Biscotti: A Blockchain System for Private and Secure Federated Learning

Muhammad Shayan Clement Fung Chris J.M. Yoon Ivan Beschastnikh
University of British Columbia

Abstract—Federated Learning is the current state of the art in supporting secure multi-party machine learning (ML): data is maintained on the owner's device and the updates to the model are aggregated through a secure protocol. However, this process assumes a trusted centralized infrastructure for coordination, and clients must trust that the central service does not use the byproducts of client data. In addition to this, a group of malicious clients could also harm the performance of the model by carrying out a poisoning attack. As a response, we propose Biscotti: a fully decentralized peer to peer (P2P) approach to multi-party ML, which uses blockchain and cryptographic primitives to coordinate a privacy-preserving ML process between peering clients. Our evaluation demonstrates that Biscotti is scalable, fault tolerant, and defends against known attacks. For example, Biscotti is able to both protect the privacy of an individual client's update and maintain the performance of the global model at scale when 30% adversaries are present in the system.

Index Terms—Distributed Machine learning, Blockchain, Privacy, Security

1 Introduction

A common requirement in machine learning (ML) applications is the collection of massive amounts of training data. This data is frequently distributed, such as among hospitals or devices in an IoT deployment. However, when training ML models in a multi-party setting, users must share their potentially sensitive information with a centralized service [1], [2], [3], [4]. Such sharing is problematic for users or companies who are not willing to trust a third party. For example, pharmaceutical companies compete with each other in drug discovery and rarely share data[1]. And, internet users are increasingly aware of the value of their data and would like to retain control over their data[2]. To avoid directly sharing sensitive data, federated learning [5], [6], [7] is a prominent solution for large-scale secure multi-party ML: clients train a shared model through a trusted aggregator without revealing their underlying data or computation [8], [9], [10]. But, doing so introduces a subtle threat: clients, who previously acted as passive data contributors, are now actively involved in the training process [11], [12], [13], presenting new privacy and security challenges to multi-party ML systems [14].

Prior work has demonstrated that adversaries can attack the shared model through poisoning attacks [11], [12], [15], [16], [17], [18], [19], in which an adversary contributes adversarial updates to shared model parameters. Adversaries can also attack the privacy of other clients in federated learning: in an information leakage attack, an adversary poses as an honest data provider and attempts to infer properties of the sensitive training data of a target client through observation of the target's shared model updates [13], [20].

Both poisoning and information leakage attacks have individually been defended in prior work through centralized anomaly detection [21], differential privacy [22], [23], [24], [25] or secure aggregation [26], [27], but to the best of our knowledge, a private and decentralized solution that solves both threats concurrently does not yet exist. Furthermore, these approaches are inapplicable in decentralized contexts that lack a trusted central authority. In this work, we focus on the decentralized P2P setting, which enables a stronger data ownership model for distributed ML.

Because ML does not require strong consensus or consistency to converge [28], [29], traditional strong consensus protocols such as Byzantine Fault Tolerant (BFT) protocols [30] are overly restrictive for machine learning workloads. Distributed ledgers (blockchains) [31] have emerged as a more appropriate system to facilitate private, verifiable, crowd-sourced computation. Through design elements such as publicly verifiable proof of work, eventual consistency, and ledger-based consensus, blockchains have been used in decentralized multi-party settings, such as currency management [31], [32], archival data storage [33], [34], financial auditing [35], privacy-preserving computation [36] and IoT edge computation [37], [38], [39].

As designed, federated learning relies on a trusted aggregator, and is unsuitable for peer-to-peer (P2P) ML settings. In this work, we propose *Biscotti*, a decentralized public P2P system that co-designs privacy-preserving multi-party ML with a blockchain ledger. In contrast to on-blockchain, layer-2 ML applications [40], we propose *proof-of-federation (PoF)*, a layer-1 blockchain consensus protocol that combines the state-of-the-art in defenses for federated learning and makes them applicable in decentralized P2P settings. Biscotti coordinates ML training between peers who are weighed by the value, or stake, that they provide to the system through PoF. Inspired by prior work [32], Biscotti uses consistent hashing based on PoF in combination with verifiable random functions (VRFs) [41] to select key roles for peers who will help coordinate the privacy and security of model updates. PoF prevents groups of colluding peers from overtaking the system without a sufficient stake ownership.

1. https://www.melloddy.eu/
2. https://www.oasislabs.com/

With Biscotti, our primary contribution is to adapt several prior techniques into one coherent system that provides secure and private multi-party machine learning in highly distributed P2P settings. In particular, Biscotti prevents peers from poisoning the model through the Multi-Krum defense [42], provides privacy through differentially private noise [22], [23], and uses Shamir secrets for secure aggregation [43].

We evaluated Biscotti on Azure and considered its performance, scalability, churn tolerance, and ability to withstand different attacks. We found that Biscotti can train an MNIST softmax model with 200 peers on a 60,000 image dataset in 266.7 minutes, while withstanding up to 30% of adversarial peers. In addition, we show that Biscotti's design is resilient to information leakage attacks [13] that require knowledge of a client's SGD update, and that Biscotti is resilient to poisoning attacks [44] from prior work. Biscotti is also fault tolerant, coping with nodes that churn every 1.875s and provides model training that converges even with node churn.

2 Challenges and Contributions

We now describe the key challenges in designing a P2P solution for multi-party ML and the key pieces in Biscotti's design that resolve each of these challenges.

Sybil attacks: consistent hashing using VRF's and PoF. In a P2P setting adversaries can collude or generate aliases to increase their influence in a sybil attack [45]. Biscotti uses a consistent hashing protocol based on the latest block hash and verifiable random functions (VRF's) (see Appendix D) to select a subset of peers that are responsible for the different stages of the PoF protocol: adding noise to updates, validating an update, and securely aggregating the update. To mitigate the effect of sybils, PoF selects peers proportional to their stake. A peer's stake is the reputation that the peer acquires by positively contributing to the shared model. This ensures that an adversary cannot increase their influence in the system by creating multiple peers without improving the model. We evaluate the security of our VRF mechanism and PoF in Appendix K.

Poisoning attacks: update validation using Multi-Krum. In multi-party ML, peers possess a disjoint and private subset of the training data. As mentioned above, adversaries can exploit the private P2P setting to stealthily execute poisoning attacks [12], [18], [46], [47], [48].

In multi-party settings, we assume that baseline validation models are not available to peers, so Biscotti validates an SGD update by evaluating it with respect to the updates submitted by other peers. Biscotti validates an SGD update using a Byzantine-tolerant aggregation scheme called *Multi-Krum* [42]. Multi-Krum is only one of several algorithm that Biscotti could use; others include median/trimmed mean and Bulyan [49], [50]. Although Multi-Krum and related aggregation schemes do not protect against all poisoning attacks [47], they are effective against some classes of attacks and Biscotti can generally support any aggregation scheme that operates only on model updates.

Multi-Krum rejects model updates that differ heavily from the direction of the majority of the updates. In our implementation of Multi-Krum, described in Section 4.5, in each round a committee of validation peers is selected by a majority vote and each member of the committee uses Multi-Krum to remove these anomalous updates. Multi-Krum guarantees convergence against f Byzantine adversaries in a system with n total clients (when $2f + 2 < n$). We demonstrate in Section 6.1 that by using Multi-Krum, Biscotti can handle poisonous updates from up to 30% malicious clients.

Information leakage attacks: VRF verifier peers and differentially-private updates using pre-committed noise. By observing a peer's model updates from each verification round, an adversary can perform an information leakage attack [13], [20], [51], [52], [53], [54] and recover details about a victim's training data. (See Appendix E for background.)

Biscotti prevents such attacks during update verification in two ways. First, the latest block hash is used to select the verifiers, ensuring that malicious peers cannot deterministically select themselves to verify a victim's gradient. Second, noising peers send differentially-private updates [22], [23] to verifier peers: before sending a gradient to a verifier, pre-committed ε-differentially private noise is added to the update, masking the peer's gradient such that no individual peer can influence or observe the model update process. (See Appendix I for details). By verifying noised model updates, verifier peers in Biscotti can verify the updates of others without directly observing their un-noised counterparts.

Utility loss with differential privacy: secure update aggregation and cryptographic commitments. The blockchain-based ledger of model updates allows for auditing of state, but this transparency is counter to the goal of privacy-preserving multi-party ML. For example, the ledger trivially leaks information if we store SGD updates directly.

Using differentially private updates during verification is one technique for preserving data privacy in Biscotti. Another technique used is secure aggregation: a block in Biscotti does not store updates from individual peers, but rather an aggregate that obfuscates any single peer's contribution in a single round. Biscotti uses a verifiable secret sharing scheme [55] to aggregate the updates so that any individual update remains hidden through cryptographic commitments (See Appendix C for background).

However, secure aggregation can be either done on the differentially-private updates or the original updates with no noise. This design choice represents a privacy-utility tradeoff in the final model. By aggregating differentially-private updates the noise is pushed into the ledger and the final model has lower utility. We illustrate this tradeoff experimentally in Appendix H and choose to aggregate the un-noised model updates, providing stronger utility guarantees that match the performance of federated learning. Furthermore, we evaluate the potential for information leakage attacks in Appendix I and show the the probability of a successful attack on Biscotti is negligible.

3 Assumptions and Threat Model

Like federated learning, Biscotti assumes a public ML system in which peers can join/leave anytime. Biscotti assumes that users are willing to collaborate on training ML models, but are unwilling to share their data [5].

3.1 Design assumptions

Proof of federation. Peers in Biscotti use proof-of-federation (PoF) to arrive at a consensus on the state of the model for each round of training. PoF is a consensus protocol for secure and private federated learning based on a recently proposed blockchain consensus mechanism called proof-of-stake (PoS) [32], [56]. Consensus using PoS is fast because it delegates the consensus responsibility to a subset of peers in each round, assigned proportionally based on their stake (see Appendix F for background). In cryptocurrencies, the *stake* of a peer refers to the amount of value (money) that a peer holds. PoS relies on the assumption that a subset of peers holding a significant fraction of the stake will not subvert the system.

In Biscotti's PoF, we define *stake* as a measure of a peer's contribution to the system. Peers acquire stake by providing beneficial model updates or by facilitating the consensus process. Thus, the stake that a peer accrues during training is proportional to their contribution to the model being trained. We assume that the stake ownership of any peer is publicly available from the current state of the blockchain.

In addition, we also assume that at any point in time, at least 70% of the stake in the system is honest and is properly bootstrapped. The initial stake distribution at the start may be derived from an online data sharing marketplace, a shared reputation score among competing agencies, or auxiliary information from a social network.

Blockchain topology. Each peer is connected to some subset of other peers in a topology that allows flooding-based dissemination of blocks to eventually reach all peers. For example, this could be a random mesh topology with flooding, similar to the one used for block dissemination in Bitcoin [31]. Peers that rejoin the system after going offline during training can recreate the latest state of the blockchain from other peers in the system.

Machine learning. We assume that all information required for P2P training is disseminated to all peers in the first block, including the model, hyperparameters, optimization algorithm, and learning objective. In a non-adversarial setting, peers have local datasets that they wish to keep private. When peers draw a sample from this data to compute a model update, we assume that this is done uniformly and independently. This ensures that the Multi-Krum [42] is accurate. In our design of Biscotti we assume stochastic gradient descent (SGD) [57] as the optimization algorithm. SGD is a general learning algorithm that can be used to train a variety of models, including deep neural networks [57].

3.2 Attacker assumptions

Adversarial peers may perform a poisoning attack by sending malicious updates or an information leakage attack by observing a target peer's updates. In doing so, we assume that the adversary may control multiple peers in a sybil attack [45] but does not control more than 30% of the total stake. Although adversaries may be able to increase the number of peers they control in the system, we assume that adversaries cannot artificially increase their stake in the system except by providing valid updates that pass Multi-Krum [42].

When adversaries perform a *poisoning attack*, we assume that their goal is to harm the performance of the final global model. Our defense relies on filtering out malicious updates that are sufficiently different from the honest clients and push the global model towards an sub-optimal objective. For the purposes of this work, we limit adversaries to poisoning attacks [44] in which data is mislabelled to a different class, causing the trained model to misclassify it. This does not include poisoning attacks that leverage unused parts of the model topology, like backdoor attacks [12], attacks based on gradient-ascent [58], or adaptive attacks based on knowledge of the poisoning defense [47]. However, Biscotti is compatible with any other aggregation approach on SGD updates, including future approaches that may handle backdoor and gradient-ascent attacks.

When adversaries perform an *information leakage attack*, we assume that they aim to learn properties of a victim's local dataset. Specifically, we provide record-level privacy, which protects against the de-anonymization of a single example from the user's dataset. Due to the vulnerabilities of secure aggregation, we do not consider information leakage attacks with side information [51], [59] or attacks against class-level privacy [20], which attempt to learn the properties of an entire target class.

4 Biscotti Design

Biscotti's design has the following goals: (1) converge to the optimal global model (the same model trained without adversaries in a federated learning setting), (2) prevent poisoning by verifying peer model updates, (3) keep peer training data private by preventing information leakage attacks, and (4) prevent colluding peers from gaining influence without acquiring sufficient stake. Biscotti meets these goals with a custom blockchain design that we now describe.

Design overview. Peers join Biscotti and collaboratively train a global model. Each block in the distributed ledger represents a single iteration of SGD and the ledger contains the state of the global model at each iteration. Figure 1 overviews the Biscotti design with a step-by-step illustration of what happens during a *single* SGD iteration in which a single block is generated.

In each iteration, peers locally compute SGD updates (step ① in Figure 1). Since SGD updates must be kept private, each peer first *masks* their update using differentially private noise. This noise is collected from a set of noising peers unique to each client and is selected by a VRF [41] (step ② and ③).

The masked updates are validated by a verification committee to defend against poisoning. Each member in the verification committee signs the commitment to the peer's *unmasked* update if it passes Multi-Krum (step ④). If the majority of the committee signs an update (step ⑤), the signed update is divided into Shamir secret shares (step ⑥) and given to a aggregation committee. The aggregation committee uses a secure protocol to aggregate the unmasked updates (step ⑦). All peers who contribute a share to the final update along with the peers chosen for the verification and aggregation committees receive additional stake in the system. The aggregate of the updates is added to the global model in a newly created block which is disseminated to all

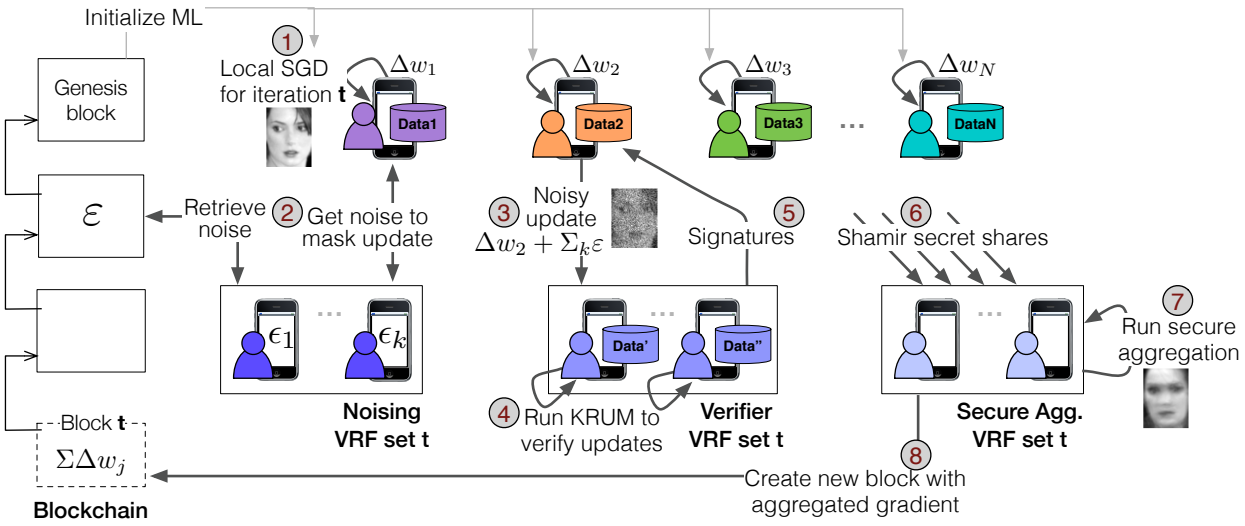

Fig. 1. The ordered steps in a single iteration of the Biscotti algorithm, from step 1 (local SGD) to step 8 (block generation).

the peers and appended to the ledger (step ⑧). Using the updated global model and stake, the peers repeat (step ①). Next, we describe how we bootstrap the training process.

4.1 Initializing the training

Biscotti peers initialize the training process using information in the first (genesis) block. We assume that a trusted authority facilitates and bootstraps the training process by publicly distributing the genesis block out of band to all peers in the system. The authority is only trusted for this step: they are not entrusted with the individual SGD updates of the peers, which could potentially leak private information of a peer's data. Each peer obtains the following information from the genesis block: (1) the initial model state w_0 and expected number of iterations T, (2) the public key PK for creating commitments to SGD updates, (see Appendix C), (3) the public keys PK_i of all other peers in the system, which are used to create and verify signatures during verification, (4) pre-commitments to differentially private noise for T SGD iterations $\zeta_{1..T}$ by each peer, (see Figure 3 and Appendix B), (5) the initial distribution of stake among peers, and (6) a stake update function that executes when new blocks are appended.

4.2 Blockchain design

Distributed ledgers are constructed by appending read-only blocks to a chain structure and disseminating blocks through a gossip protocol. Each block holds a pointer to its previous block as a cryptographic hash of its contents.

Each block in Biscotti (Figure 2) contains, in addition to the previous block hash pointer, an aggregate (Δw) of SGD updates from multiple peers and a snapshot of the global model w_t at iteration t. Newly appended blocks to the ledger store the aggregated updates $\sum \Delta w_i$ of multiple peers. To verify that the aggregate was honestly computed, individual updates need to be included in the block. However, storing them individually leaks information about individual private training data [13], [20]. We solve this

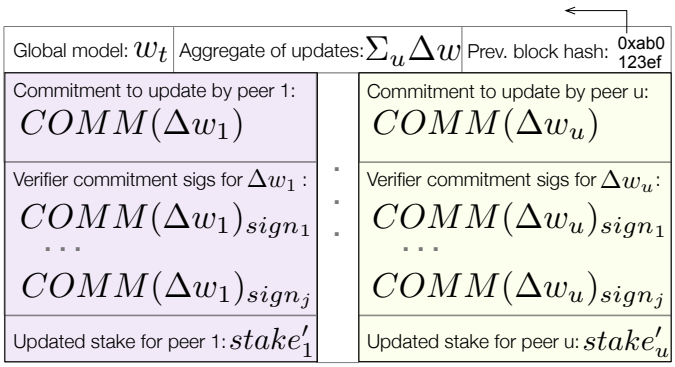

Fig. 2. Block contents at iteration t. Note that w_t is computed using $w_{t-1} + \Sigma w_u$ where w_{t-1} is the global model stored in the block at iteration $t-1$ and j is the set of verifiers for iteration t.

problem by using polynomial commitments [55]. Polynomial commitments take an SGD update and map it to a point on an elliptic curve (see Appendix C for details). By including a list of commitments for each peer i's update $COMM$ (Δw_i) in the block, we can provide both privacy and verifiability of the aggregate. The commitments provide privacy by hiding the individual updates yet can be homomorphically combined to verify that the update to the global model by the aggregator $\sum \Delta w_i$ was computed honestly. The following equality holds if the list of committed updates equals the aggregate sum:

$$COMM(\sum \Delta w_i) = \prod_i COMM(\Delta w_i)$$

The training process continues for a specified number of iterations T, at which point the learning process stops and each peer extracts the global model from the final block.

4.3 Using stake for role selection

In each iteration in Biscotti, a consistent hash weighted by peer stake designates roles (noiser, verifier, aggregator) to some peers in the system. PoF ensures that the influence of a peer is bounded by their stake (i.e., adversaries cannot

trivially increase their influence through sybils). Peers may be assigned multiple roles in a given iteration but cannot be a verifier and aggregator in the same round. The verification and aggregation committees are the same for all peers but the noising committee is unique to each peer.

The initial SHA-256 hash of the last block is repeatedly re-hashed: each new hash result is mapped onto a hash ring where portions of the ring are proportionally assigned to peers based on their stake. The hash is repeated until verifier/aggregator committees of the correct size are obtained. This provides the same stake-based property as Algorand [32]: a peer's probability of winning a lottery is proportional to their stake. Since an adversary cannot predict the future state of a block until it is created, they cannot speculate on outputs of consistent hashing and strategically perform attacks. Unlike verification and aggregation, the noising committee is different for each peer for additional privacy. Each peer can arrive at a unique committee for itself via consistent hashing by using a different initial hash. The hash computed should be random yet globally verifiable by other peers in the system. In Biscotti, a peer computes this hash by executing their secret key SK_i and the SHA-256 hash of the last block through a verifiable random function (VRF) (see Appendix D). By virtue of the peer's secret key, the hash computed is unique to the peer resulting in distinct committees for different peers. The VRF hash is also accompanied by a proof that can be combined with a peer's public key PK_i, allowing other peers to determine that the correct noising committee is selected by the peer.

At each iteration, peers run the consistent hashing protocol to determine whether they are an aggregator or verifier. Peers that are part of the verification and aggregation committees do not contribute updates for that round. Each peer that is not an aggregator or verifier computes their noising committee, obtains the noise, and hides their updates by following the noising protocol.

4.4 Noising protocol

To prevent information leakage attacks, peers use differential privacy to hide their updates during verification by adding noise sampled from a normal distribution. This ensures that each step is (ϵ, δ)-differentially private [23]. (See Appendix B for formalisms).

Using pre-committed noise to thwart poisoning. Attackers may maliciously use the noising protocol to execute poisoning or information leakage attacks. For example, a peer can send a poisonous update Δw_{poison}, and add noise ζ_p that *unpoisons* this update to resemble an honest update Δw, such that $\Delta w_{poison} + \zeta_p = \Delta w$. By doing this, the honest update Δw passes verification, but the poisonous update Δw_{poison} is applied to the model since the noise is removed in the final aggregate.

An information leakage attack may also occur when a noising peer A and a verifier B collude against a victim peer C. The noising peer A can commit a set of zero noise that does not hide the original gradient value at all. When the victim peer C sends its noised gradient to the verifier B, B can perform an information leakage attack on client C's gradient to reconstruct C's training data.

To prevent such attacks, Biscotti requires that every peer pre-commits the noise vector ζ_t for every future iteration $t \in [1..T]$ in the genesis block, and that each iteration uses a private VRF to select a *group* of noising peers based on the victim C's secret key. In doing so, an adversary cannot pre-determine whether their noise will be used in a specific verification round by a particular victim, and also cannot pre-determine if the other peers in the noising committee will be malicious in a particular round. We analyze this probability in Appendix I and show that the probability of an information leakage is negligible given a sufficient number of noising peers.

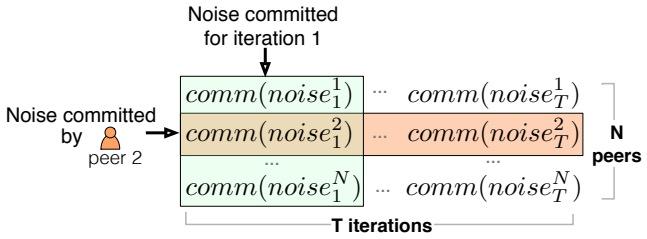

Fig. 3. Peers commit noise to an N by T structure. Each row i contains all the noise committed by a single peer i, and each column t contains potential noise to be used during iteration t. When committing noise at an iteration i, peers execute a VRF and request ζ_i^k from peer k

Protocol description. For an ML workload that may be expected to run for a specific number of iterations T, each peer i generates T noise vectors ζ_t and commits these noise vectors into the ledger, storing a table of size N by T (Figure 3). When a peer is ready to contribute an update in an iteration, it runs the noising VRF and contacts each noising peer k, requesting the noise vector ζ_k pre-committed in the genesis block $COMM(\zeta_k)$. The peer then uses a verifier VRF to determine the set of verifiers. The peer *masks* their update using this noise and submits to these verifiers the *masked* update, a commitment to the noise, and a commitment to the *unmasked* update. It also submits the noise VRF proof that attests to the verifier that its noise is sourced from peers that are a part of their noise VRF set.

4.5 Verification protocol

Verifier peers run Multi-Krum on the received set of updates and filter out malicious updates by accepting the top majority of the updates received in each round. Each verifier receives the following from each peer i: (1) the masked SGD update: $(\Delta w_i + \sum_k \zeta_k)$, (2) a commitment to the SGD update: $COMM(\Delta w_i)$, (3) a set of k noise commitments: $\{COMM(\zeta_1), COMM(\zeta_2), ..., COMM(\zeta_k))\}$ and (4) a VRF proof confirming the identity of the k noise peers.

When a verifier receives a masked update from another peer, it can confirm that the *masked* SGD update is consistent with the commitments to the *unmasked* update and the noise by using the homomorphic property of the commitments [55]. A masked update is legitimate if the following equality holds:

$$COMM(\Delta w_i + \sum_k \zeta_k) = COMM(\Delta w_i) * \prod_k COMM(\Delta \zeta_k)$$

Once the verifier receives a sufficiently large number of updates R, it proceeds with selecting the best updates using Multi-Krum, as follows:

1) For every peer i, the verifier calculates a score $s(i)$ which is the sum of Euclidean distances of i's update to the closest $R-f-2$ updates. It is given by: $s(i) = \sum_{i \to j} \|(\Delta w_i + \sum_{i,k} \zeta_{i,k}) - (\Delta w_j + \sum_{j,k} \zeta_{j,k})|^2$ where $i \to j$ denotes the fact that $(\Delta w_j + \sum_{j,k} \zeta_{j,k})$ belongs to the $R-f-2$ closest updates to $(\Delta w_i + \sum_{i,k} \zeta_{i,k})$.
2) The $R-f$ peers with the lowest scores are selected while the rest are rejected.
3) The verifier signs the $COMM(\Delta w_i)$ using its public key for all the accepted updates.

To prevent a malicious verifier from accepting all updates of its colluders in this stage, we require a peer to obtain signatures from the majority of verifiers before their update is accepted and disseminated for aggregation.

4.6 Aggregation protocol

All peers with a sufficient number of signatures from the verification stage submit their SGD updates for aggregation into the global model. The update equation in SGD (see Appendix A) can be re-written as:

$$w_{t+1} = w_t + \sum_{i=1}^{\Delta w_{verified}} \Delta w_i$$

where Δw_i is the verified SGD update of peer i and w_t is the global model at iteration t.

The aggregation protocol enables a set of m aggregators, predetermined by consistent hashing, to compute $\sum_i \Delta w_i$ without observing any individual updates. Biscotti uses a technique that preserves privacy of the individual updates if at least half of the m aggregators participate honestly in the aggregation phase. This guarantee holds if consistent hashing selects a majority of honest aggregators, which is likely when the majority of stake is honest. Biscotti achieves the above guarantees using polynomial commitments (described in Appendix C) and verifiable secret sharing [43] for aggregation of individual updates. We describe the details of the aggregation protocol in Appendix G.

4.7 Blockchain consensus

Since subsets based on consistent hashing are globally observable by each peer and rely on the SHA-256 hash of the latest block in the chain, ledger forks should rarely occur in Biscotti. For an update to be included in the ledger at any iteration, the same noising/verification/aggregation committees are used. Thus, race conditions between aggregators will not cause forks in the ledger to occur as frequently as in e.g., BitCoin [31].

When a peer observes a new block through the gossip protocol, it can verify that the computation performed is correct by running the consistent hashing protocol for the latest block and verifying the signatures of the designated verifiers and aggregators for each new block.

In Biscotti, each verification and aggregation step occurs only for a specified duration. Any updates that are not successfully propagated in this period of time are dropped: Biscotti does not append stale updates to the model once competing blocks have been committed to the ledger. This synchronous SGD model is acceptable for large scale ML workloads which have been shown to be tolerant of bounded asynchrony [28]. However, these stale updates could be leveraged in future iterations if their learning rate is decayed [60]. We leave this optimization for future work.

TABLE 1
The default parameters used for all our experiments, unless stated otherwise.

Parameter	Default Value
Privacy budget (ε)	2
Delta (δ)	10^{-5}
Number of peers	100
Number of noisers	2
Number of verifiers	3
Number of aggregators	3
Number of samples needed for Multi-Krum (R)	70
Adversary upper bound ($f < \frac{R-2}{2}$)	33
Number of updates/block ($u = \frac{R}{2}$)	35
Initial stake	Uniform, 10 each
Stake update	Linear, + 5

TABLE 2
The dataset/model types used in the experiments

Dataset	Model Type	Train/Test Examples	Params (d)
Credit Card	LogReg	21000/9000	25
MNIST	Softmax	60000/10000	7850

5 Implementation

We implemented Biscotti in 4,500 lines of Go 1.10 and 1,000 lines of Python 2.7.12 and released it as an open source project[3]. We use Go to handle all networking and distributed systems aspects of our design. We used PyTorch 0.4.1 [61] to generate SGD updates and noise during training. By building on the general-purpose API in PyTorch, Biscotti can support any model that can be optimized using SGD. We use the *go-python v1.0* [62] library to interface between Python and Go. Since go-python is incompatible with Python 3, we were limited to using Python 2.7 for our implementation.

We use the *kyber v.2* [63] and *CONIKS 0.1.1* [64] libraries to implement the cryptographic parts of our system. We use CONIKS to implement our VRF function and kyber to implement the commitment scheme and public/private key mechanisms. To bootstrap clients with the noise commitments and public keys, we use an initial genesis block. We used the bn256 curve API in kyber for generating our commitments and public keys that form the basis of the aggregation protocol and verifier signatures. For signing updates, we use the Schnorr signature [65] scheme instead of ECDSA because multiple verifier Schnorr signatures can be aggregated together into one signature [66]. Therefore, our block size remains constant as the verifier set grows.

6 Evaluation

We had several goals in the evaluation of our Biscotti prototype. We wanted to demonstrate that (1) Biscotti is robust

3. https://github.com/DistributedML/Biscotti

to poisoning attacks, (2) Biscotti protects the privacy of an individual client's data and (3) Biscotti is scalable, fault-tolerant and can be used to train different ML models.

For experiments done in a distributed setting, we deployed Biscotti across 20 Azure A4m v2 virtual machines, with 4 CPU cores and 32 GB of RAM. We deployed a varying number of peers in each of the VMs. The VM's were spread across six locations: West US, East US, Central India, Japan East, Australia East and Western Europe. Each experiment was executed for 100 iterations and the test error of the global model was recorded. To evaluate Biscotti's defense mechanisms, we ran information leakage and poisoning attacks on federated learning from prior work [13], [44] and measured their effectiveness under various attack scenarios and Biscotti parameters. We also evaluated the performance implications of our design by isolating specific components of Biscotti across varying committee sizes. For all our experiments, we deployed Biscotti with the parameter values in Table 1 unless stated otherwise.

We evaluated Biscotti with both a logistic regression and softmax classifiers. We evaluate logistic regression with a Credit Card fraud dataset [67], which uses an individual's financial and personal information to predict if they will default on their next credit card payment. Our softmax classifier contains a 1-layer neural network with a binary cross entropy loss, and we evaluate it on the MNIST [68] dataset, a task that involves predicting a digit based on its image. We claim that the general-purpose PyTorch API allows Biscotti to support models of arbitrary size and complexity, as long as they can be optimized with SGD and can be stored in our block structure.

The MNIST and Credit Card datasets have 60,000 and 21,000 training examples respectively. We used 5-fold cross validation on the training set to determine training parameters for each model. The Credit Card logistic regression model uses a batch size of 10, learning rate of 0.01 and no momentum. The MNIST softmax model uses a batch size of 10, learning rate of 0.001 and momentum of 0.75.

The MNIST/Credit Card models were tested using a separately held-out test set of 10,000 and 9,000 examples respectively. For all experiments, the training dataset was divided equally among the peers unless stated otherwise. As a result, each client possesses 600 training examples for MNIST and 210 examples for the credit card dataset. Table 2 shows the details of our datasets. In local training we were able to train a model on the Credit Card and MNIST datasets with accuracy of 98% and 92%, respectively.

6.1 Tolerating poisoning attacks

In this section, we evaluate Biscotti's performance against a label-flipping poisoning attack [44]. We also investigate the different parameter settings required for Biscotti to successfully defend against a poisoning attack and evaluate how well it performs under attack from 30% malicious nodes compared to a federated learning baseline.

Biscotti requires each peer in the verification committee to collect a sufficient sample of updates before running Multi-Krum. We evaluate the effect of varying the percentage of total updates collected in each round by Multi-Krum against varying poisoning attack rates in MNIST to evaluate

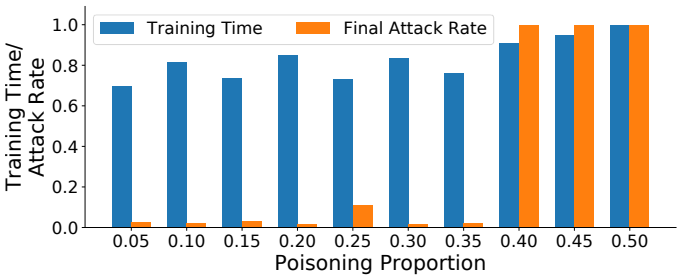

Fig. 4. Biscotti's performance on the MNIST dataset with a varying number of poisoners, from 5% to 50%. Both the total execution time and the final attack rate are scaled such that the maximum is 1.

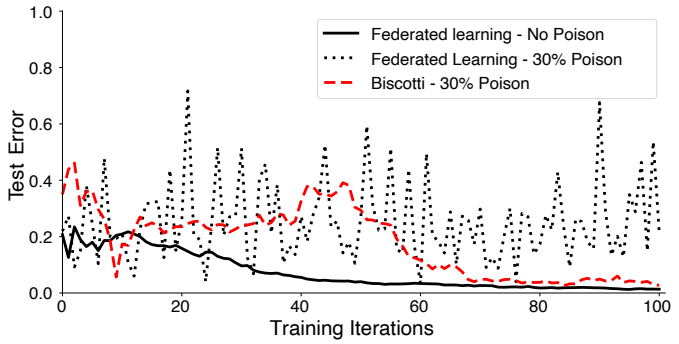

Fig. 5. Federated learning and Biscotti's test error on the CreditCard dataset with 30% of poisoners.

Multi-Krum's effectiveness. To ensure uniformity and to eliminate latency effects in the collection of updates, in these experiments the verifiers waits to collect all updates from all peers that are not assigned to a committee and then randomly samples a specified number of these updates. In addition, we also ensured that all verifiers deterministically sampled the *same* set of updates by using the last block hash as the random seed. This allows us to investigate how well Multi-Krum performs in a decentralized ML setting like Biscotti unlike the original Multi-Krum paper [42] in which updates from *all* clients are collected before running Multi-Krum.

To evaluate the success of an attack, we define *attack rate* as the fraction of target labels that are incorrectly predicted. An *attack rate* of 1 specifies a successful attack while a value close to zero indicates an unsuccessful one. Figure 4 shows both the total execution time and the resulting attack rate when Biscotti is attacked by a varying number of poisoners. Although our theoretical guarantee ensures that Biscotti can withstand poisoning attacks from up to 30% poisoners, Biscotti performs well even against 35% poisoners. Beyond 35%, Multi-Krum is unable to prevent the poisoning attack from occurring, impacting both the final model performance and the total execution time. Multi-Krum is also impacted by the privacy parameter ε and the number of samples used when removing anomalies. Experiments detailing the impact of both parameters on Multi-Krum's poisoning deterrence are in Appendix J.

Biscotti vs Federated Learning Baseline. We deployed Biscotti and federated learning and subjected both systems to a poisoning attack while training on the Credit Card and MNIST datasets. Using the parameters from the above

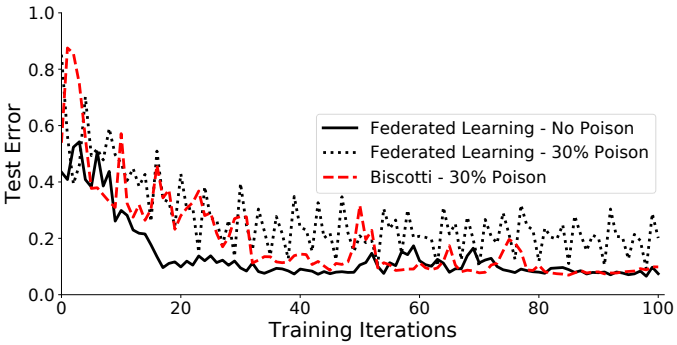

Fig. 6. Federated learning and Biscotti's test error on the MNIST dataset with 30% of poisoners.

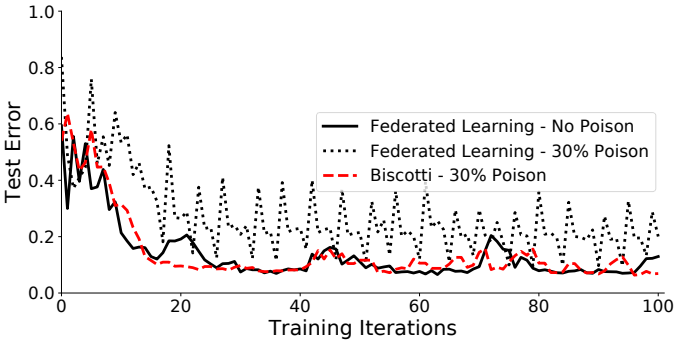

Fig. 7. Federated learning and Biscotti's test error on the MNIST dataset with 200 total nodes, 30% poisoners, with a verification and aggregation committee size of 26.

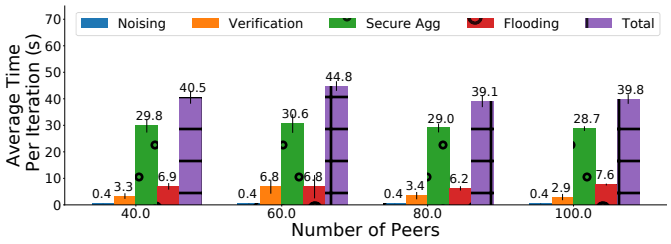

Fig. 8. Breakdown of time spent in different mechanisms in Biscotti (Figure 1) in deployments with varying number of peers.

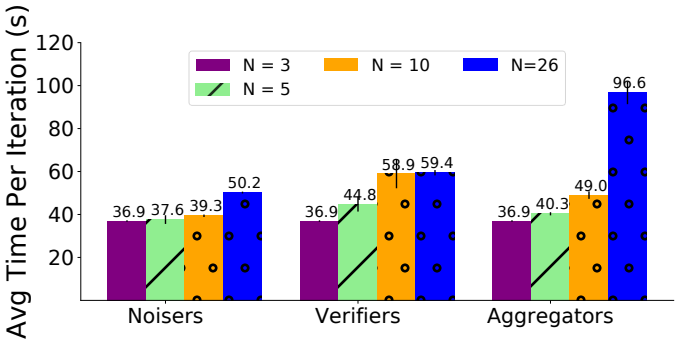

Fig. 9. The average amount of time taken per iteration with an increasing number of noisers, verifiers, or aggregators.

experiments, Biscotti sampled 70% of updates and used a value of 2 for the privacy budget ϵ.

We introduced 30% of the peers into the system with the same malicious dataset: for the Credit Card dataset they labeled all defaults as non-defaults, and for MNIST these peers labeled all 1s as 7s. Figures 5 and 6 show the test error when compared to federated learning. The results show that for both datasets, the poisoned federated learning deployment struggles to converge. In contrast, Biscotti performs as well as the baseline un-poisoned federated learning deployment on the test set.

Biscotti requires a larger number of iterations to converge than un-poisoned federated learning. The convergence is slower for Biscotti because a small verification committee of 3 peers was used, which allows the poisoning peers to control a majority in the committee frequently and accept updates from fellow malicious peers. Biscotti finally converges as the honest peers gain enough stake over time, thereby reducing the influence of malicious peers in the system. Over the course of this experiment, where a constant +5 stake update function is used, the stake of the honest clients increases from 70% to 87%.

To demonstrate the effect of the committee size on convergence under attack, we re-ran the experiment on the MNIST dataset with a larger verification and aggregation committee size of 26 peers, which we analytically determine is large enough to provide protection against an adversary that controls 30% of the stake in the system (shown in Appendix K). Since peers in the verification or aggregation committees do not contribute updates in that particular round, 100 nodes cannot generate the 70% of updates needed to protect against Multi-Krum. Therefore, for this experiment we increased the number of nodes to 200. The results in Figure 7 show that Biscotti, with a larger verification and aggregation committee size, converges in the same number of iterations as unpoisoned federated learning.

6.2 Performance, scalability and fault tolerance

In this section, we evaluate the overhead of each stage in Biscotti and investigate the effect as the number of peers increase. We also measure Biscotti's performance as we scale the size of different committees and compare its performance against federated learning. Finally, we evaluate if client churn has any effect on Biscotti's convergence.

Performance cost break-down. In breaking down the overhead in Biscotti, we deployed Biscotti over a varying number of peers in training an MNIST softmax model. We capture the amount of time spent in each of the major stages of our algorithm in Figure 1: collecting the noise from the noising clients (steps ② and ③), verification via Multi-Krum and signature collection (steps ④ and ⑤) and secure aggregation of the SGD update (steps ⑥ and ⑦). Figure 8 shows the average cost per iteration for each stage under a deployment of 40, 60, 80 and 100 nodes over 3 runs. During this experiment, the committee sizes were kept constant to the default values in Table 1.

The results show that the cost of each stage is almost constant as we vary the number of peers in the system. Biscotti spends most of its time in the aggregation phase since it requires the aggregators to collect secret shares of all the accepted updates and share the aggregated shares with each other to generate a block. The noising phase is

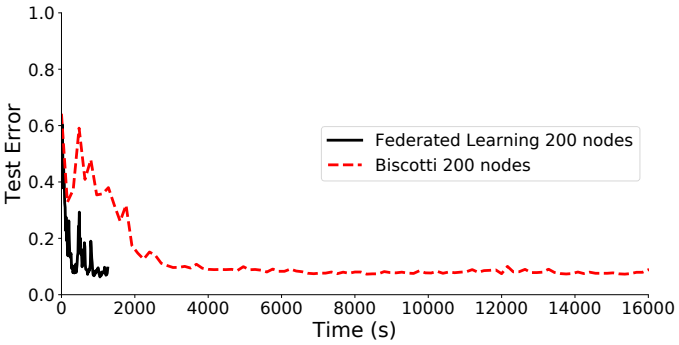

Fig. 10. Comparing the convergence of Biscotti to a federated learning baseline over time, while training an MNIST model over 200 nodes.

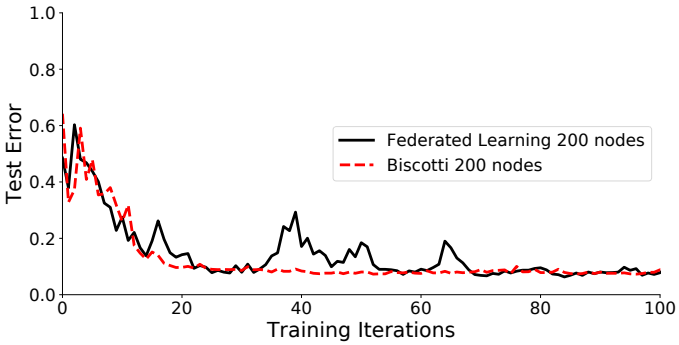

Fig. 11. Comparing the convergence of Biscotti to a federated learning baseline over the number of iterations, while training an MNIST model over 200 nodes.

the fastest since it only involves a single round trip to each member of the noising committee while the verification stage involves collecting a predefined percentage (70%) of updates to run Multi-Krum in a asynchronous manner from all the nodes. The time per iteration also stays fairly constant as the number of nodes in the system increases.

Scaling up committee sizes in Biscotti. We evaluate Biscotti's performance as we change the size of the noiser/verifier/aggregator sets. For this we deploy Biscotti on Azure with the MNIST dataset with a fixed size of 100 peers, and only vary the number of noisers needed for each SGD update, number of verifiers used for each verification committee, and the number of aggregators used in secure aggregation. Each time one of these sizes was changed, the change was performed in isolation; the rest of the committees used a set of 3. Figure 9 plots the average time taken per iteration over 3 executions of the system.

The results show that increasing the size of the noising, verifiers or aggregator sets increases the time per iteration. The iteration time increases slightly with noising committee since additional peers must be contacted to determine the total noise. Increasing the number of aggregators leads to a larger overhead because the secret shares are exchanged between more peers and recovering the aggregate requires coordination among more peers. Lastly, a large number of verifiers results in frequent timeouts in the aggregation stage. Verifiers wait for the first 70 updates and select 37 updates while aggregators wait for shares from the first 35 updates before initiating the aggregate recovery process. With an increased verifier set, the size of the intersection of updates accepted by a majority of verifiers frequently falls below 35, as each verifier runs Multi-Krum on a different set of updates. Hence, the aggregators wait until the update timeout is hit. Since the timeout is a constant value of 90 seconds, the verifier overhead does not increase significantly when increasing the verifier set from 10 to 26. However, our design could decrease this verifier overhead by decreasing the number of updates that aggregators wait for before starting their coordination.

Baseline performance. We compare Biscotti to the original federated learning baseline [5] with an execution using 5 noisers, 26 verifiers and 26 aggregators. We analytically determine that these committee sizes provide a guarantee of protecting against an adversary that holds 30% stake from unmasking updates in the noising (Appendix I) and aggregation stages (Appendix K). These committee sizes also ensure convergence under poisoning from an attack by 30% of the nodes (Section 6.1).

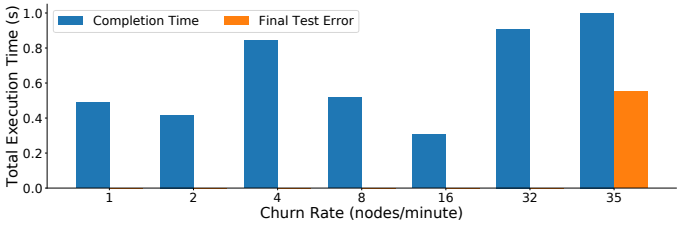

Fig. 12. The impact of churn on model performance. Churn is injected by failing/joining random nodes at a pre-determined rate.

We divide the MNIST dataset [68] into 200 equal partitions, each of which was shared with an honest peer deployed in one of 20 Azure VMs, with each VM hosting 10 peers. These peers collaborated on training an MNIST softmax classifier, and after 100 iterations both Biscotti and federated learning approached the global optimum. To ensure a fair comparison, the number of updates included in the model in each round are kept the same. In federated learning, we receive updates from 35% of the nodes selected at random for every round, while in Biscotti we include the first 35% verified updates in the block. The convergence for both systems are shown in Figures 10 and Figure 11, showing time and the number of iterations respectively. In this deployment, Biscotti takes about 13.8 times longer than Federated Learning (20.8 minutes vs 266.7 minutes), yet achieves similar model performance (92% accuracy) after the same 100 iterations.

We also evaluated the effect on Biscotti of an increased dataset size and model parameters in Appendix L. For dataset size, we increased the MNIST dataset to 600,000 and 6,000,000 training examples respectively by replicating the data available at each node. Our results shows that, despite increased dataset size, the Biscotti overhead stays at 14x. When increasing the model parameter size, we were able to train models with up to 117,540 parameters before the system failed. Biscotti can support larger models with better tuning of timeouts and a reduced communication overhead, which is discussed in Section 7.

Training with node churn. A key feature of Biscotti's P2P design is that it is resilient to node churn (node joins and

failures). For example, the failure of any Biscotti node does not block or prevent the system from converging. We evaluate Biscotti's resilience to peer churn by performing a Credit Card deployment with 50 peers, failing a peer at random at a specified rate. For each specified time interval, a peer is chosen randomly and is killed. In the next time interval, a new peer joins the network, maintaining a constant total number of 50 peers. Figure 12 shows the time to converge for varying churn rates. When a verifier or an aggregator fails, Biscotti defaults to the next iteration after a timeout, so this does not harm convergence, but does introduce variance in execution time. When the churn rate increases to 35 nodes/minute, the system does not converge, likely since 35 is the minimum updates required to create a block in our setup. Even with churn Biscotti makes progress towards the global objective; we found that Biscotti is resilient to churn rates up to 32 nodes per minute (1 node joining and 1 node failing every 1.875 seconds).

7 Limitations and Future Work

Multi-Krum limitations. For Multi-Krum to be effective, it needs to observe a large number of honest samples in each round. This may not always be possible in a decentralized system with node churn. In addition, Multi-Krum could also reject updates from peers that have non-iid data e.g., a peer that only possesses one class in the model. We did not face this issue in our experiments because we partitioned the data uniformly. However, Biscotti is compatible with other SGD aggregation approaches [49], [50]. For example, in an earlier version of our design we used a version of RONI [69] to validate peer updates.

Deep Learning. Biscotti currently does not scale to large deep learning models with millions of parameters due to the communication overhead. Previous work in federated learning addressed this problem by reducing the model size updates through learning in a restricted parameter space or compressing model updates [70], [71]. There is also extensive research outside of federated learning in training more compact and efficient neural networks with techniques such as weight quantization [72], network pruning [73], knowledge distillation [74], and designs for resource-constrained settings [75]. Another strategy to increase scalability involves improved communication efficiency. This can be achieved through transfer learning [76] on a restricted parameter space or by better partitioning and rearranging of the tensor updates [77]. We leave the application of these techniques to Biscotti as future work.

Leakage from the aggregate model. Since no noise is added to the updates present in the ledger, Biscotti is vulnerable to attacks that exploit privacy leakage from the model itself [51], [52], [59]. Apart from differential privacy, these attacks can be mitigated by adding regularization like dropout [13], [51].

Stake limitations. A client's stake plays a significant role in their chance of being selected as a noiser, verifier, or aggregator. We assume that a large stake-holder will not subvert the system because (1) they accrue more stake by participating, and (2) their stake is tied to a monetary reward at the end of training. However, a malicious client could pose as honest, accrue stake, and finally switch to acting maliciously to subvert our system.

8 Related work

Securing ML. AUROR [21] and ANTIDOTE [78] are alternative techniques to Multi-Krum to defend against poisoning. AUROR has been proposed for the model averaging use case and uses k-means clustering on a subset of important features to detect anomalies. ANTIDOTE uses a robust PCA detector to protect against attackers on anomaly-detection models.

Other defenses like TRIM [79] and RONI [69] filter out poisoned data from a centralized training set based on their impact on performance on the dataset. TRIM trains a model to fit a subset of samples selected at random, and identifies a training sample as an outlier if the error when fitting the model to the sample is higher than a threshold. RONI trains a model with and without a data point and rejects it if it degrades the performance of the classifier. Recent work has also demonstrated the performance of robust aggregation schemes, such as the median or trimmed-mean [49] as a defense for federated learning. However, these techniques can be manipulated by adversaries [18], [48].

Finally, Baracaldo et al. [80] employ data provenance as a measure against poisoning attacks by tracking the history of the training data and removing information from anomalous sources.

Poisoning attacks. Apart from label flipping attacks [44], which are handled by Biscotti, gradient ascent techniques [58], [79] are another popular way of generating poisoned samples one sample at a time by solving a bi-level optimization problem. Backdoor attacks have also been used to produce a classifier that misclassifes a manipulated image with certain pixels or a backdoor pattern [12], [17], [81]. Defending against such training attacks is a difficult and open problem.

Privacy attacks. Shokri et al. [82] demonstrated a membership inference attack using shadow model training that learns if a victim's data was used to train the model. Follow up work [51] showed that it is quite easy to launch this attack in a black-box setting. In addition, model inversion attacks [59] have been proposed to invert class images from the final trained model. However, these are attacks on class-level privacy, which we do not protect against in Biscotti. These are only effective against a user's privacy if a class represents a significant chunk of a person's data, such as in facial recognition systems.

Privacy-preserving ML systems. Although a variety of recent work has proposed the use of blockchain systems for distributed learning [38], [39], [40], [83], [84], [85], [86], Biscotti addresses a unique point in the space of solutions. The existing prior work either relies on a trusted central authority, does not address poisoning attacks, does not handle privacy attacks, or are actually layer-2 blockchain solutions, which are machine learning applications built on top of an existing blockchain systems such as Ethereum [87]. We summarize previous work that combines blockchains with ML in Appendix M.

Other solutions that do not rely on blockchain use multi-party party computation [88], [89] or trusted execution

environments [36] to encrypt the model parameters and the training data when performing multi-party ML. Biscotti does not rely on such techniques to provide privacy.

9 Conclusion

The emergence of large scale multi-party ML workloads and distributed ledgers for scalable consensus have produced two rapid and powerful trends. Biscotti's design lies at their confluence. To the best of our knowledge, this is the first system to provide privacy-preserving P2P ML through the design of a secure layer-1 distributed blockchain ledger. Unlike prior work, we do not rely on a centralized service, trusted execution environments or specialized hardware to provide defenses against adversaries. In our evaluation we demonstrate that Biscotti can coordinate P2P ML across 200 peers and produces a final model that is similar in utility to federated learning. We also illustrated its ability to withstand poisoning attacks and node churn (one node joining and one node leaving every 1.875 seconds). Our prototype is open source, runs on commodity hardware, and interfaces with PyTorch, a popular framework for machine learning: https://github.com/DistributedML/Biscotti

Acknowledgments

We would like to express our thanks to Margo Seltzer for her helpful feedback and comments. We would like to further thank Matheus Stolet for his help in evaluating the prototype at scale, and Heming Zhang for his help with bootstrapping the deployment of Biscotti on PyTorch. We would also like to thank Gleb Naumenko, Nico Ritschel, Jodi Spacek, and Michael Hou for their work on the initial Biscotti prototype.

This research has been sponsored by the Huawei Innovation Research Program (HIRP), Project No: HO2018085305. We also acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), 2014-04870.

References

[1] Watcharapichat, Pijika and Morales, Victoria Lopez and Fernandez, Raul Castro and Pietzuch, Peter, "Ako: Decentralised deep learning with partial gradient exchange," in *SoCC*, 2016.

[2] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *ICDCS*, 2017.

[3] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching LAN speeds," in *NSDI*, 2017.

[4] J. Xu and F. Wang, "Federated learning for healthcare informatics," *arXiv:1911.06270*, 2019.

[5] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *AISTATS*, 2017.

[6] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *NIPS*, 2017.

[7] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. M. Kiddon, J. Konečný, S. Mazzocchi, B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," in *SysML*, 2019.

[8] Nguyen, Thien Duc and Marchal, Samuel and Miettinen, Markus and Fereidooni, Hossein and Asokan, N and Sadeghi, Ahmad-Reza, "DÏoT: A federated self-learning anomaly detection system for IoT," in *ICDCS*, 2019.

[9] L. Lyu, J. Yu, K. Nandakumar, Y. Li, X. Ma, J. Jin, H. Yu, and K. S. Ng, "Towards Fair and Privacy-Preserving Federated Deep Models," *arXiv:1906.01167*, 2019.

[10] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *arXiv:1812.00564*, 2018.

[11] C. Fung, C. J. Yoon, and I. Beschastnikh, "The Limitations of Federated Learning in Sybil Settings," in *RAID*, 2020.

[12] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How To Backdoor Federated Learning," *arXiv:1807.00459*, 2018.

[13] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *IEEE S&P*, 2019.

[14] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *arXiv:1912.04977*, 2019.

[15] B. Biggio, B. Nelson, and P. Laskov, "Poisoning Attacks Against Support Vector Machines," in *ICML*, 2012.

[16] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. P. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia, "Exploiting Machine Learning to Subvert Your Spam Filter," in *LEET*, 2008.

[17] C. Xie, K. Huang, P.-Y. Chen, and B. Li, "DBA: Distributed backdoor attacks against federated learning," in *ICLR*, 2020.

[18] M. Fang, X. Cao, J. Jia, and N. Z. Gong, "Local model poisoning attacks to byzantine-robust federated learning," in *USENIX Security*, 2020.

[19] C. Xie, O. Koyejo, and I. Gupta, "Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance," in *ICML*, 2019.

[20] B. Hitaj, G. Ateniese, and F. Pérez-Cruz, "Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning," in *CCS*, 2017.

[21] S. Shen, S. Tople, and P. Saxena, "Auror: Defending against poisoning attacks in collaborative deep learning systems," in *ACSAC*, 2016.

[22] M. Abadi, A. Chu, I. Goodfellow, B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *CCS*, 2016.

[23] C. Dwork and A. Roth, "The Algorithmic Foundations of Differential Privacy," *Foundations and Trends in Theoretical Computer Science*, 2014.

[24] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *NIPS Workshop: Machine Learning on the Phone and other Consumer Devices*, 2017.

[25] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *CCS*, 2015.

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

[27] J. So, B. Guler, and A. S. Avestimehr, "Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning," *arXiv:2002.04156*, 2020.

[28] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *NIPS*, 2011.

[29] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent," in *NIPS*, 2017.

[30] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *OSDI*, 1999.

[31] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009.

[32] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *SOSP*, 2017.

[33] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "Medrec: Using blockchain for medical data access and permission management," in *OBD*, 2016.

[34] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," in *IEEE S&P*, 2014.

[35] N. Narula, W. Vasquez, and M. Virza, "zkledger: Privacy-preserving auditing for distributed ledgers," in *NSDI*, 2018.

[36] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *IEEE EuroS&P*, 2019.

[37] C. Xu, K. Wang, P. Li, S. Guo, J. Luo, B. Ye, and M. Guo, "Making big data open in edges: A resource-efficient blockchain-based approach," *IEEE TPDS*, 2019.

[38] Q. Wang, Y. Guo, X. Wang, T. Ji, L. Yu, and P. Li, "AI at the Edge: Blockchain-Empowered Secure Multiparty Learning with Heterogeneous Models," *IEEE Internet of Things Journal*, 2020.

[39] Y. Zhao, J. Zhao, L. Jiang, R. Tan, and D. Niyato, "Mobile Edge Computing, Blockchain and Reputation based Crowdsourcing Federated Learning: A Secure, Decentralized and Privacy-preserving System," *arXiv:1906.10893*, 2019.

[40] X. Chen, J. Ji, C. Luo, W. Liao, and P. Li, "When machine learning meets blockchain: A decentralized, privacy-preserving and secure design," in *IEEE Big Data*, 2018.

[41] S. Micali, S. Vadhan, and M. Rabin, "Verifiable random functions," in *FOCS*, 1999.

[42] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *NIPS*, 2017.

[43] A. Shamir, "How to share a secret," *Communications of the ACM*, 1979.

[44] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar, "Adversarial Machine Learning," in *AISec*, 2011.

[45] J. J. Douceur, "The sybil attack," in *IPTPS '02*.

[46] S. Mahloujifar, M. Mahmoody, and A. Mohammed, "Multi-party poisoning through generalized p-tampering," *arXiv:1809.03474*, 2018.

[47] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *ICML*, 2019.

[48] C. Xie, O. Koyejo, and I. Gupta, "Fall of empires: Breaking byzantine-tolerant SGD by inner product manipulation," in *UAI*, 2019.

[49] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *ICML*, 2018.

[50] E. M. E. Mhamdi, R. Guerraoui, and S. Rouault, "The hidden vulnerability of distributed learning in byzantium," *arXiv:1802.07927*, 2018.

[51] A. Salem, Y. Zhang, M. Humbert, M. Fritz, and M. Backes, "ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models," *arXiv:1806.01246*, 2018.

[52] L. Wang, S. Xu, X. Wang, and Q. Zhu, "Eavesdrop the composition proportion of training labels in federated learning," *arXiv:1910.06044*, 2019.

[53] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Stand-alone and federated learning under passive and active white-box inference attacks," *arXiv:1812.00910*, 2018.

[54] S. Truex, L. Liu, M. E. Gursoy, L. Yu, and W. Wei, "Towards demystifying membership inference attacks," *arXiv:1807.09173*, 2018.

[55] A. Kate and I. Zaverucha, Gregory M.and Goldberg, "Constant-size commitments to polynomials and their applications," in *ASIACRYPT*, 2010.

[56] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *CRYPTO*, 2017.

[57] L. Bottou, *Large-Scale Machine Learning with Stochastic Gradient Descent*, 2010.

[58] L. Muñoz González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli, "Towards poisoning of deep learning algorithms with back-gradient optimization," in *AISec*, 2017.

[59] M. Fredrikson, S. Jha, and T. Ristenpart, "Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures," in *CCS*, 2015.

[60] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," in *NIPS*, 2012.

[61] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS Autodiff Workshop*, 2017.

[62] "go-python," https://github.com/sbinet/go-python, 2018.

[63] "DEDIS Advanced Crypto Library for Go," https://github.com/dedis/kyber, 2018.

[64] "A CONIKS Implementation in Golang," https://github.com/coniks-sys/coniks-go, 2018.

[65] C. P. Schnorr, "Efficient identification and signatures for smart cards," in *CRYPTO*, 1990.

[66] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille, "Simple schnorr multi-signatures with applications to bitcoin," Cryptology ePrint Archive, Report 2018/068, 2018.

[67] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[68] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, 1998.

[69] M. Barreno, B. Nelson, A. D. Joseph, and J. D. Tygar, "The Security of Machine Learning," *Machine Learning*, vol. 81, no. 2, 2010.

[70] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[71] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *IEEE transactions on neural networks and learning systems*, 2019.

[72] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *ICLR*, 2016.

[73] V. Lebedev and V. Lempitsky, "Fast convnets using group-wise brain damage," in *CVPR*, 2016.

[74] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv:1503.02531*, 2015.

[75] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.

[76] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *NIPS*, 2014.

[77] Y. Peng, Y. Zhu, Y. Chen, Y. Bao, B. Yi, C. Lan, C. Wu, and C. Guo, "A generic communication scheduler for distributed dnn training acceleration," in *SOSP*, 2019.

[78] B. I. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S.-h. Lau, S. Rao, N. Taft, and J. D. Tygar, "ANTIDOTE: Understanding and Defending Against Poisoning of Anomaly Detectors," in *IMC*, 2009.

[79] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating machine learning: Poisoning attacks and countermeasures for regression learning," in *IEEE S&P*, 2018.

[80] N. Baracaldo, B. Chen, H. Ludwig, and J. A. Safavi, "Mitigating poisoning attacks on machine learning models: A data provenance based approach," in *AISec*, 2017.

[81] T. Gu, B. Dolan-Gavitt, and S. Garg, "BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain," *arXiv:1708.06733*, 2017.

[82] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *IEEE S&P*, 2017.

[83] Y. Liu, S. Sun, Z. Ai, S. Zhang, Z. Liu, and H. Yu, "Fedcoin: A peer-to-peer payment system for federated learning," *arXiv:2002.11711*, 2020.

[84] H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Blockchained On-Device Federated Learning," *IEEE Communications Letters*, 2019.

[85] S. Lugan, P. Desbordes, E. Brion, L. X. R. Tormo, A. Legay, and B. Macq, "Secure architectures implementing trusted coalitions for blockchained distributed learning (tclearn)," *IEEE Access*, vol. 7, 2019.

[86] J. Weng, J. Weng, J. Zhang, M. Li, Y. Zhang, and W. Luo, "DeepChain: Auditable and Privacy-Preserving Deep Learning with Blockchain-based Incentive," *IEEE TPDS*, 2019.

[87] V. Buterin, "A next-generation smart contract and decentralized application platform," 2014.

[88] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *IEEE S&P*, 2017.

[89] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Helen: Maliciously secure coopetitive learning for linear models," in *IEEE S&P*, 2019.

[90] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *CRYPTO*, 1992.

APPENDIX A
FEDERATED LEARNING AND DISTRIBUTED SGD

Given a set of training data, a model structure, and a proposed learning task, ML algorithms *train* an optimal set of parameters, resulting in a model that optimally performs this task. In Biscotti, we assume stochastic gradient descent (SGD) [57] as the optimization algorithm.

In federated learning [5], a shared model is updated through SGD. Each client uses their local training data and their latest snapshot of the shared model state to compute the optimal update on the model parameters. The model is then updated and clients update their local snapshot of the shared model before performing a local iteration again. The model parameters w are updated at each iteration i as follows:

$$w_{t+1} = w_t - \eta_t(\lambda w_t + \frac{1}{b} \sum_{(x_i,y_i)\in B_t} \nabla l(w_t, x_i, y_i)) \qquad (1a)$$

where η_t represents a degrading learning rate, λ is a regularization parameter that prevents over-fitting, B_t represents a gradient batch of local training data examples (x_i, y_i) of size b and ∇l represents the gradient of the loss function.

SGD is a general learning algorithm that can be used to train a variety of models, including neural networks [57]. A typical heuristic involves running SGD for a fixed number of iterations or halting when the magnitude of the gradient falls below a threshold. When this occurs, model training is considered complete and the shared model state w_t is returned as the optimal model w^*.

In a multi-party ML setting federated learning assumes that clients possess training data that is not identically and independently distributed (non-IID) across clients. In other words, each client possesses a subset of the global dataset that contains specific properties distinct from the global distribution.

When performing SGD across clients with partitioned data sources, we redefine the SGD update $\Delta_{i,t}w_g$ at iteration t of each client i to be:

$$\Delta_{i,t}w_g = \lambda\eta_t w_g + \frac{\eta_t}{b} \sum_{(x,y)\in B_{i,t}} \nabla l(w_g, x, y) \qquad (1b)$$

where the distinction is that the gradient is computed on a global model w_g, and the gradient steps are taken using a local batch $B_{i,t}$ of data from client i. When all SGD updates are collected, they are averaged and applied to the model, resulting in a new global model. The process then proceeds iteratively until convergence.

To increase privacy guarantees in federated learning, secure aggregation protocols have been added to the central server [26] such that no individual client's SGD update is directly observable by server or other clients. However, this relies on a centralized service to perform such an aggregation and does not provide security against adversarial attacks on ML.

APPENDIX B
DIFFERENTIALLY PRIVATE STOCHASTIC GRADIENT DESCENT

We use the concept of (ϵ, δ) differential privacy as explained in Abadi et al. [22]. Each SGD step becomes (ϵ, δ) differentially private if we sample normally distributed noise as shown in Algorithm 1. Each client commits noise to the genesis block for all expected iterations T. We also requires that the norm of the gradients be clipped to have a maximum norm of 1 so that the noise does not completely obfuscate the gradient.

Data: Batch size b, Learning rate η_t, Privacy parameters (ε, δ), Expected update dimension d
Result: Precommited noise for an SGD update $\zeta_t \ \forall T$
for *iteration* $t \in [1..T]$ **do**
  `// Sample noise of length` d `for each expected sample in batch`
  for *Example* $i \in [1..b]$ **do**
    Sample noise $\zeta_i = \mathcal{N}(0, \sigma^2 I)$ where $\sigma = \sqrt{2log\frac{1.25}{\delta}}/\epsilon$
  end
  $\zeta_t = \frac{\eta_t}{b}\sum_i \zeta_i$
  Commit ζ_t to the genesis block at column t.
end

Algorithm 1: Precommiting differentially Private noise for SGD, taken from Abadi et al. [22].

This precommited noise is designed such that a neutral third party aggregates a client update $\Delta_{i,t}w_g$ from Equation (1b) and precommitted noise ζ_t from Algorithm 1 without any additional information. The noise is generated without any prior knowledge of the SGD update it will be applied to while retaining the computation and guarantees provided by prior work. The noisy SGD update $\widetilde{\Delta_{i,t}w_g}$ follows from aggregation:

$$\widetilde{\Delta_{i,t}w_g} = \Delta_{i,t}w_g + \zeta_t$$

APPENDIX C
POLYNOMIAL COMMITMENTS AND VERIFIABLE SECRET SHARING

Polynomial Commitments [55] is a scheme that allows commitments to a secret polynomial for verifiable secret sharing [43]. This allows the committer to distribute secret shares for a secret polynomial among a set of nodes along with witnesses that prove in zero-knowledge that each secret share belongs to the committed polynomial. The polynomial commitment is constructed as follows:

Given two groups G_1 and G_2 with generators g_1 and g_2 of prime order p such that there exists an asymmetric bilinear pairing $e : G_1 \times G_2 \rightarrow G_T$ for which the t-SDH assumption holds, a commitment public key (PK) is generated such that $PK = \{g, g^{\alpha}, g^{(\alpha)^2}, ..., g^{(\alpha)^t}\} \in G_1^{t+1}$ where α is the secret key. The committer can create a commitment to a polynomial $\phi(x) = \sum_{j=0}^{t} \phi_j x^j$ of degree t using the commitment PK such that:

$$COMM(PK, \phi(x)) = \prod_{j=0}^{deg(\phi)} (g^{\alpha^j})^{\phi_j}$$

Given a polynomial $\phi(x)$ and a commitment $COMM(\phi(x))$, it is trivial to verify whether the commitment was generated using the given polynomial

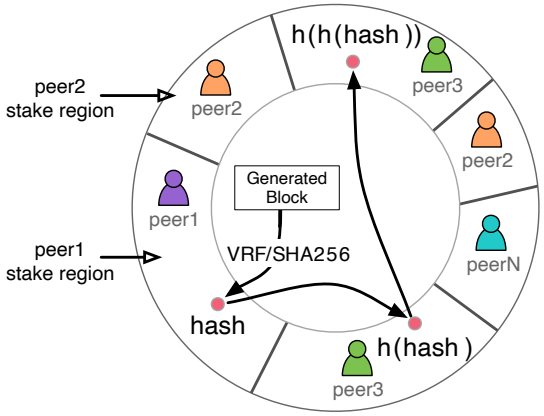

Fig. 13. Biscotti uses a consistent hashing protocol based on the current stake to determine the roles for each iteration.

or not. Moreover, we can multiply two commitments to obtain a commitment to the sum of the polynomials in the commitments by leveraging their homomorphic property:

$$COMM(\phi_1(x) + \phi_2(x)) = COMM(\phi_1(x)) * COMM(\phi_2(x))$$

Once the committer has generated $COMM(\phi(x))$, it can carry out a (n, t) - secret sharing scheme to share the polynomial among a set of n participants in such a way that in the recovery phase a subset of at least t participants can compute the secret polynomial. All secret shares $(i, \phi(i))$ shared with the participants are evaluations of the polynomial at a unique point i and are accompanied by a commitment to a witness polynomial $COMM(\psi_i(x))$ such that $\psi_i(x) = \frac{\phi(x)-\phi(i)}{x-i}$. By leveraging the divisibility property of the two polynomials $\{\phi(x), \psi(x)\}$ and the bilinear pairing function e, it is trivial to verify that the secret share comes from the committed polynomial [55]. This is carried out by evaluating whether the following equality holds:

$$e(COMM(\phi_i(x)), g_2) \stackrel{?}{=} e(COMM(\psi_i(x)), \frac{g_2^{\alpha}}{g_2^{i}}) e(g_1, g_2)^{\Delta w_i(i)}$$

If the above holds, then the share is accepted. Otherwise, the share is rejected.

This commitment scheme is unconditionally binding and computationally hiding given the Discrete Logarithm assumption holds [55]. This scheme can be easily extended to provide unconditional hiding by combining it with another scheme called Pedersen commitments [90] however we do not implement it.

Appendix D
Verifiable Random Functions

A verifiable random function $VRF_{sk}(x)$ is a function that takes in as input a random seed (x) and a secret key (sk). Subsequently, it outputs two values: a hash and a proof. The hash output is a hashlen-bit-long value that is uniquely determined by the sk therefore is unique to a peer. The proof allows anyone with the peer's public key (pk) to check that the hash has indeed been generated by a client who holds the private key. Therefore, it provides each client in the system to deterministically produce a hash that cannot be faked and is unique to the client for that seed value.

In Biscotti, a VRF is used to select a unique committees that provides noise to a peer for protecting its update. A peer uses as inputs to the VRF its own secret key and the SHA-256 hash of the previous block. To select a committee, the hash output of the VRF is input to a consistent hashing procedure. The hash output from the VRF is mapped to a hash ring where each peer is assigned a space proportional to their stake (See Figure 13). The peer in whose portion of the ring the hash lies is selected as part of the committee. The process is repeated until the peer gets a committee of the right size.

Appendix E
Information Leakage Attacks

When doing collaborative learning, each client computes their gradients by back-propagating the loss through the entire network from the last layer to the first layer. The gradient for a layer is computed by using the layer's features and the error from the preceding layer. If the layers are sequential and fully connected, then the output for layer h_{l+1} is computed as follows:

$$h_{l+1} = W_l * hl$$

where W_l is the weight matrix.

Hence, the gradient of the error (E) with respect to W_l is computed as follows:

$$\frac{dE}{dW_l} = \frac{dE}{dh_{l+1}} * h_l$$

Note that the gradient of the weights $\frac{dE}{dW_l}$ is computed by using the inner products from the layer above and also the features of that particular layer. Therefore, the values of the gradients of the first layer will be proportional to the input features. By exploiting this property, observation of gradient updates can be used to infer feature values, which are in turn based on the participants private training data.

In the information leakage attack that we launch on Biscotti, we select the values of the gradient in the first layer that correspond to a certain class, rescale the values to lie between (0,255) and visualize the resulting image.

Appendix F
Proof of Stake

Blockchain based systems need a mechanism to prevent any arbitrary peer from proposing blocks and extending the chain at the same time. Otherwise, anyone can extend the blockchain and nothing would stop the blockchain from developing forks at a rate equal to the number of users and there will be no consensus eventually. A rate limiting solution is needed to prevent the ledger from being extended infinitely by all the peers. Proof of Work (POW) and Proof of Stake (POS) are two popular solutions to this problem.

Proof of Work uses a puzzle to solve the rate limiting problem but it has its limitations. Peers who want to propose the next block have to solve a hard cryptographic puzzle

that is trivial to verify by other peers. The peer that solves the puzzle first gets to propose the next block. The puzzle acts as a rate limiter because by creating new identities (Sybils) peers do not gain any advantage in being the next proposer. In addition to preventing Sybils, it acts as a probabilistic back off mechanism that tries to limit forks in the system. It does not completely eliminate forks and users have to wait for a certain amount of time (6 blocks in Bitcoin) before their transaction is confirmed as accepted. This leads to long wait times (60 minutes for Bitcoin) for transactions to be accepted. Furthermore, it is also energy consuming since it takes a lot of computational power to solve the puzzles. Despite its limitations of long wait times and high energy consumption, POW is a widely used solution.

To mitigate the problems of Proof of Work, Proof of Stake has been recently proposed as an alternate. POS based systems assign the responsibility of proposing blocks each round to specific peers. The probability that a peer will be selected to propose is proportional to the value (stake) that they have in the system. In cryptocurrencies, the stake of a peer is equal to the amount of money that they have in the system. Based on the distribution of stake at that particular time, a predefined algorithm is used to choose a peer/subset of peers that is responsible for proposing the next block. The algorithm ensures that that at any time a group of malicious nodes holding a certain amount of stake in the system cannot act maliciously and take over the system. All other users observe the protocol messages, which allows them to learn the agreed-upon block.

For cryptocurrencies, Algorand [32] is a popular proposal for a proof of stake based system. In Algorand, each peer is assigned a priority for two particular roles: block proposal and block selection. To determine their priority for a particular role, each peer runs a verifiable random function (VRF's) (see D) using their private key, the role (selection/proposal) and a random seed which is public information on the blockchain. The VRF outputs a pseudo-random hash value that is passed by the user through cryptographic sortition to determine their influence in proposing or selecting a block for that particular round. At a high level, the cryptographic sortition is a random algorithm that assigns each user a priority such that the priority assigned is proportional to the user's account balance. If the user's priority is above a threshold, they are selected for that particular role. Subsequently, all users assigned a proposal role, propose a block based on the user's priority. To reach consensus on a single block proposal, peers assigned the selection role agree on one proposal for the next block using a multi-step Byzantine Agreement protocol. In each step, the peers responsible for that step vote for a proposal until in some step enough users have agreed on a proposal. This proposal becomes the next block in the chain.

Similar to Algorand, Biscotti uses verifiable random functions to assign roles to peers in the system. However, instead of using the cryptographic sortition algorithm, Biscotti uses a consistent hashing protocol described in Section 4.3 to select a peer for a role such that the probability of getting selected for a role is proportional to the peer's stake. In Biscotti, we define stake to be the reputation that a peer acquires over the training process by contributing updates.

APPENDIX G
SECURE AGGREGATION PROTOCOL IN BISCOTTI

In Biscotti, an update of length d is encoded as a d-degree polynomial, which can be broken down into n shares such that $(n = 2 * (d+1))$. These n shares are distributed equally among m aggregators. Since an update can be reconstructed using $(d+1)$ shares, it would require $\frac{m}{2}$ colluding aggregators to compromise the privacy of an individual update.

A peer with a verified update already possesses a commitment $C = COMM(\Delta w_i(x))$ to its SGD update signed by a majority of the verifiers from the previous step. To compute and distribute its update shares among the aggregators, peer i runs the following secret sharing procedure:

1) The peer computes the required set of secret shares $s_{m,i} = z, \Delta w_i(z) | z \in Z$ for aggregator m. In order to ensure that an adversary does not provide shares from a poisoned update, the peer computes a set of associated witnesses $wit_{m,i} = \{COMM(\Psi_z(x)) | \Psi_z(x) = \frac{\Delta w(x) - \Delta w(z)}{x-z}\}$. These witnesses will allow the aggregator to verify that the secret share belongs to the update Δw_i committed to in C. It then sends $< C, s_{m,i}, wit_{m,i} >$ to each aggregator along with the signatures obtained in the verification stage.
2) After receiving the above vector from peer i, the aggregator m runs the following sequence of validations:
   a) m ensures that C has passed the validation phase by verifying that it has the signature of the majority in the verification set.
   b) m verifies that in each share $(z, \Delta w_i(z)) \in s_{m,i}$ $\Delta w_i(z)$ is the correct evaluation at z of the polynomial committed to in C. (See Appendix C for details.)

Once every aggregator has received shares for the minimum number of updates u required for a block, each aggregator aggregates its individual shares and shares the aggregate with all of the other aggregators. As soon as a aggregator receives the aggregated $d+1$ shares from at least half of the aggregators, it can compute the aggregate sum of the updates and create the next block. The protocol to recover $\sum_{i=1}^{u} \Delta w_i$ is as follows:

1) All m aggregators broadcast the sum of their accepted shares and witnesses $< \sum_{i=1}^{u} s_{m_i}, \sum_{j=1}^{u} wit_{m_j} >$
2) Each aggregator verifies the aggregated broadcast shares made by each of the other aggregators by checking the consistency of the aggregated shares and witnesses.
3) Given that m obtains the shares from $\frac{m}{2}$ aggregators including itself, m can interpolate the aggregated shares to determine the aggregated secret $\sum_{j=1}^{u} \Delta w_j$

Once m has computed $\sum_{i=1}^{u} \Delta w_i$, it can create a block with the updated global model. All commitments to the updates and the list of signatures that contributed to the aggregate are added to the block. The block is then disseminated in the network. Any peer in the system can observe that all updates were verified by looking at the signature list and homomorphically combining the commitments to check that the update to the global model was computed honestly (see Section 4.2). If any of these conditions are violated, the peer rejects the block.

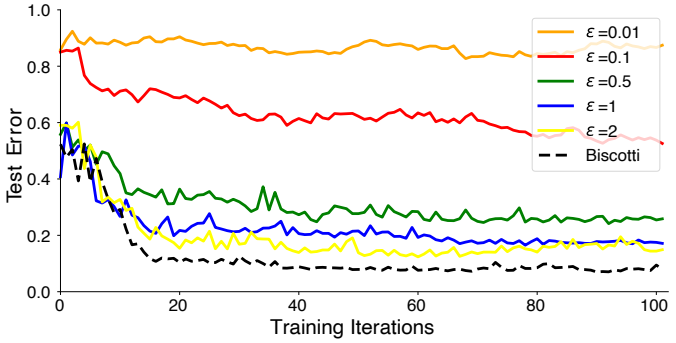

Fig. 14. The utility trade-off if differentially private noise is added directly to the updates in the ledger. Biscotti, which aggregates non-noisy updates, has the best utility (lowest test error).

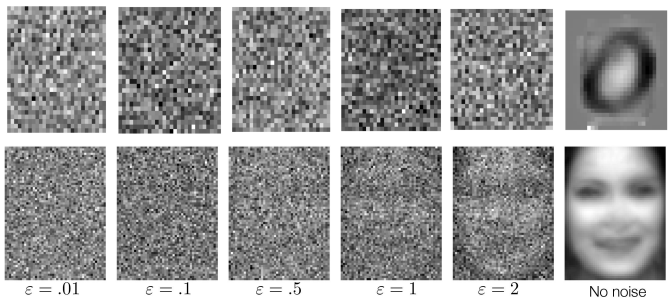

Fig. 15. Visualized privacy trade-off for curves in Figure 14.

Appendix H
Experimentally demonstrating the privacy-utility tradeoff

Figure 14 illustrates the privacy-utility trade-off for learning a softmax model to recognize hand-written digits. The model is trained using the MNIST dataset over 100 iterations for different values of ϵ. Smaller ϵ results in more noise and more privacy, but lower utility, or test error. The bottom-most line in the Figure is Biscotti, which aggregates original updates: a design choice that we have made to prioritize utility. By default Biscotti uses a batch value of 35. Figure 15 illustrates the privacy-utility trade-off visually for batches of size 35 with noisy updates as compared to Biscotti (right-most image), which aggregates un-noised updates. The images are reconstructed using an information leakage attack [13] on the aggregated gradients of two different machine learning models. The top row of images are constructed from aggregated gradients of the MNIST model with respect to the 0 digit class. The bottom row shows results for a softmax model trained to recognize gender from faces using the Labelled Faces in the Wild (LFW) dataset. The pictures are reconstructed with respect to the female class. These results demonstrate that although aggregation protects individual training examples it does reveal information about how a target class appears in aggregate. Differentially-private noise needs to be included in the aggregate to provide this additional level of protection. However, our design favours utility, therefore we choose to remove the noise before aggregating the updates in the ledger.

Appendix I
Privacy evaluation

In this section, we evaluate the privacy provided by secure aggregation in Biscotti. We subject Biscotti to an information leakage attack [13] and demonstrate that the effectiveness of this attack decreases with the number of securely aggregated updates in a block. We also show that the probability of a successful collusion attack to recover an individual client's private gradient decreases as the size of the committees grows.

Information leakage from aggregated gradients. We subject the aggregated gradients from several different datasets to the gradient-based information leakage attack described in [13]. We invert the aggregated gradient knowing that the gradient for the weights associated with each class in the fully connected softmax layer is directly proportional to the input features. To infer the original features, we take the gradients from a single class and invert them with respect to all the classes in the CIFAR-10 (10 classes of objects/animals) and LFW datasets(2 classes male/female). We also invert the gradients for three classes (0,3,5) on the MNIST dataset. We visualize the gradient in grayscale after reshaping to the original feature dimensions in Figure 16. The aggregated gradient will have data sampled from a mixture of classes including the target class. Our results show that having a larger number of participants in the aggregate decreases the impact of this attack. As shown in Figure 16, as the number of aggregates batched together increases, it becomes harder to distinguish the individual training examples.

By default Biscotti aggregates/batches 35 updates. Figure 16 illustrates how the individual class examples from the inverted images are difficult to determine. It might be easy to infer what a class looks like from the inversions. But if the class does not represent an individual's training set, secure aggregation provides privacy. For example, in LFW we get an image that represents what a male/female looks like but we do not gain any information about the individual training examples. Hence, the privacy gained is dependent on how close the training examples are to the class representative.

Recovering a client's individual gradient. We also evaluate a proposed attack on the noising protocol, which aims to de-anonymize peer gradients. This attack is performed when a verifier colludes with several malicious peers. When bootstrapping the system, the malicious peers pre-commit noise that sums to 0. As a result, when a noising committee selects these noise elements for verification, the masked gradient is not actually masked with any ε noise, allowing a malicious peer to perform an information leakage attack on the victim.

We evaluated the effectiveness of this attack by varying the proportion of malicious stake in the system and calculating the probability of the adversary being able to unmask the updates for various sizes of the noising committee. A malicious peer can unmask an update if it controls all the noisers for a peer and has at least one verifier in the verification committee. Figure 17 shows the probability of a privacy violation as the proportion of adversarial stake increases for noising committee sizes of 3, 5 and 10 respectively.

When the number of noisers for an iteration is 3, an adversary needs at least 15% of stake to successfully unmask

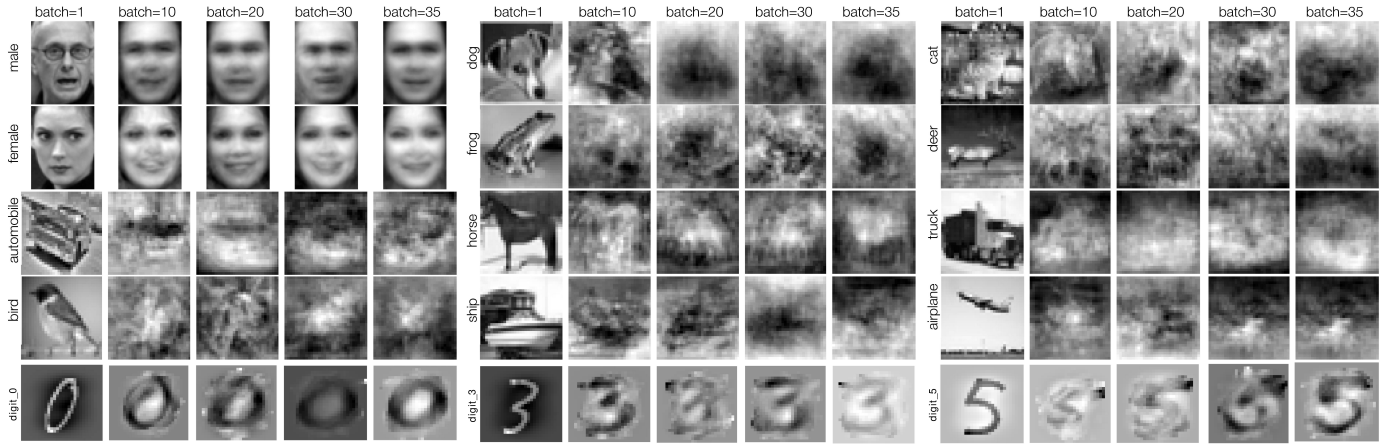

Fig. 16. The results of an information leakage attack on different number of aggregated gradients for different classes.

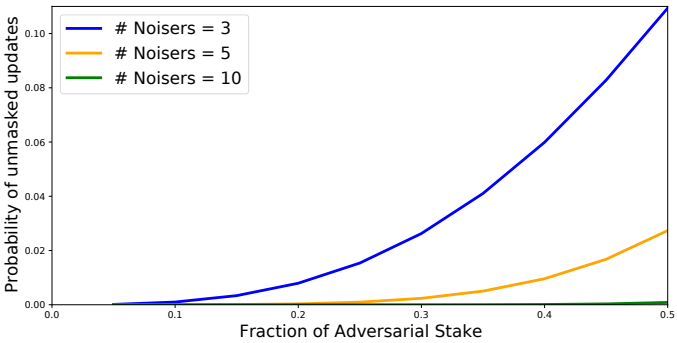

Fig. 17. Probability of a successful collusion attack to recover an individual client's gradient.

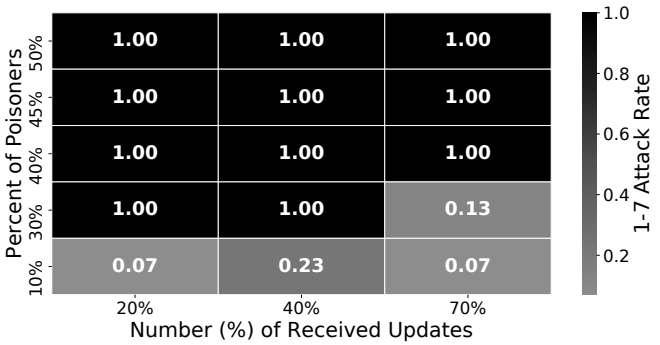

Fig. 18. Evaluating the effect of the number of sampled updates each round on Krum's performance.

an SGD update. This trend continues when 5 noisers are used: over 30% of the stake must be malicious. When the number of noisers is 10 (which has minimal additional overhead according to Figure 9), privacy violations do not occur even with 50% of malicious stake. By using a stake-based consistent hashing to select noising clients, Biscotti prevents adversaries from performing information leakage attacks on other clients unless their proportion of stake in the system is overwhelmingly large.

Appendix J
Hyperparameter effects on Multi-Krum poisoning deterrence

Attack Rate vs Number of Samples. Figure 18 shows that, as the percentage of poisoners in the system increases, a higher fraction of updates need to be collected for Multi-Krum to be effective (achieving a low attack rate). A large sample ensures that the poisoners do not gain majority in the set being fed to Multi-Krum, otherwise Multi-Krum cannot prevent poisoned updates from leaking into the model. The results show that in each round, updates need to be selected from 70% of the peers to prevent poisoning from 30% of the nodes.

Attack Rate vs Noise. To ensure that updates are kept private in the verification stage, differentially private noise is added to each update before it is sent to the verifier.

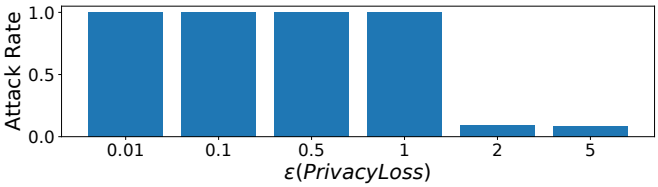

Fig. 19. Multi-Krum's performance in defending against a 30% attack on the MNIST dataset for different settings of ϵ noise.

This noise is parametrized by the ϵ and δ parameters. The δ parameter indicates the probability with which plain ϵ-differential privacy is broken and is ideally set to a value lower than $1/|d|$ where d is the dimension of the dataset. Hence, we set δ to be 10^{-5} in all our experiments. ϵ represents privacy-loss and a lower value of ϵ indicates that more noise is added to the updates. We investigate the effect of the ϵ value on the performance of Multi-Krum with 30% poisoners in a 100-node deployment with 70 received updates in each round on the MNIST dataset. Figure 19 shows that Multi-Krum loses its effectiveness at values of $\epsilon \leq 1$ but performs well on values of $\epsilon \geq 2$.

TABLE 3
Blockchain-based systems designed to support multi-party ML.

System	Layer	Distribution	Anti-poisoning	Smart contract value distribution	Privacy	Consensus	Deployability
FedCoin [83]	L2	Client-server	×	✔	✔ secure agg.	PoSV	✔ 1 server, 100 clients
BlockFL [84]	L2	P2P	×	×	×	PoW	× Simulated
LearningChain [40]	L2	P2P	✔ l-nearest agg	×	✔ diff. priv.	PoW	✔ Ethereum
BEMA [38]	L2	P2P	✔ Krum, MPMC	×	×	PoW	× Simulated
TCLearn [85]	L1	Client-server	✔ Test dataset	×	✔ hom. encryption	BFT	× Simulated
MEC BChain [39]	L1	Client-server and P2P Hybrid	× Auditing	×	✔ diff. priv.	N/A	× Simulated
DeepChain [86]	L1	P2P	× Auditing	✔	✔ hom. encryption	BFT	× Simulated
Biscotti (this paper)	L1	P2P	✔ Krum	×	✔ diff. priv. + secure agg.	PoF	✔ Azure, 200 clients

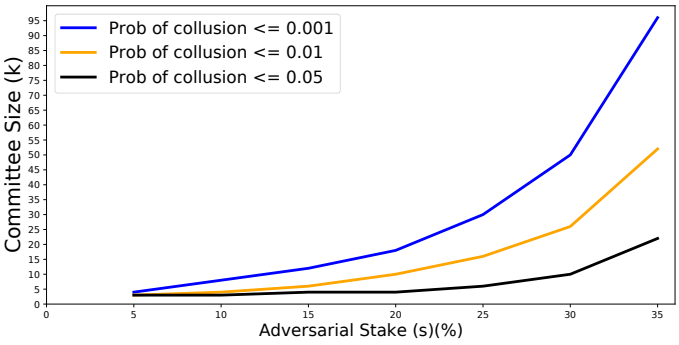

Fig. 20. Size of committee needed such that the probability of an adversary successfully colluding is below a threshold.

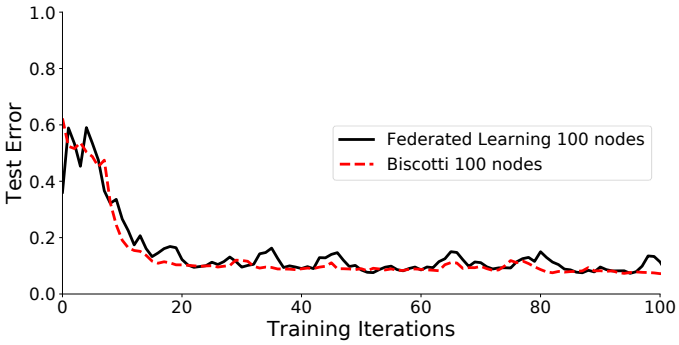

Fig. 21. Federated learning and Biscotti's test error on the MNIST dataset with 600,000 datapoints, across iterations

APPENDIX K
ANALYZING THE MINIMUM COMMITTEE SIZE TO PREVENT COLLUSION

In Biscotti, the verification and aggregation stages involve committees that use a majority voting scheme to reach consensus. By making these committee sizes large enough, we can prevent an adversary controlling a certain fraction of the stake from acting maliciously. An adversary having the majority vote can act maliciously by accepting poisoned updates or recovering an individual peer's updates during aggregation. In this section, we carry out an analysis of the least committee size needed such that the probability of an adversary having the majority is below a threshold.

Using the consistent hashing protocol, the probability of a peer being selected is proportional to their stake. Hence, the probability p of an adversary having the majority in a committee size k can be calculated by:

$$p = \sum_{(i=\frac{k}{2}+1)}^{k} \binom{k}{i} s^i (1-s)^{k-i}$$

where s is the fraction of stake controlled by the adversary.

By assuming that p follows a binomial distribution, we obtain a loose upper bound for an adversary controlling the majority in Biscotti. A binomial distribution assumes sampling peers with replacement allowing a peer to be elected more than once in the committee. Since Biscotti limits a peer to only one vote in the committee, the actual probability of the adversary controlling the majority in Biscotti is less than p.

Since p is an upper bound, we can safely use it to calculate the smallest committee size that bounds p below a threshold (t). To obtain the minimum committee size for p, we use a brute force approach and try out different committee sizes and pick the least size that causes p to fall below the threshold.

Figure 20 plots the minimum committee size needed against adversarial stake for probability thresholds of 0.01, 0.05 and 0.001 respectively. The minimum committee size is independent of the number of nodes and grows exponentially with adversarial stake in the system. Since our experimental evaluation is limited to training for a 100 rounds, the probability threshold of an adversary controlling the majority t needs to be less than 0.01. For this threshold, a committee size of 26 protects against an adversary controlling 30% of the stake in the system.

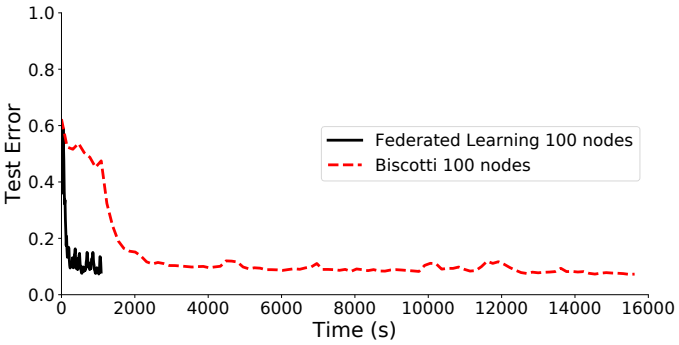

Fig. 22. Federated learning and Biscotti's test error on the MNIST dataset with 600,000 datapoints, across time

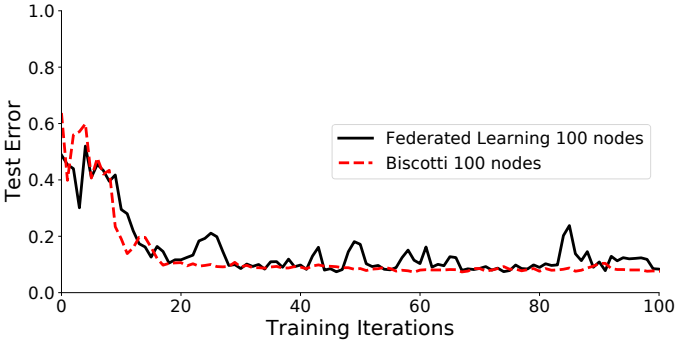

Fig. 23. Federated learning and Biscotti's test error on the MNIST dataset with 6,000,000 datapoints, across iterations

APPENDIX L
EXTRA SCALE EXPERIMENTS

To evaluate Biscotti at greater scales, we performed additional baseline experiments with 200 nodes. Cases with 600,000 training examples (Figures 21 and 22) and 6,000,000 training examples (Figures 23 and 24) are shown. The test error over training iterations (Figures 21 and 23) and seconds (Figures 22 and 24) are shown. In each case, the results are similar to the cases shown in Figures 11 and 10: the performance across iterations is similar, while the time overhead imposed is still 14X.

APPENDIX M
EXISTING BLOCKCHAIN FOR ML SYSTEMS

Table 3 reviews the existing state of the art in blockchain systems designed specifically for ML workloads. The listed systems have been all published in the last two years (2018 and later). Biscotti, the system described in this paper, is listed in the last row. The table compares the systems across seven dimensions:

- **Layer.** is the system a custom blockchain (layer 1) co-designed with the application, or is the system built on top of an existing, generic, blockchain (layer 2)?
- **Distribution.** What is the distribution model in the system? Is the system P2P or does it rely on trusted authorities, like a server?
- **Poisoning defense.** Does the system offer a defense against poisoning attacks?

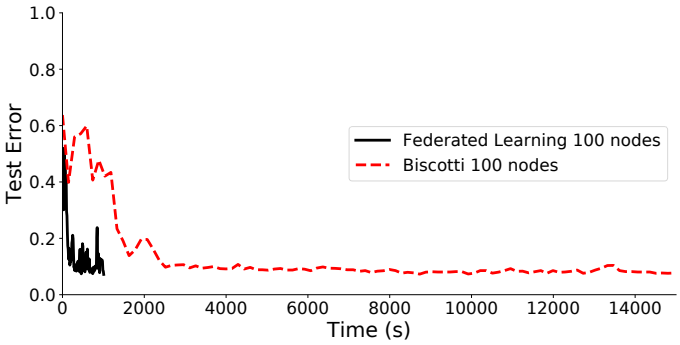

Fig. 24. Federated learning and Biscotti's test error on the MNIST dataset with 600,000 datapoints, across time

- **Smart contract value distribution.** Does the system support smart contracts to reward participants for their contributions to the learning process?
- **Privacy.** Is privacy offered by the system, and how is it provided?
- **Consensus.** What is the consensus protocol used by the blockchain?
- **Deployability.** How deployable is the system. In particular, is the presented artifact evaluated in a simulation or a real deployment?

Each system comes with various design tradeoffs, and may be suitable for different types of ML workloads. Most other solutions are inadequate for our decentralized, adversarial setting: either they do not provide a poisoning defense, they do not provide privacy for its peers, or they rely on trusted authorities and are not a true, decentralized system.

A key Biscotti's feature is its layer-1 design. Within the space of layer-2 solutions, LearningChain [40] also provides all of the above guarantees, but it is deployed as a layer-2 application solution on top of Ethereum. As a result, it's performance is tied to the Ethereum network, and it relies on Ethereum's costly proof-of-work consensus mechanism.

To the best of our knowledge, Biscotti is the only layer-1 solution that is (1) fully decentralized (P2P), (2) addresses both poisoning and privacy attacks against the global model and client datasets, and (3) is deployable in a WAN environment across hundreds of nodes. And, unlike most of the systems in Table 3, Biscotti is open-source: https://github.com/DistributedML/Biscotti