Biscotti: A Blockchain System for Private and Secure Federated Learning

Muhammad Shayan, Clement Fung, Chris J. M. Yoon, and Ivan Beschastnikh

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 percent 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

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

• *The authors are with the University of British Columbia, Vancouver, BC V6T 1Z4, Canada. E-mail: mshayanshafi@gmail.com, clementf@andrew.cmu.edu, yoon@alumni.ubc.ca, bestchai@cs.ubc.ca.*

Manuscript received 1 July 2020; accepted 20 Nov. 2020. Date of publication 11 Dec. 2020; date of current version 11 Feb. 2021.
(Corresponding author: Ivan Beschastnikh.)
Recommended for acceptance by P. Balaji, J. Zhai, and M. Si.
Digital Object Identifier no. 10.1109/TPDS.2020.3044223

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.

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 percent 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 1.875 s 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, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2020.3044223) 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, available in the online supplemental material.

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 percent 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, available in the online supplemental material, 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, available in the online supplemental material, 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, available in the online supplemental material, 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 trade-off experimentally in Appendix H, available in the online supplemental material, and choose to aggregate the un-noised model updates, providing stronger utility guarantees that

match the performance of federated learning. Furthermore, we evaluate the potentital for information leakage attacks in Appendix I, available in the online supplemental material, 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, available in the online supplemental material, 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 percent 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 percent 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. Fig. 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 Fig. 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

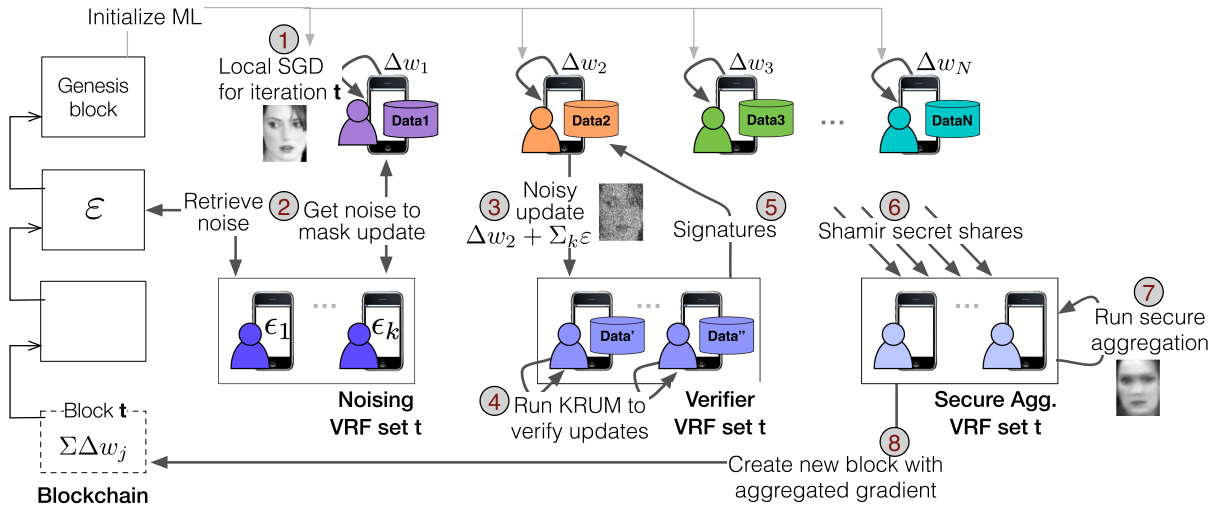

Fig. 1. The ordered steps in a single iteration of the Biscotti algorithm, from step 1 (local SGD) to step 8 (block generation).

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 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, available in the online supplemental material), (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 Fig. 3 and Appendix B, available in the online supplemental material), (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 (Fig. 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 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, available in the online supplemental material, 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.

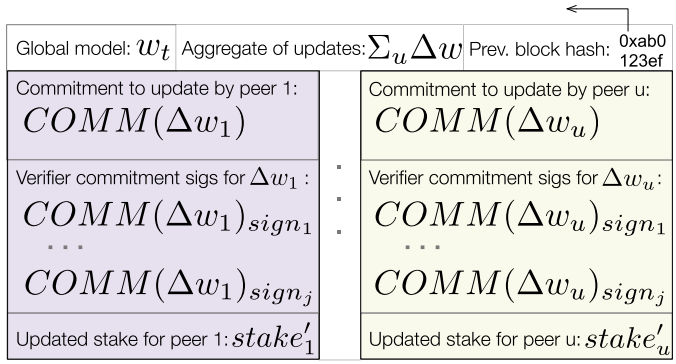

Fig. 2. Block contents at iteration t. Note that w_t is computed using $w_{t-1} + \Sigma w_u$ where w_{t-1} is the global model stored in the block at iteration $t-1$ and j is the set of verifiers for iteration t.

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 rehashed: 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, available in the online supplemental material). 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, available in the online supplemental material, 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.

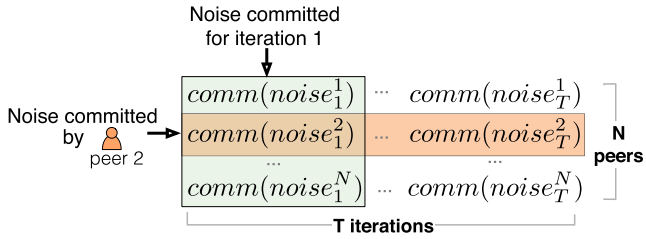

Fig. 3. Peers commit noise to an N by T structure. Each row i contains all the noise committed by a single peer i, and each column t contains potential noise to be used during iteration t. When committing noise at an iteration i, peers execute a VRF and request ζ_i^k from peer k.

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 predetermine 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, available in the online supplemental material, and show that the probability of an information leakage is negligible given a sufficient number of noising peers.

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 (Fig. 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: $\{COMM(\zeta_1), COMM(\zeta_2), \ldots, COMM(\zeta_k))\}$ 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, available in the online supplemental material) 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, available in the online supplemental material) and verifiable secret sharing [43] for aggregation of individual updates. We describe the details of the aggregation protocol in Appendix G, available in the online supplemental material.

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.

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 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

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

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

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 percent, 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 percent malicious nodes compared to a federated learning baseline.

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

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 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. Fig. 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 percent poisoners, Biscotti performs well even against 35 percent poisoners. Beyond 35 percent, 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, available in the online supplemental material.

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

We introduced 30 percent 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 1 as 7 s. Figs. 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

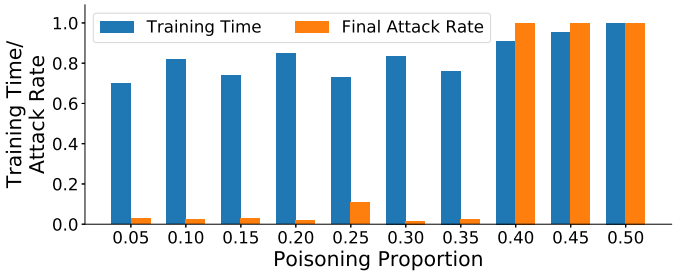

Fig. 4. Biscotti's performance on the MNIST dataset with a varying number of poisoners, from 5 to 50 percent. Both the total execution time and the final attack rate are scaled such that the maximum is 1.

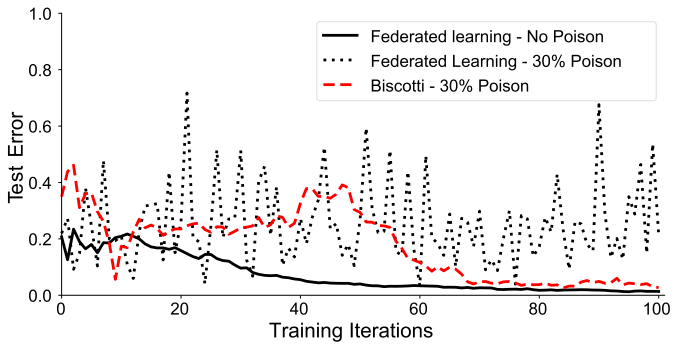

Fig. 5. Federated learning and Biscotti's test error on the CreditCard dataset with 30 percent of poisoners.

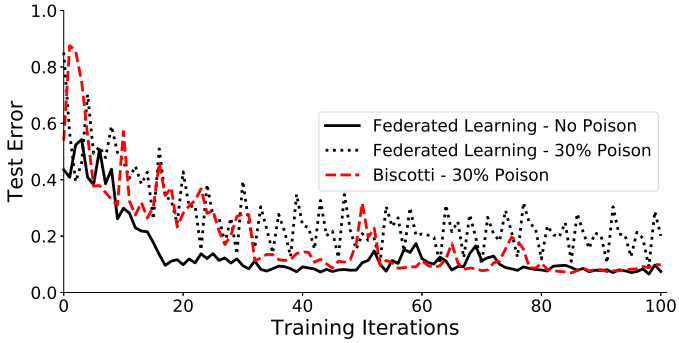

Fig. 6. Federated learning and Biscotti's test error on the MNIST dataset with 30 percent of poisoners.

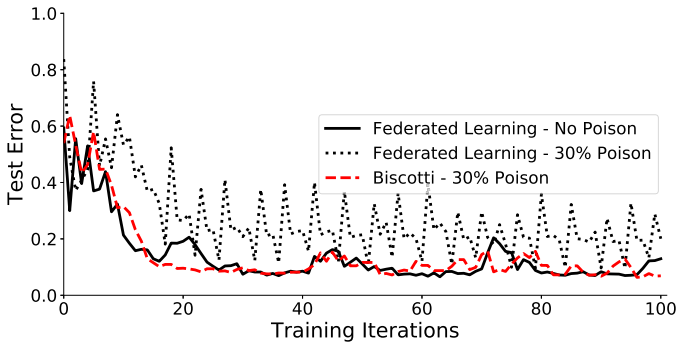

Fig. 7. Federated learning and Biscotti's test error on the MNIST dataset with 200 total nodes, 30 percent poisoners, with a verification and aggregation committee size of 26.

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 percent.

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 percent of the stake in the system (shown in Appendix K, available in the online supplemental material). Since peers in the verification or aggregation committees do not contribute updates in that particular round, 100 nodes cannot generate the 70 percent of updates needed to protect against Multi-Krum. Therefore, for this experiment we increased the number of nodes to 200. The results in Fig. 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 Fig. 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). Fig. 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 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 percent) 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. Fig. 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.

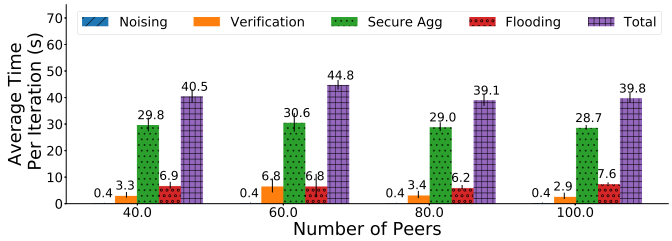

Fig. 8. Breakdown of time spent in different mechanisms in Biscotti (Fig. 1) in deployments with varying number of peers.

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 percent stake from unmasking updates in the noising (Appendix I, available in the online supplemental material) and aggregation stages (Appendix K, available in the online supplemental material). These committee sizes also ensure convergence under poisoning from an attack by 30 percent of the nodes (Section 6.1).

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,

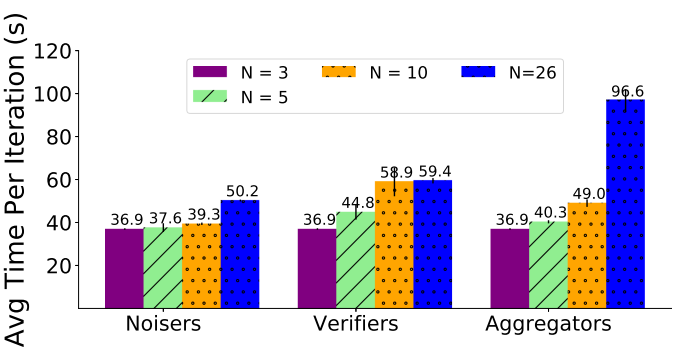

Fig. 9. The average amount of time taken per iteration with an increasing number of noisers, verifiers, or aggregators.

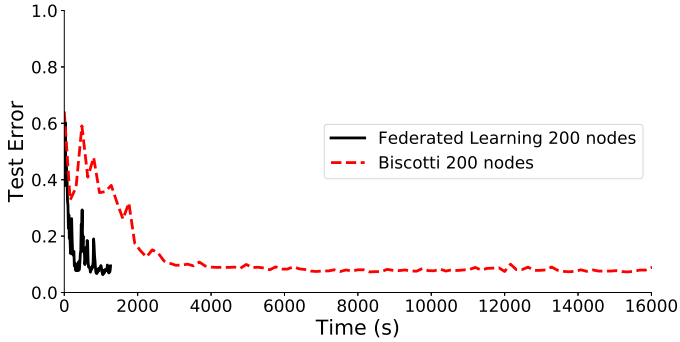

Fig. 10. Comparing the convergence of Biscotti to a federated learning baseline over time, while training an MNIST model over 200 nodes.

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 percent of the nodes selected at random for every round, while in Biscotti we include the first 35 percent verified updates in the block. The convergence for both systems are shown in Figs. 10 and 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 versus 266.7 minutes), yet achieves similar model performance (92 percent accuracy) after the same 100 iterations.

We also evaluated the effect on Biscotti of an increased dataset size and model parameters in Appendix L, available in the online supplemental material. 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

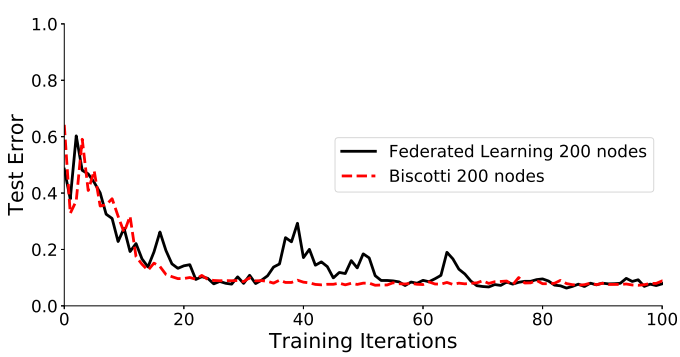

Fig. 11. Comparing the convergence of Biscotti to a federated learning baseline over the number of iterations, while training an MNIST model over 200 nodes.

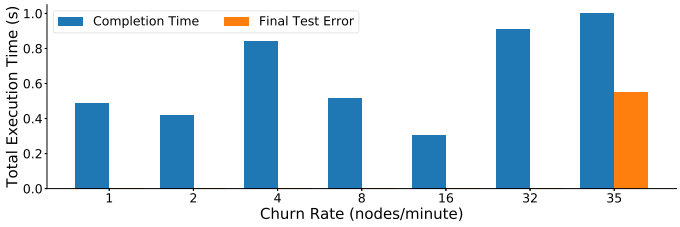

Fig. 12. The impact of churn on model performance. Churn is injected by failing/joining random nodes at a pre-determined rate.

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. Fig. 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, available in the online supplemental material.

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

The authors would like to thank Margo Seltzer for her helpful feedback and comments. They would also 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. They would also like to thank Gleb Naumenko, Nico Ritschel, Jodi Spacek, and Michael Hou for their work on the initial Biscotti prototype. This work has been sponsored by the Huawei Innovation Research Program (HIRP), Project No: HO2018085305. They also acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), 2014-04870.

References

[1] P. Watcharapichat, V. L. Morales, R. C. Fernandez, P. Pietzuch, "Ako: Decentralised deep learning with partial gradient exchange," in *Proc. 7th ACM Symp. Cloud Comput.*, 2016, pp. 84–97.

[2] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 328–339.

[3] K. Hsieh *et al.*, "Gaia: Geo-distributed machine learning approaching LAN speeds," in *Proc. 14th USENIX Conf. Netw. Syst. Des. Implementation*, 2017, pp. 629–647.

[4] J. Xu *et al.*, "Federated learning for healthcare informatics," *J. Healthcare Inform. Res.*, pp. 1–19, 2020.

[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 *Proc. 20th Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.

[6] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4427–4437.

[7] K. Bonawitz *et al.*, "Towards federated learning at scale: System design," in *Proc. Conf. Syst. Mach. Learn.*, 2019.

[8] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, "DÏoT: A federated self-learning anomaly detection system for IoT," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 756–767.

[9] L. Lyu *et al.*, "Towards fair and privacy-preserving federated deep models," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 11, pp. 2524–2541, 2020.

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

[11] C. Fung, C. J. Yoon, and I. Beschastnikh, "The limitations of federated learning in sybil settings," in *Proc. 23rd Int. Symp. Res. Attacks Intrusions Defenses*, 2020, pp. 301–316.

[12] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2020, pp. 2938–2948.

[13] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 691–706.

[14] P. Kairouz *et al.*, "Advances and open problems in federated learning," 2019, *arXiv: 1912.04977*.

[15] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in *Proc. 29th Int. Conf. Mach. Learn.*, 2012, pp. 1467–1474.

[16] B. Nelson *et al.*, "Exploiting machine learning to subvert your spam filter," in *Proc. 1st Usenix Workshop Large-Scale Exploits Emergent Threats*, 2008, Art. no. 7.

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

[18] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to Byzantine-robust federated learning," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 1605–1622.

[19] C. Xie, O. Koyejo, and I. Gupta, "Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 6893–6901.

[20] B. Hitaj, G. Ateniese, and F. Pérez-Cruz, "Deep models under the GAN: Information leakage from collaborative deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 603–618.

[21] S. Shen, S. Tople, and P. Saxena, "Auror: Defending against poisoning attacks in collaborative deep learning systems," in *Proc. 32nd Annu. Conf. Comput. Secur. Appl.*, 2016, pp. 508–519.

[22] M. Abadi *et al.*, "Deep learning with differential privacy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 308–318.

[23] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, pp. 211–407, 2014.

[24] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," in *Proc. NIPS Workshop: Mach. Learn. Phone Consum. Devices*, 2017.

[25] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 1310–1321.

[26] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1175–1191.

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

[28] B. Recht, C. Re, S. Wright, and F. Niu, "HOGWILD!: A lock-free approach to parallelizing stochastic gradient descent," in *Proc. 24th Int. Conf. Neural Inf. Process. Syst.*, 2011, pp. 693–701.

[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 *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5336–5346.

[30] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proc. 3rd Symp. Operating Syst. Des. Implementation*, 1999, pp. 173–186.

[31] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*. Cham, Switzerland: Springer, 2009. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[32] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proc. 26th Symp. Operating Syst. Princ.*, 2017, pp. 51–68.

[33] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "MedRec: Using blockchain for medical data access and permission management," in *Proc. 2nd Int. Conf. Open Big Data*, 2016, pp. 25–30.

[34] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," in *Proc. IEEE Symp. Secur. Privacy*, 2014, pp. 475–490.

[35] N. Narula, W. Vasquez, and M. Virza, "zkLedger: Privacy-preserving auditing for distributed ledgers," in *Proc. 15th USENIX Conf. Netw. Syst. Des. Implementation*, 2018, pp. 65–80.

[36] R. Cheng *et al.*, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2019, pp. 185–200.

[37] C. Xu *et al.*, "Making big data open in edges: A resource-efficient blockchain-based approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 4, pp. 870–882, Apr. 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 Things J.*, vol. 7, no. 10, pp. 9600–9610, Oct. 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," 2019, *arXiv: 1906.10893*.

[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 *Proc. IEEE Int. Conf. Big Data*, 2018, pp. 1178–1187.

[41] S. Micali, S. Vadhan, and M. Rabin, "Verifiable random functions," in *Proc. 40th Annu. Symp. Found. Comput. Sci.*, 1999, Art. no. 120.

[42] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 118–128.

[43] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, pp. 612–613, 1979.

[44] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar, "Adversarial machine learning," in *Proc. 4th ACM Workshop Secur. Artif. Intell.*, 2011, pp. 43–58.

[45] J. J. Douceur, "The sybil attack," in *Proc. Int. Workshop Peer-to-Peer Syst.*, 2002, pp. 251–260.

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

[47] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 634–643.

[48] C. Xie, O. Koyejo, and I. Gupta, "Fall of empires: Breaking Byzantine-tolerant SGD by inner product manipulation," in *Proc. Uncertainty Artif. Intell. Conf.*, 2020, pp. 261–270.

[49] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 5650–5659.

[50] R. Guerraoui and S. Rouault, "The hidden vulnerability of distributed learning in byzantium," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 3521–3530.

[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," in *Proc. Annu. Netw. Distrib. Syst. Security Symp.*, 2019.

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

[53] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," *IEEE Symp. Secur. Privacy*, pp. 739–753, 2019, doi: doi: 10.1109/SP.2019.00065.

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

[55] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2010, pp. 177–194.

[56] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Proc. Annu. Int. Cryptol. Conf.*, 2017, pp. 357–388.

[57] L. Bottou, "Large-Scale machine learning with stochastic gradient descent," in *Proc. Int. Conf. Comput. Statist.*, 2010, pp. 177–186.

[58] L. Muñoz González *et al.*, "Towards poisoning of deep learning algorithms with back-gradient optimization," in *Proc. 10th ACM Workshop Artif. Intell. Secur.*, 2017, pp. 27–38.

[59] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 1322–1333.

[60] J. Dean *et al.*, "Large scale distributed deep networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1223–1231.

[61] A. Paszke *et al.*, "Automatic differentiation in pytorch," 2017.

[62] go-Python, 2018. [Online]. Available: https://github.com/sbinet/go-python

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

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

[65] C. P. Schnorr, "Efficient identification and signatures for smart cards," in *Proc. Workshop Theory Appl. Cryptogr. Techn. Advances Cryptol.*, 1990, pp. 688–689.

[66] G. Maxwell *et al.*, "Simple schnorr multi-signatures with applications to bitcoin," *Des. Codes Cryptography*, vol. 87, pp. 2139–2164, 2019. [Online]. Available: https://doi.org/10.1007/s10623-019-00608-x

[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," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[69] M. Barreno, B. Nelson, A. D. Joseph, and J. D. Tygar, "The security of machine learning," *Mach. Learn.*, vol. 81, no. 2, pp. 121–148, 2010.

[70] J. Konečný *et al.*, "Federated learning: Strategies for improving communication efficiency," 2016, *arXiv:1610.05492*.

[71] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-IID Data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 9, pp. 3400–3413, Sep. 2020.

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

[73] V. Lebedev and V. Lempitsky, "Fast ConvNets using group-wise brain damage," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2554–2564.

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

[75] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv: 1704.04861*.

[76] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 3320–3328.

[77] Y. Peng *et al.*, "A generic communication scheduler for distributed DNN training acceleration," in *Proc. 27th ACM Symp. Operating Syst. Princ.*, 2019, pp. 16–29.

[78] B. I. Rubinstein *et al.*, "ANTIDOTE: Understanding and defending against poisoning of anomaly detectors," in *Proc. 9th ACM SIGCOMM Conf. Internet Meas.*, 2009, pp. 1–14.

[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 *Proc. IEEE Symp. Secur. Privacy*, 2018, pp. 19–35.

[80] N. Baracaldo, B. Chen, H. Ludwig, and J. A. Safavi, "Mitigating poisoning attacks on machine learning models: A data provenance based approach," in *Proc. 10th ACM Workshop Artif. Intell. Secur.*, 2017, pp. 103–110.

[81] T. Gu, B. Dolan-Gavitt, and S. Garg, "BadNets: Identifying vulnerabilities in the machine learning model supply chain," 2017, *arXiv: 1708.06733*.

[82] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 3–18.

[83] Y. Liu, Z. Ai, S. Sun, S. Zhang, Z. Liu, and H. Yu, "FedCoin: A peer-to-peer payment system for federated learning," in *Federated Learning. Lecture Notes in Computer Science*, Q. Yang, L. Fan, H. Yu Eds., Springer, Cham., vol 12500, 2020. [Online]. Available: https://doi.org/10.1007/978-3-030-63076-8_9

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

[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, pp. 181789–181799, 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 Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2019.2952332.
[87] V. Buterin, "A next-generation smart contract and decentralized application platform," White Paper 3.37, 2014.
[88] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 19–38.
[89] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Helen: Maliciously secure coopetitive learning for linear models," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 724–738.
[90] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," *Lecture Notes Comput. Sci.*, vol. 576, pp. 129–140. 1992. [Online]. Available: https://doi.org/10.1007/3-540-46766-1_9

Muhammad Shayan received the MSc degree in computer science from the University of British Columbia, Vancouver, Canada. His research efforts have focused on developing machine learning systems that protect user privacy and guard against malicious actors. For more information please visit: https://m-shayanshafi.github.io

Clement Fung received the MSc degree in computer science from the University of British Columbia, Vancouver, Canada. He is currently working toward the PhD degree with the School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania. His research interests center on the privacy and security of systems that rely on machine learning. For more information please visit: https://clementfung.github.io/.

Chris J. M. Yoon is currently working toward the BSc degree in computer science at the University of British Columbia, Vancouver, Canada. He is interested in computer science research with applications to healthcare and has worked on projects in private and secure multi-party machine learning, and medical image analysis.

Ivan Beschastnikh is an associate professor with the Department of Computer Science, University of British Columbia. He has broad research interests. His recent projects span distributed systems, machine learning, security and privacy, and formal methods. For more information please visit: http://www.cs.ubc.ca/~bestchai/.