# Biscotti - A Ledger for Private and Secure Peer to Peer Machine Learning

by

Muhammad Shayan

BSc, Lahore University of Management Sciences, 2017

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

**Master of Science**

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

December 2019

© Muhammad Shayan, 2019

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, a thesis entitled:

**Biscotti - A Ledger for Private and Secure Peer to Peer Machine Learning**

submitted by **Muhammad Shayan** in partial fulfillment of the requirements

for the degree of **Master of Science**

in **Computer Science**.

**Examining Committee:**

Ivan Beschastnikh, Computer Science
*Supervisor*

Margo Seltzer, Computer Science
*Supervisory Committee Member*

# 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 [12]. 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. [25]

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 protect the privacy of an individual client's update and the performance of the global model at scale when 30% of adversaries are trying to poison the model [25].

# Lay Summary

The amount of data being generated is growing at an unprecedented rate. This data holds insights that humans cannot extract alone. Machine learning is a technique used to build models that capture insights from large amounts of data automatically.

However, current practices for machine learning require data to be centralized in a single location. This centralization might be infeasible because data might be of a private nature.

We design and implement *Biscotti*, a system that enables multiple parties to build machine learning models collaboratively without relying on a centralized entity for data collection or coordinating the model building process.

Biscotti protects the privacy of data providers during model building while ensuring a group of malicious data providers cannot harm the performance of the model. We evaluate Biscotti and show that it outperforms current collaborative machine learning systems in simultaneously protecting against privacy and security threats.

# Preface

This thesis presents original, unpublished work performed by the author in collaboration with Clement Fung, Chris J.M Yoon, Heming Zhang and Matheus Stolet under the supervision of Dr Ivan Beschastnikh in the Networks, Systems and Security lab at the University of British Columbia.

The complete design of the Biscotti system in Chapter 4 is the work of the author in collaboration with Clement and Chris. Chris and Heming assisted with implementing Pytorch models for the system. Chris also implemented the privacy attack and generated Figure 6.6. Clement designed and implemented the stake for role selection protocol described in Chapter 4 . Clement also added the differential privacy implementation to the system as described in Appendix A. Clement also helped with writing an earlier draft of a paper submission based on this thesis. Matheus Stolet helped with evaluating the prototype at scale and generating Figures 6.8, 6.9, 6.10. All other implementation/experiments were conducted by the author.

# Table of Contents

**Abstract** . . . iii

**Lay Summary** . . . iv

**Preface** . . . v

**Table of Contents** . . . vi

**List of Tables** . . . viii

**List of Figures** . . . ix

**Glossary** . . . xi

**Acknowledgements** . . . xii

**1 Introduction** . . . 1

**2 Challenges and Contributions** . . . 4

**3 Assumptions and Threat Model** . . . 8
- 3.1 Design assumptions . . . 8
- 3.2 Attacker assumptions . . . 9

**4 Design** . . . 11
- 4.1 Initializing the training . . . 12
- 4.2 Blockchain design . . . 13

4.3 Using stake for role selection . . . . . . . . . . . . . . . . . . . . 14
4.4 Noising protocol . . . . . . . . . . . . . . . . . . . . . . . . . . 16
4.5 Verification protocol . . . . . . . . . . . . . . . . . . . . . . . . 17
4.6 Aggregation protocol . . . . . . . . . . . . . . . . . . . . . . . . 19
4.7 Blockchain consensus . . . . . . . . . . . . . . . . . . . . . . . . 21

**5 Implementation . . . . . . . . . . . . . . . . . . . . . . . . . . 23**

**6 Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24**
6.1 Minimum committee size to prevent collusion . . . . . . . . . . . 26
6.2 Tolerating poisoning attacks . . . . . . . . . . . . . . . . . . . . 27
6.3 Privacy evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . 33
6.4 Performance, scalability and fault tolerance . . . . . . . . . . . . 35

**7 Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41**

**8 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43**

**9 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45**

**Bibliography . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46**

**Appendix : Supporting Materials . . . . . . . . . . . . . . . . . . . . 52**
A.1 Federated learning and distributed stochastic gradient descent . . . 52
A.2 Differentially private stochastic gradient descent . . . . . . . . . . 53
A.3 Polynomial commitments and verifiable secret sharing . . . . . . 54
A.4 Verifiable random functions . . . . . . . . . . . . . . . . . . . . . 56
A.5 Information leakage attacks . . . . . . . . . . . . . . . . . . . . . 56
A.6 Proof of stake . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57

# List of Tables

Table 6.1 The dataset/model types used in the experiments . . . . . . . . 25
Table 6.2 The hyperparameters used in the experiments . . . . . . . . . . 25
Table 6.3 The default parameters used for all our experiments, unless stated otherwise. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26

# List of Figures


Figure 2.1 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). . . . 6

Figure 2.2 Visualized privacy trade-off for curves in Figure 2.1. . . . . . 6

Figure 4.1 The ordered steps in a single iteration of the Biscotti algorithm. 12

Figure 4.2 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 $t$. . . 13

Figure 4.3 Biscotti uses a consistent hashing protocol based on the current stake to determine the roles for each iteration. . . . . . . . . 15

Figure 4.4 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 $\zeta_i^k$ from peer $k$ . . . . . . . . . . . . . . . . . . . . . 18

Figure 6.1 Size of committee needed such that the probability of an adversary successfully colluding is below a threshold. . . . . . . 28

Figure 6.2 Evaluating the effect of the number of sampled updates each round on KRUM's performance. . . . . . . . . . . . . . . . . . 28

Figure 6.3 KRUM's performance in defending against a 30% attack on the MNIST dataset for different settings of $\varepsilon$ noise. . . . . . . 29

Figure 6.4 Federated learning and Biscotti's test error on the CreditCard dataset with 30% of poisoners. . . . . . . . . . . . . . . . . . 30

Figure 6.5 Federated learning and Biscotti's test error on the MNIST dataset with 30% of poisoners. . . . . . . . . . . . . . . . . . . . . . . 30

Figure 6.6 Attack rate comparison of federated learning and Biscotti's on the MNIST dataset with 30% of poisoners. . . . . . . . . . . 31

Figure 6.7 Federated learning and Biscotti's test error on the MNIST dataset with 30% of poisoners with larger committee size. . . . . . . 31

Figure 6.8 Attack rate comparison of federated learning and Biscotti's on the MNIST dataset with 30% of poisoners with larger committee size. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32

Figure 6.9 The results of an information leakage attack on different number of aggregated gradients for different classes. . . . . . . . . 34

Figure 6.10 Probability of a successful collusion attack to recover an individual client's gradient. . . . . . . . . . . . . . . . . . . . . . 34

Figure 6.11 Breakdown of time spent in different mechanisms in Biscotti (Figure 4.1) in deployments with varying number of peers. . . 36

Figure 6.12 The average amount of time taken per iteration with an increasing number of noisers, verifiers, or aggregators. . . . . . . . . 37

Figure 6.13 Comparing the time to convergence of Biscotti to a federated learning baseline. . . . . . . . . . . . . . . . . . . . . . . . . 38

Figure 6.14 Comparing the iterations to convergence of Biscotti to a federated learning baseline. . . . . . . . . . . . . . . . . . . . . . 38

Figure 6.15 The impact of churn on model performance. . . . . . . . . 39

# Glossary

**DP** Differential Privacy

**FL** Federated Learning

**ML** Machine Learning

**P2P** Peer to Peer

**PK** Public Key

**POS** Proof of Stake

**POW** Proof of Work

**SK** Secret Key

**SGD** Stochastic Gradient Descent

**VRF** Verifiable Random Functions

# Acknowledgements

First and foremost, I would like to thank my supervisor, Ivan Beschastnikh for his fantastic supervision and being a source of constant guidance throughout the project. I would also like to thank Clement Fung for his contributions to the thesis and for mentoring me throughout the process. In addition to this, I would like to thank Chris J.M Yoon, Heming Zhang and Matheus Stolet for their contributions to the project. I would also like to thank Margo Seltzer for being the second reader for my thesis and taking the time out to provide helpful feedback. This thesis would not have been possible without the contributions of all the above people.

In addition, I would like to thank my parents and family who acted as a network of support despite the distance. I would also like to thank all my NSS-lab mates for all their help in navigating my journey through grad school.

Finally, I would like the sponsors, Huawei and NSERC Canada for supporting this research. This research has been sponsored by the Huawei Innovation Research Program (HIRP), Project No: HO2018085305 and the Natural Sciences and Engineering Research Council of Canada (NSERC), 2014-04870.

# Chapter 1

# Introduction

A common thread in machine learning (ML) applications is the collection of massive amounts of training data, which is often centralized for analysis. However, when training ML models in a multi-party setting, users must share their potentially sensitive information with a centralized service.

Federated learning [12, 32] is a prominent solution for high scale secure multi-party ML: clients train a shared model with a secure aggregator without revealing their underlying data or computation. But, doing so introduces a subtle threat: clients, who previously acted as passive data contributors, are now actively involved in the training process [25]. This presents new privacy and security challenges.

Prior work has demonstrated that adversaries can attack the shared model through poisoning attacks [10, 40], in which an adversary contributes adversarial updates to influence 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 client and attempts to steal or de-anonymize sensitive training data through careful observation and isolation of a victim's model updates [24, 33].

Solutions to these two attacks are at tension and are inherently difficult to achieve together: client contributions or data can be made public and verifiable to prevent poisoning, but this violates the privacy guarantees of federated learning. Client contributions can be made more private, but this eliminates the potential for accountability from adversaries. Prior work has attempted to solve these two

attacks individually through centralized anomaly detection [48], differential privacy [4, 20, 22] or secure aggregation [12]. However, a private and decentralized solution that solves both attacks concurrently does not yet exist. In addition, these approaches are inapplicable in contexts where a trusted centralized authority does not exist. This is the focus of our work.

Because ML does not require strong consensus or consistency to converge [43], traditional strong consensus protocols such as Byzantine Fault Tolerant (BFT) protocols [15] are too restrictive for machine learning workloads. To facilitate private, verifiable, crowd-sourced computation, distributed ledgers (blockchains) [38] have emerged. Through design elements, such as publicly verifiable proof of work, eventual consistency, and ledger-based consensus, blockchains have been used for a variety of decentralized multi-party tasks such as currency management [23, 38], archival data storage [5, 35] and financial auditing [39]. Despite this wide range of applications, a fully realized, accessible system for large scale multi-party ML that is robust to both attacks on the global model and attacks on other clients does not exist.

We propose *Biscotti*, a decentralized public peer to peer system that co-designs a privacy-preserving multi-party ML process with a blockchain ledger. Peering clients join Biscotti and contribute to a ledger to train a global model, under the assumption that peers are willing to collaborate on building ML models, but are unwilling to share their data. Each peer has a local data set that could be used to train the model. Biscotti is designed to support stochastic gradient descent (SGD) [14], an optimization algorithm that iteratively selects a batch of training examples, computes their gradients with respect to the current model parameters, and takes steps in the direction that minimizes the loss function. SGD is general purpose and can be used to train a variety of models, including neural networks [17].

The Biscotti blockchain coordinates ML training between the peers. Peers in the system are weighed by the value, or stake, that they have in the system. Inspired by prior work [23], Biscotti uses proof of stake in combination with verifiable random functions (VRFs) [34] to select key roles that help to arrange the privacy and security of peer SGD updates. Our use of stake prevents groups of colluding peers from overtaking the system without a sufficient stake ownership.

*With Biscotti's design our primary contribution is to combine several prior*

*techniques into one coherent system that provides secure and private multi-party machine learning in a highly distributed setting.* In particular, Biscotti prevents peers from poisoning the model through the Multi-KRUM defense [11] and provides privacy through differentially private noise [4] and Shamir secrets for secure aggregation [47]. Most importantly, Biscotti combines these techniques *without* impacting the accuracy of the final model, achieving identical model performance as federated learning.

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 64 minutes, which is 14x slower than a similar federated learning deployment. Biscotti is fault tolerant to node churn every 15 seconds across 100 nodes, and converges even with such churn. Furthermore, we show that Biscotti is resilient to information leakage attacks [33] that require knowledge of a client's SGD update and a label-flipping poisoning attack [25] from prior work.

# Chapter 2

# Challenges and Contributions

We now describe the key challenges in designing a peer to peer (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 proof of stake.**

In a P2P setting adversaries can collude or generate aliases to increase their influence in a sybil attack [19].

Biscotti uses a consistent hashing protocol based on the hash of the last block and verifiable random functions (VRF's) (see Appendix A.4) to select a subset of peers that are responsible for the different stages of the training process: adding noise to updates, validating an update, and securely aggregating the update. To mitigate the effect of sybils, the protocol selects peers proportionally to peer stake. A peer's stake is reputation that the peer acquires by contributing in the system. This ensures that an adversary cannot increase their influence in the system by creating multiple peers without increasing their total contribution/ stake.

**Poisoning attacks: update validation using KRUM.** In multi-party ML, peers possess a relatively disjoint and private sub-set of the training data. As mentioned above, privacy can be exploited by adversaries to provide cover for poisoning attacks.

In multi-party settings, known baseline models are not available to peers, so Biscotti validates an SGD update by evaluating an update with respect to the updates submitted by other peers. Biscotti validates an SGD update using the *Multi-*

*KRUM* algorithm [11], which rejects updates that push the model away from the direction of the majority of the updates. More precisely, in each round, a committee of peers is selected to filter out poisoned updates by a majority vote and each member of the committee uses Multi-KRUM to filter out poisoned SGD updates. Multi-KRUM guarantees to filter out $f$ out of $n$ malicious updates such that $2f+2<n$. We demonstrate that by using Multi-KRUM, Biscotti can handle poisonous updates from up to 30% malicious clients.

**Information leakage attacks: random verifier peers and differentially private updates using pre-committed noise.** By observing a peer's SGD updates from each verification round, an adversary can perform an information leakage attack [33] and recover details about a victim's training data. (For details on information leakage attacks, see Appendix A.5)

Biscotti prevents such attacks during update verification in two ways. First, it uses a consistent hashing on the SHA-256 hash of the last block to ensure that malicious peers cannot deterministically select themselves to verify a victim's gradient. Second, peers send differentially-private updates [4, 20] to verifier peers: before sending a gradient to a verifier, pre-committed $\varepsilon$-differentially private noise is added to the update, masking the peer's gradient in a way that neither the peer nor the attacker can influence or observe(See Figure 2.2). By verifying noised SGD updates, peers in Biscotti can verify the updates of other peers without observing their un-noised versions.

**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.

Verification of differentially private updates discussed above is one piece of the puzzle. The second piece is secure update aggregation: a block in Biscotti does not store updates from individual peers, but rather an aggregate that obfuscates any single peer's contribution during that round of SGD. Biscotti uses a verifiable secret sharing scheme [27] to aggregate the updates so that any individual update remains hidden through cryptographic commitments (For a background, see Appendix A.3).

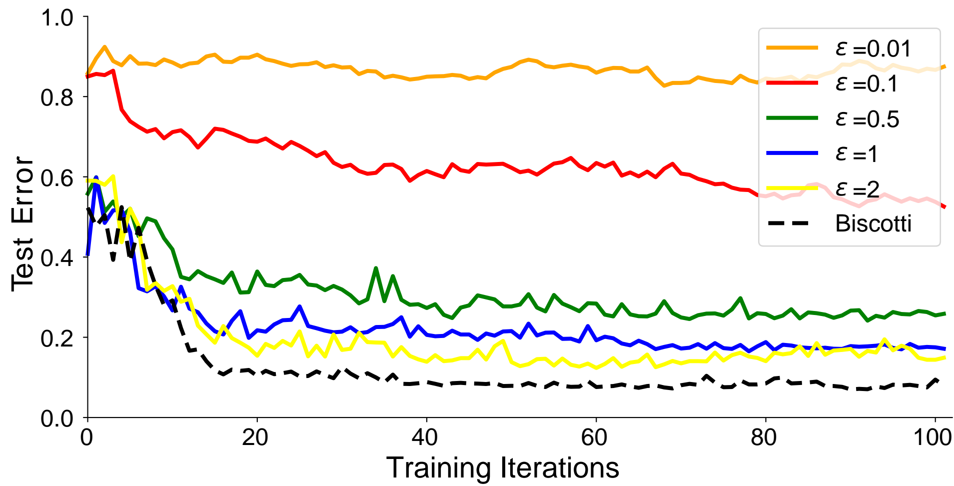


**Figure 2.1:** The utility trade-off if differentially private noise is added directly to the updates in the ledger. Biscotti, which aggregates non-noisy updates, has the best utility (lowest test error).

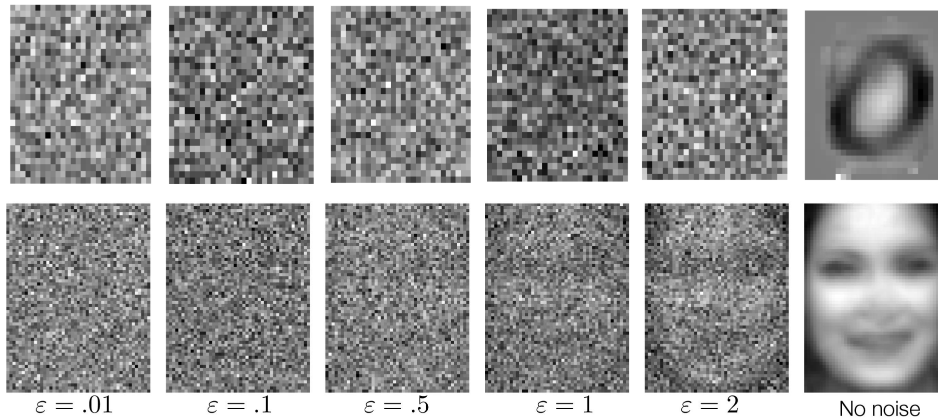


**Figure 2.2:** Visualized privacy trade-off for curves in Figure 2.1.

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 trained model. By aggregating differentially-private updates the noise is pushed into the ledger and the final model has lower utility. By aggregating original updates we can achieve utility that is identical to feder-

ated learning training. Figure 2.1 illustrates this 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 $\varepsilon$. Smaller $\varepsilon$ means 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 2.2 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 [33] 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 certain class looks like on average in the aggregate. Differential 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.

# Chapter 3

# Assumptions and Threat Model

Like federated learning, Biscotti assumes a public machine learning system which peers can join/leave anytime. Biscotti assumes that users are willing to collaborate on building ML models, but are unwilling to directly share their data when doing so [32]. This need for collaboration arises because a model trained only on an individual peer's data would exhibit poor performance, but a model trained by all participants will have near-optimal performance.

## 3.1 Design assumptions

**Proof of stake**.

Peers in Biscotti need to arrive at a consensus on the state of the model for each round of training. Proof of Stake (POS) [23, 28] is a recently proposed consensus mechanism for cryptocurrencies. Consensus using POS is fast because it delegates the consensus responsibility to a subset of peers each round. The peers responsible are selected each round based on the amount of stake that they possess in the system. 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 attempt to subvert the system. For a detailed background on Proof of Stake , please refer to Appendix A.6.

In Biscotti, we define *stake* as a measure of peer's contribution to the system. Peers acquire stake by by providing SGD updates or by facilitating the consensus

process. Thus, the stake that a peer accrues during the training process is proportional to their contribution to the trained model. We assume that all nodes can obtain the fraction of stake of any other peer using a function from the current stake of the blockchain. In addition to this, we also assume that at any point in time more than 70% of the stake in the system is honest and that the stake function 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 on 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 that 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 [38]. Peers that go offline during training and join later are bootstrapped on the current state of the blockchain by other peers in the system.

**Machine learning**. We assume that ML training parameters are known to all peers: the model, its hyperparameters, its optimization algorithm and the learning goal of the system (these are distributed in the first block). 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 an SGD update, we assume that this is done uniformly and independently [13]. This ensures that the Multi-KRUM gradient validation [11] is accurate.

## 3.2 Attacker assumptions

Peers may be adversarial and send malicious updates to perform a poisoning attack on the shared model or an information leakage attack against a target peer victim. In doing so, we assume that the adversary may control multiple peers in a sybil attack [19] but does not control more than 30% of the total peers. Although peers 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 [11].

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 are pushing the global model towards some sub-optimal objective. For the purposes of this work, we limit the adversaries only to a label flipping poisoning attack [25] in which they mislabel a certain class in their dataset causing the model to learn to misclassify it. This does not include attacks on unused parts of the model topology, like backdoor attacks [6], attacks based on gradient-ascent [37] techniques or adaptive attacks based on knowledge of the poisoning defense [9].

When adversaries perform an *information leakage attack*, we assume that they aim to learn properties of a victim's local dataset. Due to the vulnerabilities of secure aggregation, we do not consider information leakage attacks with side information [21, 45] and class-level information leakage attacks on federated learning [24], which attempt to learn the properties of an entire target class.

# Chapter 4

# Design

Biscotti implements peer-to-peer ML trained using SGD. For this process, Biscotti's design has the following goals:

- Convergence to an optimal global model (the model trained without adversaries in a federated learning setting)
- Poisoning is prevented by verifying peer contributions to the model
- Peer training data is kept private and information leakage attacks on training data are prevented
- Colluding peers cannot gain influence without acquiring sufficient stake

Biscotti meets these goals through a blockchain-based design that we describe in this section.

**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 4.1 overviews the Biscotti design with a step-by-step of 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 4.1). Since SGD updates need to be kept private, each peer first *masks* their update using differentially private noise. This noise is obtained from a unique set of noising peers for each client selected by a VRF [34] (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

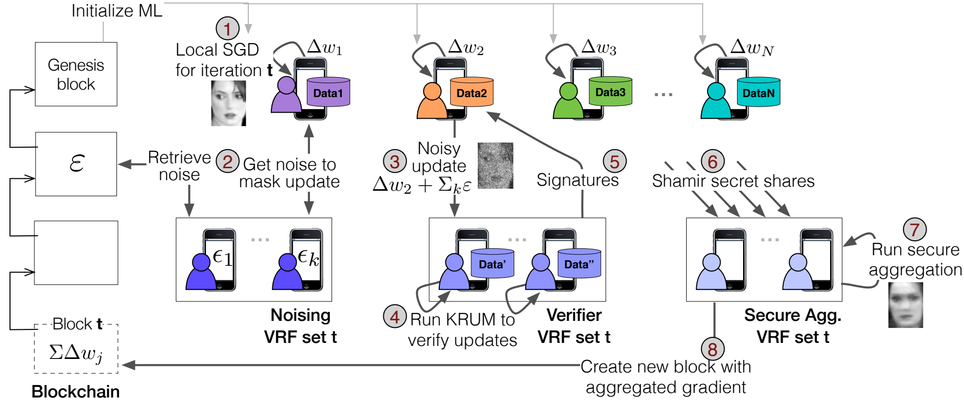


**Figure 4.1:** The ordered steps in a single iteration of the Biscotti algorithm.

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 this block is distributed out of band by a trusted authority and the information within it is reliable. The centralized trusted authority only plays the role of facilitating and bootstrapping the training process by distributing the genesis block that makes public information available to all peers in the system. It is not entrusted with individual SGD updates of the peers that could potentially leak private information of a peer's data. Each peer that joins Biscotti obtains the following information from the genesis block:

| Global model: $w_t$ | Aggregate of updates: $\sum_u \Delta w$ | Prev. block hash: 0xab0 123ef |
|---|---|---|

Commitment to update by peer 1: $COMM(\Delta w_1)$

Verifier commitment sigs for $\Delta w_1$ : $COMM(\Delta w_1)_{sign_1}$ ... $COMM(\Delta w_1)_{sign_j}$

Updated stake for peer 1: $stake'_1$

Commitment to update by peer u: $COMM(\Delta w_u)$

Verifier commitment sigs for $\Delta w_u$: $COMM(\Delta w_u)_{sign_1}$ ... $COMM(\Delta w_u)_{sign_j}$

Updated stake for peer u: $stake'_u$

**Figure 4.2:** 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 $t$.

- Initial model state $w_0$, and expected number of iterations $T$
- The commitment public key $PK$ for creating commitments to SGD updates (see Appendix A.3)
- The public keys $PK_i$ of all other peers in the system that are used to create and verify signatures during verification.
- Pre-commitments to $T$ iterations of differentially private noise $\zeta_{1..T}$ for each SGD iteration by each peer (see Figure 4.4 and Appendix A.2)
- The initial stake distribution among the peers
- A stake update function for updating a peer's stake when a new block is appended

## 4.2 Blockchain design

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

Each block in Biscotti (Figure 4.2) contains, in addition to the previous block

hash pointer, an aggregate ($\Delta w$) 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 made part of the block. However, storing them would leak information about private training data of the individuals. We solve this problem by using polynomial commitments. [27]. Polynomial commitments take an SGD update and map it to a point on an elliptic curve. (see Appendix A.3 for details). By including a list of commitments for each peer $i$'s update $COMM\ (\Delta w_i)$ 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:

$$COMM(\sum \Delta w_i) = \prod_i COMM(\Delta w_i) \tag{2}$$

The training process continues for a specified number of iterations $T$ upon which the learning process is terminated and each peer extracts the global model from the final block.

## 4.3 Using stake for role selection

For each iteration in Biscotti, a consistent hashing protocol weighted by stake designates roles (noiser, verifier, aggregator) to some peers in the system. The protocol ensures that the influence of a peer is bounded by their stake (i.e. adversaries cannot trivially increase their influence through sybils). Peers can take on 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.

Biscotti uses consistent hashing to select peers for the verification and aggregation committees (Figure 4.3). 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 peer

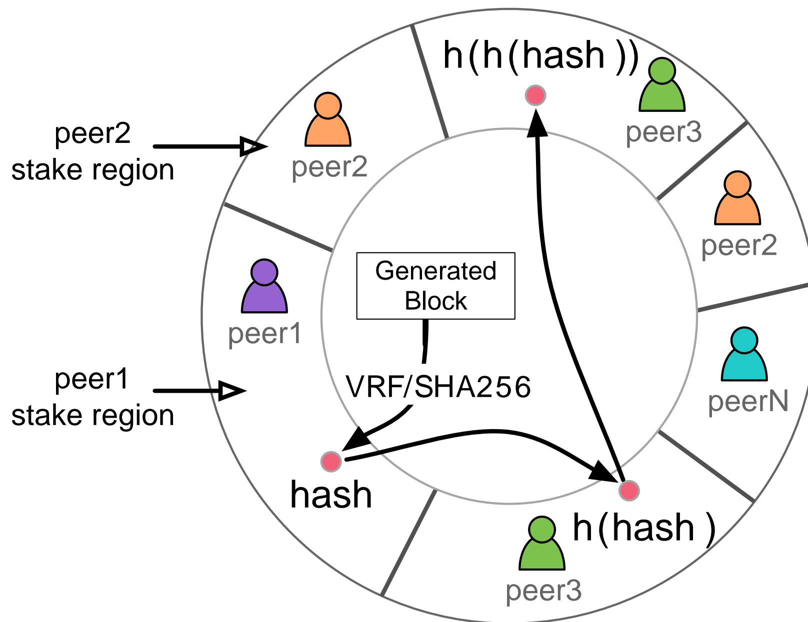


**Figure 4.3:** Biscotti uses a consistent hashing protocol based on the current stake to determine the roles for each iteration.

in whose space the hash lies is chosen for the verifier/aggregator role. The process is repeated until you get verifier/aggregator committees of the right size. This provides the same stake-based property as Algorand [23]: a peer's probability of winning this 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 passing their secret key $SK_i$ and the SHA-256 hash of the last block to a verifiable random function (VRF). 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 peers public key allowing others peer to determine that the correct noise providers

are 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 set of noise providers. The contributing peers then obtain noise to hide their updates by following the noising protocol.

## 4.4 Noising protocol

Sending a peer's SGD update directly to another peer may leak sensitive information about their private dataset [33]. To prevent this, peers use differential privacy to hide their updates prior to verification by adding noise sampled from a normal distribution. This ensures that each step is $(\varepsilon, \delta)$ differentially private by standard arguments in [20]. (see Appendix A.2 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 $\Delta w_{poison}$, and add noise $\zeta_p$ that *unpoisons* this update to resemble an honest update $\Delta w$, such that $\Delta w_{poison} + \zeta_p = \Delta w$. By doing this, a verifier observes the honest update $\Delta w$, but the poisonous update $\Delta w_{poison}$ is applied to the model because the noise is removed in the final aggregate.

To prevent this, Biscotti requires that every peer pre-commits the noise vector $\zeta_t$ for every iteration $t \in [1..T]$ in the genesis block. Since the updates cannot be generated in advance without the knowledge of the global model, a peer cannot effectively commit noise that unpoisons an update. Furthermore, Biscotti requires that the noise that is added is taken from a *different* peer than the one creating the update. This peer is determined using a noising VRF and further restricts the control that a malicious peer has over the noise used to sneak poisoned updates past verification.

**Using a VRF-chosen set of noisers to thwart information leakage.** A further issue may occur in which a noising peer $A$ and a verifier $B$ can collude in an information leakage attack 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 $B$, $B$ performs an information leakage attack on client $C$'s gradient back to its original training data. This attack is viable because the verifier $B$ knows that $A$ is providing the random noise and $A$ provides zero noise.

This attack motivates Biscotti's use of a private VRF that selects 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. Our results in Figure 6.10 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 $\zeta_t$ and commits these noise vectors into the ledger, storing a table of size $N$ by $T$ (Figure 4.4). 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 $\zeta_k$ 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

The verifier peers are responsible for filtering out malicious updates in a round by running Multi-KRUM on the received pool of updates and accepting the top majority of the updates received each round. Each verifier receives the following from each peer $i$:

- The masked SGD update: $(\Delta w_i + \sum_k \zeta_k)$
- Commitment to the SGD update: $COMM(\Delta w_i)$
- The set of $k$ noise commitments:

$\{COMM(\zeta_1), COMM(\zeta_2), ..., COMM(\zeta_k))\}$

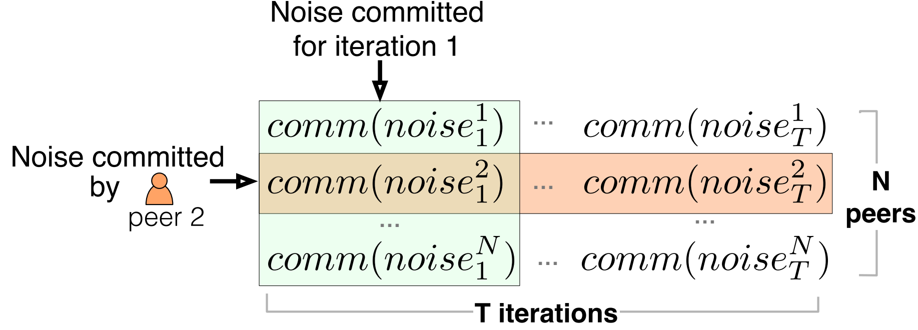


**Figure 4.4:** Peers commit noise to an $N$ by $T$ structure. Each row $i$ contains all the noise committed by a single peer $i$, and each column $t$ contains potential noise to be used during iteration $t$. When committing noise at an iteration $i$, peers execute a VRF and request $\zeta_i^k$ from peer $k$

- 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 [27]. 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:

- 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 $(\Delta w_i + \sum_{i,k} \zeta_{i,k})$.

- The $R-f$ peers with the lowest scores are selected while the rest are rejected.
- The verifier signs the $COMM(\Delta w_i)$ using its public key for all the updates that are accepted.

To prevent a malicious sybil verifier from accepting all updates of its colluders in this stage, we require a peer to obtain signatures from majority of the verifiers to get their update accepted. Once a peer receives a majority number of signatures from the verifiers, the update can be disseminated for aggregation.

## 4.6 Aggregation protocol

All peers with a sufficient number of signatures in the verification stage submit their SGD updates for aggregation which will be eventually appended to the global model. The update equation in SGD (see Appendix A.1) can be re-written as:

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

where $\Delta w_i$ is the verified SGD update of peer $i$ and $w_t$ is the global model at iteration $t$.

However, individual updates contain sensitive information and cannot be directly shared for aggregation. This presents a privacy dilemma: no peer should observe an update from any other peer, but the sum of the updates must be stored in a block.

The objective of the aggregation protocol is to enable 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 (see Appendix A.3) combined with verifiable secret sharing [47] of individual updates. The 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. Therefore, given that any majority of aggregators is honest and does not collude, the privacy of an individual peer update is preserved.

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 $s_{m,i} = \{z, \Delta w_i(z) | z \in \mathbb{Z}\}$ 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 $\Delta w_i$ 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 $C$. (For details, see Appendix A.3)

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

## 4.7 Blockchain consensus

Because consistent hashing based subsets are globally observable by each peer, and based only 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 [38].

When a peer observes a more recent ledger state through the gossip protocol, it can catch up by verifying that the computation performed is correct by running the consistent hashing protocol for the ledger state and by 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 [43]. However, these stale updates could be leveraged in future iterations if their learning rate is decayed [17]. We leave this for future work.

# Chapter 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[1]. We use Go to handle all networking and distributed systems aspects of our design. We used PyTorch 0.4.1 [41] 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* [3] library to interface between Python and Go. Since go-python does not support Python >= 2.7.12 therefore we were limited to using Python 2.7 for our implementation.

We use the *kyber v.2* [2] and *CONIKS 0.1.1* [1] 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 [46] scheme instead of ECDSA because multiple verifier Schnorr signatures can be aggregated together into one signature [31]. Therefore, our block size remains constant as the verifier set grows.

[1] https://github.com/DistributedML/Biscotti

# Chapter 6

# Evaluation

We had several goals when designing 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 to train an ML model 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. We measured the error of the global model against a test set and ran each experiment for 100 iterations. To evaluate Biscotti's defense mechanisms, we ran known inference and poisoning attacks on federated learning [25, 33] 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 our system and varying the committee sizes with different numbers of peers. For all our experiments, we deployed Biscotti with the parameter values in Table 6.3 unless stated otherwise.

We evaluated Biscotti with logistic regression and softmax classifiers on the Credit Card and MNIST dataset respectively. The softmax classifier contains a one layer neural network with a binary cross entropy loss at the end. However, due to the general-purpose PyTorch API, we claim that Biscotti can generalize to models of arbitrary size and complexity, as long as they can be optimized with

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

**Table 6.1:** The dataset/model types used in the experiments

| Model/Dataset | Batch Size | Learning Rate | Momentum |
|---|---|---|---|
| **Credit Card/LogReg** | 10 | 0.01 | 0 |
| **MNIST/Softmax** | 10 | 0.001 | 0.75 |

**Table 6.2:** The hyperparameters used in the experiments

SGD and can be stored in our block structure. We evaluate logistic regression with a credit card fraud dataset from the UC Irvine data repository [18], which uses an individual's financial and personal information to predict whether or not they will default on their next credit card payment. We evaluate softmax with the canonical MNIST [30] dataset, a task that involves predicting a digit based on its image.

The MNIST and Credit Card datasets have 60,000 and 21000 training examples respectively. We used 5-fold cross validation on the training set to decide on hyperparameters of batch size, momentum and learning rate indicated in Table 6.2. The MNIST/Credit Card models were tested using a separately held-out test set of 10,000 and 9000 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. The size of dataset/training examples per peer is small for our experimental setting and might not be representative of large-scale datasets.

In summary, Table 6.1 and 6.2 show the datasets, types of model, number of training examples, the number of parameters $d$ in the models and the hyperparameters that were used in our experiments.

In local training we were able to train a model on the Credit Card and MNIST datasets with accuracy of 98% and 92%, respectively.

| Parameter | Default Value |
|---|---|
| Privacy budget ($\varepsilon$) | 2 |
| Delta ($\delta$) | $10^{-5}$ |
| 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 6.3:** The default parameters used for all our experiments, unless stated otherwise.

## 6.1 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 6.1 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.

## 6.2 Tolerating poisoning attacks

In this section, we evaluate Biscotti's performance when we subject the system to a label flipping poisoning attack as in Huang et.al [25]. We investigate the different parameter settings required for Biscotti to successfully defend against a poisoning attack and then evaluate how well it performs under attack from 30% malicious nodes compared to a Federated Learning baseline.

**Attack Rate vs Received Updates.** Biscotti requires each peer in the verification committee to collect a sufficient sample of updates before running Multi-KRUM. We evaluated the effect of collecting varying percentages of total updates in each round on the effectiveness of Multi-KRUM with different poisoning rates for the MNIST dataset. To ensure uniformity and to eliminate latency effects in the collection of updates, in these experiments the verifiers waited for updates from all peers that are not assigned to a committee and then randomly sampled a specified number of 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.

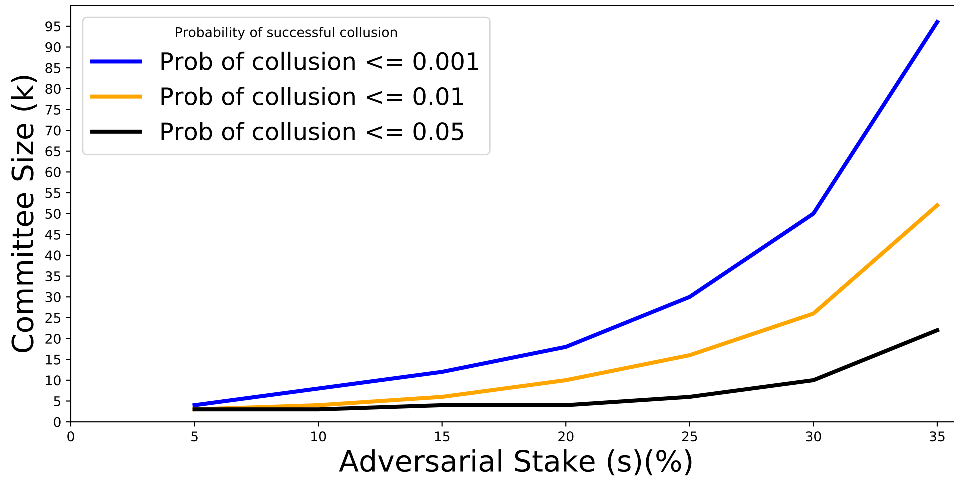


**Figure 6.1:** Size of committee needed such that the probability of an adversary successfully colluding is below a threshold.

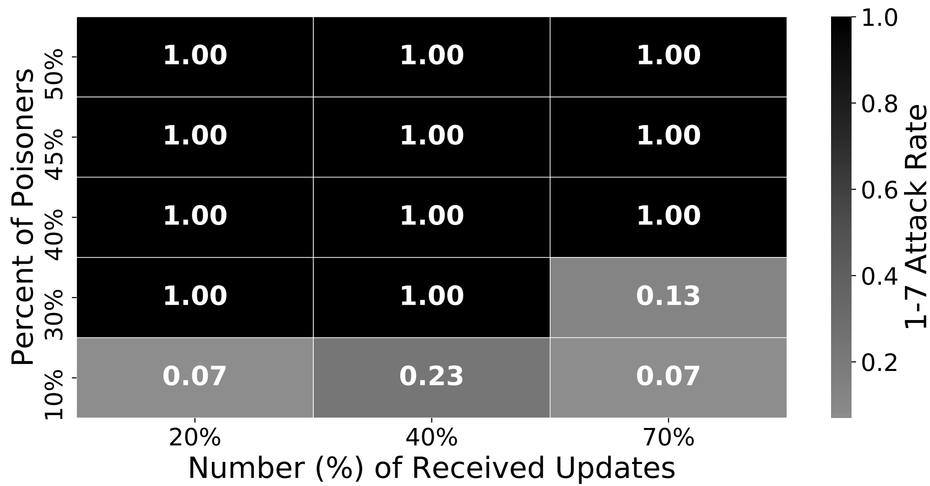


**Figure 6.2:** Evaluating the effect of the number of sampled updates each round on KRUM's performance.

This allows us to investigate how well Multi-KRUM performs in a decentralized ML setting like Biscotti unlike the original Multi-KRUM paper [11] 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

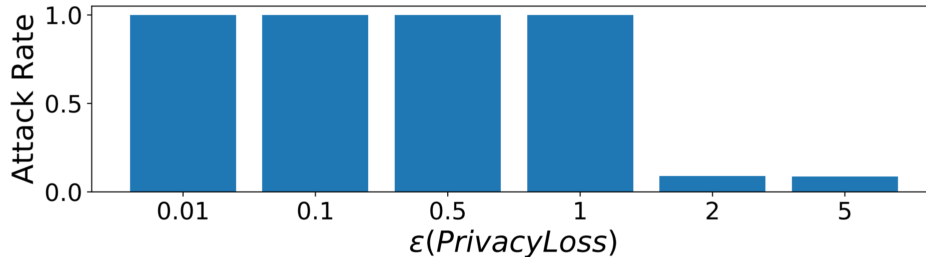


**Figure 6.3:** KRUM's performance in defending against a 30% attack on the MNIST dataset for different settings of $\varepsilon$ noise.

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 6.2 show the results. As the percentage of poisoners in the system increases, a higher fraction of updates need to be collected for Multi-KRUM to be effective (to achieve 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 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. This noise is parametrized by the $\varepsilon$ and $\delta$ parameters. The $\delta$ parameter indicates the probability with which plain $\varepsilon$-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 $\delta$ to be $10^{-5}$ in all our experiments. $\varepsilon$ represents privacy-loss and a lower value of $\varepsilon$ indicates that more noise is added to the updates. We investigate the effect of the $\varepsilon$ 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 6.3 shows that KRUM loses its effectiveness at values of $\varepsilon \leq 1$ but performs well on values of $\varepsilon \geq 2$.

**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 exper-

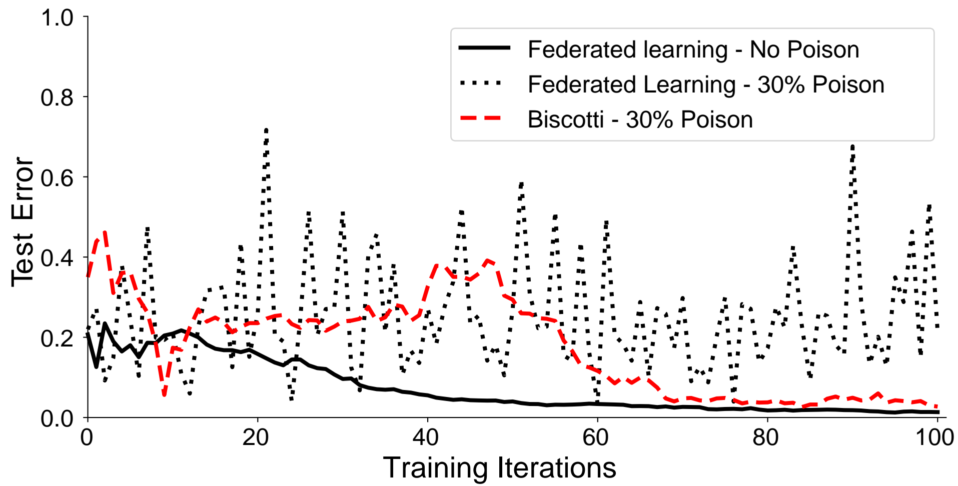


**Figure 6.4:** Federated learning and Biscotti's test error on the CreditCard dataset with 30% of poisoners.

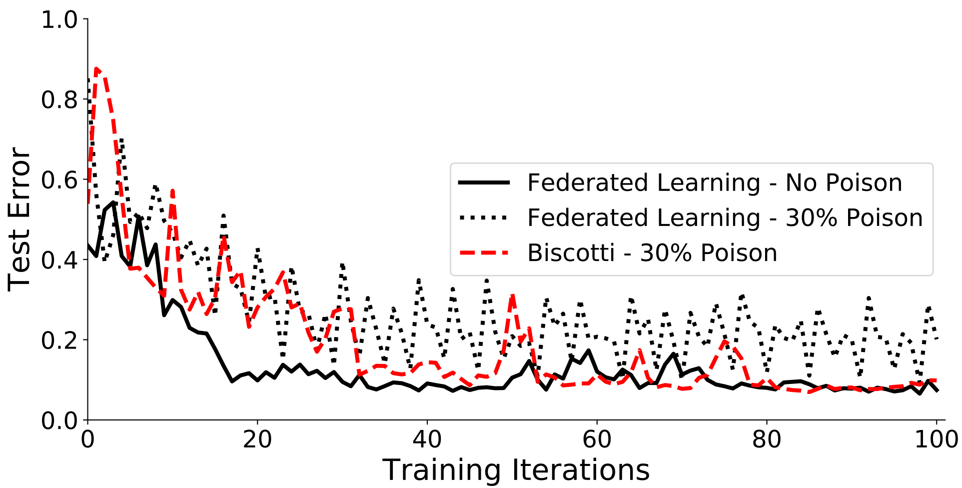


**Figure 6.5:** Federated learning and Biscotti's test error on the MNIST dataset with 30% of poisoners.

iments, Biscotti sampled 70% of updates and used a value of 2 for the privacy budget $\varepsilon$.

We introduced 30% of the peers into the system with the same malicious dataset: for credit card they labeled all defaults as non-defaults, and for MNIST

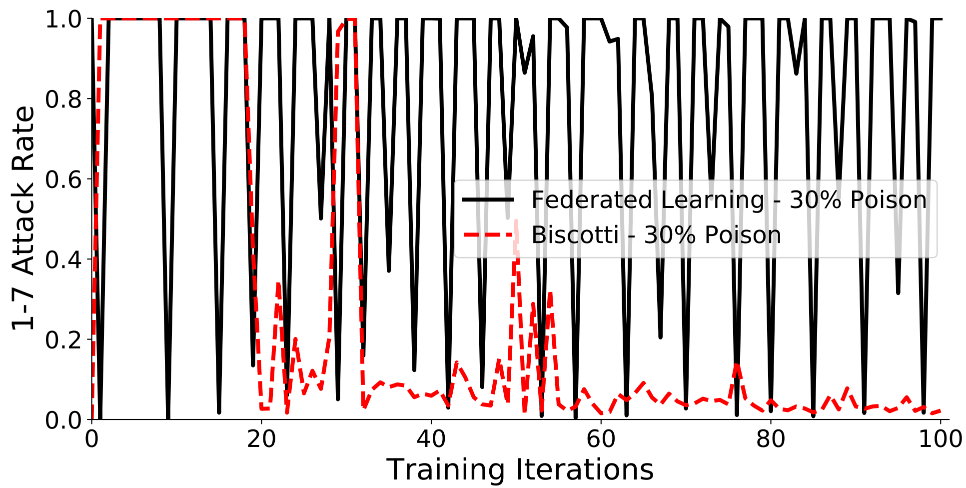


**Figure 6.6:** Attack rate comparison of federated learning and Biscotti's on the MNIST dataset with 30% of poisoners.

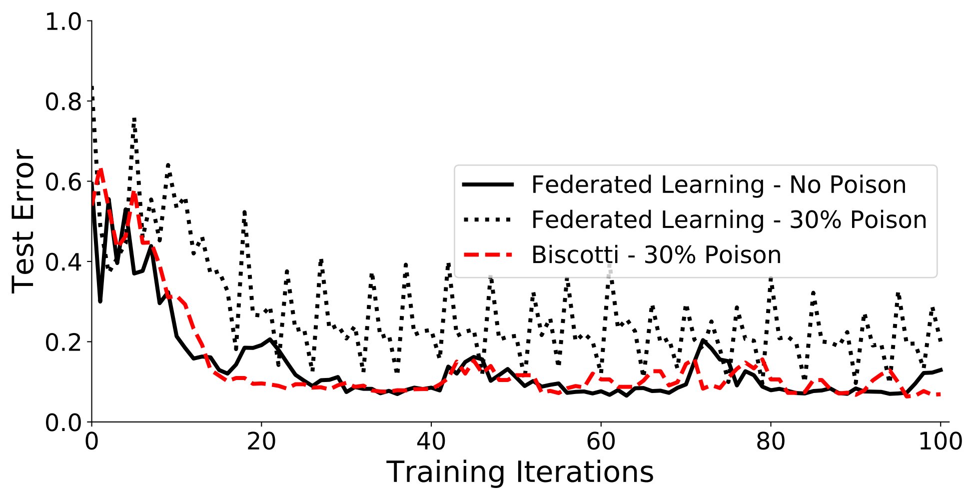


**Figure 6.7:** Federated learning and Biscotti's test error on the MNIST dataset with 30% of poisoners with larger committee size.

these peers labeled all 1s as 7s. Figure 6.4 and 6.5 shows the test error for both datasets as compared to federated learning. The results show that for both datasets the poisoned federated learning deployment struggled to converge. By contrast, Biscotti performed as well as the baseline un-poisoned federated learning deploy-

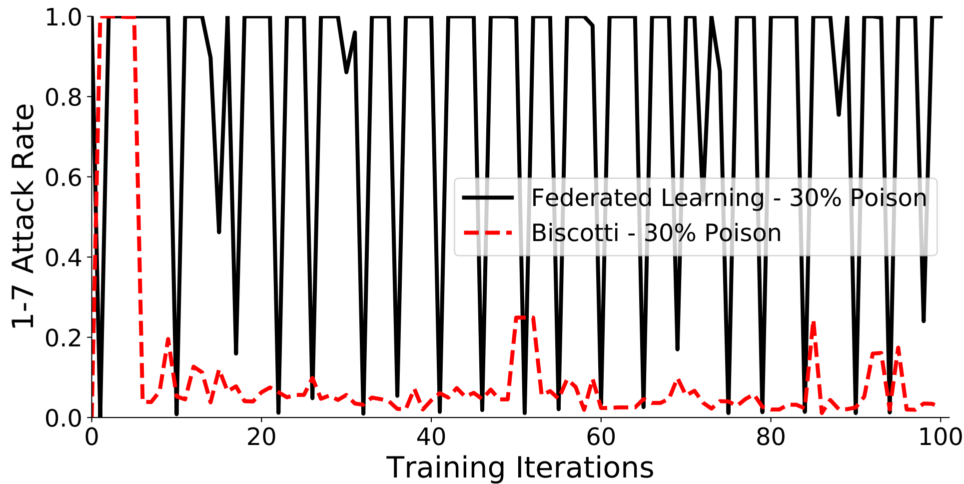


**Figure 6.8:** Attack rate comparison of federated learning and Biscotti's on the MNIST dataset with 30% of poisoners with larger committee size.

ment on the test set.

We show the attack rate in Biscotti and Federated Learning for MNIST in Figure 6.6. For the credit card dataset, the attack rate is the same as the test error since there is only one class in the dataset. The figure illustrates that in federated learning, since no defense mechanisms exist, a poisoning attack creates a tug of war between honest and malicious peers. On the other hand, in Biscotti the attack rate settles after some initial jostling.

The jostling results in Biscotti taking 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 for this experiment. A smaller committee size allowed the poisoning peers to control a majority in the committee frequently and accept updates from fellow malicious peers. Biscotti finally converged because the honest peers gained influence over time thereby reducing the chance of malicious peers from taking control of the majority vote each round. In this experiment, the stake of the honest clients went from 70% to 87%.

To demonstrate the effect of the committee size on convergence under attack, we run the same experiment for the MNIST dataset with an increased committee size of 26 peers. As shown in Section 6.1, a committee size of 26 provides protec-

tion against an adversary controlling 30% of the stake in the system. Since peers part of committees do not contribute updates in that particular round, 100 nodes is not enough to collect 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 6.8 shows that Biscotti converges in the same number of iterations as unpoisoned Federated Learning with an increased committee size. Figure 6.8 shows that the attack rate stays less than 24.9% after the first 5 iteration because the poisoners were unable to get their updates accepted during the training process.

## 6.3 Privacy evaluation

In this section, we evaluate the privacy provided by secure aggregation in Biscotti. We subject Biscotti to an information leakage attack [33] 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 [33]. 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 6.9. 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 6.9, 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 6.9 illustrates how the individual class examples from the inverted images are difficult to determine.

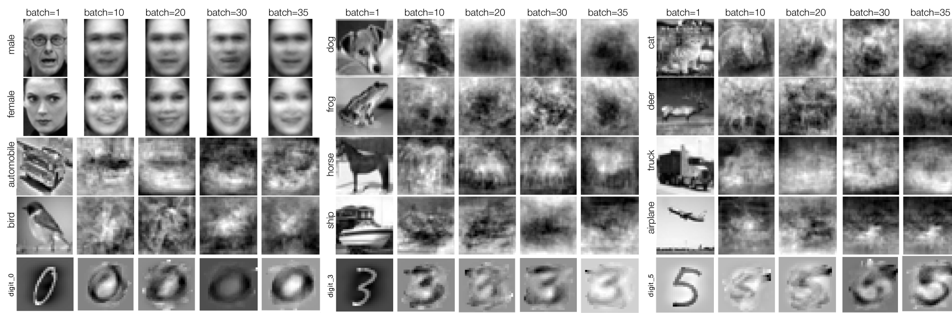


**Figure 6.9:** The results of an information leakage attack on different number of aggregated gradients for different classes.

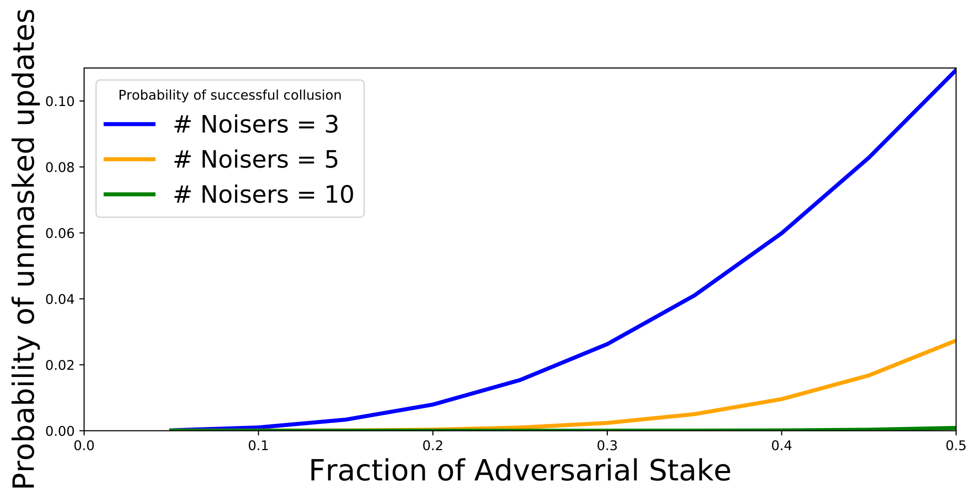


**Figure 6.10:** Probability of a successful collusion attack to recover an individual client's gradient.

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 precommit 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 $\varepsilon$ 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 atleast one verifier in the verification committee. Figure 6.10 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 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 6.12), 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.

## 6.4 Performance, scalability and fault tolerance

In this section, we evaluate the overhead of each stage in Biscotti and investigate the effect on the overhead as the number of peers increase. In addition to this, we also measure the effect on Biscotti's performance as we scale the size of different committees. We compare the performance of Biscotti on federated learning. Finally, we see if 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 peers in training on MNIST. We captured the total amount of time spent in each of the major phases of our algorithm in Figure 4.1: collecting the noise from each of the noising clients (steps (2) and

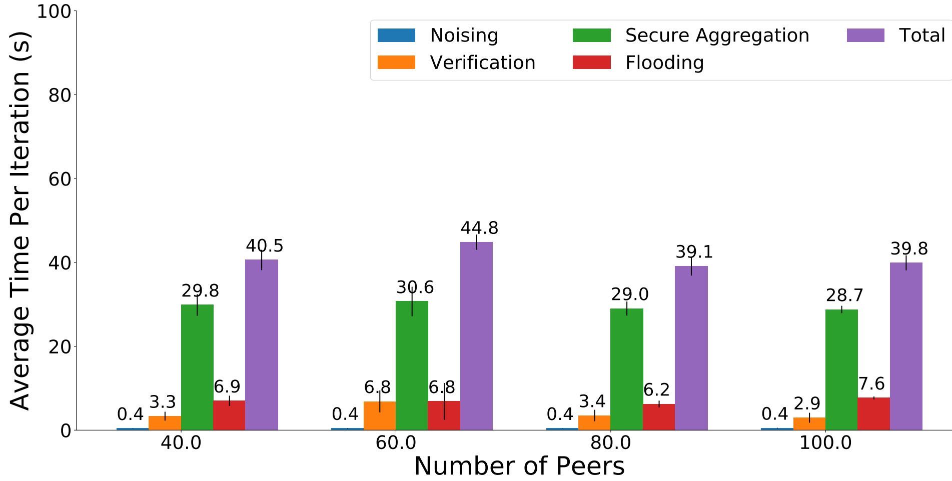


**Figure 6.11:** Breakdown of time spent in different mechanisms in Biscotti (Figure 4.1) in deployments with varying number of peers.

(3)), executing verification via Multi-KRUM and collecting the signatures (steps (4) and (5)) and securely aggregating the SGD update (steps (6) and (7)). Figure 6.11 shows the breakdown of 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 6.3.

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 making a round trip to each of the noisers 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 increase.

**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

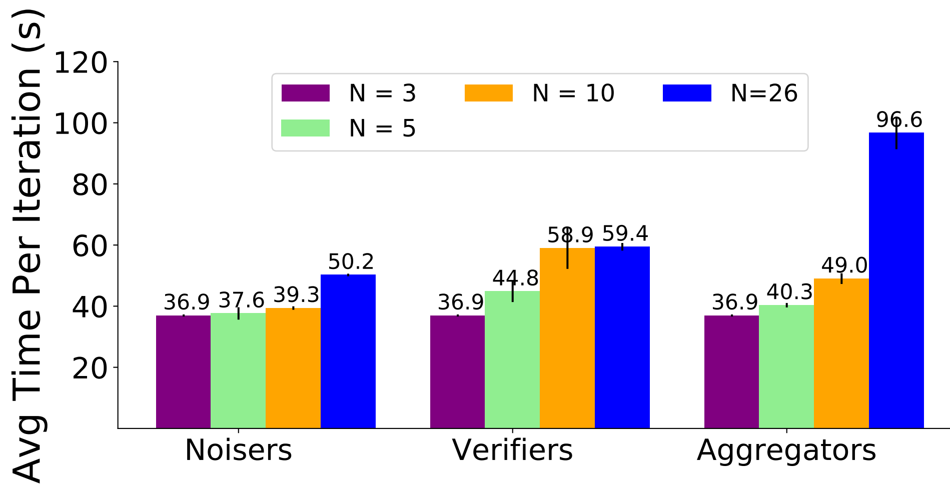


**Figure 6.12:** The average amount of time taken per iteration with an increasing number of noisers, verifiers, or aggregators.

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 6.12 plots the average time taken per iteration over 3 runs of the system.

The results show that increasing the number of noisers, verifiers and aggregator sets increases the time per iteration. The iteration time increases slightly with noisers because it requires contacting additional peers for the noise. Increasing the aggregators leads to a larger overhead because the secret shares are shared with more aggregators and recovering the aggregate requires coordination among more peers. Lastly, a large number of verifiers results in frequent timeouts in the mining stage. Verifiers wait for the first 70 updates and select 37 updates while miners 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 majority verifiers falls frequently below 35 because each of them run Multi-KRUM on a different set of updates. Hence, the aggregators wait until a timeout for 35 updates when the actual number of updates accepted during the round is less. 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

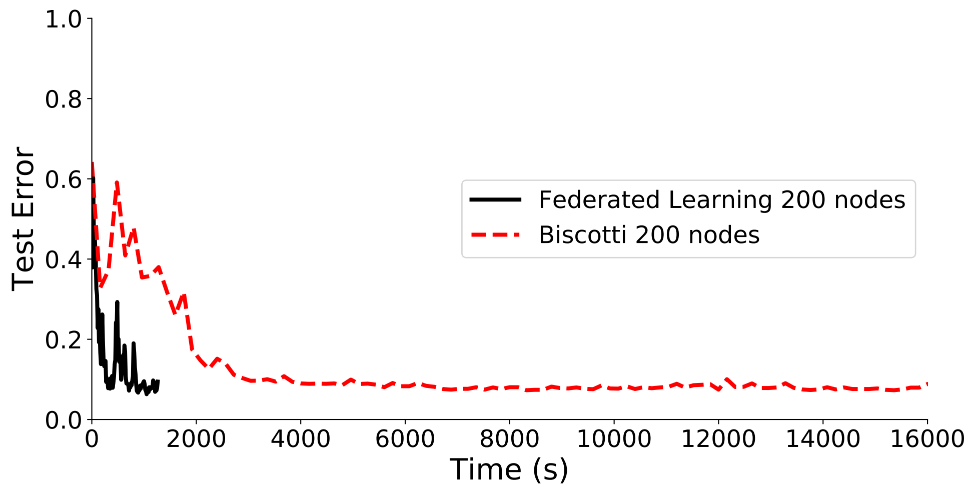


**Figure 6.13:** Comparing the time to convergence of Biscotti to a federated learning baseline.

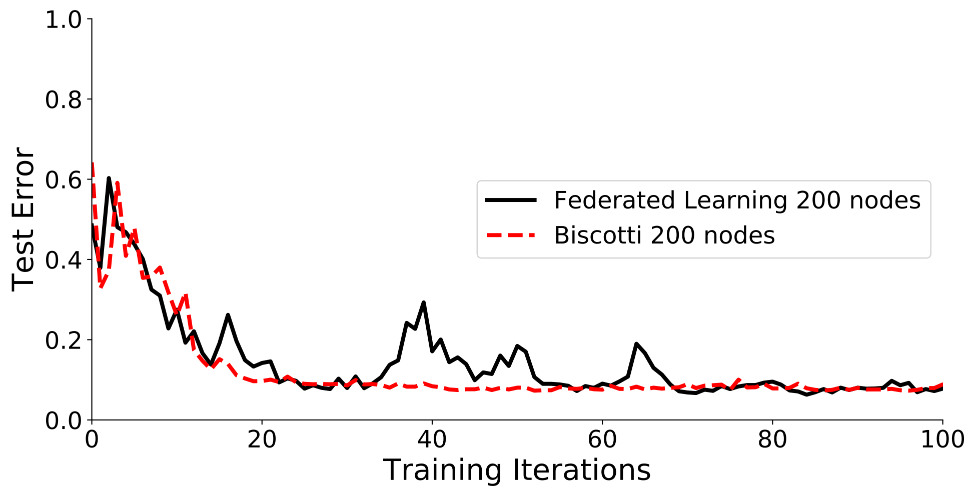


**Figure 6.14:** Comparing the iterations to convergence of Biscotti to a federated learning baseline.

26. However, this increased verifier overhead could be lowered by decreasing the number of updates that aggregators wait for before starting the coordination.

**Baseline performance**

We execute Biscotti in a baseline deployment and compare it to the original

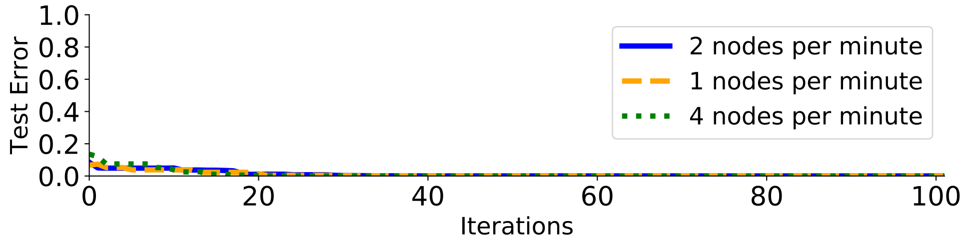


**Figure 6.15:** The impact of churn on model performance.

federated learning baseline [32]. We run Biscotti with 5 noisers, 26 verifiers and 26 aggregators. As demonstrated earlier, these committee sizes provide a guarantee of protecting against an adversary holding 30% stake from unmasking updates in the noising (Section 6.3) and aggregation stages (Section 6.1). They also ensure convergence under poisoning from an attack by 30% of the nodes. (Section 6.2) The MNIST dataset [30] into 200 equal partitions, each of which was shared with an honest peer on an Azure cluster of 20 VMs, with each VM hosting 10 peers. These Biscotti/Federated Learning peers collaborated on training an MNIST softmax classifier, and after 100 iterations both models approached the global optimum. To ensure a fair comparison, the number of updates included in the model each round are kept the same. Federated Learning selects 35% nodes at random out of 200 for every round and receives updates from them while Biscotti includes the first 35% verified updates in the block. The convergence rate over time for both systems is plotted in Figure 6.13 and the same convergence over the number of iterations is shown in Figure 6.14. 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 100 iterations.

**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 100 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 and is bootstrapped, maintaining a constant total number of 100 peers. Figure 6.15 shows the rate of convergence for varying failure rates of

4, 2, and 1 peer(s) failing and joining per minute. When a verifier or an aggregator fails, Biscotti defaults to the next iteration after a timeout, so this does not harm convergence.

Even with churn Biscotti is able to make progress towards the global objective. We found that Biscotti is resilient to churn rates up to 4 nodes per minute (1 node joining and 1 node failing every 15 seconds).

# Chapter 7

# Limitations

**KRUM limitations.** For KRUM to be effective against a large number of poisoners, it needs to observe a large number samples in each round. This may not always be possible in a decentralized system where there is node churn. In particular, this may compromise the system because adversaries may race to contribute a majority of the samples. In addition to this, Multi-KRUM could also reject updates of peers that have different data from the rest of the peers e.g only one peer has 1's in the system. We did not face this issue in our experiments because we partitioned the data uniformly. However, Biscotti is compatible with other poisoning detection approaches and can integrate any poisoning detection mechanism which uses SGD updates and does not require individual update histories. For example, in a previous version of Biscotti we used RONI [8] to validate updates.

**Deep Learning.** We showed that Biscotti is able to train a softmax and logistic regression model with 7,850 and 25 parameters respectively. But, Biscotti might not scale to a large deep learning model with millions of parameters due to the communication overhead. Strategies to reduce this might require learning updates in a restricted parameter space or compressing model updates [29]. We leave the training of deep learning models with Biscotti to future work.

**Leakage from the aggregate model.** Since there is no differential privacy added to the updates present in the ledger, Biscotti is vulnerable to attacks that exploit privacy leakage from the model itself [21, 45]. Apart from differential privacy, these attacks can also be mitigated by adding proper regularization [33, 45] like

dropout, or they may be irrelevant if a class does not represent an individual's training data.

**Stake limitations.** The stake that a client possesses plays a significant role in determining the chance that a node is selected as a noiser/verifier/aggregator. Our assumption is 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 honest and accrue stake for a pre-defined period of time and then turn malicious to subvert our system. We leave the design of a robust stake mechanism that protects against such attacks to future work.

# Chapter 8

# Related Work

Biscotti's novelty lies in its ability to simultaneously handle poisoning attacks and preserve privacy in a P2P multi-party training context.

**Securing ML.** Similar to KRUM, AUROR [48] and ANTIDOTE [44] are alternative techniques to defend against poisoning that rely on anomaly detection. AUROR has been mainly 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 trying to evade anomaly-based methods.

Other defenses like TRIM [26] and RONI [8] 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 by a certain amount.

Finally, Baracaldo et al. [7] 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 [25] we evaluated, gradient ascent techniques [26, 37] are a popular way of generating poisoned samples one sample at a time by solving a bi-level optimization problem. Backdoor attacks have also been proposed to make the classifier misclassify an image if it contains certain

pixels or a backdoor key. Defending against such attacks during training is a hard and open problem in the literature.

**Privacy attacks.** Shokri et. al [50] 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 [45] showed that it is quite easy to launch this attack in a black-box setting. In addition, model inversion attacks [21] have been proposed to invert class images from the final trained model. However, we assume that these are only effective if a class represents a significant chunk of a person's data e.g., facial recognition systems.

Additional security work in federated learning has required that public key infrastructure exists which validates the identity of users [12], but prior sybil work has shown that relying on public key infrastructure for user validation is insufficient [52, 53].

**Privacy-preserving ML.** Cheng et al. recently proposed the use of TEEs and privacy-preserving smart contracts to provide security and privacy to multi-party ML tasks [16]. But, some TEEs have been found to be vulnerable [51]. Another solution uses two party computation [36] and encrypts both the model parameters and the training data when performing multi-party ML.

Differential privacy is often applied to multi-party ML systems to provide privacy [4, 22, 49]. However, its use introduces privacy-utility tradeoffs. Biscotti uses differential privacy during update validation, but does not push noise into the trained model, thus favouring utility.

# Chapter 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 peer-to-peer ML through a distributed ledger, while simultaneously considering poisoning attacks. And, unlike prior work, Biscotti does not rely on trusted execution environments or specialized hardware. In our evaluation we demonstrated that Biscotti can coordinate a collaborative learning process across 100 peers and produces a final model that is similar in utility to state of the art federated learning alternatives. We also illustrated its ability to withstand poisoning and information leakage attacks, and frequent failures and joining of nodes (one node joining, one node leaving every 15 seconds).

Our Biscotti prototype is open source, runs on commodity hardware, and interfaces with PyTorch, a popular framework for machine learning.

# Bibliography

[1] A CONIKS Implementation in Golang. https://github.com/coniks-sys/coniks-go, 2018. → page 23

[2] DEDIS Advanced Crypto Library for Go. https://github.com/dedis/kyber, 2018. → page 23

[3] go-python. https://github.com/sbinet/go-python, 2018. → page 23

[4] M. Abadi, A. Chu, I. Goodfellow, B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *23rd ACM Conference on Computer and Communications Security*, CCS, 2016. → pages 2, 3, 5, 44, 53, 54

[5] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman. Medrec: Using blockchain for medical data access and permission management. OBD '16. → page 2

[6] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov. How To Backdoor Federated Learning. *ArXiv e-prints*, 2018. → page 10

[7] N. Baracaldo, B. Chen, H. Ludwig, and J. A. Safavi. Mitigating poisoning attacks on machine learning models: A data provenance based approach. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, AISec, 2017. → page 43

[8] M. Barreno, B. Nelson, A. D. Joseph, and J. D. Tygar. The Security of Machine Learning. *Machine Learning*, 81(2), 2010. → pages 41, 43

[9] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo. Analyzing federated learning through an adversarial lens. In *Proceedings of the 36th International Conference on Machine Learning*, 2019. → page 10

[10] B. Biggio, B. Nelson, and P. Laskov. Poisoning Attacks Against Support Vector Machines. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML, 2012. → page 1

[11] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems 30*, NIPS. 2017. → pages 3, 5, 9, 28

[12] 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 *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS, 2017. → pages iii, 1, 2, 44, 53

[13] L. Bottou. On-line learning in neural networks. Cambridge University Press, 1999. → page 9

[14] L. Bottou. *Large-Scale Machine Learning with Stochastic Gradient Descent*. COMPSTAT '10. 2010. → pages 2, 52

[15] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, 1999. → page 2

[16] 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 contract execution. *ArXiv e-prints*, 2018. → page 44

[17] 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 *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS, 2012. → pages 2, 22

[18] D. Dheeru and E. Karra Taniskidou. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml. → page 25

[19] J. J. Douceur. The sybil attack. IPTPS '02. → pages 4, 9

[20] C. Dwork and A. Roth. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4), 2014. → pages 2, 5, 16

[21] M. Fredrikson, S. Jha, and T. Ristenpart. Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures. In *Proceedings*

*of the 2015 ACM SIGSAC Conference on Computer and Communications Security*, CCS, 2015. → pages 10, 41, 44

[22] 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. → pages 2, 44

[23] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP, 2017. → pages 2, 8, 15, 58

[24] B. Hitaj, G. Ateniese, and F. Pérez-Cruz. Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS, 2017. → pages 1, 10

[25] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar. Adversarial Machine Learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, AISec, 2011. → pages iii, 1, 3, 10, 24, 27, 43

[26] 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 *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. → page 43

[27] A. Kate and I. Zaverucha, Gregory M.and Goldberg. Constant-size commitments to polynomials and their applications. 2010. → pages 5, 14, 18, 54, 55

[28] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Advances in Cryptology – CRYPTO 2017*, 2017. → page 8

[29] J. Konen, 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. → page 41

[30] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998. → pages 25, 39

[31] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille. Simple schnorr multi-signatures with applications to bitcoin. Cryptology ePrint Archive, Report 2018/068, 2018. → page 23

[32] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, AISTATS, 2017. → pages 1, 8, 39, 52

[33] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov. Inference Attacks Against Collaborative Learning. *ArXiv e-prints*, 2018. → pages 1, 3, 5, 7, 16, 24, 33, 41

[34] S. Micali, S. Vadhan, and M. Rabin. Verifiable random functions. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS, 1999. → pages 2, 11

[35] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz. Permacoin: Repurposing bitcoin work for data preservation. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, S&P, 2014. → page 2

[36] P. Mohassel and Y. Zhang. Secureml: A system for scalable privacy-preserving machine learning, 2017. → page 44

[37] 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 *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security 2017*. → pages 10, 43

[38] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2009. → pages 2, 9, 21

[39] N. Narula, W. Vasquez, and M. Virza. zkledger: Privacy-preserving auditing for distributed ledgers. In *15th USENIX Symposium on Networked Systems Design and Implementation*, NSDI, 2018. → page 2

[40] 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 *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, LEET, 2008. → page 1

[41] 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. → page 23

[42] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference*, CRYPTO, 1992. → page 55

[43] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems 24*, NIPS. 2011. → pages 2, 22

[44] 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 *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, IMC, 2009. → page 43

[45] 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. 2018. → pages 10, 41, 44

[46] C. P. Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology — CRYPTO' 89 Proceedings*, 1990. → page 23

[47] A. Shamir. How to share a secret. *Communications of the ACM*, 1979. → pages 3, 19, 54

[48] S. Shen, S. Tople, and P. Saxena. Auror: Defending against poisoning attacks in collaborative deep learning systems. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ACSAC, 2016. → pages 2, 43

[49] R. Shokri and V. Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 2015 ACM SIGSAC Conference on Computer and Communications Security*, CCS, 2015. → page 44

[50] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *IEEE Symposium on Security and Privacy*, pages 3–18. IEEE Computer Society, 2017. → page 44

[51] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *Proceedings of the 27th USENIX Security Symposium*, USENIX SEC, 2018. → page 44

[52] B. Viswanath, M. A. Bashir, M. B. Zafar, S. Bouget, S. Guha, K. P. Gummadi, A. Kate, and A. Mislove. Strength in numbers: Robust tamper detection in crowd computations. In *Proceedings of the 3rd ACM Conference on Online Social Networks*, COSN, 2015. → page 44

[53] G. Wang, B. Wang, T. Wang, A. Nika, H. Zheng, and B. Y. Zhao. Defending against sybil devices in crowdsourced mapping services. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys, 2016. → page 44

# Appendix: Supporting Materials

## A.1 Federated learning and distributed stochastic gradient descent

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) [14] as the optimization algorithm.

In federated learning [32], 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:

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

where $\eta_t$ represents a degrading learning rate, $\lambda$ 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 $\nabla l$ 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 [14]. 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:

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

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 [12] 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.

## A.2 Differentially private stochastic gradient descent

We use the concept of $(\varepsilon, \delta)$ differential privacy as explained in Abadi et al. [4]. Each SGD step becomes $(\varepsilon, \delta)$ 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.

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

**Data:** Batch size $b$, Learning rate $\eta_t$, Privacy parameters $(\varepsilon, \delta)$, 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 $\sigma = \sqrt{2log\frac{1.25}{\delta}}/\varepsilon$
**end**
$\zeta_t = \frac{\eta_t}{b}\sum_i \zeta_i$
Commit $\zeta_t$ to the genesis block at column $t$.
**end**

**Algorithm 1:** Precommiting differentially Private noise for SGD, taken from Abadi et al. [4].

follows from aggregation:

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

## A.3 Polynomial commitments and verifiable secret sharing

Polynomial Commitments [27] is a scheme that allows commitments to a secret polynomial for verifiable secret sharing [47]. 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 $\alpha$ 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 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 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 [27]. 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 [27]. This scheme can be easily extended to provide unconditional hiding by combining it with another scheme called Pedersen commitments [42] however we do not implement it.

## A.4 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 4.3). 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.

## A.5 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 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.

## A.6 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 [23] 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 A.4) 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 al-

gorithm, 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.