risk assessment using statistical and machine learning: basic methodology and risk modeling applications. *Computational Economics* 15, 1 (2000), 107–143.
- [34] Zahra Ghodsi, Mojan Javaheripi, Nojan Sheybani, Xinqiao Zhang, Ke Huang, and Farinaz Koushanfar. 2023. zPROBE: Zero Peek Robustness Checks for Federated Learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 4860–4870.
- [35] Maryellen L Giger. 2018. Machine learning in medical imaging. *Journal of the American College of Radiology* 15, 3 (2018), 512–520.
- [36] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*. 51–68.
- [37] Oded Goldreich. 2007. *Foundations of cryptography: volume 1, basic tools*. Cambridge university press.
- [38] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 305–326.
- [39] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems* 28 (2015).
- [40] Burak Hasircioglu and Deniz Gündüz. 2021. Private wireless federated learning with anonymous over-the-air computation. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 5195–5199.
- [41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
- [42] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. 2020. Zcash protocol specification. *GitHub: San Francisco, CA, USA* (2020).
- [43] Thomas Humphries, Simon Oya, Lindsey Tulloch, Matthew Rafuse, Ian Goldberg, Urs Hengartner, and Florian Kerschbaum. 2020. Investigating membership inference attacks under data dependencies. *arXiv preprint arXiv:2010.12112* (2020), 2.
- [44] Swanand Kadhe, Nived Rajaraman, O Ozan Koyluoglu, and Kannan Ramchandran. 2020. Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning. *arXiv preprint arXiv:2009.11248* (2020).
- [45] Peter Kairouz, Ziyu Liu, and Thomas Steinke. 2021. The distributed discrete gaussian mechanism for federated learning with secure aggregation. In *International Conference on Machine Learning*. PMLR, 5201–5212.
- [46] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing bitcoin security and performance with strong consistency via collective signing. In *Usenix Security*.
- [47] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE symposium on security and privacy (SP)*. IEEE, 583–598.
- [48] Philip Koshy, Diana Koshy, and Patrick McDaniel. 2014. An analysis of anonymity in bitcoin using p2p network traffic. In *International Conference on Financial Cryptography and Data Security*. Springer, 469–485.
- [49] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2010. Cifar-10 (canadian institute for advanced research). URL <http://www.cs.toronto.edu/kriz/cifar.html> 5 (2010).

- [50] Yann LeCun. 1998. The MNIST database of handwritten digits. <http://yann.lecun.com/exdb/mnist/> (1998).
- [51] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
- [52] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. 2007. t-closeness: Privacy beyond k-anonymity and l-diversity. In *2007 IEEE 23rd international conference on data engineering*. IEEE, 106–115.
- [53] Yijing Li, Xiaofeng Tao, Xuefei Zhang, Junjie Liu, and Jin Xu. 2022. Privacy-preserved federated learning for autonomous driving. *IEEE Transactions on Intelligent Transportation Systems* (2022).
- [54] Hidde Lycklama, Lukas Burkhalter, Alexander Viand, Nicolas Küchler, and Anwar Hithnawi. 2023. RoFL: Robustness of Secure Federated Learning. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 453–476.
- [55] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkatasubramanian. 2007. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1, 1 (2007), 3–es.
- [56] Mohamad Mansouri, Melek Önen, Wafa Ben Jaballah, and Mauro Conti. 2023. Sok: Secure aggregation based on cryptographic schemes for federated learning. *Proceedings on Privacy Enhancing Technologies* (2023).
- [57] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 1273–1282.
- [58] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 691–706.
- [59] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2021. PPFL: privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. 94–108.
- [60] Hamid Mozaffari, Virat Shejwalkar, and Amir Houmansadr. 2023. Every Vote Counts: Ranking-Based Training of Federated Learning to Resist Poisoning Attacks. In *32nd USENIX Security Symposium (USENIX Security 23)*.
- [61] Pratyay Mukherjee and Daniel Wichs. 2016. Two round multiparty computation via multi-key FHE. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 735–763.
- [62] Arjun Narayan, Ariel Feldman, Antonis Papadimitriou, and Andreas Haeberlen. 2015. Verifiable differential privacy. In *Proceedings of the Tenth European Conference on Computer Systems*. 1–14.
- [63] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2019. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE symposium on security and privacy (SP)*. IEEE, 739–753.
- [64] Milad Nasr, Shuang Songi, Abhradeep Thakurta, Nicolas Papernot, and Nicholas Carlini. 2021. Adversary instantiation: Lower bounds for differentially private machine learning. In *2021 IEEE Symposium on security and privacy (SP)*. IEEE, 866–882.
- [65] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*. Springer, 223–238.
- [66] Dario Pasquini, Danilo Francati, and Giuseppe Ateniese. 2021. Eluding Secure Aggregation in Federated Learning via Model Inconsistency. *arXiv preprint arXiv:2111.07380* (2021).
- [67] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019), 8026–8037.
- [68] Fergal Reid and Martin Harrigan. 2013. An analysis of anonymity in the bitcoin system. In *Security and privacy in social networks*. Springer, 197–223.
- [69] Théo Ryffel, Pierre Tholoni, David Pointcheval, and Francis Bach. 2020. Ariann: Low-interaction privacy-preserving deep learning via function secret sharing. *arXiv preprint arXiv:2006.04593* (2020).
- [70] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 459–474.
- [71] Jinhyun So, Başak Güler, and A Salman Avestimehr. 2021. Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning. *IEEE Journal on Selected Areas in Information Theory* 2, 1 (2021), 479–489.
- [72] Latanya Sweeney. 2002. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10, 05 (2002), 557–570.
- [73] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. 2019. A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*. 1–11.
- [74] Emily Vogels. 2020. About one-in-five Americans use a smart watch or fitness tracker. <https://www.pewresearch.org/short-reads/2020/01/09/about-one-in-five-americans-use-a-smart-watch-or-fitness-tracker/>.
- [75] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. 2020. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security* 15 (2020), 3454–3469.
- [76] Nicole Wetsman. 2019. Data from health apps offers opportunities and obstacles to researchers. <https://www.theverge.com/2019/7/3/20681254/data-health-apps-clue-period-tracking-sleep-fitness-research>.
- [77] Yuxin Wu and Kaiming He. 2018. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*. 3–19.
- [78] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar, and Heiko Ludwig. 2019. Hybridalpha: An efficient approach for privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*. 13–23.
- [79] Chien-Sheng Yang, Jinhyun So, Chaoyang He, Songze Li, Qian Yu, and Salman Avestimehr. 2021. LightSecAgg: Rethinking Secure Aggregation in Federated Learning. *arXiv preprint arXiv:2109.14236* (2021).
- [80] Leon Yao and John Miller. 2015. Tiny imagenet classification with convolutional neural networks. *CS 231N* 2, 5 (2015), 8.
- [81] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. 2018. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE, 268–282.
- [82] Da Yu, Huishuai Zhang, Wei Chen, and Tie-Yan Liu. 2021. Do not let privacy overbill utility: Gradient embedding perturbation for private learning. *arXiv preprint arXiv:2102.12677* (2021).
- [83] Bin Zhao, Kai Fan, Kan Yang, Zilong Wang, Hui Li, and Yintang Yang. 2021. Anonymous and privacy-preserving federated learning with industrial big data. *IEEE Transactions on Industrial Informatics* 17, 9 (2021), 6314–6323.
- [84] Tianqi Zhou, Jian Shen, Pandi Vijayakumar, Md Zakirul Alam Bhuiyan, and Audithan Sivaraman. 2023. Anonymous Authentication Scheme for Federated Learning. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 1–6.

A Extensions

Addressing a stronger adversary model—training side. AnoFel assumes semi-honest clients in the training phase. Therefore, mitigating threats of using a legitimate (registered and certified) dataset in a training activity of totally different type—e.g., use medical data to train a model concerned with vehicles, are out of scope. We can make our adversary model stronger by considering a semi-malicious client who may attempt this attack. This can be done by requiring the certifier to add a dataset type dt to the dataset certificate, and having the client prove that a dataset with the correct type has been used in training. Each training activity will have a designated type dt , and a certifier will check that a dataset D_i is indeed of type dt (so it can be used to train any model of type dt) before signing $H(D_i) \parallel \text{mpk}_i \parallel dt$. Also, the ZKP circuit a client uses in training must check that \mathcal{AG} (that will receive a ciphertext of the model updates) is managing a training activity with an identical dt . Otherwise, a valid proof cannot be generated.

Addressing malicious clients during training, i.e., these who may deviate arbitrarily from the protocol, can be done using ZKPs (as discussed in Section 6). In a generic way, this involves requiring a submitted model update to be accompanied with a ZKP proof on well-formedness, meaning that the registered dataset and the actual initial model parameters were used and training was done correctly. Extending AnoFel to support that while preserving its efficiency level is part of our future work.

Addressing a stronger adversary model—aggregation side. An extra layer of precaution can be added to our system design to strengthen client participation anonymity. In particular, we want to protect against an adversary who may know that certain encrypted updates have been submitted by a certain client (without knowing

any information about the dataset itself). Since the updates are encrypted under \mathcal{AG} 's public key, the adversary might be able to determine the dataset type based on the training activity type that \mathcal{AG} is managing.¹⁶ This will invade privacy, e.g., reveal that a client suffers from a particular disease merely based on the dataset or training task type. Thus, we need a technique to hide which training activity model updates are submitted to.

We can address this case by creating an anonymity set for the aggregators. That is, the system will have several ongoing training activities, each of which with its own \mathcal{AG} committee. When submitting a model update, the client will choose a set of aggregator committees $AG = \{\mathcal{AG}_1, \dots, \mathcal{AG}_u\}$ including the target \mathcal{AG} who is managing the training activity the client is interested in.¹⁷ The client then shuffles AG to avoid any ordering attacks (e.g., if the target \mathcal{AG} is always placed first, this reveals the target training activity). After that, it encrypts the updates under the public keys of this set—encrypt the actual updates for the target \mathcal{AG} while encrypt 0 for the rest. This will produce c ; a vector of u ciphertexts. Consequently, even if it is revealed that a client has submitted model updates, this will not expose the target training activity.

Another case is related to the aggregators themselves; now we consider semi-honest aggregators, thus they will correctly aggregate the updates, generate a noise value that respects the desired DP parameters, and produce valid partial decryptions of the ciphertext of the aggregated updates. Malicious aggregators may deviate arbitrarily. For example, those who are colluding with the server may produce zero noise value, or disclose the generated noise to the server, which will impact the DP privacy guarantees. Also, malicious aggregators may target the server itself by not aggregating the updates correctly, or even produce invalid partial decryptions. This leaves the server the with dilemma of which t partial decryptions should be used to obtain valid aggregated updates.

Addressing malicious aggregators can be done in a generic way using ZKPs. Encrypted noise values will have a ZKP attesting to their correctness, and same for partial decryptions. We leave this direction as part of our future work, which also includes investigating the use of verifiable DP solutions [14, 62] in the context of anonymous and private federated learning.

Instantiating the bulletin board. The concrete instantiation of the board impacts runtime. The board mediates communication between parties and must validate all information postings before accepting them. To speed up this process, the board can be formed as a sequence of blocks of information maintained by a committee of validators to distribute trust (similar to a permissioned blockchain) or using a permissionless blockchain with an honest majority assumption on the miners/validators. For fast processing, confirmation, and block finality, we recommend the use of a consensus protocol utilizing variants of practical Byzantine fault tolerant (PBFT) [20] as in, e.g., [36, 46, 47]. That is, for each epoch, a committee will manage the board; it proposes the next block of information to be added to the board (based on the messages received from clients, the server, and the aggregators). The

committee members would agree on the block by signing it. Once a majority of votes is collected, the block is considered final and added to the board. The validity/correctness rules of the board are derived from the federated learning protocol itself; so in AnoFel, for example, there will be format checking, ZKP verification, and signature verification. Thus, many building blocks, functionalities, and optimizations from existing blockchain paradigms can be utilized.

Reducing storage costs of the bulletin board. ML models may involve thousands of parameters. The server needs to post the initial values of these parameters on the board for each training iteration. Also, a client posts the (encrypted) updated version of all these parameters on the board. This is a large storage cost that may create a scalability problem. To address this issue, we can employ the scalability solutions currently used by the blockchain community; for example, store the (ciphertext of) model parameters on a decentralized storage network, and post only the hash of them on the board (with a pointer to where the actual data is stored). Furthermore, once the data is used, i.e., a training iteration concluded, initial model, noise values posted by aggregators, and all client updates can be discarded, which reduces the storage cost significantly. Note that the initial model parameters and encrypted noise values are posted by the server and aggregators, respectively, who are not anonymous. While the encrypted updates are posted by clients, and thus, an anonymous off-chain storage must be used (like an anonymous sidechain) to avoid compromising their anonymity.

To get a sense of the communication cost added due to the use of a bulletin board instantiated as a blockchain, we report overhead numbers from state-of-the-art literature on PBFT-based blockchain architecture. We also assume an optimized storage solution, e.g., the one above where a hash (of size 32 bytes and any additional bytes for metadata based on the board protocol) of a submitted update is posted on the board that points to the actual update stored on a storage network or a sidechain. Verifying updates before posting them on the board in AnoFel is dominated by verifying the ZKP, which as shown in Table 2, takes around 4.7 ms per proof. Running the PBFT agreement, as reported in Figure 10 in [47], for a consensus group of size 256 and data size of 1 MB, ByzCoinX takes up to 15 sec to conclude block consensus. Thus, from the client view, it will take around 15 sec for their submitted updates to appear on the board (in addition to communication link delays that are on the order of 100 ms to send these updates to the board validators). Take the numbers of client runtime for TinyimageNet (the heaviest dataset) from Figure 6, a client would need around 8 sec to finish training and preparing the encrypted updates along with needed ZKP. While, based on Figure 5, on the aggregator side, it needs around 30 sec to finish aggregation and decryption of the aggregated updates (in addition to the time needed to retrieve the encrypted updates from the storage network) also for TinyimageNet. Also, it would need another 15 sec to conclude consensus to publish the new updated model by the model owner on the board before a new iteration can start. Thus, while accounting for the consensus and communication delays, a training iteration would need around 70 sec.

Indeed, consensus time can be further optimized when using a permissioned setting which allows for a smaller number of permissioned validators to run consensus. As shown in Figure 10 in [47], when having 8 validators, the consensus latency drops to around 2

¹⁶Information about the training task could be public knowledge, and even if it is secret, the adversary can collude with any member of \mathcal{AG} and obtain this information.

¹⁷A client will have a fixed AG . Changing AG between iterations must be done carefully; for a new AG' , if $AG \cap AG' = \mathcal{AG}$, it would be trivial to tell which training activity a client is part of.

sec (not to mention removing any additional cost needed to select validator committee at random for each new epoch). In a setting like FL, we favor a permissioned board setup to optimize performance.

B Proof of Theorem 1

To prove Theorem 1, we have to prove that AnoFel does not violate the error bound on training correctness, and that no PPT adversary can win the security games defined in Section 2 for anonymity and dataset privacy with a probability larger than the defined success bounds of these games.

Intuitively, AnoFel satisfies these properties by relying on the correctness and security of the underlying cryptographic primitives, and the bounds on accuracy and privacy loss offered by DP. The use of a secure ZKP system guarantees: completeness (a valid proof generated by a client will be accepted by the bulletin board validators and the aggregators \mathcal{AG}), soundness (a client that does not own a certified dataset cannot register, and a client that does not belong to the registered set cannot generate valid proofs during training), and zero knowledge (so the proof does not reveal anything about the master public key of the client or her dataset, and cannot be linked back to the client registration information).

Furthermore, the use of a semantically secure threshold homomorphic encryption scheme guarantees that the ciphertexts of the model updates do not reveal anything about the underlying (plaintext) updates, and homomorphic add will produce a valid result of the sum of these updates. The use of a secure commitment scheme guarantees that a commitment posted by a client cl_i hides the dataset D_i and binds this client to D_i . The security of the digital signature scheme and the PKI guarantee that a malicious adversary cannot forge a certificate for a corrupted dataset, and that a man-in-the-middle attacker cannot manipulate any of the messages that a client, aggregator, or a server send. Also, under the assumption that at least t members of \mathcal{AG} are honest, it is guaranteed that S will not have access to the individual updates submitted by clients.

Moreover, the use of a (ϵ, δ) -differential privacy leads to an error (or loss in accuracy) bound α as detailed in Section 3, as well as bounds to the adversary success in distinguishing a model trained with dataset D from a model trained with a dataset $D' \neq D$. The parameter ϵ control this indistinguishability level, and the parameter δ makes it easier to satisfy DP by allowing a small failure in the privacy guarantees. Thus, the bound γ will depend on the values of ϵ and δ , thus covering the success probability in violating privacy regardless of the strategy that an attacker uses.¹⁸

Formally, the proof of Theorem 1 requires proving three lemmas showing that AnoFel is correct, anonymous, and supports dataset privacy. For correctness, note that AnoFel does not impact training correctness and accuracy. So if a non-cryptographic defense mechanism is employed, and this mechanism provides a trade-off

with respect to accuracy (as in DP), AnoFel will not impact that trade-off.

LEMMA 1. *AnoFel satisfies the correctness property as defined in Definition 1.*

PROOF. Correctness follows by the correctness of the homomorphic encryption scheme and the security of the digital signature scheme, as well as the accuracy level provided by DP. A semi-honest client in the training phase will perform training as required and encrypt the updates and post them on the board. Since AnoFel uses an existential unforgeable digital signature scheme, a malicious attacker \mathcal{A} cannot modify the ciphertext of the updates without invalidating the signature, and \mathcal{A} cannot forge a valid signature over a modified ciphertext. Thus, it is guaranteed that all accepted model update ciphertexts are the ones produced by the client.

Also, since AnoFel uses a correct (and secure) homomorphic encryption scheme, the homomorphic addition of the ciphertexts will produce a ciphertext of the sum of the actual updates (along with the noise level required by DP). By the correctness of the decryption algorithm, after decrypting the sum ciphertext, the server will obtain the correct value of the aggregated updates in each training iteration. This trained model differs from the actual one by the error bounds α obtained from DP, thus satisfying α -correctness. This completes the proof. \square

For anonymity, as an extra step, a technique can be used to preprocess the dataset to remove sensitive attributes from the dataset. Thus, any attack against anonymity that assumes that the \mathcal{A} got a hold on datapoint, without knowing the identity of owner client, will be ineffective (note if the attacker gets a hold on a client dataset and he knows the client identity, then he already compromised privacy of that client). The definition of our anonymity property does not assume the adversary knows the dataset or identity of the clients involved in the challenge.¹⁹

LEMMA 2. *AnoFel satisfies the anonymity property as defined in Definition 1.*

PROOF. Under the assumption that at least one honest client (other than the honest cl in the game) has submitted updates during the challenge training iteration (as described in the game definition earlier), accessing the aggregated model updates at the end of any iteration will not provide \mathcal{A} with any non-negligible advantage in winning the game. Thus, the proof is reduced to showing that all actions introduced by AnoFel preserve anonymity. We prove that using a similar proof technique to the one in [31], where we show a series of hybrids starting with an AnonGame with $b = 0$ (Hybrid₀), and finishing with an AnonGame game with $b = 1$ (Hybrid₇). By showing that all these hybrids are indistinguishable, this proves that \mathcal{A} cannot tell which client was chosen for the challenge train command in AnonGame. Now, we proceed with a sequence of hybrid games as follows:

Hybrid₀: The game AnonGame with $b = 0$.

Hybrid₁: Same as Hybrid₀, but we replace the zero-knowledge proofs with simulated ones, i.e., we invoke the zero-knowledge

¹⁹Even if we allow that, we can define γ -anonymity property where the attacker wins with probability bounded by γ inherited from DP.

¹⁸We note that deriving an exact formula for γ in terms of ϵ and δ is outside the scope of this work. Several works derived bounds for DP-based privacy guarantees as discussed in Section 3, and so γ will be the union bound of the success probabilities of all possible strategies. An alternative approach is to consider the success probability in the DINDGame in terms of a multiplicative term, e^ϵ , and an additive term δ over 0.5 (i.e., winning the game by just making a random guess), in a similar way to how DP guarantee is defined in Definition 2. However, we keep the definition general, by considering the additive term γ , so it can be used with any PAFL scheme including these that may use different non-cryptographic privacy techniques other than DP.

property simulator for each of the register and train queries, and we replace the actual proofs in the output of these queries with simulated ones. The hybrids Hybrid_0 and Hybrid_1 are indistinguishable by the zero-knowledge property of the ZKP system that AnoFel uses. That is, if \mathcal{A} can distinguish them, then we can build another adversary \mathcal{A}' that can break the zero-knowledge property of the ZKP system, which is a contradiction.

Hybrid₂: Same as Hybrid_1 , but we replace $(cl_0, aux_0, cl_1, aux_1)$ with fresh output $(cl'_0, aux'_0, cl'_1, aux'_1)$. That is, we choose fresh datasets and register two fresh clients using them. So if setup created a state with n clients, any register query for any of the n clients other than cl_0 and cl_1 will proceed as in Hybrid_1 . However, if it is for cl_0 or cl_1 , we replace them with cl'_0 or cl'_1 and proceed.

Hybrid_1 and Hybrid_2 are indistinguishable by the zero-knowledge property of the ZKP system and the hiding property of the commitment scheme that AnoFel uses (which implies that client registration is indistinguishable). That is, if \mathcal{A} can distinguish them, then we can build two adversaries: \mathcal{A}' that can break the zero-knowledge property of the ZKP system, and \mathcal{A}'' that can break the hiding property of the commitment scheme, which is a contradiction.

Hybrid₃: Same as Hybrid_2 , but we replace the output of training any of $(cl_0, aux_0, cl_1, aux_1)$ with fresh output produced by training $(cl'_0, aux'_0, cl'_1, aux'_1)$. As above, if setup created a state with n client registrations, any train query for any of the n clients other than cl_0 and cl_1 will proceed as in Hybrid_2 . However, if the train query is for cl_0 or cl_1 , we replace them with training output based on the fresh datasets owned by cl'_0 or cl'_1 and proceed.

Hybrid_2 and Hybrid_3 are indistinguishable by the zero knowledge property of the ZKP system and the semantic security of the homomorphic encryption scheme used in AnoFel (which implies that training is indistinguishable). If \mathcal{A} can distinguish them, then we can build two adversaries: \mathcal{A}' that can break the zero-knowledge property of the ZKP system, and \mathcal{A}'' that can break the semantic security of the encryption scheme, which is a contradiction.

Hybrid₄: Same as Hybrid_3 , but with $b = 1$. The hybrids Hybrid_3 and Hybrid_4 are indistinguishable by the indistinguishability of model training as described above.

Hybrid₅: Same as Hybrid_4 , but with $(cl_0, aux_0, cl_1, aux_1)$ used in training as in the original game. So this is Hybrid_3 with $b = 1$. The hybrids Hybrid_4 and Hybrid_5 are indistinguishable by the indistinguishability argument of hybrids Hybrid_3 and Hybrid_2 .

Hybrid₆: Same as Hybrid_5 , but with $(cl_0, aux_0, cl_1, aux_1)$ used in registration as in the original game. So this is Hybrid_2 with $b = 1$. The hybrids Hybrid_5 and Hybrid_6 are indistinguishable by the indistinguishability argument of hybrids Hybrid_2 and Hybrid_1 .

Hybrid₇: Same as Hybrid_6 , but with real ZKPs instead of the simulated ones. So this is the original AnonGame with $b = 1$. The hybrids Hybrid_6 and Hybrid_7 are indistinguishable by the indistinguishability argument of hybrids Hybrid_1 and Hybrid_0 .

This shows that AnonGame with $b = 0$ is indistinguishable from AnonGame with $b = 1$, which completes the proof. \square

As for dataset privacy, the use of DP will allow \mathcal{A} to win the dataset privacy game with advantage γ since he selects the datasets

involved in the challenge (e.g., \mathcal{A} can perform a membership attack against the trained model using datapoints from the datasets he chose). Thus, our proofs proceed in two stages: first, we show that the cryptographic primitives used in AnoFel do not provide \mathcal{A} with any non-negligible advantage, and second, by the security guarantees of DP, this attacker has an advantage bounded by γ based on DP privacy guarantees with respect to the output (i.e., the trained model).

LEMMA 3. *AnoFel satisfies the dataset privacy property as defined in Definition 1.*

PROOF. In DINDGame , adversary \mathcal{A} chooses two datasets D_0 and D_1 . The challenger picks one of them at random, registers a client with the chosen dataset, and invokes the train command for that client. \mathcal{A} gets to see the output of the registration and training commands, which are the messages and signatures that a client sends in the setup and training phases of AnoFel.

In order to win the DINDGame , \mathcal{A} can attack the registration or the training processes. That is, for the former \mathcal{A} tries to reveal which dataset is hidden in the posted commitment or obtain information about the witness underlying the submitted proof, which contains the dataset D_b in this case. While for the latter, \mathcal{A} may try to infer any information about the plaintext of the model updates ciphertext. Note that attacking the ZKP to reveal any information about the commitment that was used, and then attacking that commitment, reduces to the same case of attacking the registration process.

Attacking registration means attempting to break the hiding property of the commitment and the zero-knowledge property of the ZKP. Since AnoFel uses a secure commitment scheme, the former will succeed with negligible probability $\text{negl}_1(\lambda)$. Also, since AnoFel uses a secure ZKP system, the latter will succeed with negligible probability $\text{negl}_2(\lambda)$. Attacking the training process means attempting to break the semantic security of the encryption scheme. Since AnoFel uses a semantically secure encryption scheme, such an attack will succeed with negligible probability $\text{negl}_3(\lambda)$.

Accordingly, \mathcal{A} 's advantage by the cryptographic primitives that we use is $\text{negl}_1(\lambda) + \text{negl}_2(\lambda) + \text{negl}_3(\lambda) = \text{negl}(\lambda)$.

Now, \mathcal{A} can query the oracle $\mathcal{O}_{\text{PAFL}}$ to access the updated model and perform any training output-related attacks, e.g., membership or inference attacks. That is, \mathcal{A} knows both datasets and query the model over various datapoints, or perform any other strategy, to distinguish which dataset was used in training. The success of this strategy is bounded by the privacy loss γ obtained by DP.

Thus, the probability that \mathcal{A} wins in the DINDGame is $\frac{1}{2} + \text{negl}(\lambda) + \gamma$, which completes the proof. \square

Proof of Theorem 1. Follows by Lemmas 1, 2, and 3.

C Accuracy Evaluation

We evaluate the model test accuracy in AnoFel incorporating DP and compare to a non-DP baseline in Figure 9. We present evaluations for number of clients $N = \{16, 64, 256\}$, IID and non-IID data for MNIST, CIFAR10, and TinyImageNet benchmarks. Across all data distributions and client sizes, the accuracy drop due to DP for the MNIST dataset is between 0.05%–0.22%, for CIFAR10 is between 0.79%–3.68%, and for TinyImageNet is between 8.11%–12.27%.

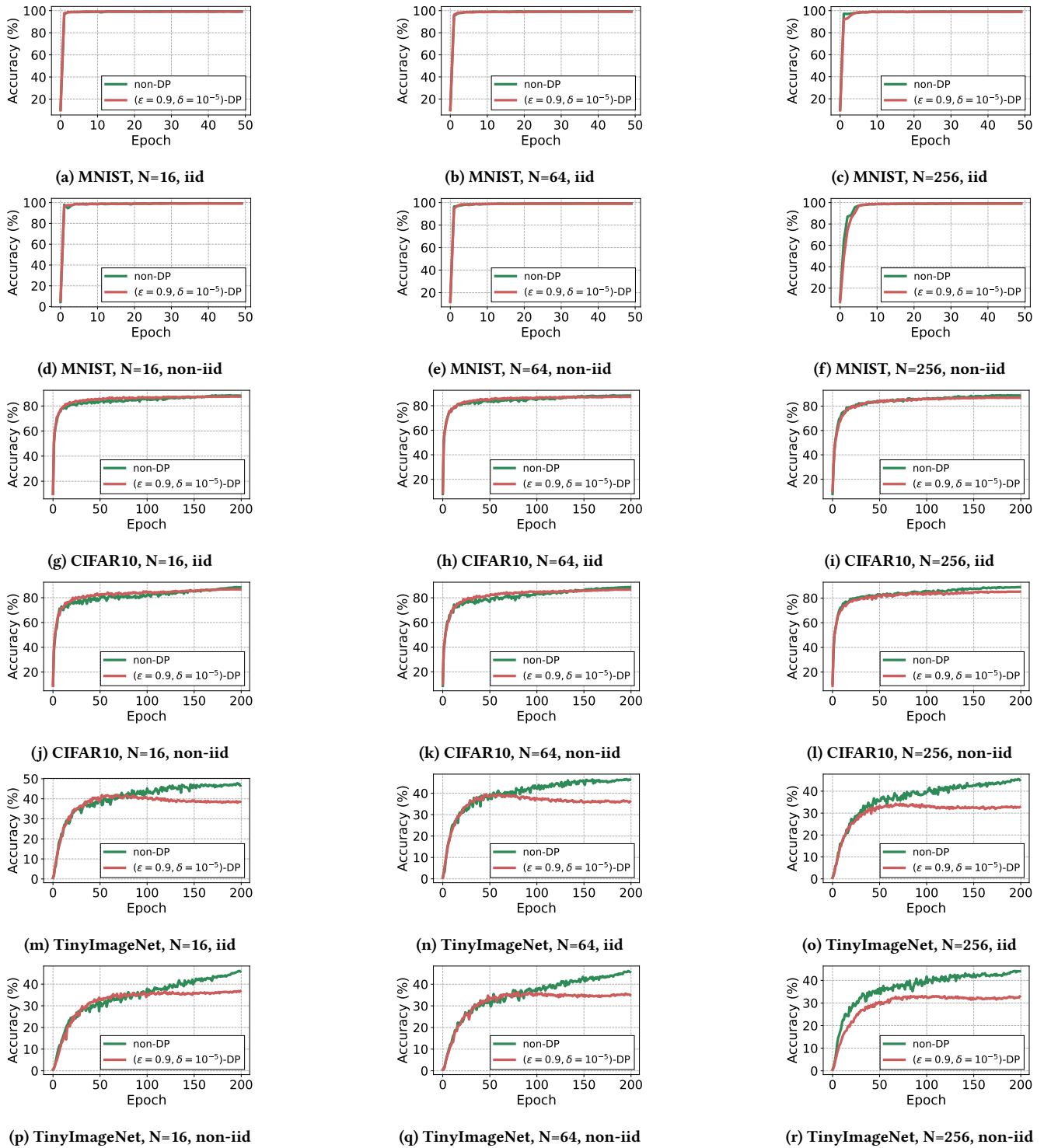


Figure 9: Accuracy of AnoFel per epoch compared with a non-DP baseline for MNIST, CIFAR10, and TinyImageNet.