Private and secure distributed ML

Ivan Beschastnikh, Clement Fung,
Stewart Grant, Michael Hou, Jamie Koerner,
Shayan Muhammad, Gleb Naumenko, Chris Yoon

University of British Columbia

Networks Systems Security lab
nss.cs.ubc.ca

Private and secure distributed ML

Defense for Sybil-based poisoning in Fed Learning

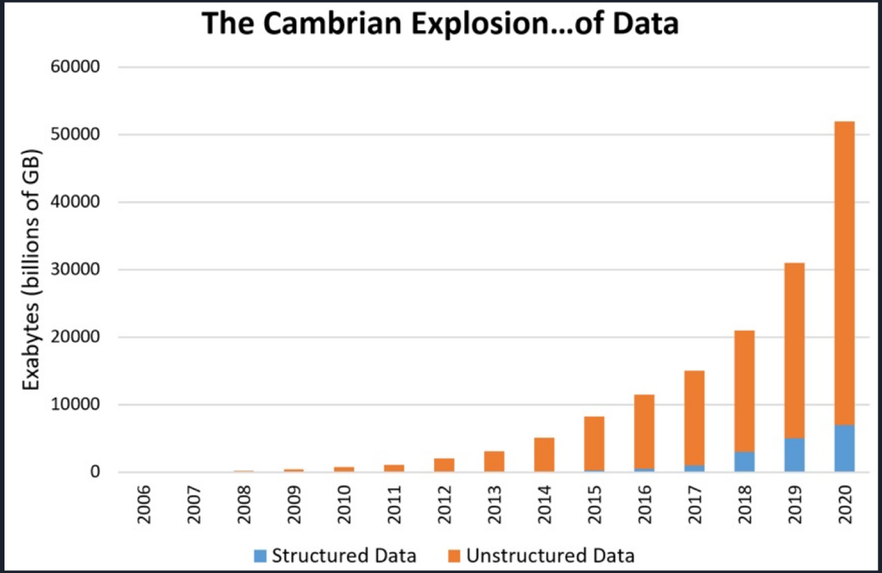
P2P ML via a Blockchain

Ivan Beschastnikh, Clement Fung, Stewart Grant, Michael Hou, Jamie Koerner, Shayan Muhammad, Gleb Naumenko, Chris Yoon

University of British Columbia

Networks Systems Security lab
nss.cs.ubc.ca

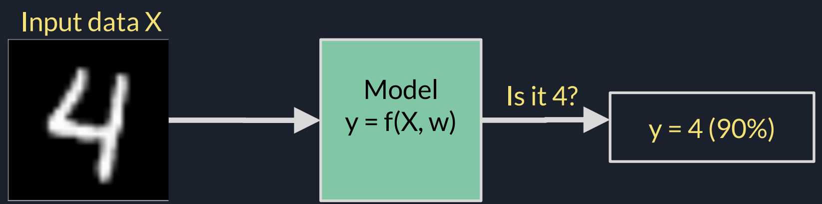
The Explosion of Data and Machine Learning



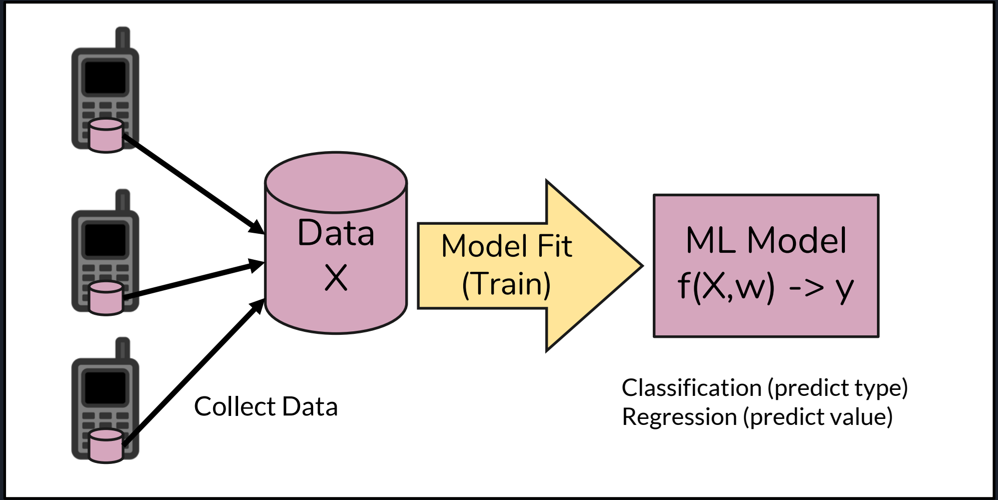
"By 2020, the amount of data is predicted to sit at 53 zettabytes - increasing 50 times since 2003."

-- Hal Varian, Chief Economist Google

Machine Learning helps to extract insights from immense amounts of heterogeneous data without human intervention:



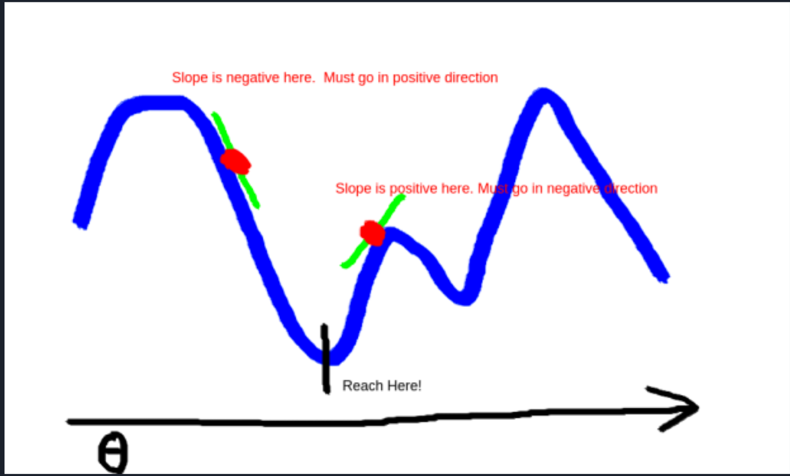
(Very) Gentle ML Overview



- **X**: labelled data features
  - E.g., Square footage
- **y**: predicted output
  - E.g., House value
  - Categorical or numerical
- **w**: model parameters
  - Feature weighting
  - Depends on model type
    - Assume arbitrary vector of floats

ML: Stochastic Gradient Descent

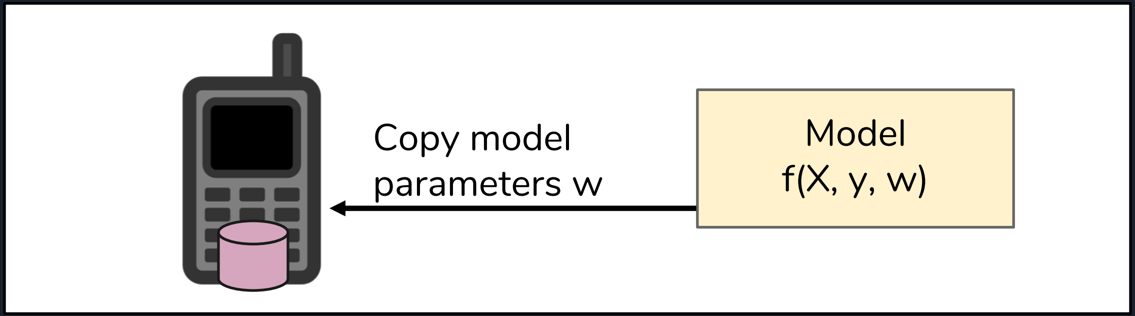
- SGD: General <u>iterative</u> algorithm for model training [1]
  - Can apply to regressions, deep learning, etc.



[1] Léon Bottou "*Large-Scale Machine Learning with Stochastic Gradient Descent*", COMPSTAT '10

ML: Stochastic Gradient Descent

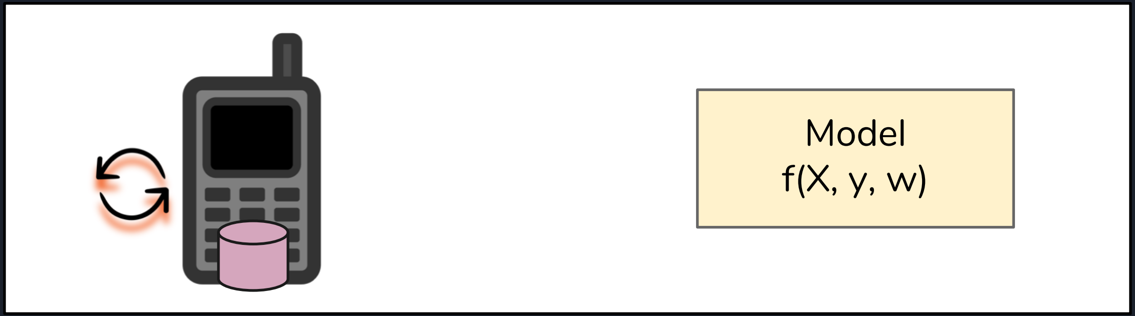
- SGD: General <u>iterative</u> algorithm for model training [1]
  - Can apply to regressions, deep learning, etc.



[1] Léon Bottou "*Large-Scale Machine Learning with Stochastic Gradient Descent*", COMPSTAT '10

ML: Stochastic Gradient Descent

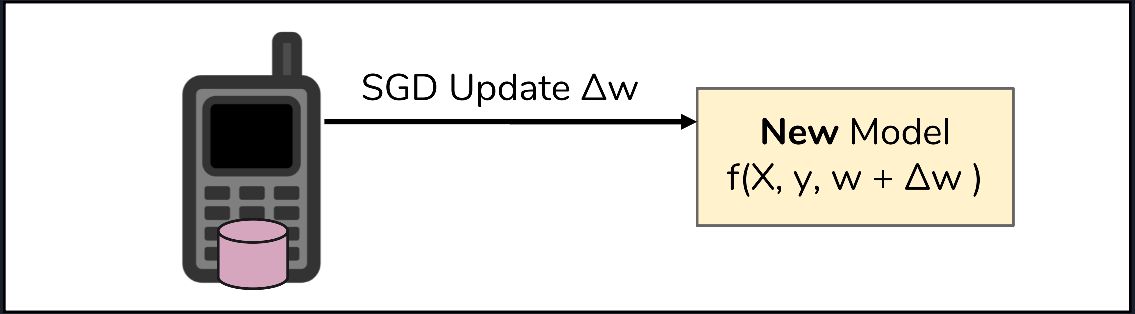
- SGD: General <u>iterative</u> algorithm for model training [1]
  - Can apply to regressions, deep learning, etc.



[1] Léon Bottou "*Large-Scale Machine Learning with Stochastic Gradient Descent*", COMPSTAT '10

ML: Stochastic Gradient Descent

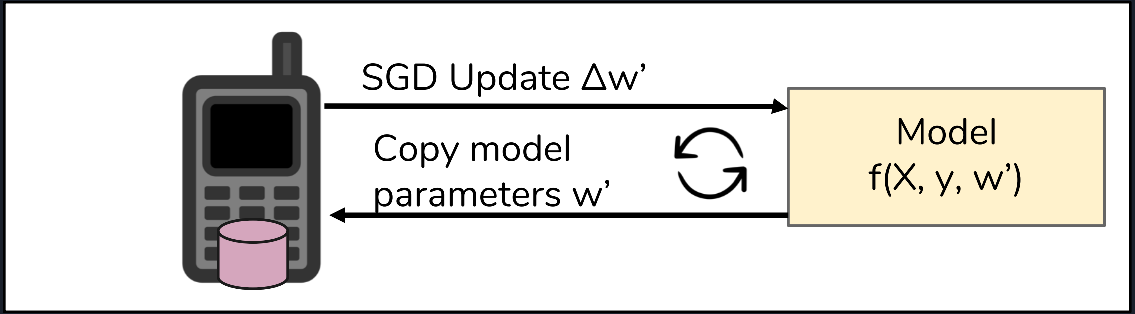
- SGD: General iterative algorithm for model training [1]
  - Can apply to regressions, deep learning, etc.



[1] Léon Bottou "*Large-Scale Machine Learning with Stochastic Gradient Descent*", COMPSTAT '10

ML: Stochastic Gradient Descent

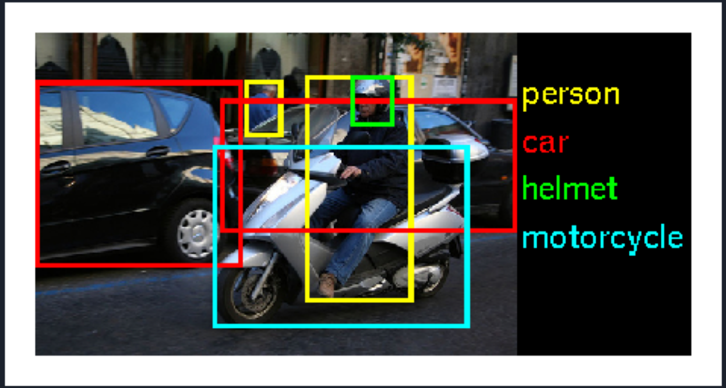
- **Repeat** until done!
  - Using some convergence gradient metric
  - For a fixed number of iterations



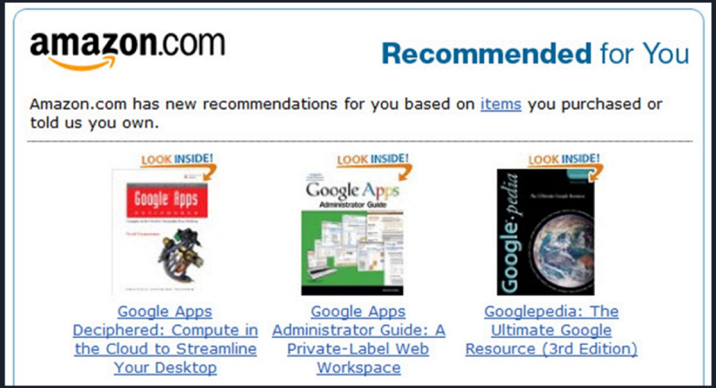
[1] Léon Bottou “*Large-Scale Machine Learning with Stochastic Gradient Descent*”, COMPSTAT ‘10

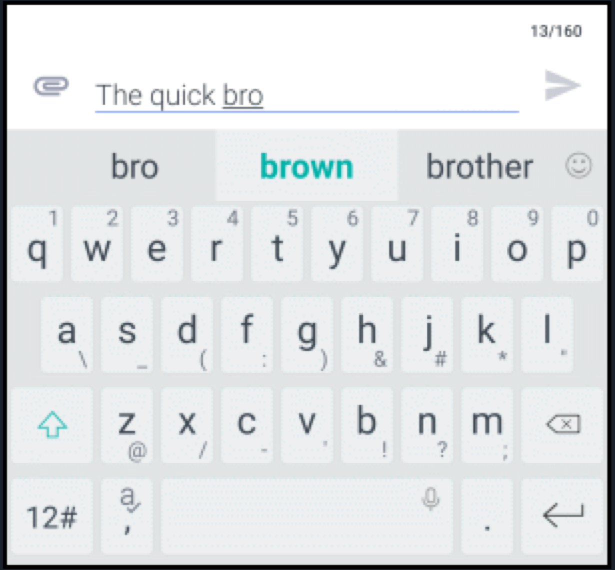
ML Use Cases

Computer Vision



Recommender Systems

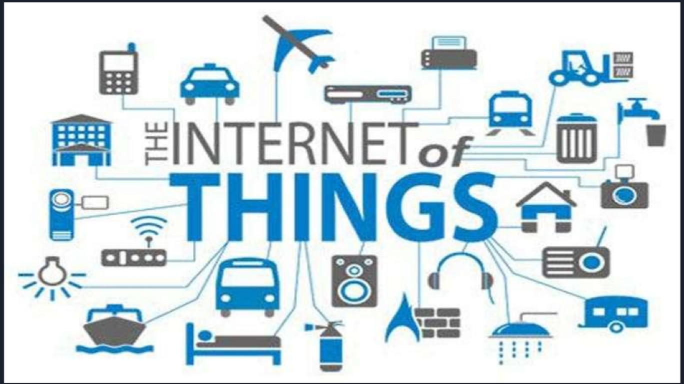




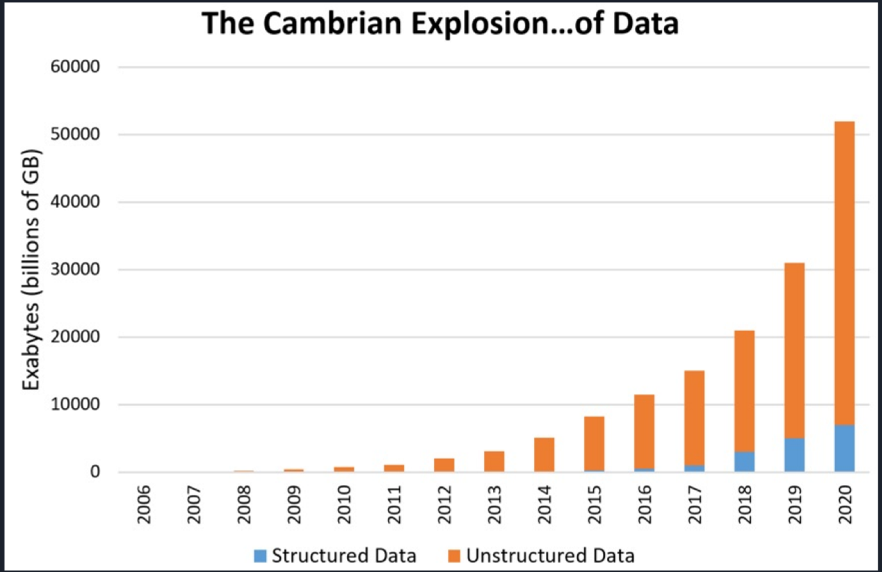
NLP

Key observation: Many Data and Analyses are **Decentralized**

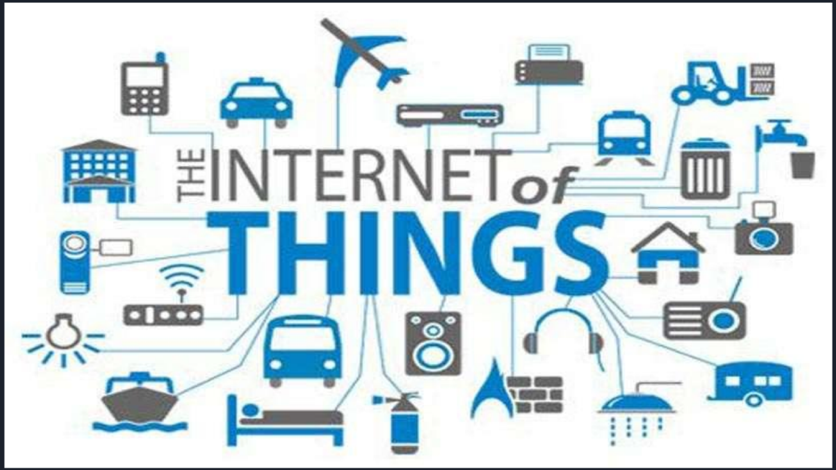
- Internet of things (large scale sensor networks)
- Live mobile analytics (maps/routing/traffic)



So... how do we process all this decentralized data with ML?



+



Modern Large Scale ML Solutions

- Modern solutions: centralize data and centralize compute
  - Copy + store all data in a data centre and train on it
  - Facebook has to periodically train on 100s of TBs of data that can take days to complete [1]
  - 3 example scale-out ML solutions:

[1] Hazelwood et al. *"Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective"* HPCA 18

Modern Large Scale ML Solutions

- Modern solutions: centralize data and centralize compute
    - Copy + store all data in a data centre and train on it

Problem: this is **costly** and **lacks privacy**

[1] Hazelwood et al. *"Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective"* HPCA 18

Costs of Centralization


. . .

- ~2.3 billion smartphones and rising
- Collecting data, keeping it updated is expensive
- Recent improvements: perform ML **without** data transfer
  - Aggregating locally trained models
  - Training over the network:
    - Gaia [1] : build ML models using data across data centers
    - **Federated Learning** [2] : build ML models from data on handheld devices around the world

[1] Hsieh et al. *"Gaia: Geo-distributed Machine Learning Approaching LAN Speeds"* NSDI 17
[2] McMahan et al. *"Communication Efficient Learning of Deep Networks from Decentralized Data"* AISTATS 17

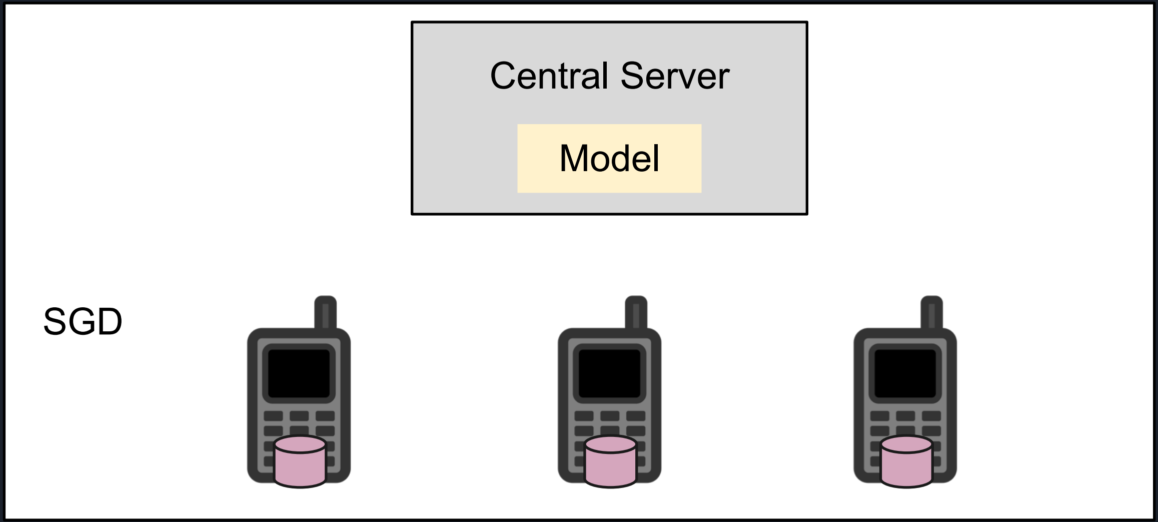
Distributed ML: Federated Learning

Federated Learning [1] (Google's new 2017 algorithm): Data never leaves the client, as good as centralized

[1] McMahan et al. *"Communication-Efficient Learning of Deep Networks from Decentralized Data" AISTATS 17*

Distributed ML: Federated Learning

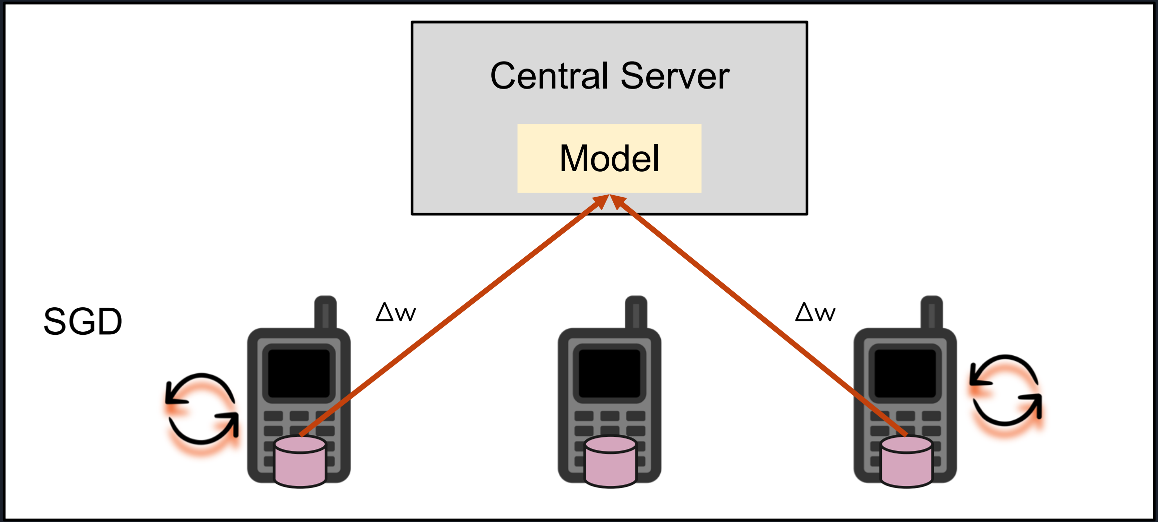
- Key idea: send SGD updates over network



[1] McMahan et al. *"Communication-Efficient Learning of Deep Networks from Decentralized Data" AISTATS 17*

Distributed ML: Federated Learning

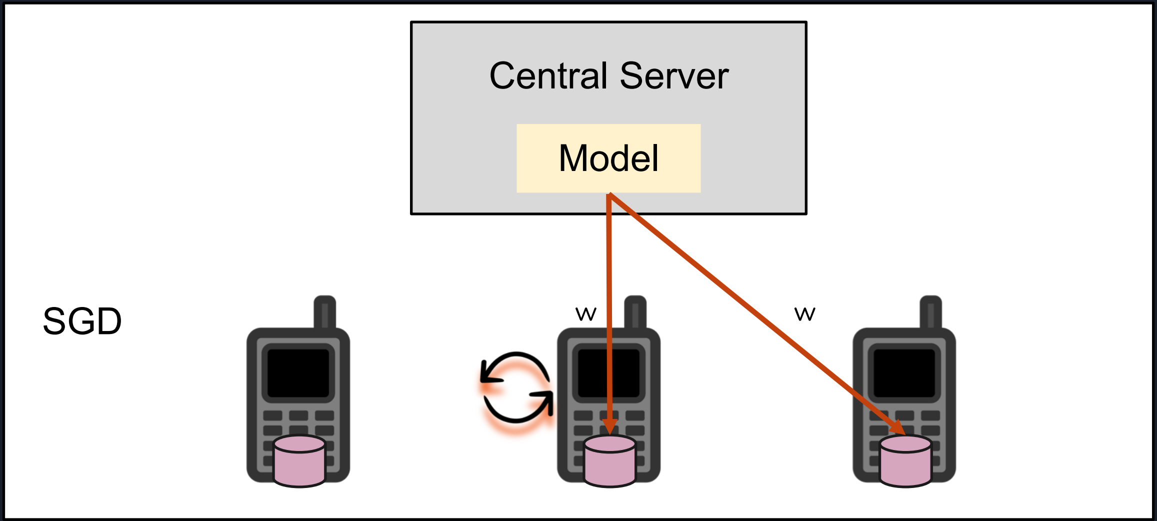
- Key idea: send SGD updates over network



[1] McMahan et al. *"Communication-Efficient Learning of Deep Networks from Decentralized Data" AISTATS 17*

Distributed ML: Federated Learning

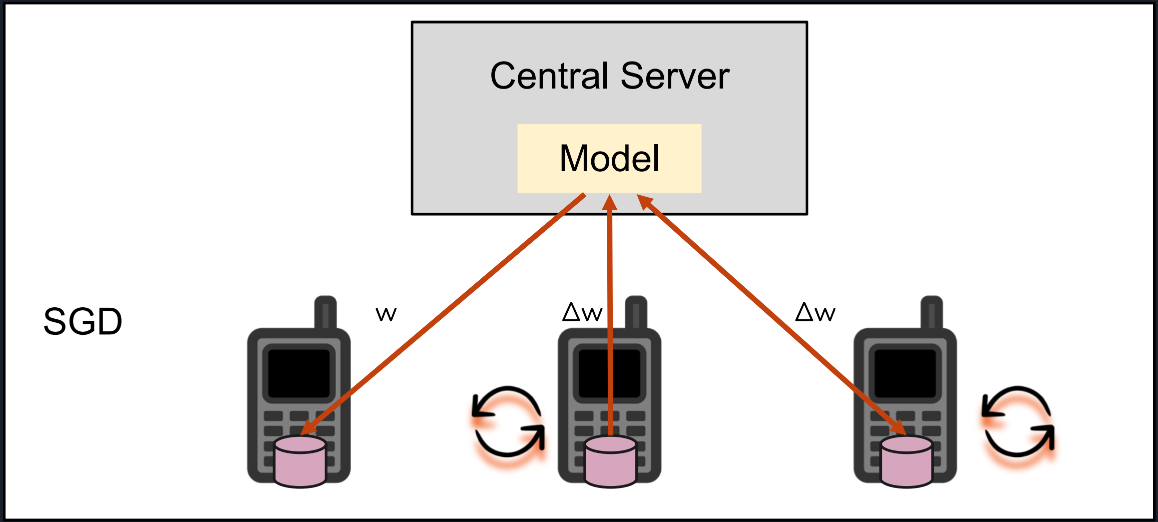
- Key idea: send SGD updates over network



[1] McMahan et al. *"Communication-Efficient Learning of Deep Networks from Decentralized Data" AISTATS 17*

Distributed ML: Federated Learning

- Key idea: send SGD updates over network



[1] McMahan et al. *“Communication-Efficient Learning of Deep Networks from Decentralized Data” AISTATS 17*

Federated Learning Tradeoffs

- **Benefits**: client centric compute enables privacy
  - Data remains with client
  - Perform SGD locally at client
    - Can modify the protocol for further privacy
  - Client churn, asynchrony, non-IID data OK!

- **Drawbacks**: less control for server
  - Clients used to just provide data
  - Clients capable of many new attacks

[1] McMahan et al. *"Communication-Efficient Learning of Deep Networks from Decentralized Data" AISTATS 17*

In this talk

- Introduction: cloud machine learning (ML)

Two projects with two points of view:

1. **Federated learning is here to stay**
    - FoolsGold : Countering sybil poisoning in fed. learning
2. **The future is decentralized**
    - Biscotti: P2P ML on the blockchain

In this talk

- Introduction: cloud machine learning (ML)

Two projects with two points of view:

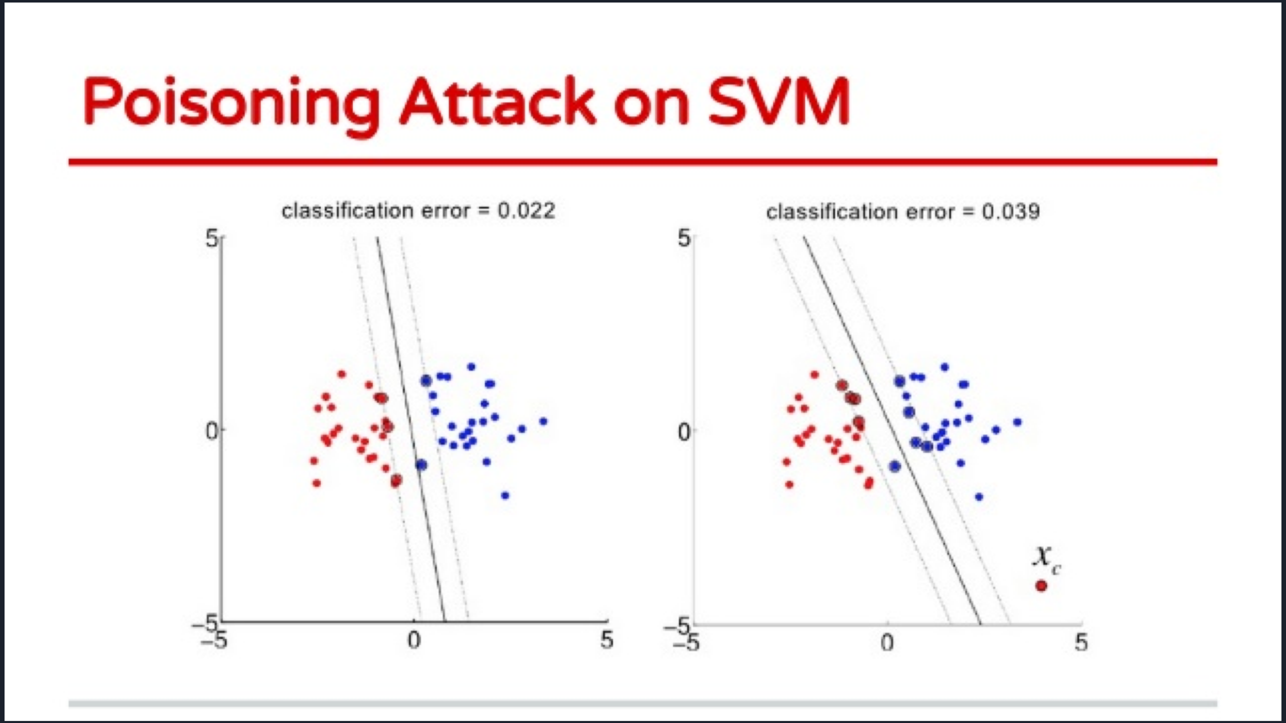
1. **Federated learning is here to stay**
    - FoolsGold : Countering sybil poisoning in fed. learning
2. **The future is decentralized**
    - Biscotti: P2P ML on the blockchain

Poisoning Attacks

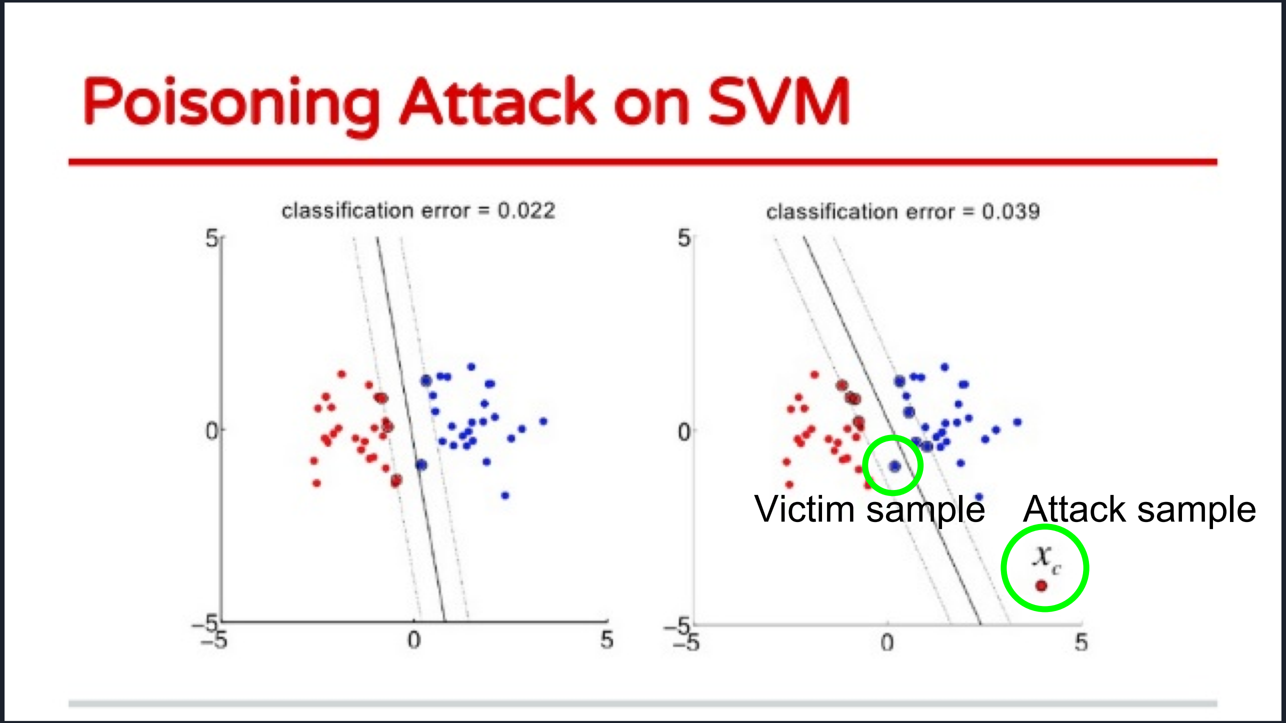
- Influence model prediction outputs
- Supply malicious training data
- Two types: [1]
    - Random attack: Aim to decrease model accuracy
    - **Targeted attack:** Aim to increase/decrease classification of a specific point
        - I want my email to pass a spam filter
        - I want my advertisement to be displayed more

[1] Huang et al. "*Adversarial Machine Learning*". AISec '11

Targeted Poisoning Attacks



Targeted Poisoning Attacks



Sybil Attacks

- Fake accounts created for additional leverage [1]
- In ML setting:
  - Attacks can become **more** powerful (poisoning, leakage)

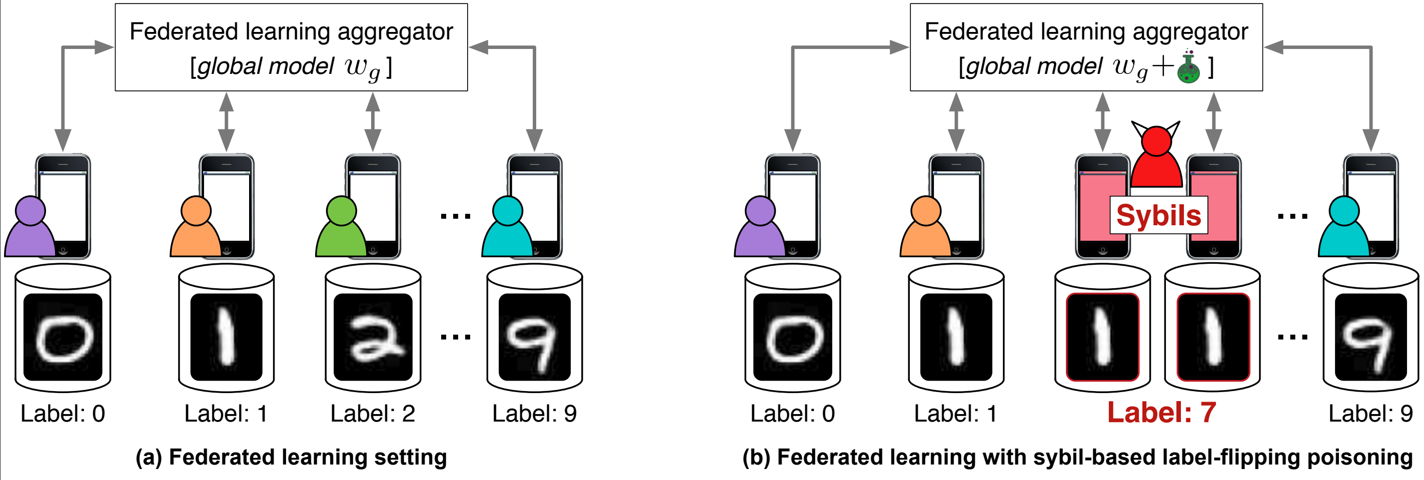
[1] Doucuer et al. "*The Sybil Attack*" IPTPS '01

[2] Wang et al. "*Defending against Sybil Devices in Crowdsourced Mapping Services*", MobiSys '16

Figure 1: Before the attack (left), Waze shows the fastest route for the user. After the attack (right), the user gets automatically re-routed by the fake traffic jam.

Sybil-based poisoning in federated learning

- Problem:
  - Federated Learning actively involves clients
  - Easy for a client to poison model using sybils

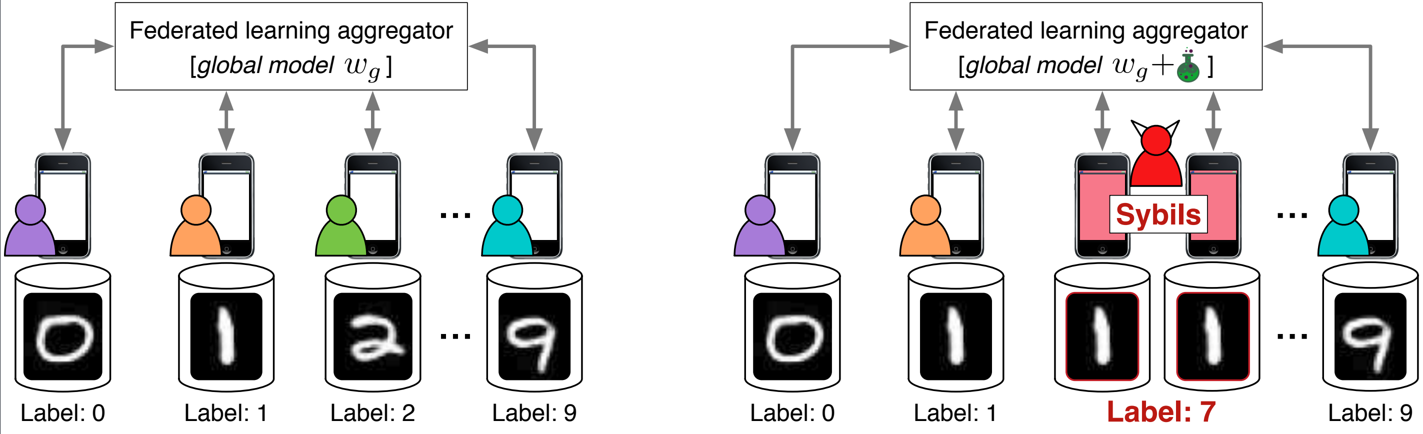


Sybil-based poisoning

TABLE I. THE ACCURACY AND ATTACK SUCCESS RATES FOR BASELINE (NO ATTACK), AND ATTACKS WITH 1 AND 2 SYBILS IN A FEDERATED LEARNING CONTEXT WITH MNIST DATASET.

	Baseline	Attack 1	Attack 2
# honest clients	10	10	10
# malicious sybils	0	1	2
Accuracy (digits: 0, 2-9)	90.2%	89.4%	88.8%
Accuracy (digit: 1)	96.5%	60.7%	0.0%
Attack success rate	0.0%	35.9%	96.2%

- Problem:
  - Federated Learning actively involves clients
  - Easy for a client to poison model using sybils



(a) Federated learning setting

(b) Federated learning with sybil-based label-flipping poisoning

Sybil-based poisoning in federated learning

- Problem:
  - Federated Learning actively involves clients
  - Easy for a client to poison model using sybils

- Existing solutions:
  - Involve detecting malicious data/robust ML [1,2]
  - Assumptions about data: poor match for fed. learn. setting!
  - Only work up to a limit: “*at most 33% attackers*” [3]

[1] Rubinstein et al. “*ANTIDOTE: Understanding and Defending against Poisoning of Anomaly Detectors*” IMC ‘09
[2] Shen et al. “*Auror: Defending Against Posoning ATtacks in Collaborative Deep Learning Systems*” ACSAC’16
[3] Blanchard et al. “*Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent*”. NIPS ’17

Towards More Robust Poisoning Defenses

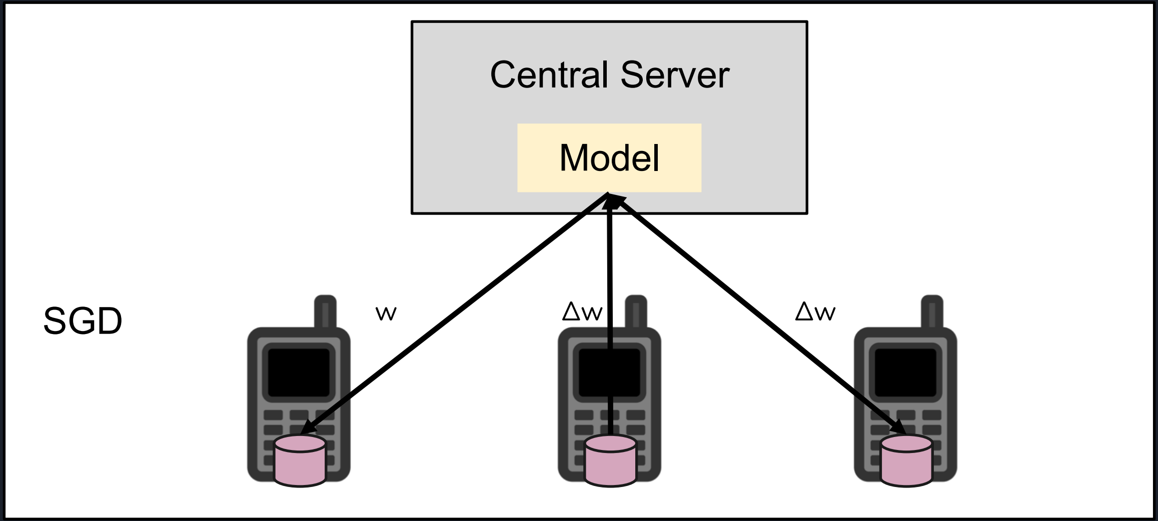
- Our approach (FoolsGold):
  - Combine ideas from graph defense and anomalous behaviour defense to ML context [1,2]
    - Use update similarity and correctness
  - Instead of robustness, detect and reject Sybils

[1] Viswanath et al. "*Strength in Numbers: Robust Tamper Detection in Crowd Computations* " COSN '15
[2] Tran et al. "*Sybil-Resilient Online Content Voting*" NSDI '09

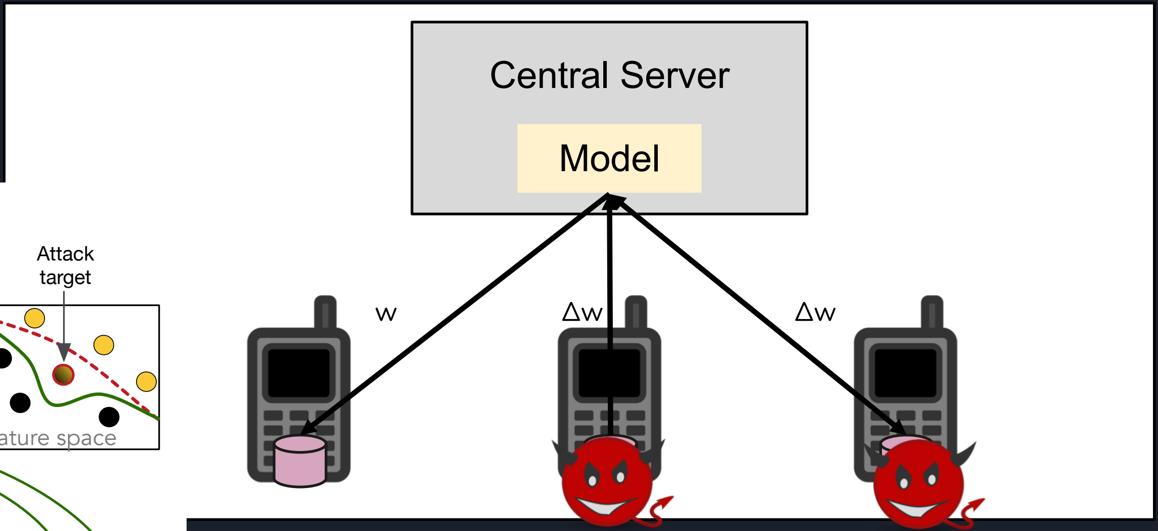
FoolsGold goals

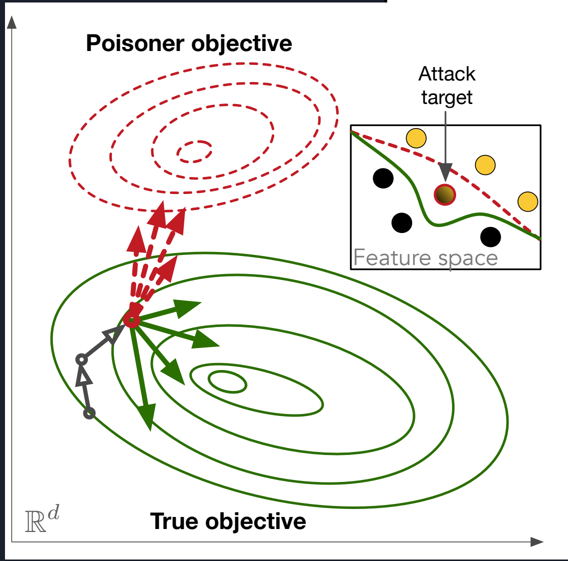
1. Preserve Fed. Learning performance when no attacks
2. Devalue contributions that are similar
3. Be robust to an increasing number of sybils
4. Preserve honest gradients
5. Make few assumptions (e.g., # of attackers)

FoolsGold: how it works

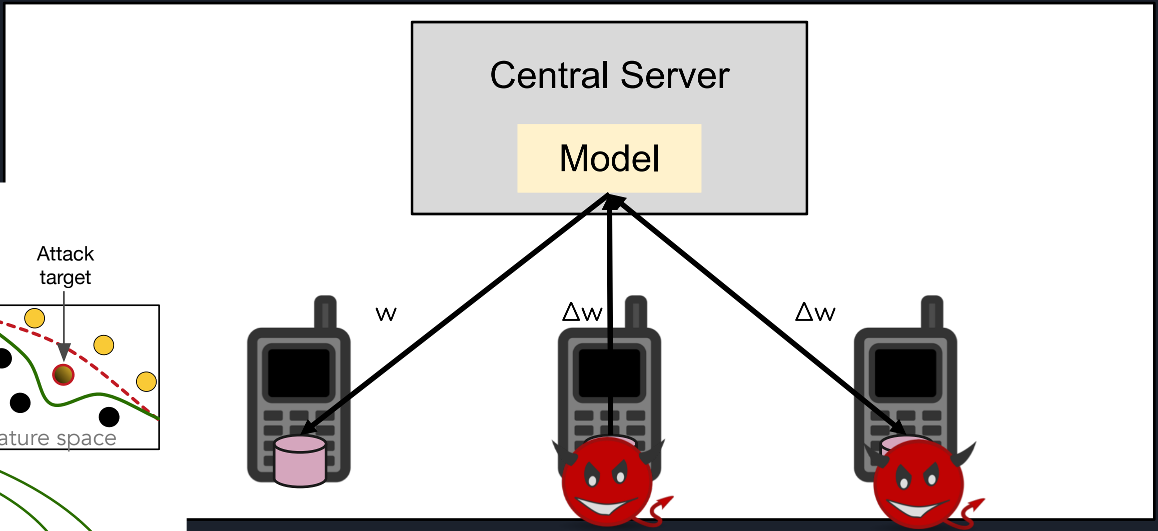


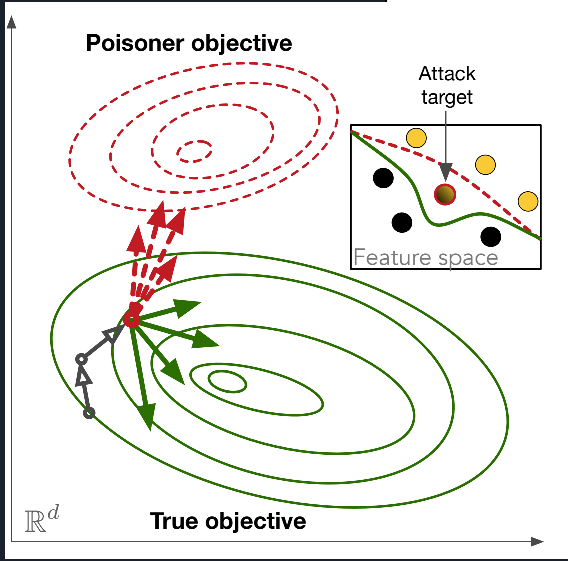
FoolsGold: how it works





FoolsGold: how it works

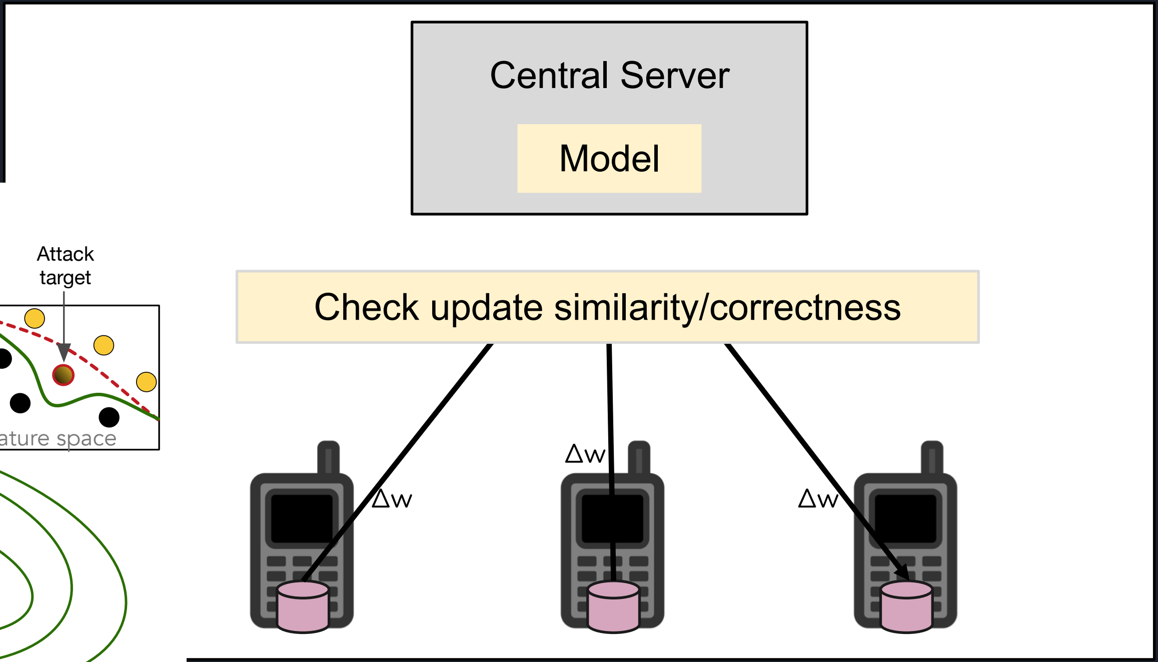


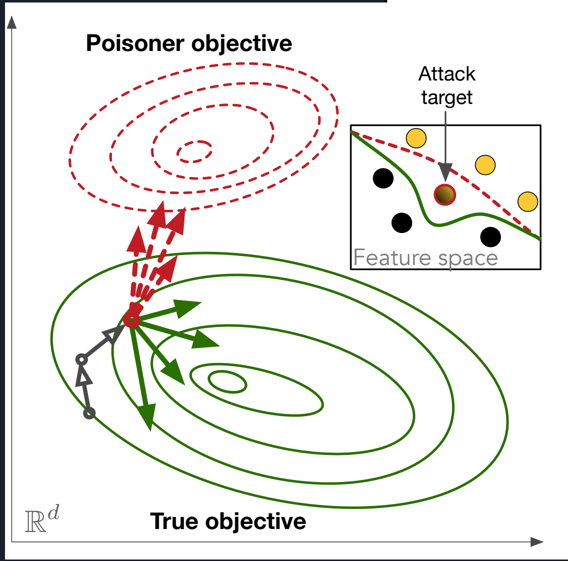


Key ideas:

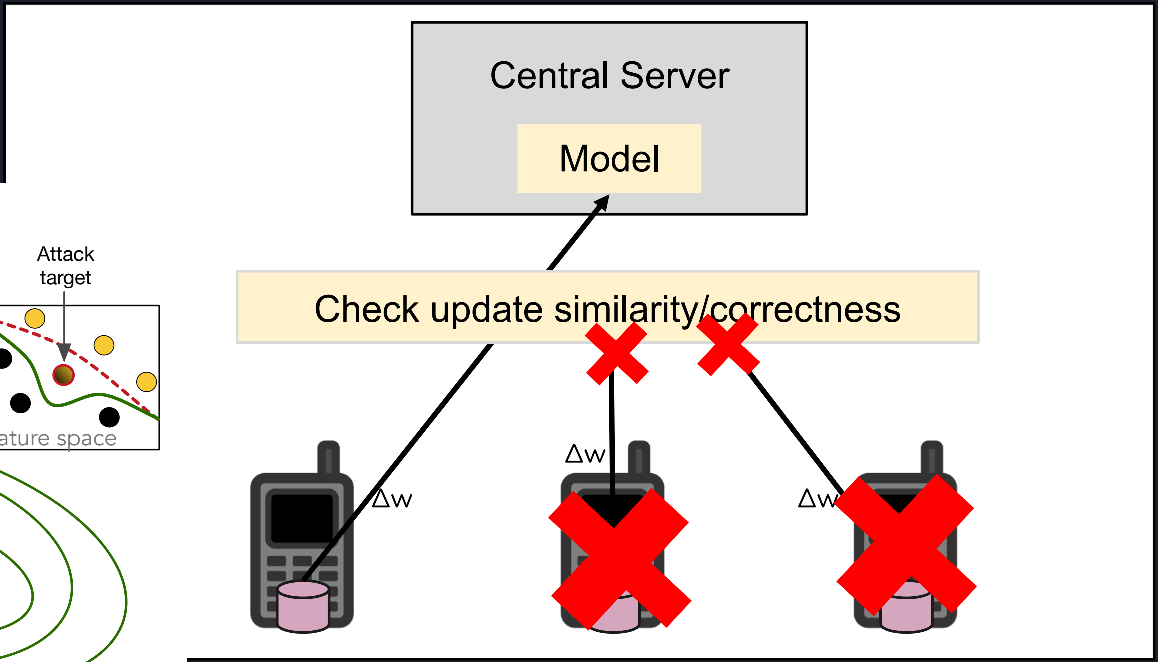
1. Limit attacker ability to influence model with similar-looking data
2. Use shape of updates to identify and reject Sybil contributions

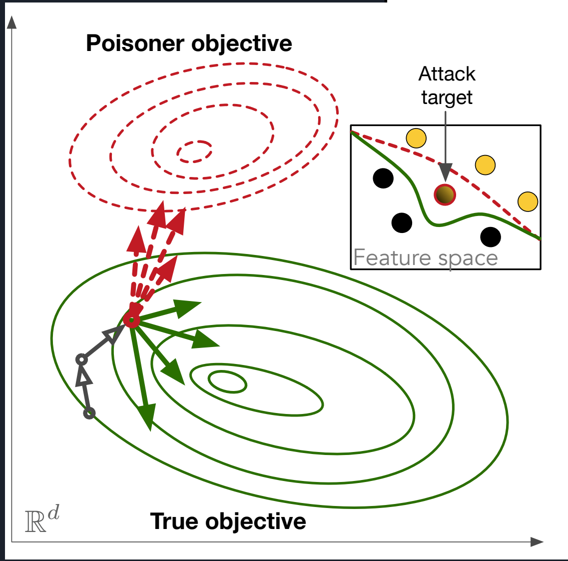
FoolsGold: how it works



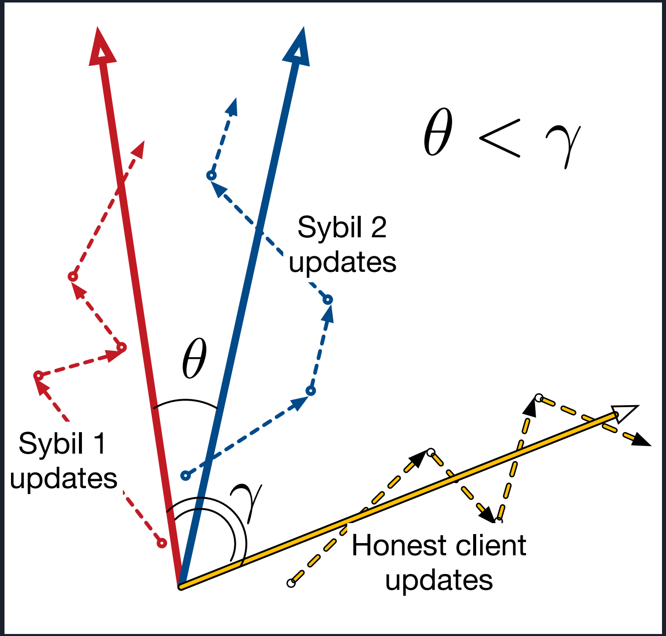


FoolsGold: how it works

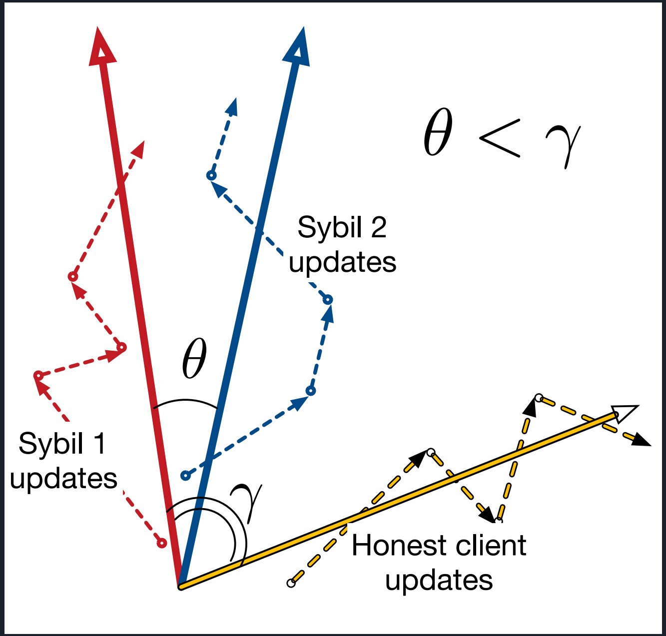




FoolsGold: more details



FoolsGold: more details

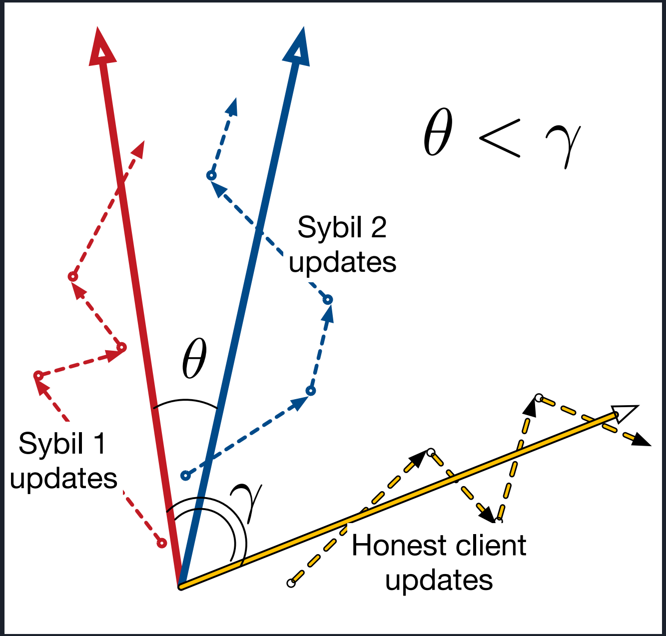


Data: Δ_t from all clients at iteration t
Result: A client weight vector v

for *iteration* t **do**
    for *All clients* i **do**
        // Updates history
        Let H_i be the aggregate historical vector $\sum_{t=1}^{T} \Delta_{i,t}$
        // Feature importance
        Let S_t be the weight of indicative features at iteration t
        for *All other clients* j **do**
            Find weighted cosine similarity cs_{ij} between H_i and H_j using S_t
        end
        // Pardoning
        for *All other clients* j **do**
            if $\max_j(cs) > \max_i(cs)$ **then**
                cs_{ij} *= $\max_i(cs)/\max_j(cs)$
            end
        end
        Let $v_i = 1 - \max_j(cs_i)$
    end
    // Logit function
    $v = v/\max(v)$
    $v = \ln(\frac{v}{(1-v)} + 0.5)$
    return v
end

Algorithm 1: FoolsGold algorithm.

FoolsGold: more details

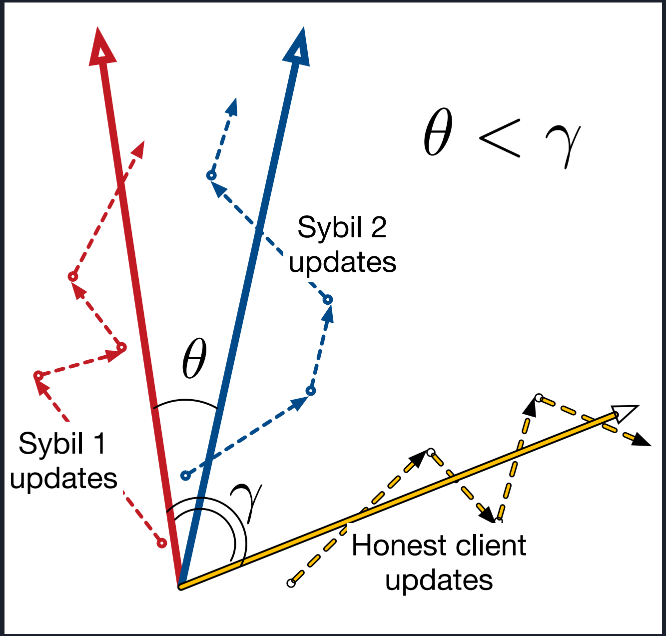


Data: Δ_t from all clients at iteration t
Result: A client weight vector v

for *iteration* t **do**
  for *All clients* i **do**
    // Updates history
    Let H_i be the aggregate historical vector $\sum_{t=1}^{T} \Delta_{i,t}$
    // Feature importance
    Let S_t be the weight of indicative features at iteration t
    for *All other clients* j **do**
      Find weighted cosine similarity cs_{ij} between H_i and H_j using S_t
    end
    // Pardoning
    for *All other clients* j **do**
      if $\max_j(cs) > \max_i(cs)$ **then**
        cs_{ij} *= $\max_i(cs)/\max_j(cs)$
      end
    end
    Let $v_i = 1 - \max_j(cs_i)$
  end
  // Logit function
  $v = v/\max(v)$
  $v = \ln(\frac{v}{(1-v)} + 0.5)$
  return v
end

Algorithm 1: FoolsGold algorithm.

FoolsGold: more details



Data: Δ_t from all clients at iteration t
Result: A client weight vector v

for *iteration* t **do**
  for *All clients* i **do**
    // Updates history
    Let H_i be the aggregate historical vector $\sum_{t=1}^{T} \Delta_{i,t}$
    // Feature importance
    Let S_t be the weight of indicative features at iteration t
    for *All other clients* j **do**
      Find weighted cosine similarity cs_{ij} between H_i and H_j using S_t
    end
    // Pardoning
    for *All other clients* j **do**
      if $\max_j(cs) > \max_i(cs)$ **then**
        cs_{ij} *= $\max_i(cs) / \max_j(cs)$
      end
    end
    Let $v_i = 1 - \max_j(cs_i)$
  end
  // Logit function
  $v = v / \max(v)$
  $v = \ln(\frac{v}{(1-v)} + 0.5)$
  return v
end

Algorithm 1: FoolsGold algorithm.

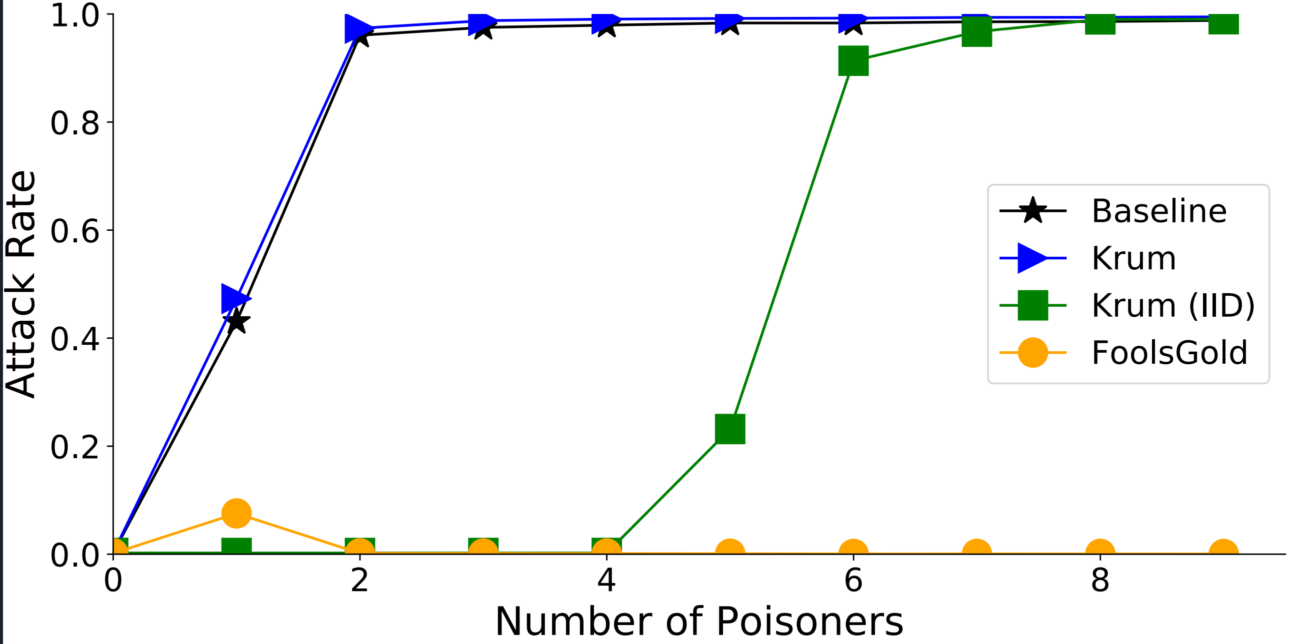
Evaluation (highlights)

TABLE II. DATASETS USED IN THIS EVALUATION.

Dataset	Examples	Classes	Features
MNIST	60,000	10	784
KDDCup	494,020	23	41
Amazon	1,500	50	10,000

Attack	Description
A-1	Single client attack.
A-5	5 clients attack.
A-2x5	2 sets of 5 clients, concurrent attacks.
A-5x5	5 sets of 5 clients, concurrent attacks.
A-AllOnOne	5 clients executing 5 attacks on the same target class.
A-99	99% adversarial clients, performing the same attack.

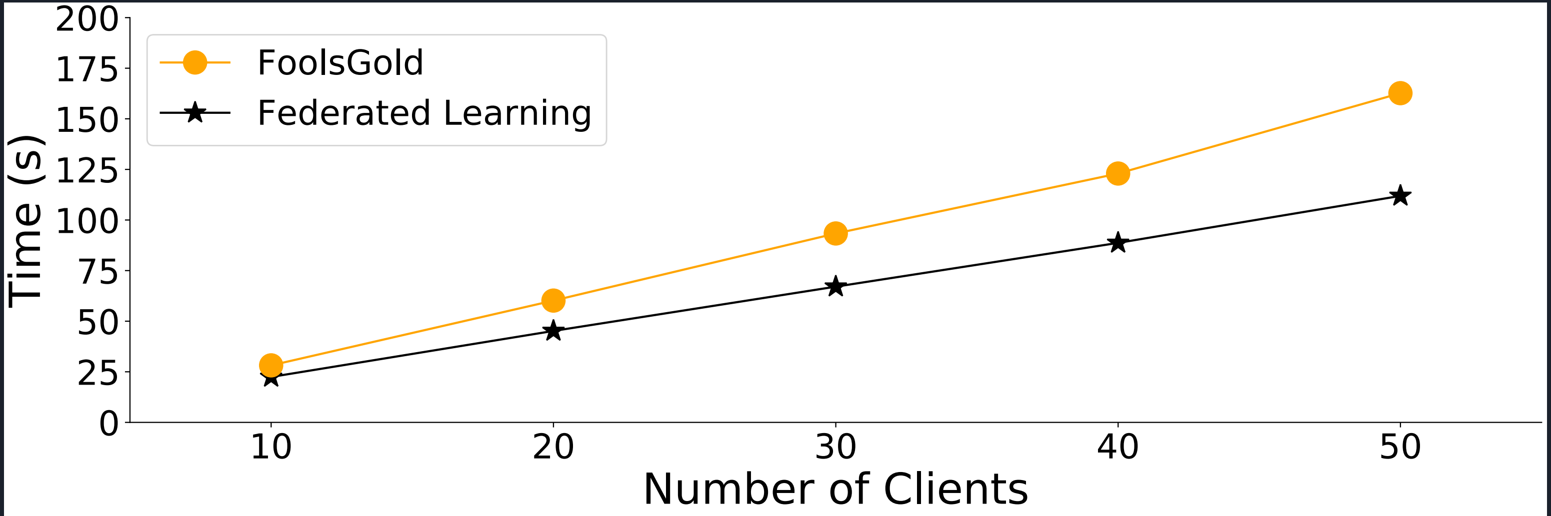
FoolsGold versus Krum [1]



- Multi-krum with byzantine # clients, f = 2
- Krum does poorly in non-IID setting (IID exp also shown)

[1] Blanchard et al. *"Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent"*. NIPS '17

Performance impact



- Train MNIST for 3K iterations
- Pairwise cosine similarity most expensive computation
- Python sub-optimal + better algorithms for cosine sim exist [1]

[1] Andoni et al. *"Practical and optimal lsh for angular distance,"* NIPS 2015

FoolsGold summary

- Federated learning: actively involves clients
- Sybil-based poisoning a concern
- Key idea: Use contribution similarity to detect sybils
- Simple algorithm that runs on server
- Effective for large number of sybils across 3 datasets

More info:
Fung et al. "*Mitigating Sybils in Federated Learning Poisoning*" Arxiv 2018

In this talk

- Introduction: cloud machine learning (ML)

Two projects with two points of view:

1. **Federated learning is here to stay**
   - FoolsGold : Countering sybil poisoning in fed. learning
2. **The future is decentralized**
   - Biscotti: P2P ML on the blockchain

Modern Large Scale ML Solutions

- Modern solutions: centralize data and centralize compute
  - Copy + store all data in a data centre and train on it

Problem: this is costly and **lacks privacy**

[1] Hazelwood et al. *"Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective"* HPCA 18

The Need for Privacy

- Data can be sensitive
  - Photos, location info, voice recordings.. our entire lives!
- Typically, a centralized service performs model training
  - Do we have to trust ____________ with our data?

large company

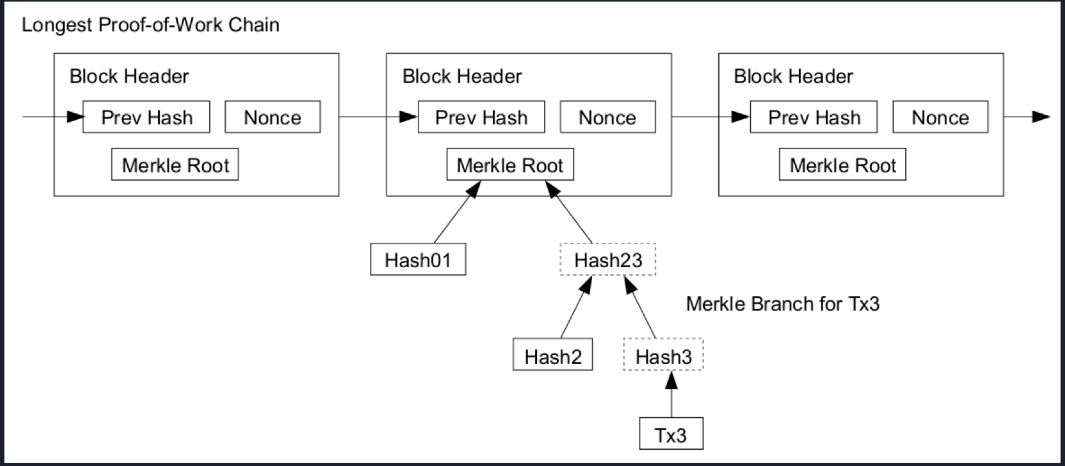
Private P2P Federated Learning

- Major issue for federated learning style systems:
  - Coordination and consistency of many clients
  - Security against Sybil attacks
- There is a modern solution that provides this in a peer to peer (P2P) network...

- …we just have to figure out privacy ☺

Blockchain Based Learning

BLOCKCHAIN



Use cases for such a system

- Health-care
    - Privacy regulations prohibit sharing of patient data
      Poisoning leads to inaccurate models/wrong diagnosis
- IoT
    - Personal data in IOT devices/sensors
    - Models for smart homes, self driving cars
    - Poisoned models could have disastrous consequences e.g., loss of lives
- Other fed. learning use cases without a trusted entity
    - Gboard -> Predicting next word in text messages

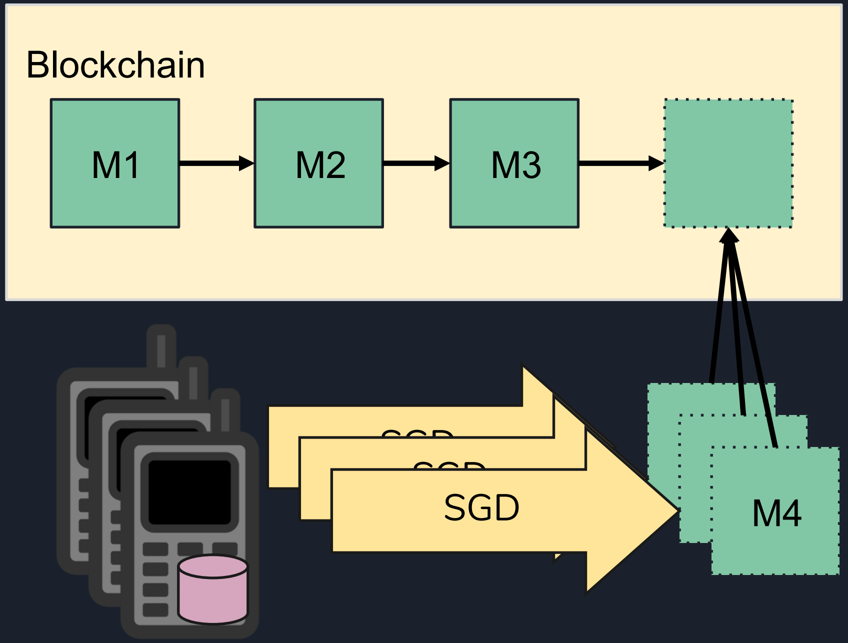
Blockchain Based Learning: how?

- We propose an alternative solution to distributed ML based on blockchain
  - Blockchain as a consensus protocol
  - Blockchain acts as shared state and coordinator
- Requires mapping of traditional blockchain ideas to ML
  - Proof of work/stake/something else?
  - SGD deltas dissemination
  - What does a block represent?
  - Block validation
  - Concurrency control (longest chain wins?)

Biscotti overview

Key ideas

1. Store global model structure in blockchain (secure aggregation)
2. Peers verify updates to defend against malicious updates
3. Use diff. priv. noise to protect updates



Verified updates agg. into blocks

Biscotti design overview

Goal	Mechanism
1. Support universal model types	1. Stochastic Gradient Descent (SGD) [1]
2. Peer to peer ML: no central coordinator	2. Blockchain
3. Prevent model poisoning	3. Verification through RONI [4]
4. Preserve privacy of the peers' data	4. Differentially private noise [3] , secret sharing [2]
5. Maintain defenses against sybils	5. Stake weighted VRFs [5] for 3 and 4.

[1] *Leon Bottou. "**Large scale Machine Learning with Stochastic Gradient Descent**" COMPSTAT 10*
[2] *Adil Shamir "**How to share a secret.**" ACM 1979*
[3] *Cynthia Dwork "**Differential Privacy**" ICALP 06*
[4] *Barreno et al "**The security of machine learning**" Machine Learning Journal 2010*
[5] *Micali et al "**Verifiable Random Functions**" FOCS 99*

Biscotti threat model

B. protects against:

- A malicious trusted entity. Biscotti does not assume a trusted component.
- Peers sending malicious updates to perform a poisoning attack[1] against a specific class.
- Peers colluding [2] with other peers to launch a poisoning attack.
- Peers colluding to perform a targeted attack to recover a victim's data.[3]

B. does not protect against:

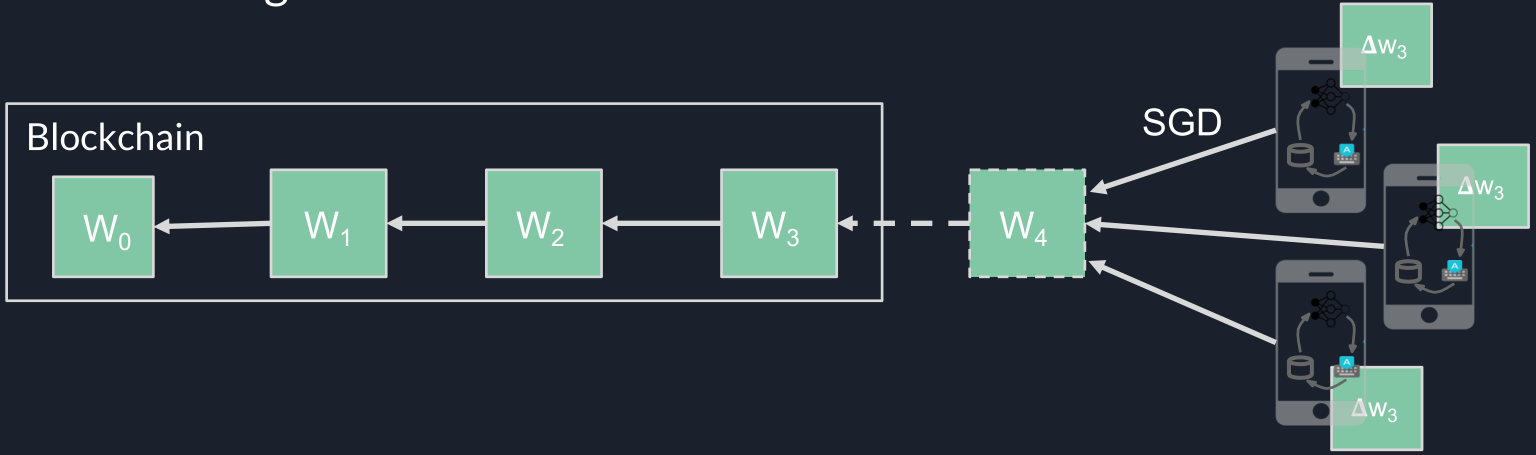
- Class-level information leakage from the global model (revealing all information about a specific class)
- Poisoning attacks that are unrelated to class-level information (targeted poisoning, backdoors, adversarial examples)
- Settings in which an adversary controls over half the resources in the system

We make fewer assumption about the malicious nature of peers unlike federated learning!

[1] *Barreno et al "**The security of machine learning**" Machine Learning Journal 2010*
[2] *Doucer et al **"The sybil attack"** IPTPS 01*
[3] *Hitaj et al "**Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning** " CCS17*

The easy part: SGD + Blockchain

- Each block stores a set of SGD updates from multiple peers
  - Each peer computes SGD using their blockchain state
  - With each block, the set of updates is added, updating the global model



Biscotti: design overview

Goal	Mechanism
1. Support universal model types	1. Stochastic Gradient Descent (SGD) [1]
2. Peer to peer ML: no central coordinator	2. Blockchain
3. Prevent model poisoning	**3. Verification through RONI [4]**
4. Preserve privacy of the peers' data	**4. Differentially private noise [3] , secret sharing [2]**
5. Maintain defenses against sybils	**5. Stake weighted VRFs [5] for 3 and 4.**

[1] *Leon Bottou. "**Large scale Machine Learning with Stochastic Gradient Descent**" COMPSTAT 10*

[2] *Adil Shamir "**How to share a secret.**" ACM 1979*

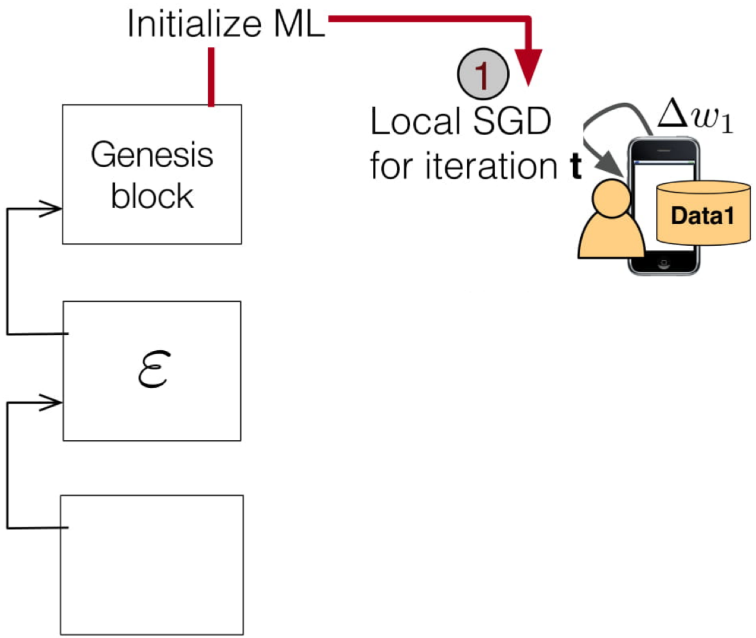
[3] *Cynthia Dwork **"Differential Privacy"** ICALP 06*

[4] *Barreno et al "**The security of machine learning**" Machine Learning Journal 2010*

[5] *Micali et al "**Verifiable Random Functions**" FOCS 99*

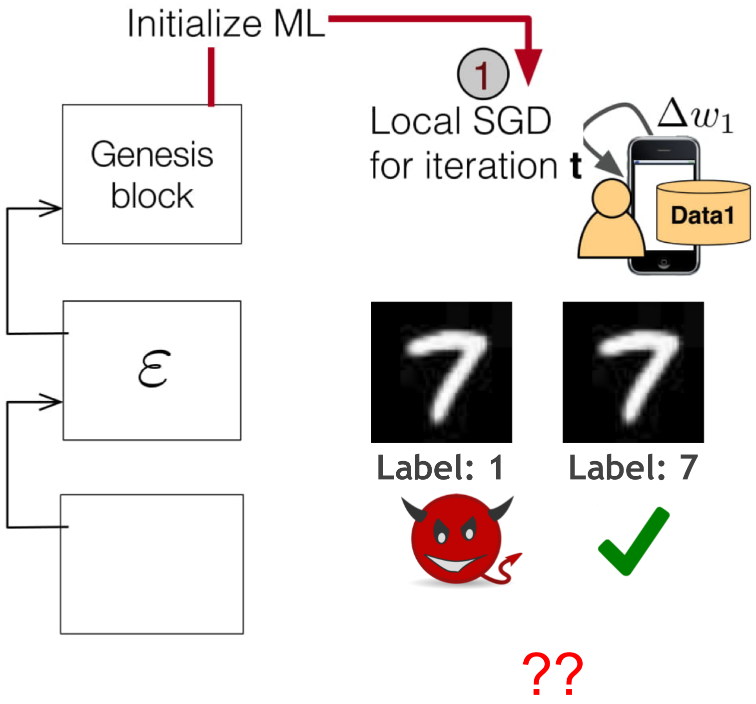
Biscotti: Defending against poisoning attacks

- A client computes an SGD update.



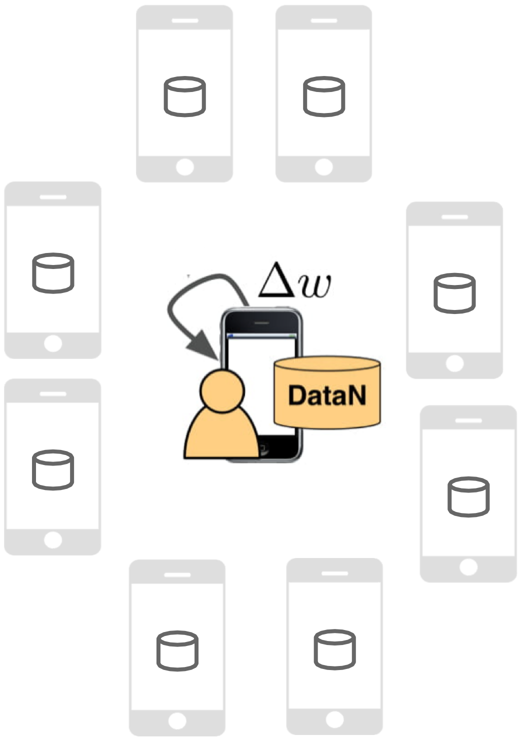
Biscotti: Defending against poisoning attacks

- A client computes an SGD update.
- How do we check if it is poisoned?



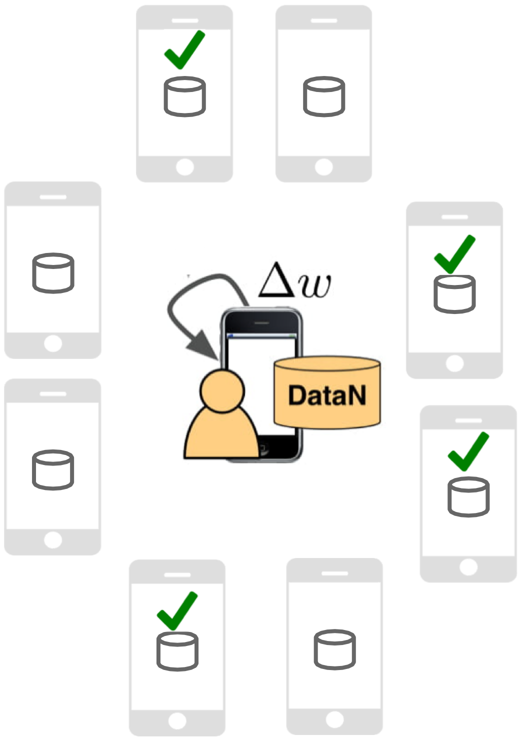
Biscotti: Defending against poisoning attacks

- A client computes an SGD update.
- How do we check if it is poisoned?
    - The P2P system has a wealth of verification data at each node's disposal



Biscotti: Defending against poisoning attacks

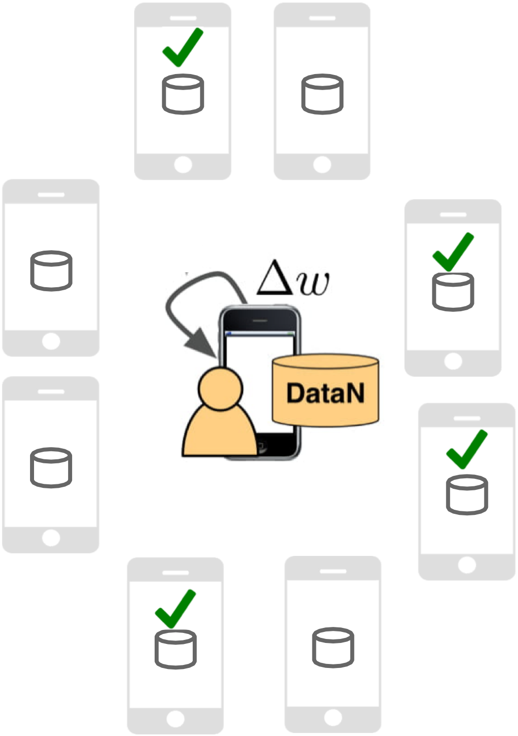
- A client computes an SGD update.
- How do we check if it is poisoned?
    - The P2P system has a wealth of verification data at each node's disposal
    - Select a subset of clients to act as a verification committee.



Biscotti: Defending against poisoning attacks

- A client computes an SGD update.
- How do we check if it is poisoned?
  - The P2P system has a wealth of verification data at each node's disposal
  - Select a subset of clients to act as a verification committee.

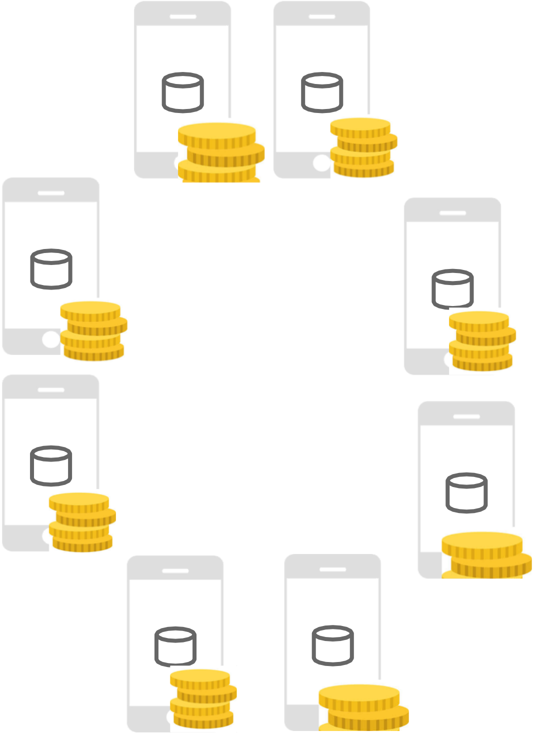
    Problem: How do we select this committee in a way that it is completely random and prevents collusion among clients?



Key idea: Proof of stake[1] and VRF[3]

- Each client has some **stake proportional to their contribution**
- Clients accumulate stake by contributing updates
- POS is a popular alternative to POW to achieve Byzantine fault tolerance (e.g. Algorand [2])
- In each iteration, a verifiable random function (VRF) uses stake + randomness to select a committee responsible for validating updates

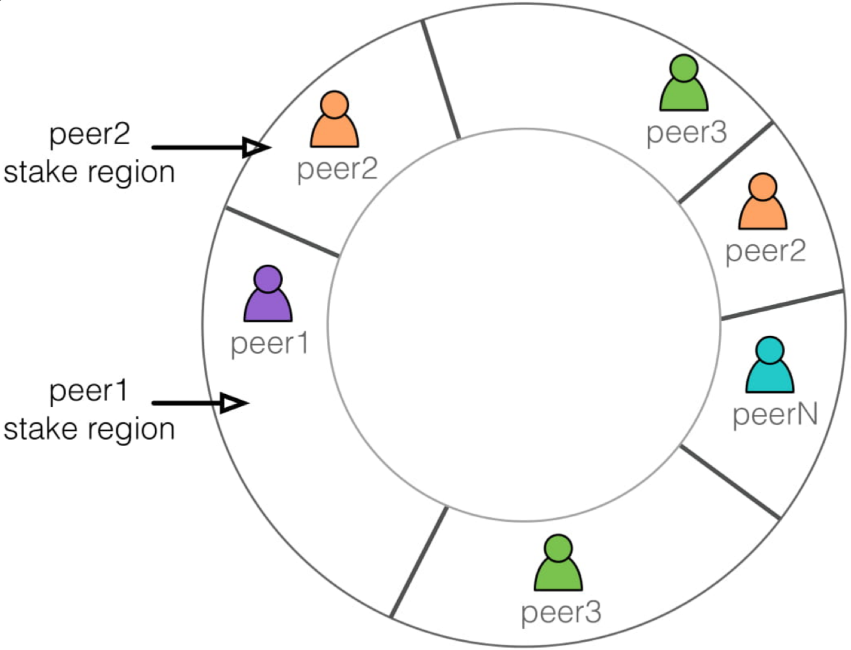
Assumption: At any time in the system the majority of stake in the system is honest.



[1] https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs
[2] *Gilad et al* "**Algorand: Scaling Byzantine Agreements for Cryptocurrencies**" *SOSP 17*
[3] *Micali et al* "**Verifiable random functions**" *SOSP 17*

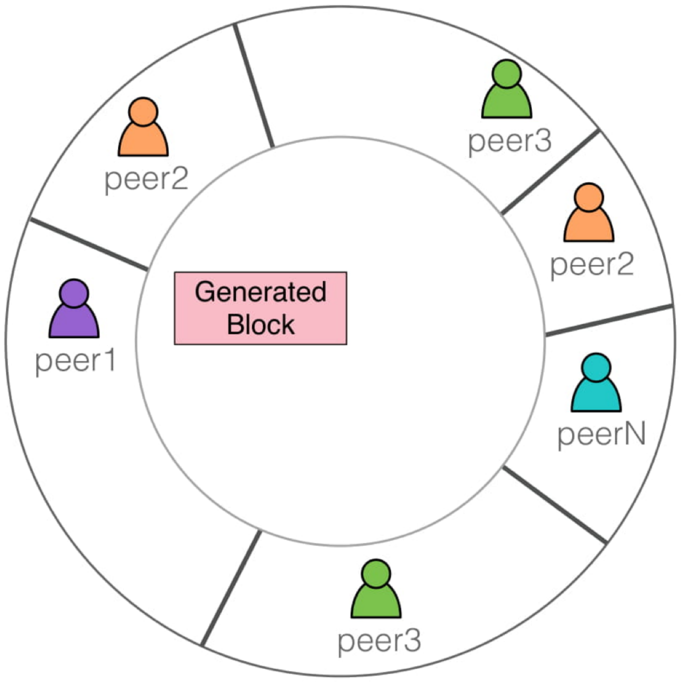
Biscotti: selecting verifiers

- Each client has a region over a hash ring weighted by **client stake** allocated via **consistent hashing**.



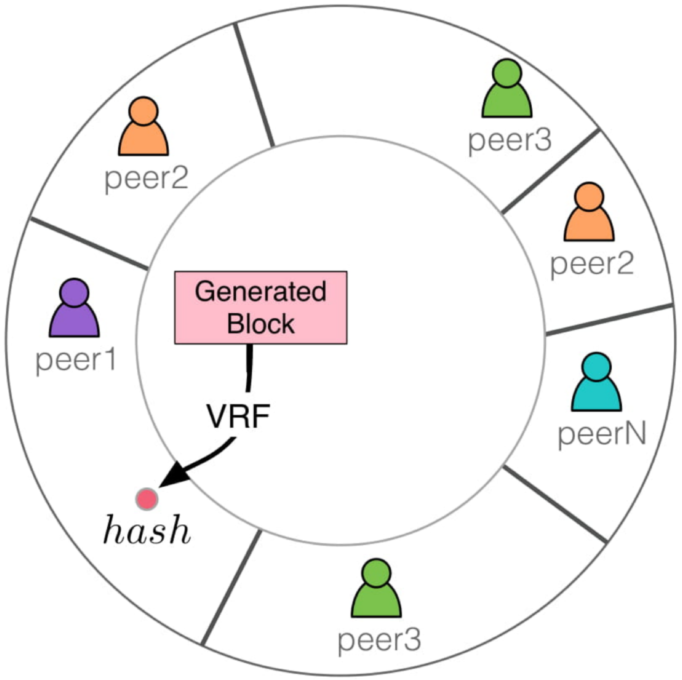
Biscotti: selecting verifiers

- Each client has a region over a hash ring weighted by **client stake** allocated via **consistent hashing**.
- A VRF hash using SHA-256 hash of the last generated block as the seed is computed.



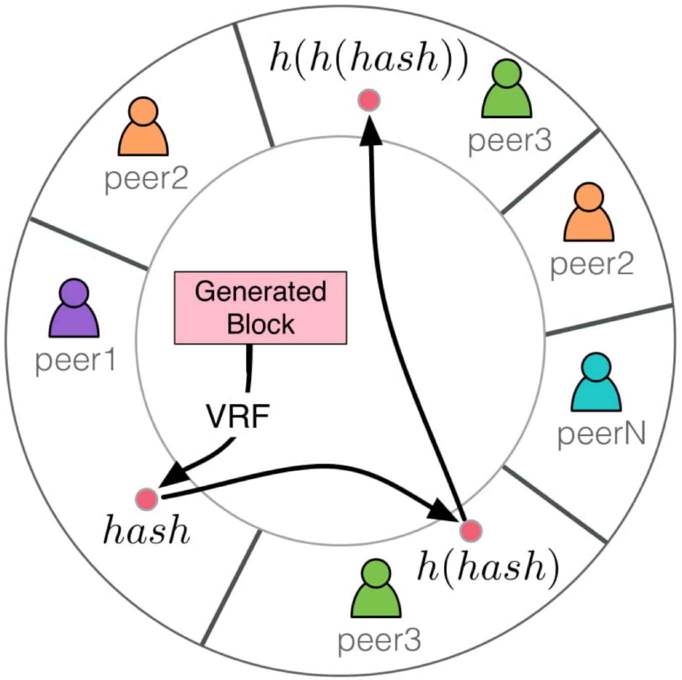
Biscotti: selecting verifiers

- Each client has a region over a hash ring weighted by **client stake** allocated via **consistent hashing**.
- A VRF hash using SHA-256 hash of the last generated block as the seed is computed.
- The peer in whose region the hash lies is selected as the verifier.



Biscotti: selecting verifiers

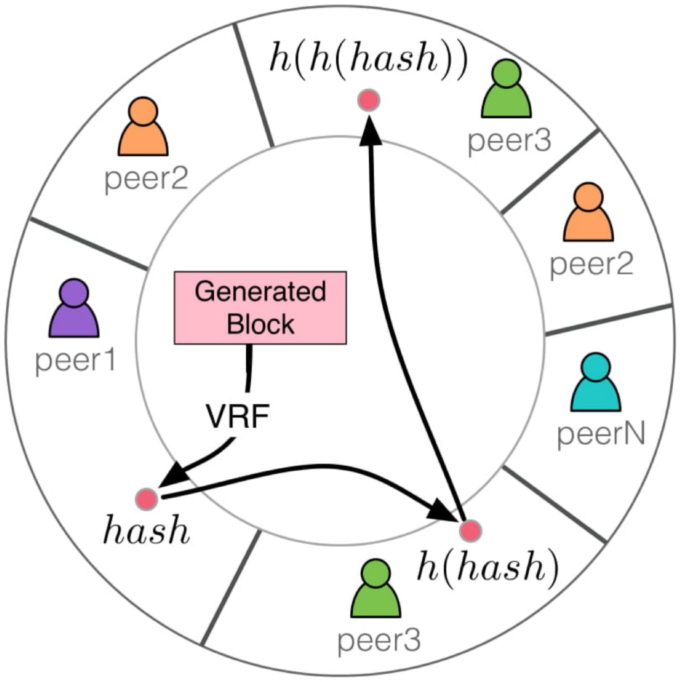
- Each client has a region over a hash ring weighted by **client stake** allocated via **consistent hashing**.
- A VRF hash using SHA-256 hash of the last generated block as the seed is computed.
- The peer in whose region the hash lies is selected as the verifier.
- The VRF hash is re-computed using the previous VRF hash to select multiple verifiers.



Biscotti: selecting verifiers

- Each client has a region over a hash ring weighted by **client stake** allocated via **consistent hashing**.
- A VRF hash using SHA-256 hash of the last generated block as the seed is computed.
- The peer in whose region the hash lies is selected as the verifier.
- The VRF hash is re-computed using the previous VRF hash to select multiple verifiers.

The verifier selection is completely random and cannot be manipulated by any client

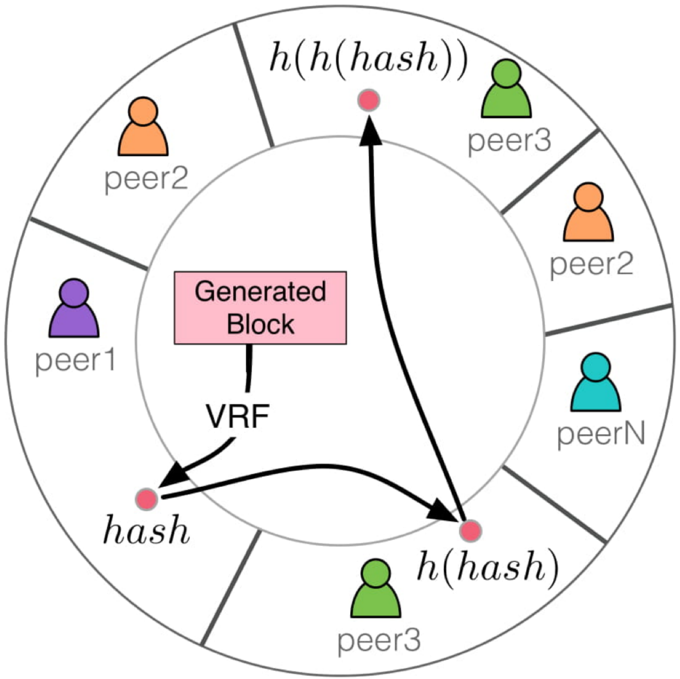


Biscotti: selecting verifiers

- Each client has a region over a hash ring weighted by **client stake** allocated via **consistent hashing**.
- A VRF hash using SHA-256 hash of the last generated block as the seed is computed.
- The peer in whose region the hash lies is selected as the verifier.
- The VRF hash is re-computed using the previous VRF hash to select multiple verifiers.

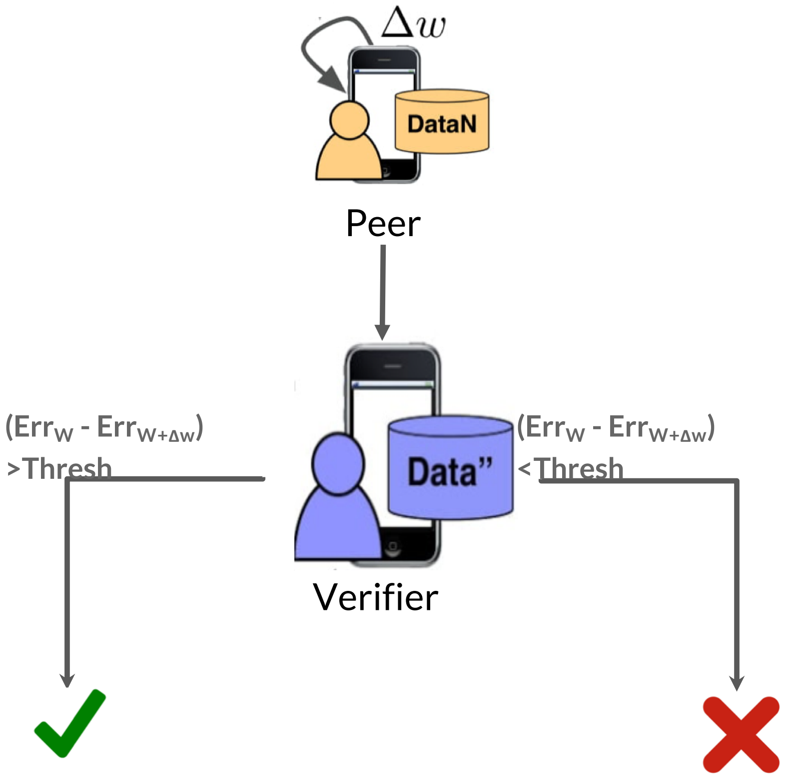
The verifier selection is completely random and cannot be manipulated by any client

Once selected, how do the verifiers check updates for poisoning?



Biscotti: Verification using RONI [1]

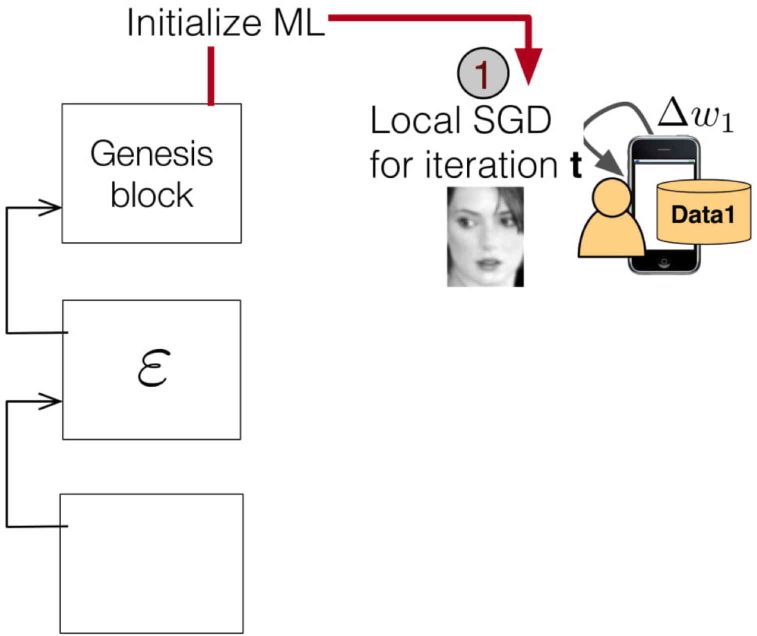
- Verifier determines whether the update improves performance of the model w.r.t his own data.
- Measures validation error of the current state of the model on his data (Err_W).
- Measures validation error of the model + update on his data ($Err_{W+\Delta W}$).
- If (Err_W - $Err_{W+\Delta W}$) > Threshold
  - Accept Update
- Otherwise:
  - Reject Update



[1] *Barreno et al "**The security of machine learning**" Machine Learning Journal 2010*

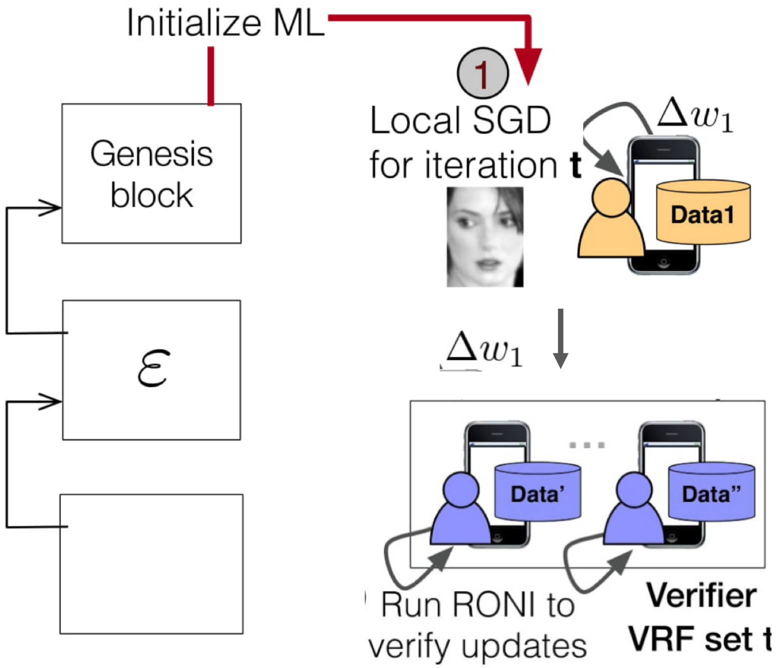
Biscotti: a robust poisoning defense

- The client computes a SGD update.



Biscotti: a robust poisoning defense

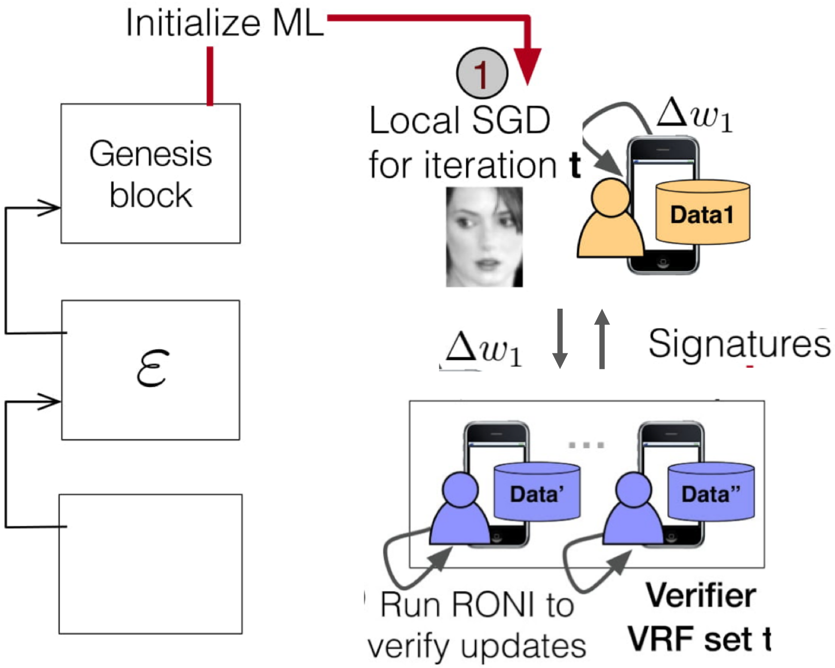
- The client computes a SGD update.
- The client sends the update to each verifier selected for RONI [1] verification.



[1] *Barreno et al “**The security of machine learning**” Machine Learning Journal 2010*

Biscotti: a robust poisoning defense

- The client computes a SGD update.
- The client sends the update to each verifier selected for RONI [1] verification.
- Verifiers return signatures if the update passes, and an update is accepted if it receives a majority of updates.



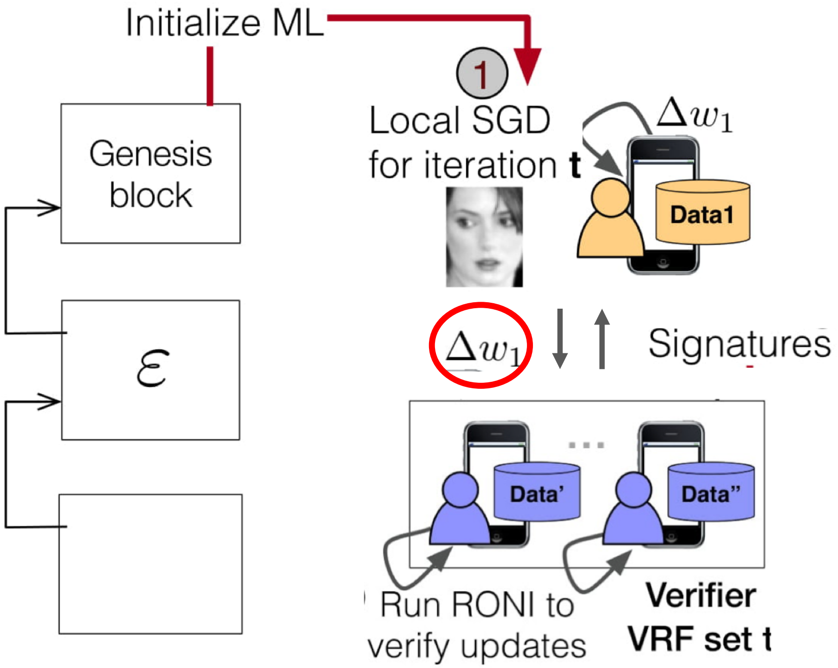
[1] *Barreno et al* ***"The security of machine learning"*** *Machine Learning Journal 2010*

Biscotti: a robust poisoning defense

- The client computes a SGD update.
- The client sends the update to each verifier selected for RONI [1] verification.
- Verifiers return signatures if the update passes, and an update is accepted if it receives a majority of updates.

Problem remains: No Privacy!

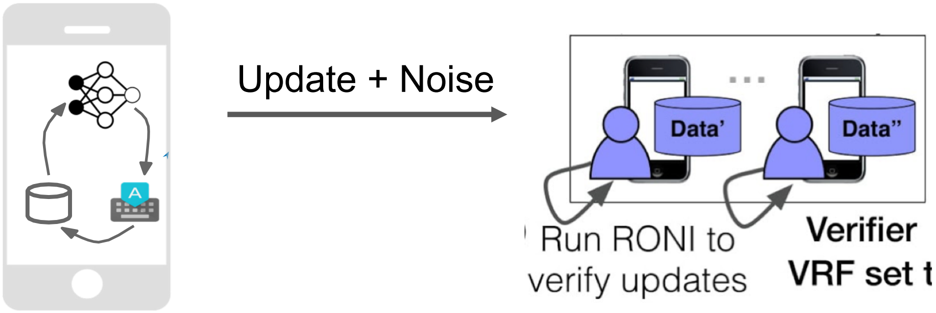
Inversion Attack [2]



[1] *Barreno et al* ***"The security of machine learning"*** *Machine Learning Journal 2010*
[2] *Hitaj et al* ***"Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning "*** *CCS17*

Biscotti: Adding privacy using noise

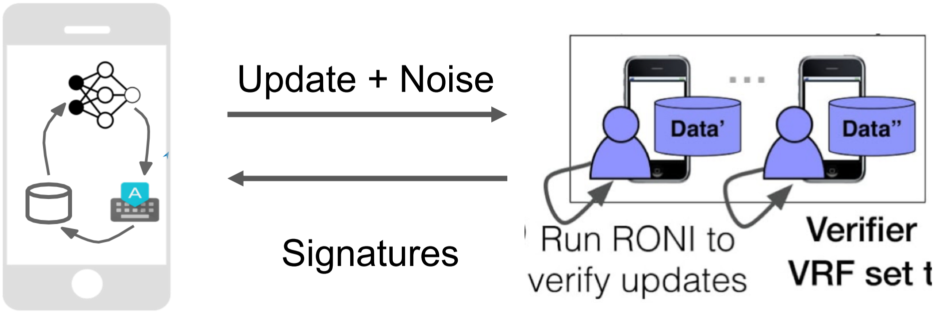
- Since the SGD update cannot be revealed to the verifier, the update has differentially private noise [1] added to it.
  - With noise added, the data gets obfuscated.



[1] *Cynthia Dwork* ***"Differential Privacy"*** *ICALP 06*

Biscotti: Adding privacy using noise

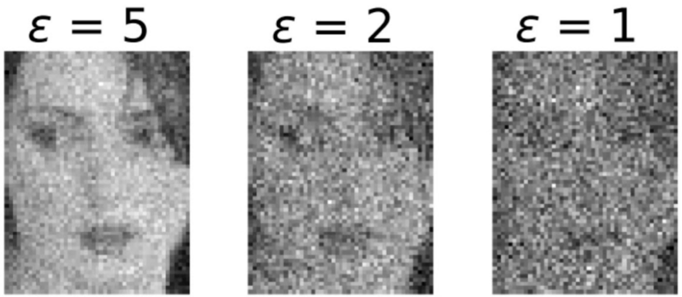
- Since the SGD update cannot be revealed to the verifier, the update has differentially private noise [1] added to it.
  - With noise added, the data gets obfuscated.
  - Verifiers verify the noisy update, and return signatures if they pass RONI.



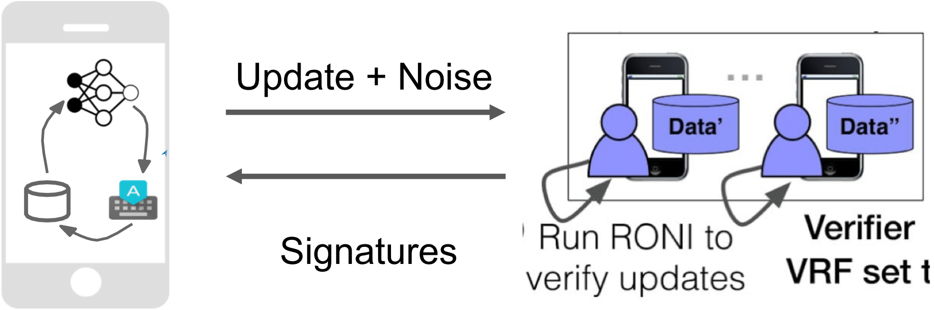
[1] *Cynthia Dwork **"Differential Privacy"** ICALP 06*

Biscotti: Adding privacy using noise

- Since the SGD update cannot be revealed to the verifier, the update has differentially private noise [1] noise added to it.
    - With noise added, the data gets obfuscated.
    - Verifiers verify the noisy update, and return signatures if they pass RONI.



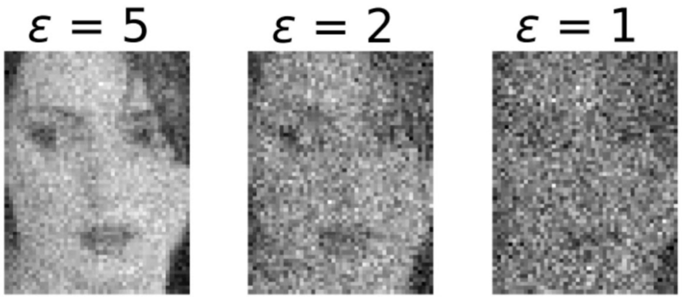
Problem remains: noise harms utility



[1] *Cynthia Dwork **"Differential Privacy"** ICALP 06*

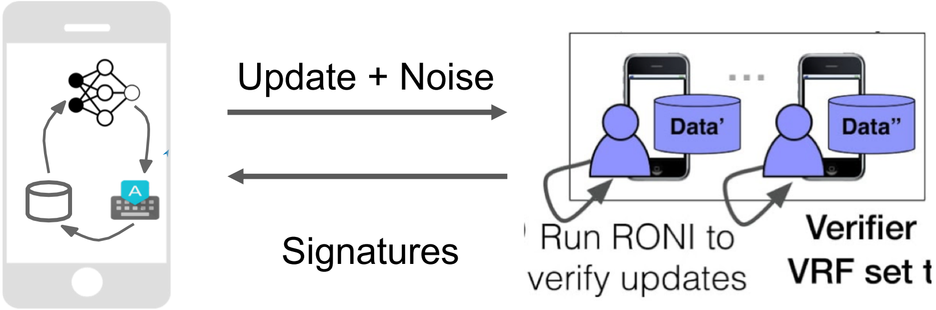
Biscotti: Adding privacy using noise

- Since the SGD update cannot be revealed to the verifier, the update has noise added to it.
  - With noise added, the data gets obfuscated.
  - Verifiers verify the noisy update, and return signatures if they pass RONI.



Problem remains: noise harms utility

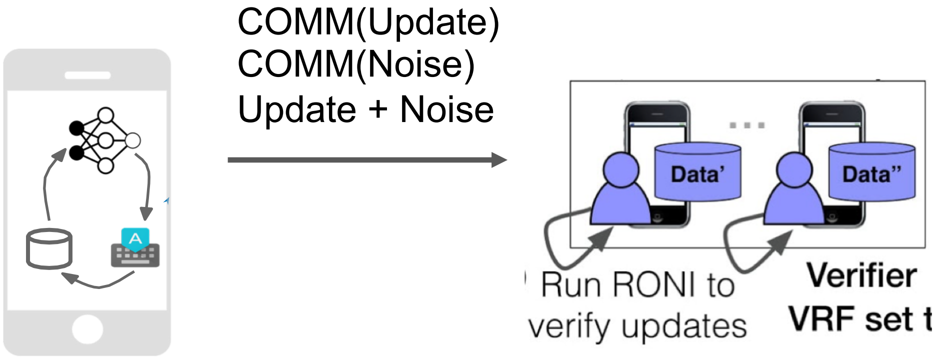
Verifier signs the updates without the noise.



[1] *Cynthia Dwork **"Differential Privacy"** ICALP 06*

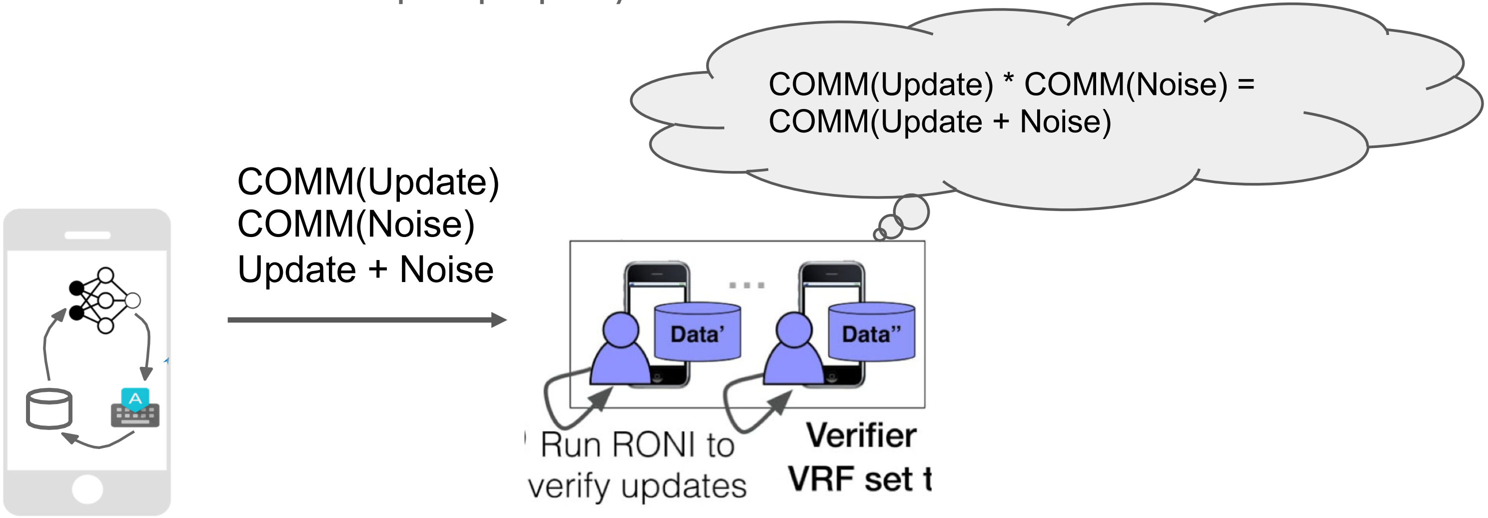
Biscotti: Adding privacy with commitments

- Peer uses commitments to hide individual updates and noise.
  - Reveals commitments and the sum to verifier



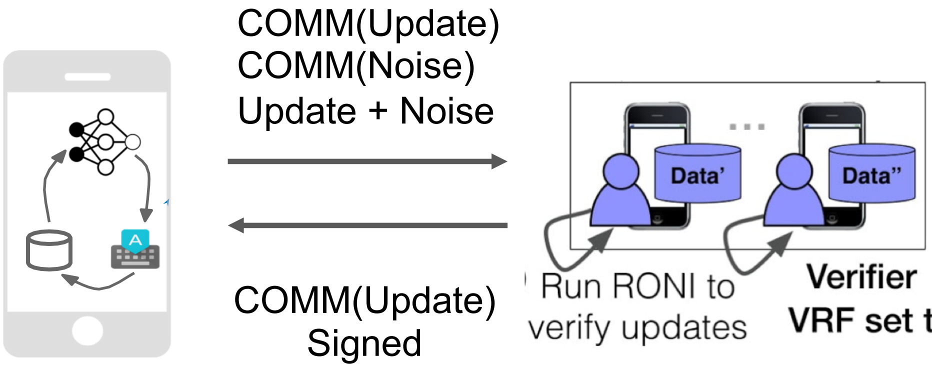
Biscotti: Adding privacy with commitments

- Peer uses commitments to hide individual updates and noise.
  - Reveals commitments and the sum to verifier
- Trivial for verifiers to verify that commitments add up to the revealed value because of homomorphic property



Biscotti: Adding privacy with commitments

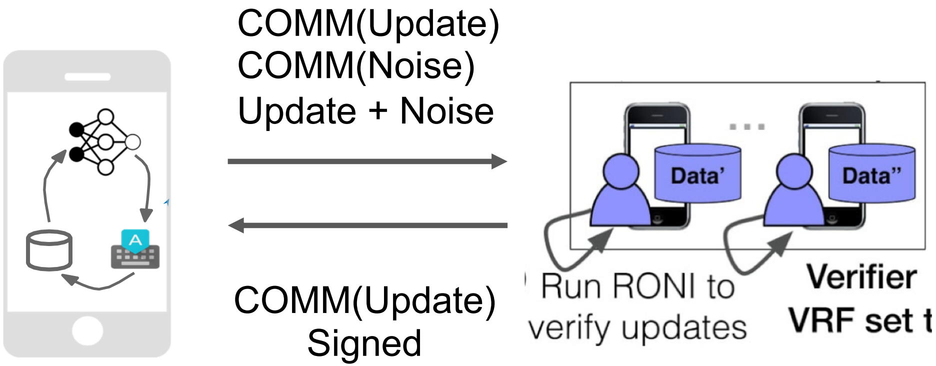
- Peer uses commitments to hide individual updates and noise.
    - Reveals commitments and the sum to verifier
- Trivial for verifiers to verify that commitments add up to the revealed value because of homomorphic property
- Returns a signature of the **commitment to the update** if verification passes



Biscotti: Adding privacy with commitments

- Peer uses commitments to hide individual updates and noise.
  - Reveals commitments and the sum to verifier
- Trivial for verifiers to verify that commitments add up to the revealed value because of homomorphic property
- Returns a signature of the **commitment to the update** if verification passes

This bypasses verification and goes into the block!

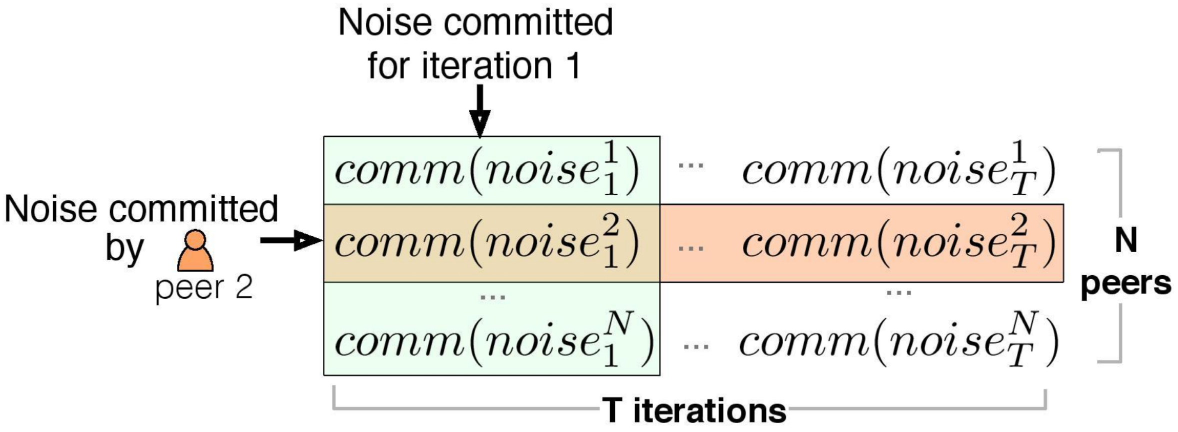


Problem:

Noise can be used to make poisoned update look good after addition of noise

Biscotti: Pre-committing noise

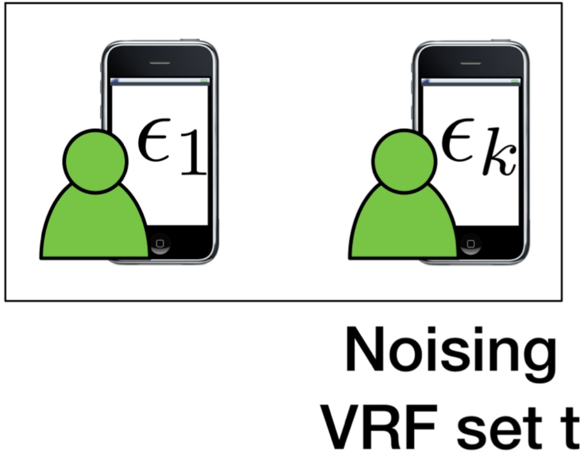
- We cannot give a peer control of the noise used for their update
  - Could be used to manipulate verification result
- Solution: Noise is pre-committed in the genesis block in a matrix.
  - SGD training usually done for a predefined number of iterations. [1]
  - Pre-commit for N peers over T iterations. . Can't go back on commitment
- Only the **commitments** to the noise are published



[1] *Leon Bottou. "**Large scale Machine Learning with Stochastic Gradient Descent**" COMPSTAT 10*

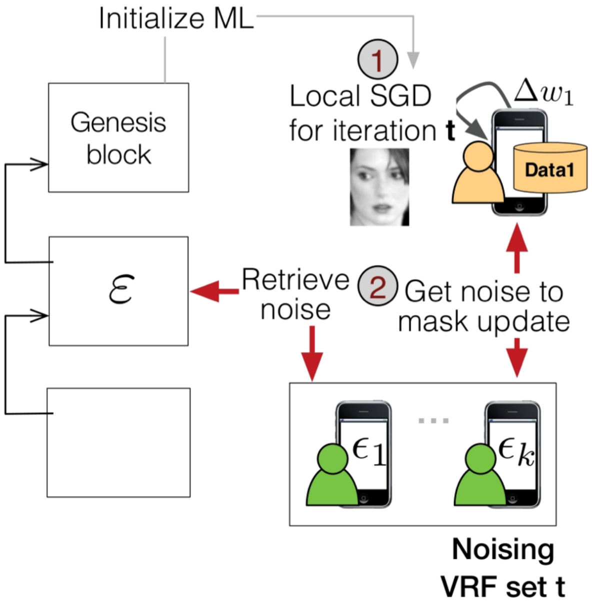
Biscotti: Adding the pre-committed noise and revealing to verifiers

- Each peer computes a VRF that outputs the indices of noise to use
  - The index selects from the precommitted values, and is unique to each peer.



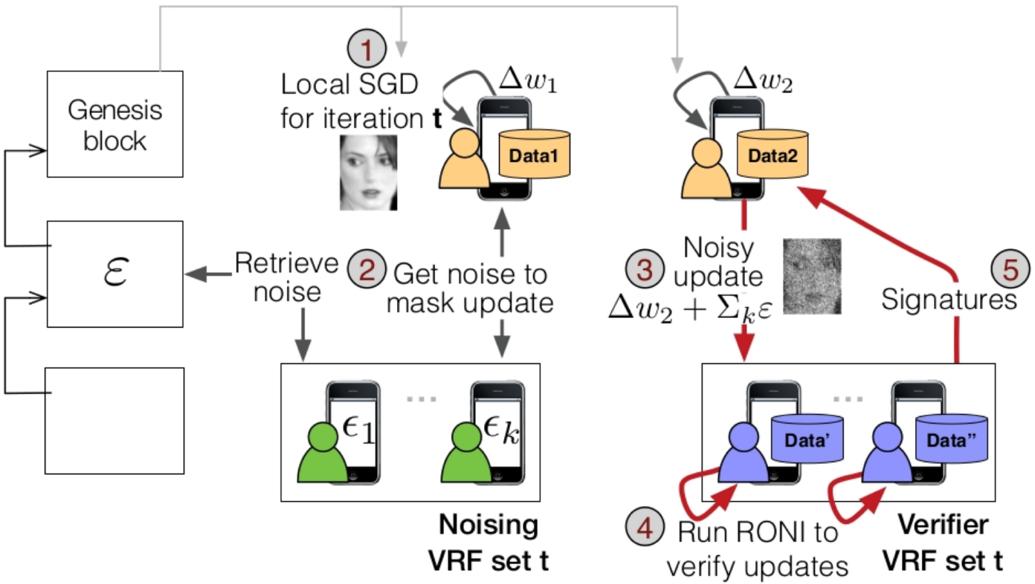
Biscotti: Adding the pre-committed noise and revealing to verifiers

- The peer retrieves the noise vectors from each noising peer
  - The noising peer committee is unique to each client.
  - Noise can be verified by matching with corresponding commitment in the genesis block



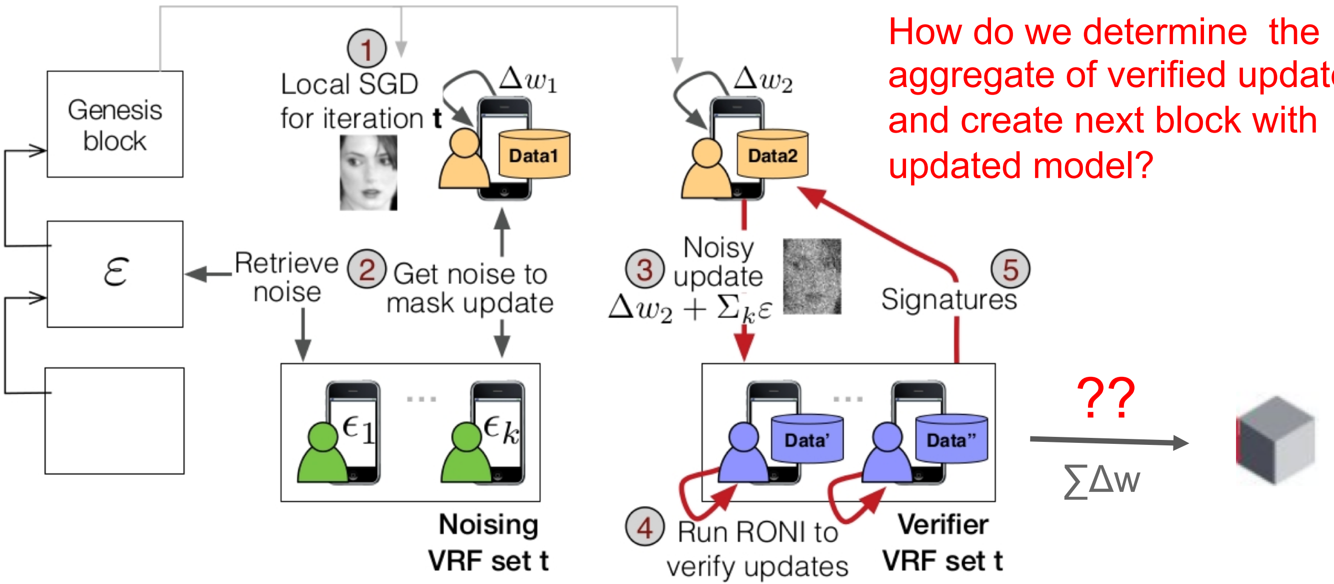
Biscotti: Adding the pre-committed noise and revealing to verifiers

- The peer combines the precommitted noise vectors from the respective noising peers
  - Sends commitment to the update, commitments to the noise vectors, and noisy update
  - Peer collects signatures from verifiers



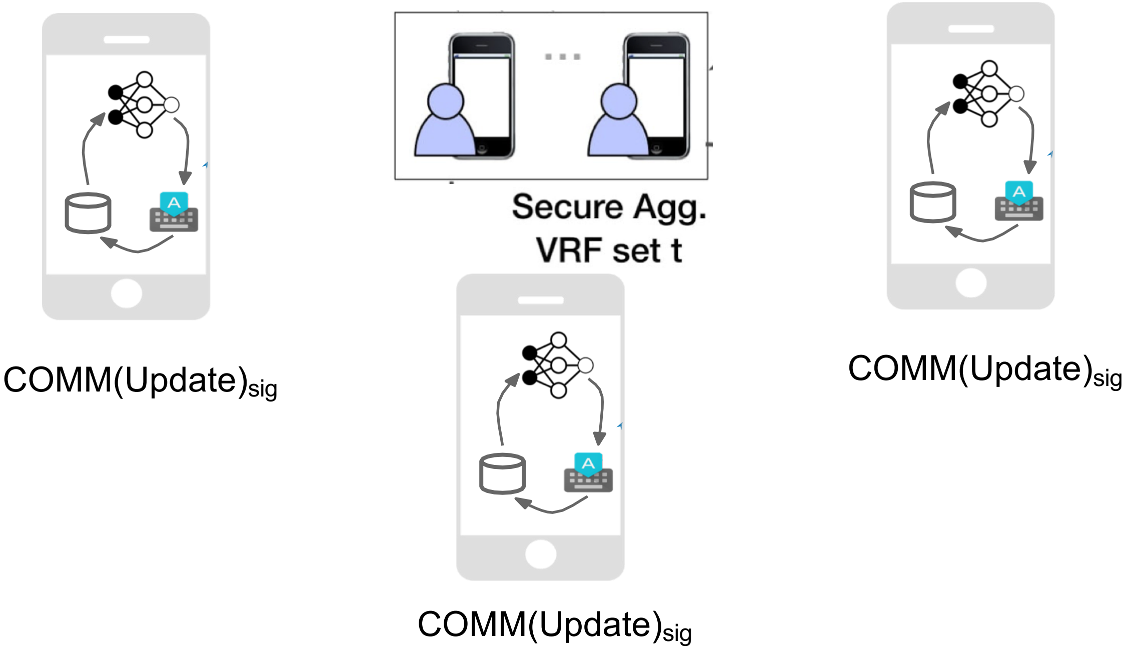
Biscotti: Adding the pre-committed noise and revealing to verifiers

- The peer combines the precommitted noise vectors from the respective noising peers
  - Sends commitment to the update, commitments to the noise vectors, and noisy update
  - Peer collects signatures from verifiers



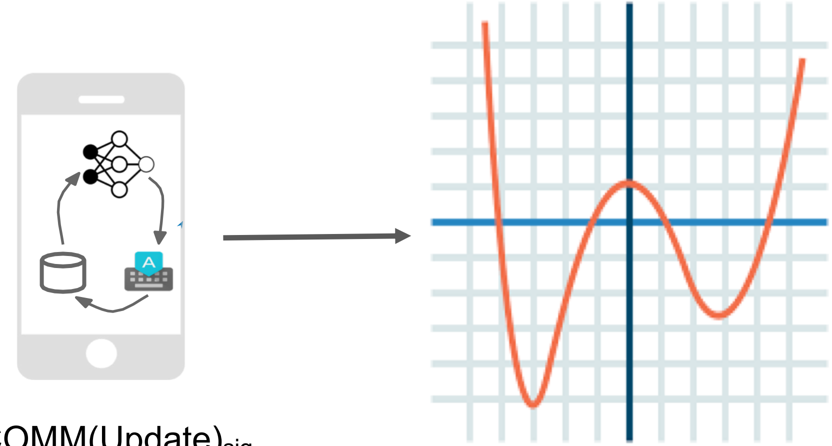
Biscotti: Aggregation and Block Generation

- Similar to verification, an aggregation committee is selected for creating the new block using another stake-weighted VRF function



Biscotti: Aggregation using Shamir secret sharing[1]

- Each peer’s update can be considered to be a d-degree polynomial.

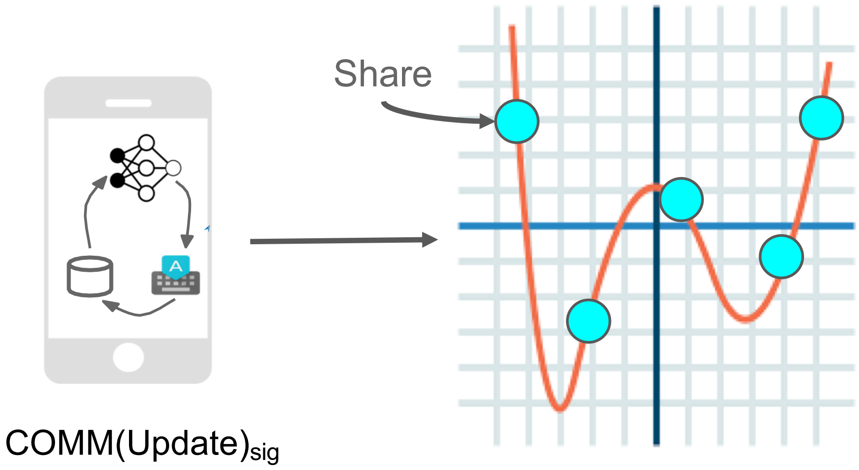


COMM(Update)$_{sig}$

Update as a polynomial

Biscotti: Aggregation using Shamir secret sharing[1]

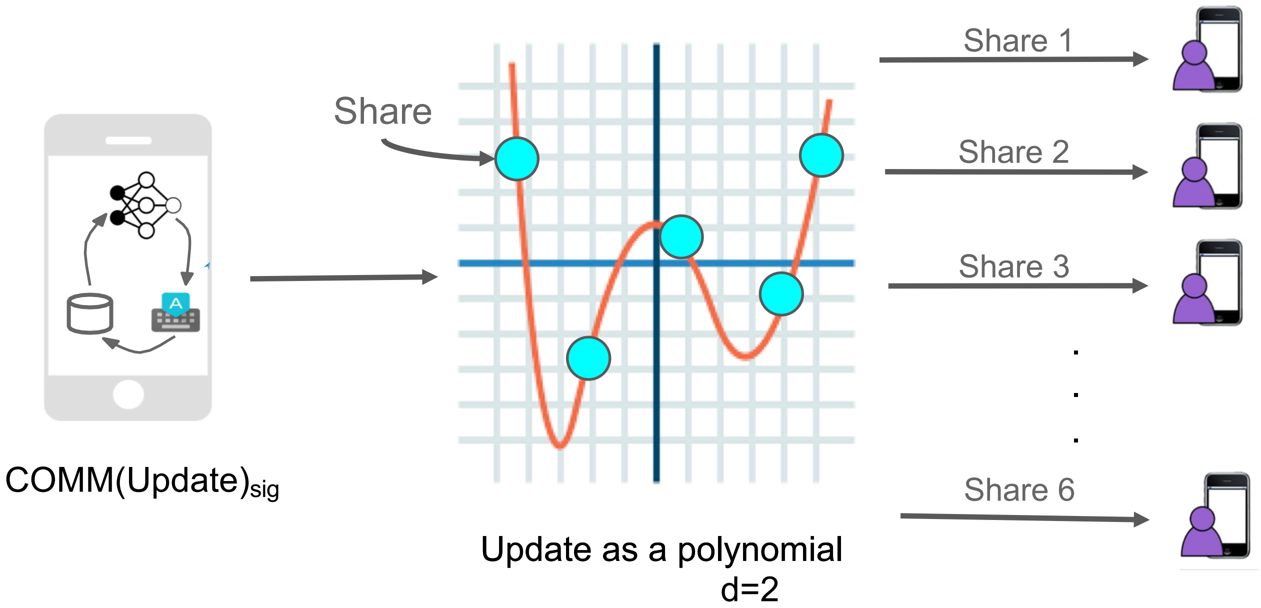
- Each peer's update can be considered to be a d-degree polynomial.
- Update can be broken into n-secret shares such d+1 are needed to reconstruct.



Update as a polynomial
d=2

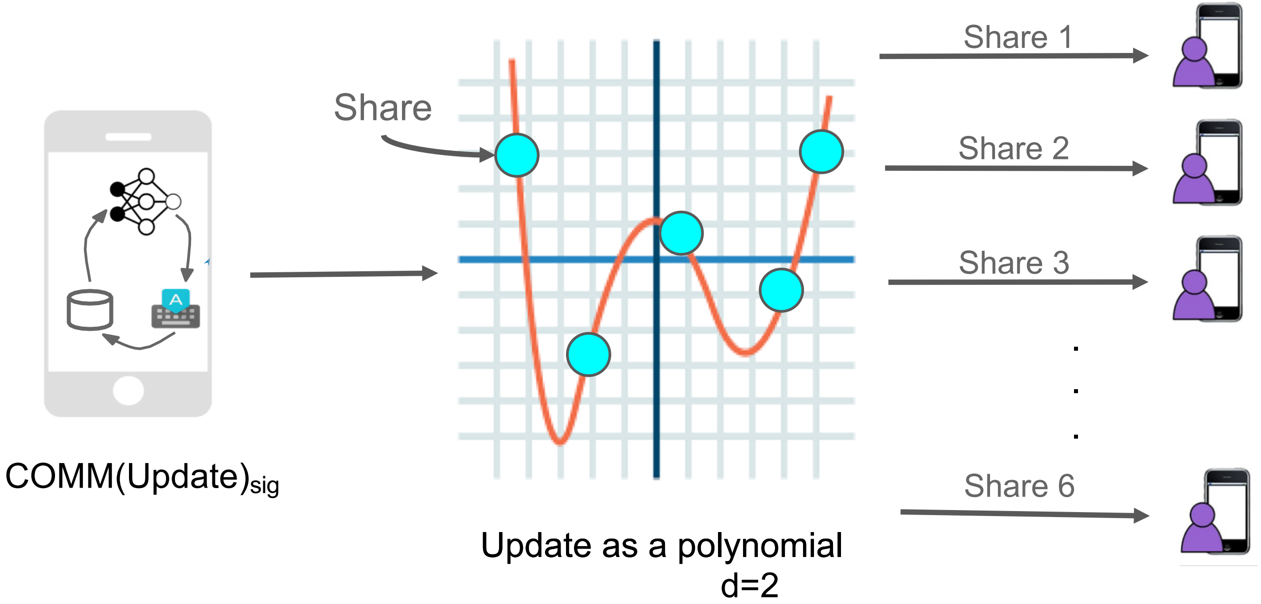
Biscotti: Aggregation using Shamir secret sharing[1]

- Each peer's update can be considered to be a d-degree polynomial.
- Update can be broken into n-secret shares such d+1 are needed to reconstruct.
- These shares are distributed among aggregators equally such that n=2*(d+1)



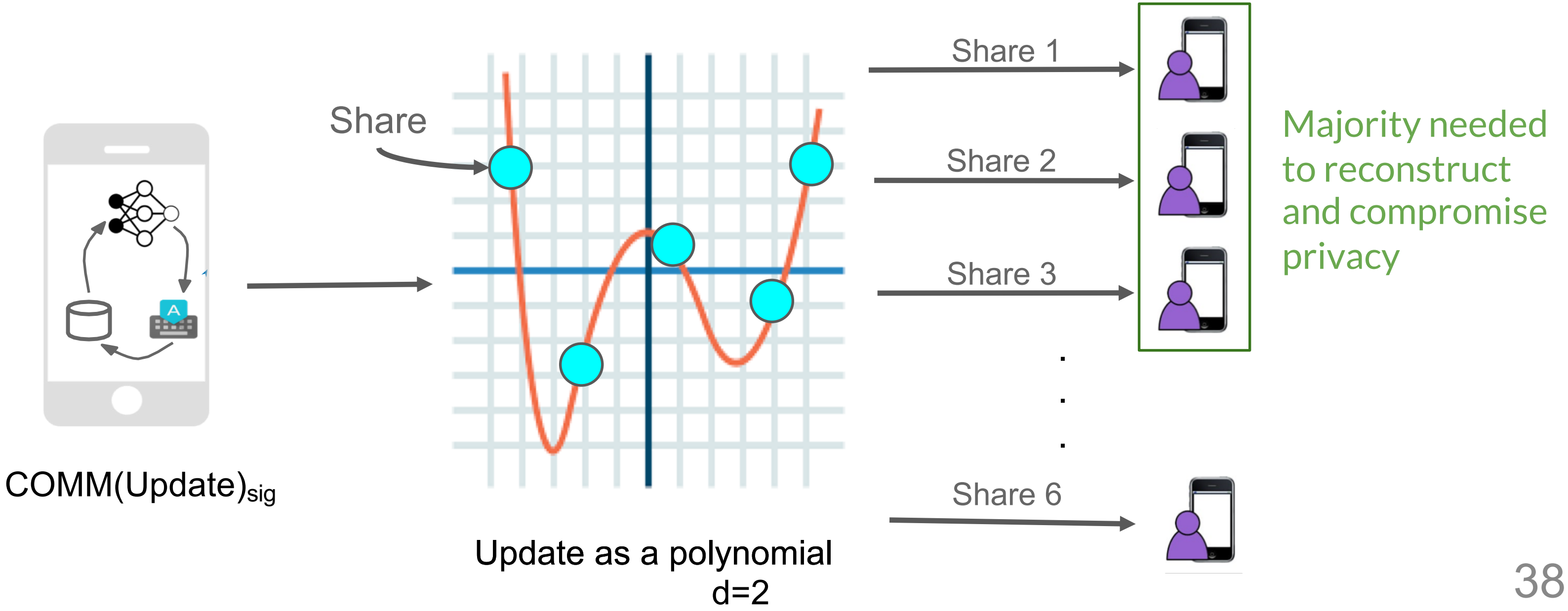
Biscotti: Aggregation using Shamir secret sharing[1]

- Each peer's update can be considered to be a d-degree polynomial.
- Update can be broken into n-secret shares such d+1 are needed to reconstruct.
- These shares are distributed among aggregators equally such that n=2*(d+1)



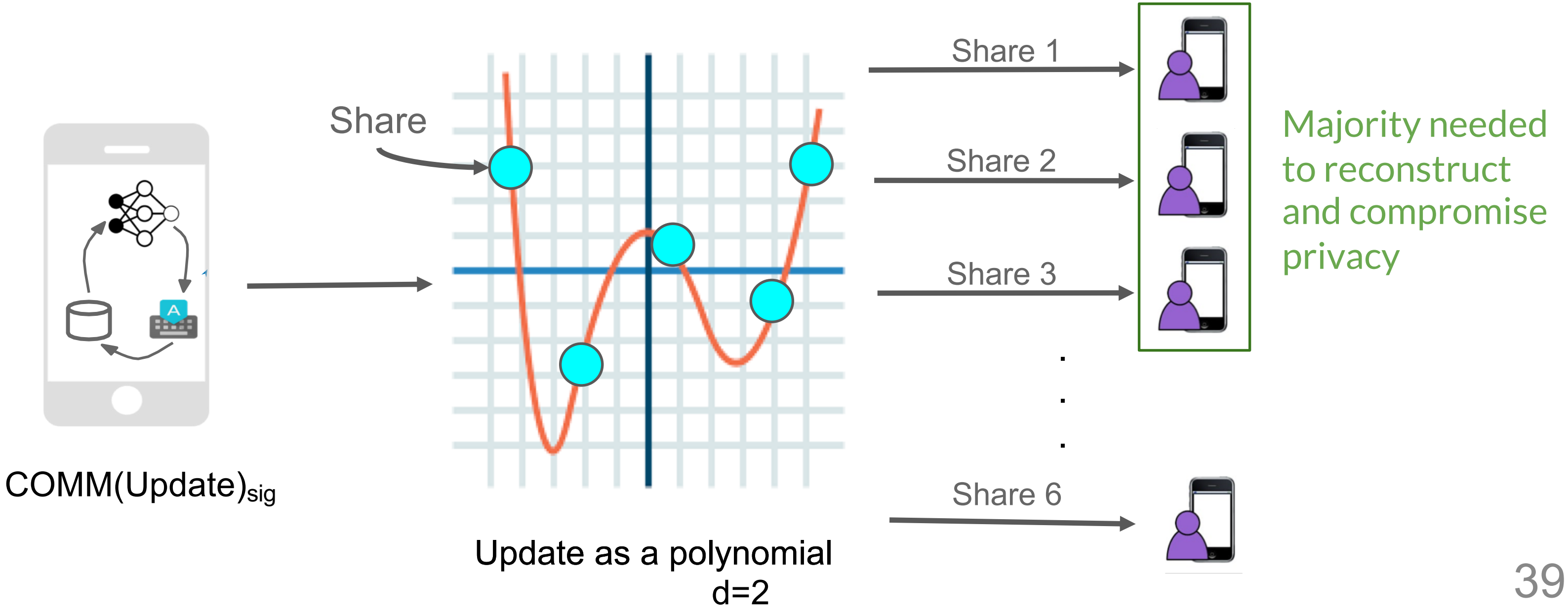
Biscotti: Aggregation using Shamir secret sharing[1]

- Each peer's update can be considered to be a d-degree polynomial.
- Update can be broken into n-secret shares such d+1 are needed to reconstruct.
- These shares are distributed among aggregators equally such that n=2*(d+1)



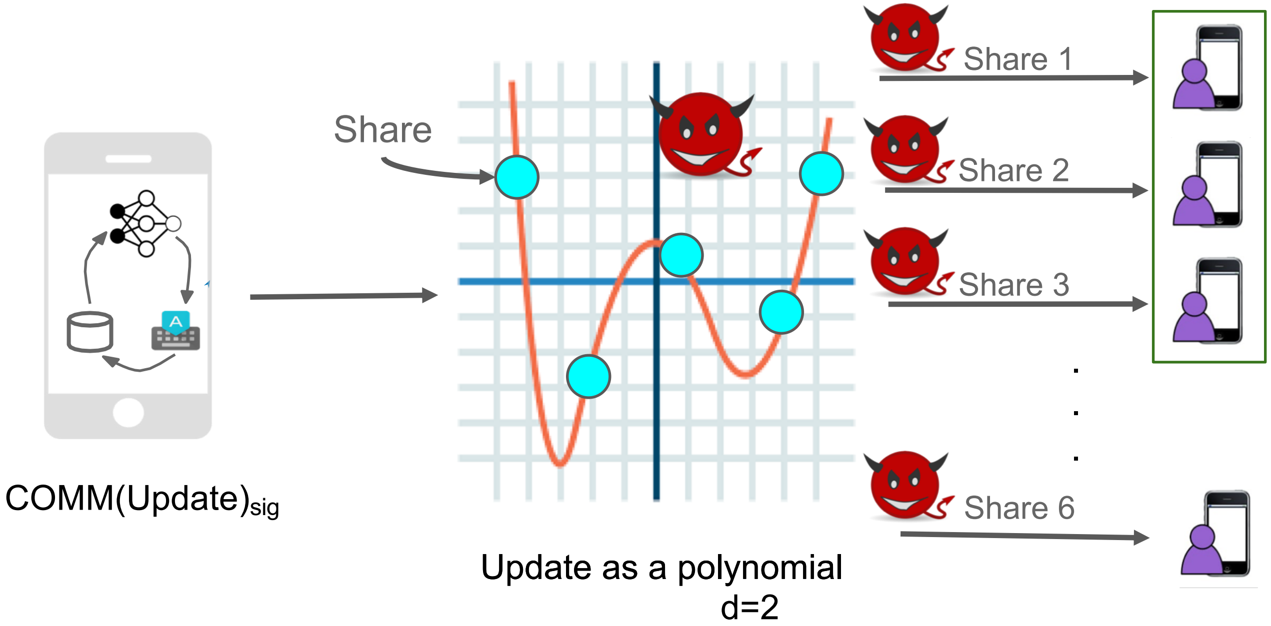
Biscotti: Aggregation using Shamir secret sharing[1]

- Each peer's update can be considered to be a d-degree polynomial.
- Update can be broken into n-secret shares such d+1 are needed to reconstruct.
- These shares are distributed among aggregators equally such that n=2*(d+1)



Biscotti: Aggregation using Shamir secret sharing[1]

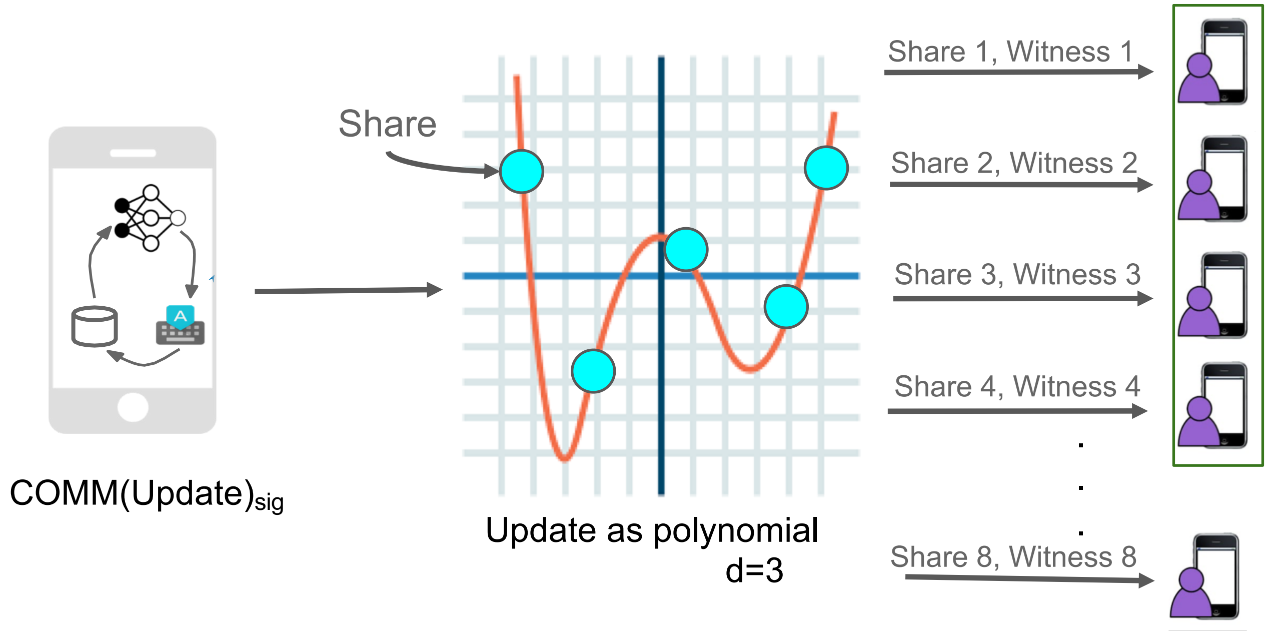
- Each peer's update can be considered to be a d-degree polynomial.
- Update can be broken into n-secret shares such d+1 are needed to reconstruct.
- These shares are distributed among miners equally such that n=2*(d+1)



The peer could provide shares for a malicious update.

Biscotti: Aggregation using Shamir secret sharing[1]

- Each share is accompanied by a witness that proves in zk the validity of the share.



Biscotti: Aggregation using Shamir secret sharing[1]

- Each share is accompanied by a witness that proves in zk the validity of the share.

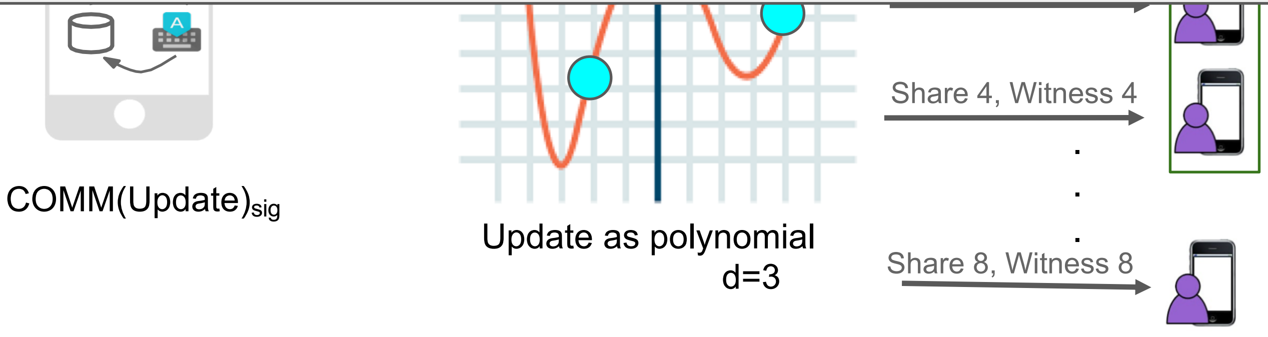
What is a witness?

Given an update polynomial $\Delta w(x)$ and a secret share $(\mathbf{i}, \mathbf{\Delta w}(\mathbf{i}))$

- A witness is a commitment to a polynomial ϕ(x) => COMM(ϕ(x))

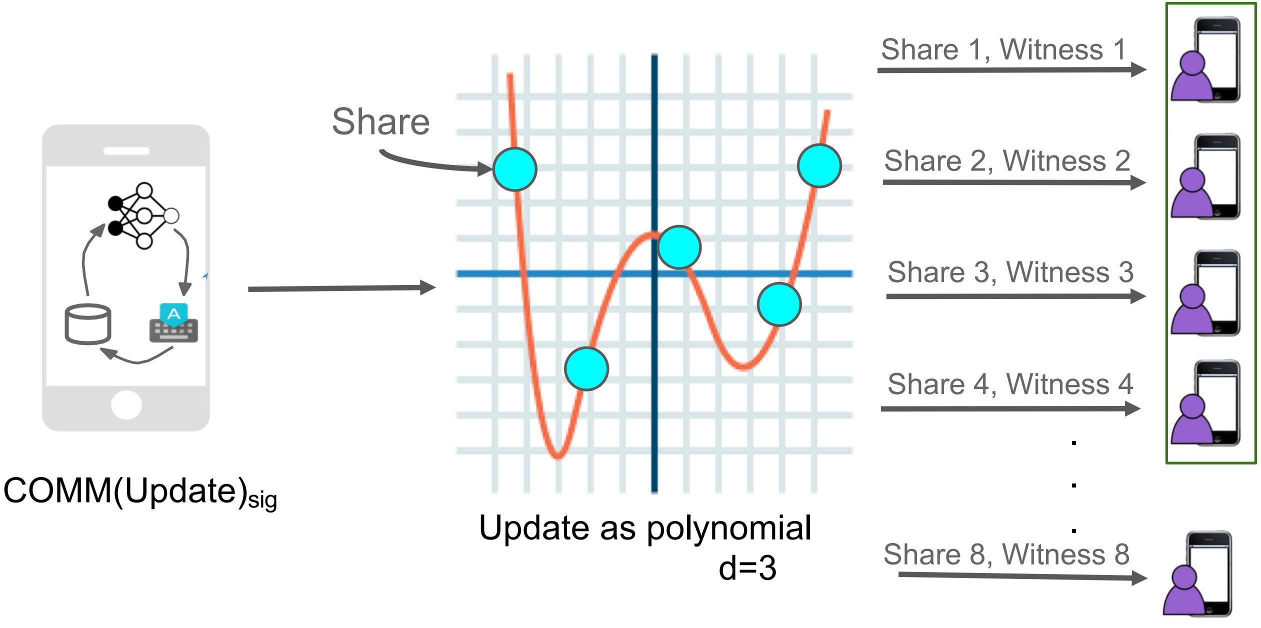
$$\phi(x) = \frac{\Delta w(x) - \Delta w(i)}{x - i}$$

- The witness polynomial divides the update polynomial by the secret share.



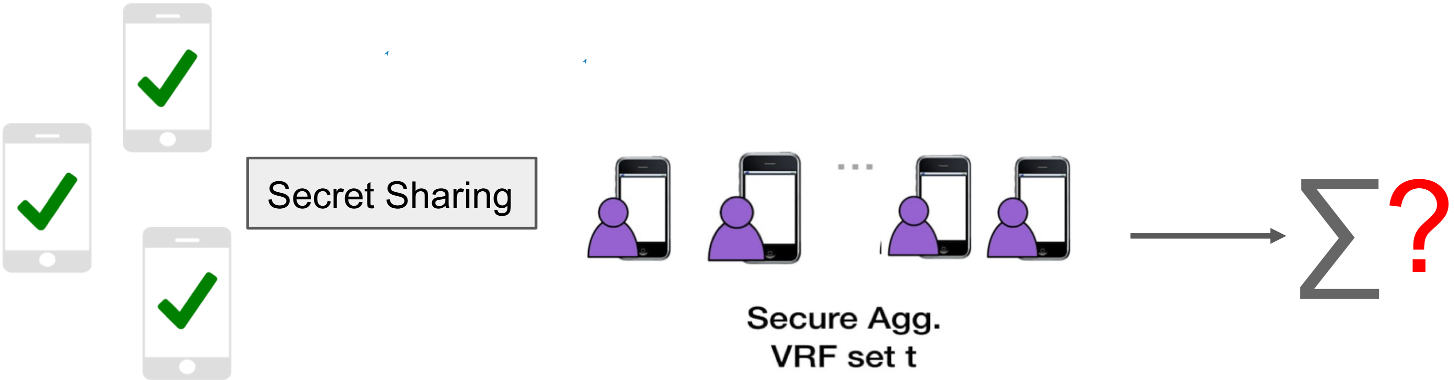
Biscotti: Aggregation using Shamir secret sharing[1]

- Each share is accompanied by a witness that proves in zk the validity of the share.
- Using the divisibility property of the witness and the update polynomial
  - Aggregator can verify share came from the signed update



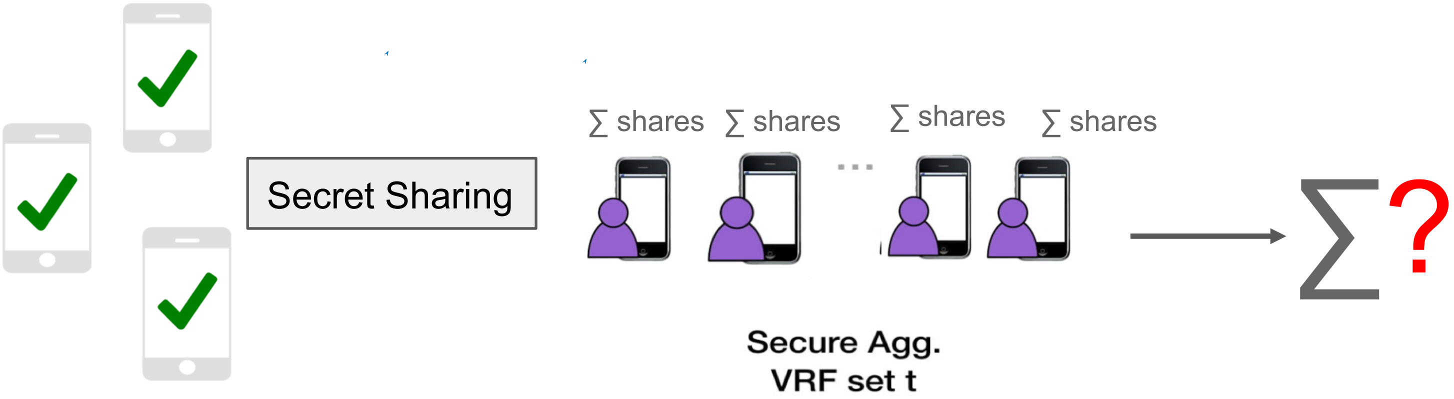
Biscotti: Aggregation using Shamir secret sharing[1]

- How do we recover the aggregate of the updates?



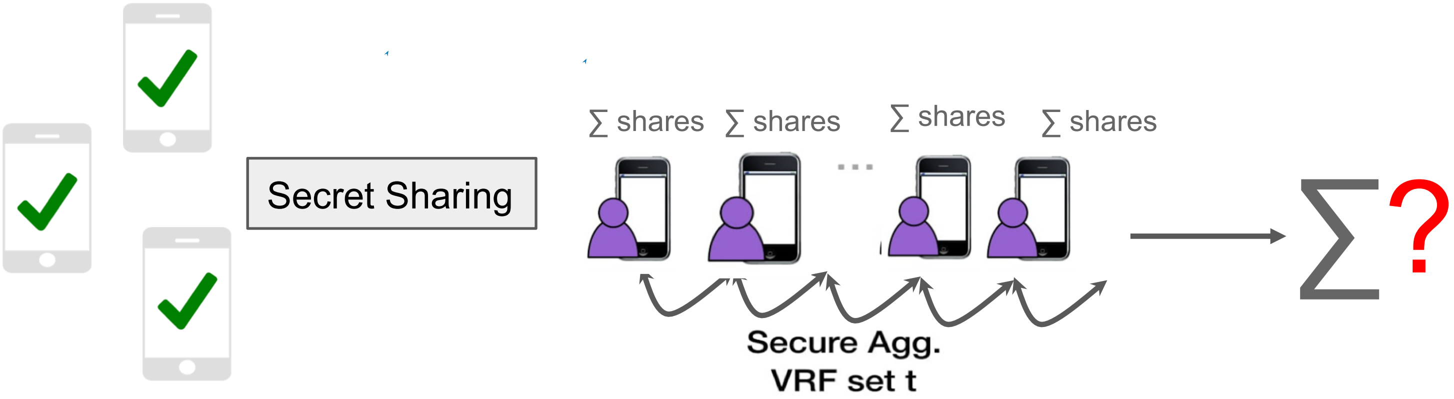
Biscotti: Aggregation using Shamir secret sharing[1]

- The aggregators compute the sum of their secret shares.



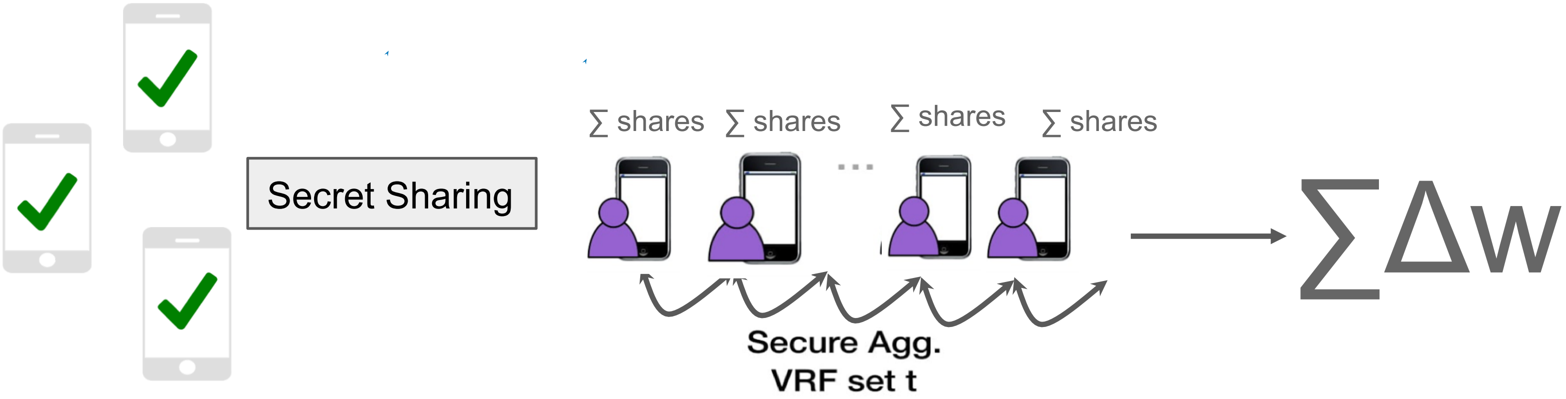
Biscotti: Aggregation using Shamir secret sharing[1]

- The aggregators compute the sum of their secret shares.
- The aggregated shares are shared with the rest.



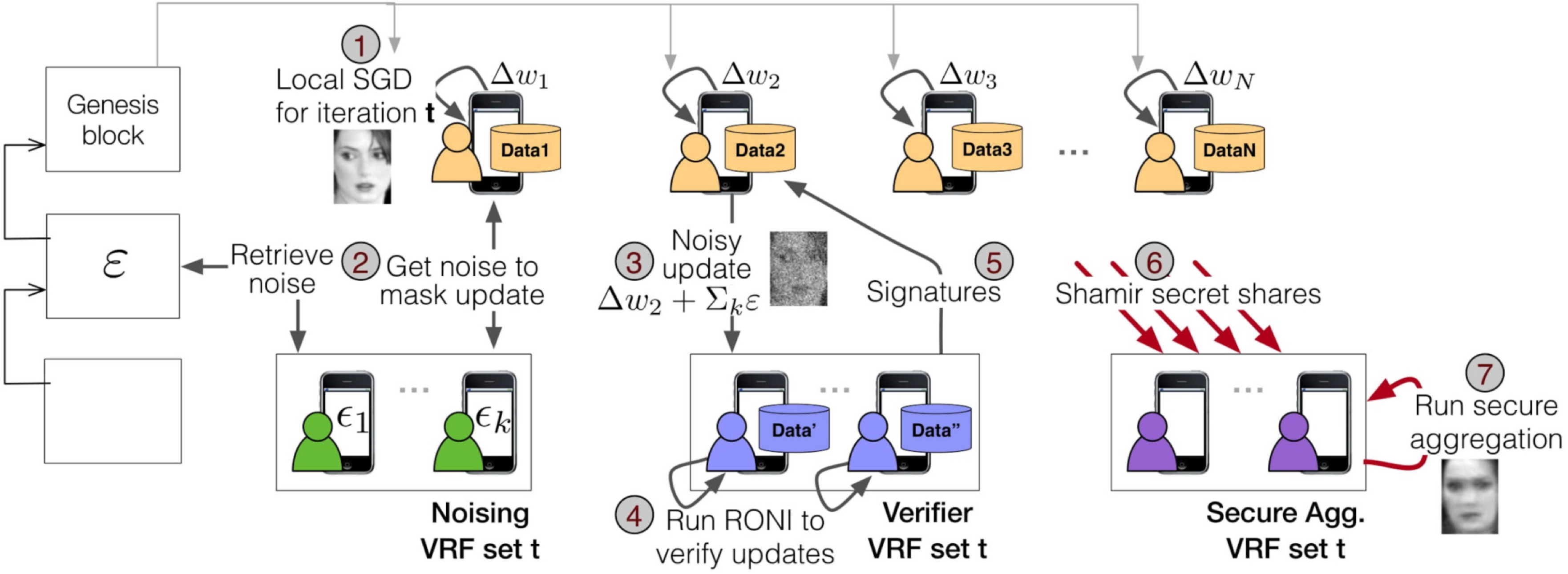
Biscotti: Aggregation using secret sharing

- The aggregators compute the sum of their secret shares.
- The aggregated shares are shared with the rest.
    - If the aggregated shares are interpolated, the aggregate of the updates can be computed.



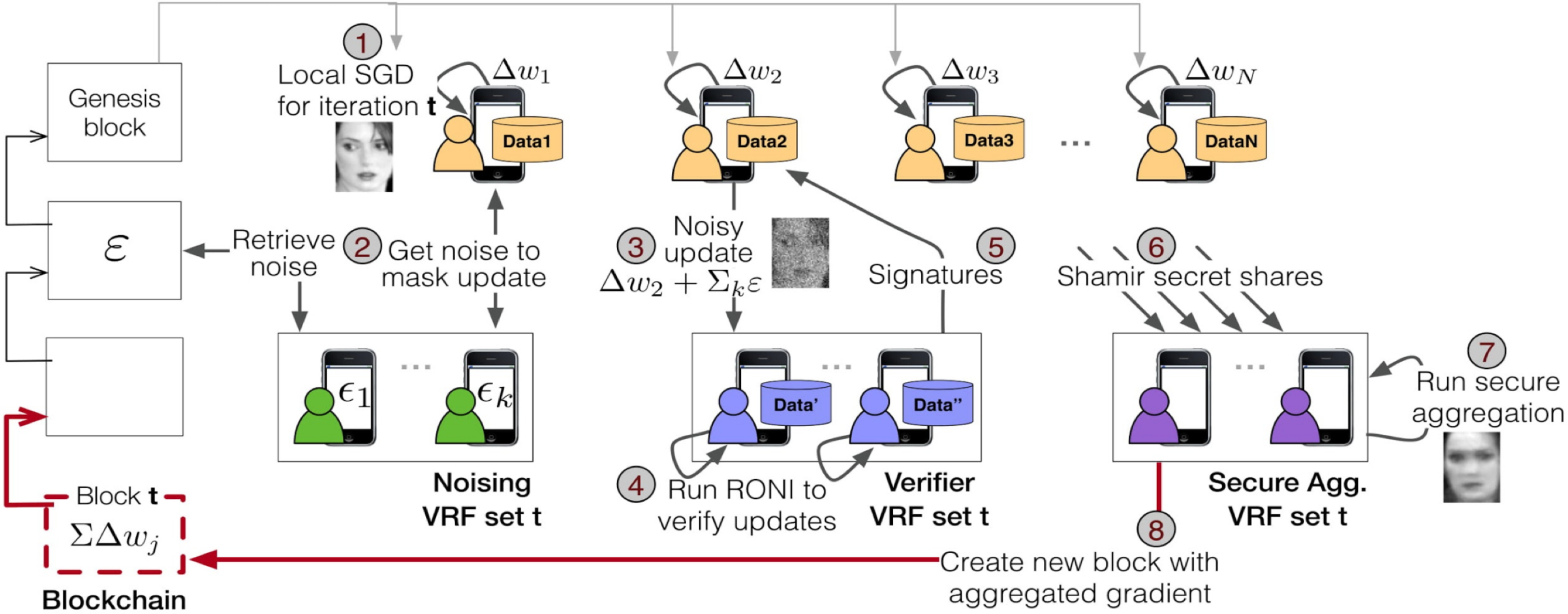
Biscotti: Aggregation and Block generation

- The aggregators figure out out the aggregate of updates without the noise using secret sharing



Biscotti: Aggregation and Block generation

- The block with the updated global model is created and added to blockchain. The model has optimal utility since it does not have noise added in the final output.



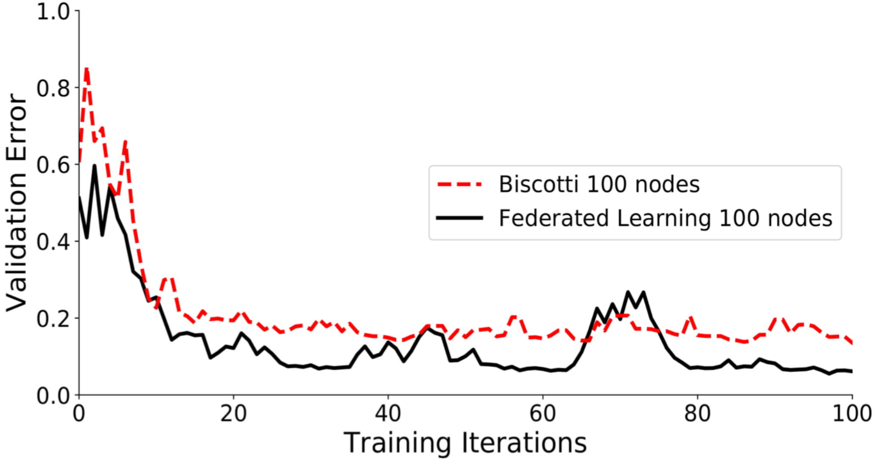
Evaluation: goals

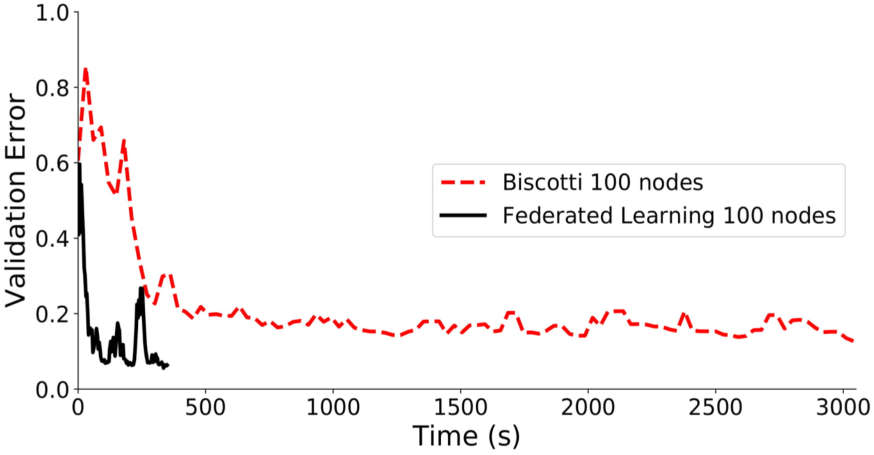
- Performance
  - Biscotti's performance compare to federated learning.
  - Performance bottlenecks in Biscotti.
  - Variation in performance as the size of verifier, noiser and aggregator sets increase
- Security and privacy
  - Poisoning attacks
  - Privacy with Sybils
- Fault tolerance
  - Performance with node churn

Dataset	**Model Type**	**Examples (n)**	**Params (d)**
Credit Card	LogReg	21000	25
MNIST	Softmax	60000	7850

Parameter	**Default Value**
Privacy budget (ε)	2
Number of noisers	2
Number of verifiers	3
Number of aggregators	3
Proportion of secret shares needed u	0.125
Initial stake	Uniform, 10 each
Stake update function	Linear, +/- 5

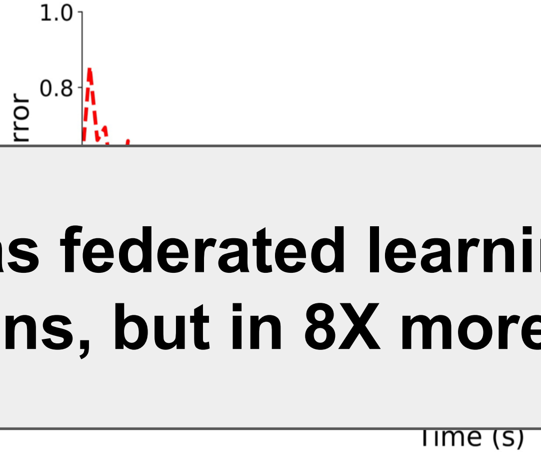
Evaluation - Baseline (Biscotti vs Federated Learning)





- Deployed on 20 Azure VMs with 5 peers each.
- Biscotti achieves similar training error in a similar number of iterations
- Biscotti has a 6x performance overhead compared to federated learning

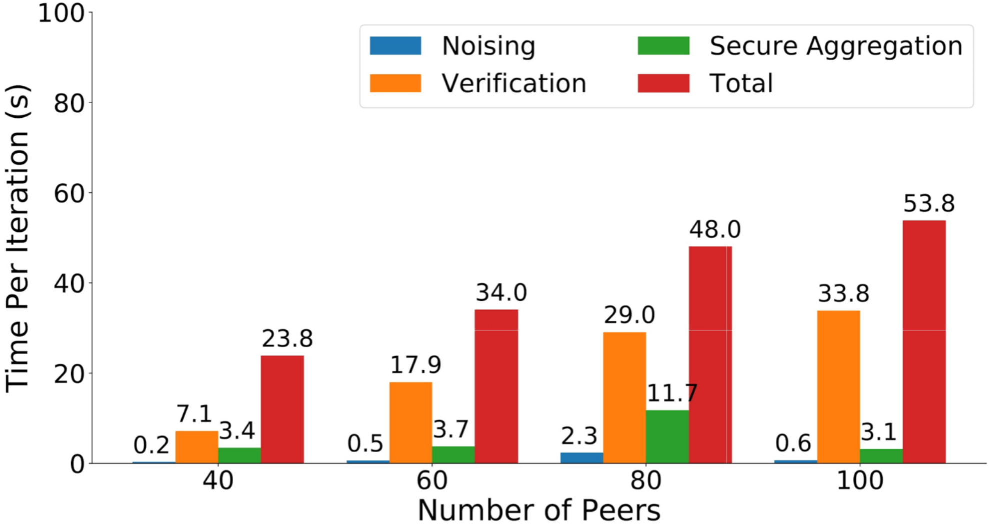
Evaluation - Baseline (Biscotti vs Federated Learning)



Biscotti performs as well as federated learning in the same number of iterations, but in 8X more time.

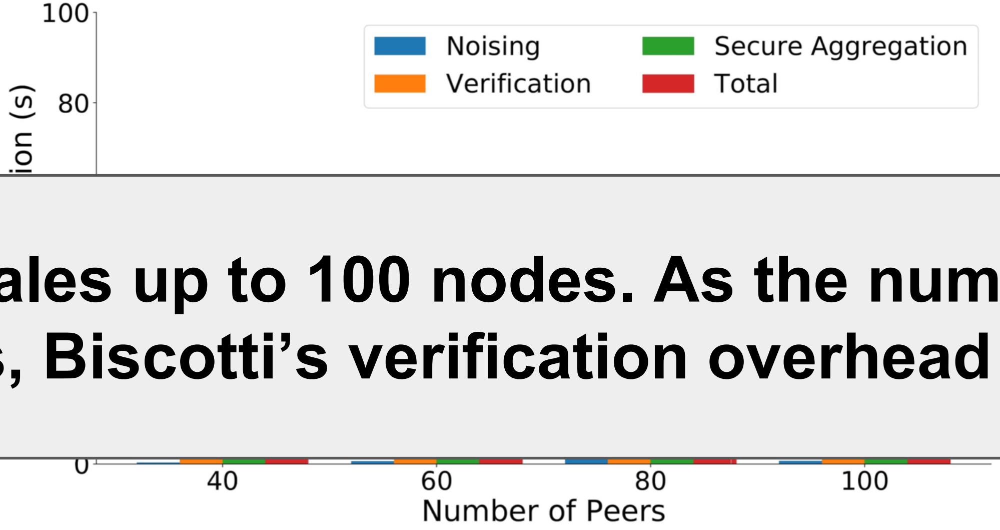
- Deployed on 20 Azure VMs with 5 peers each.
- Biscotti achieves similar training error in a similar number of iterations
- Biscotti has a 6x performance overhead compared to federated learning

Evaluation - Component-wise Performance Breakdown



- Time per iteration increases exponentially with the number of nodes
- Verification of updates is the most costly operation.
- Noising and aggregation have negligible overhead on the total performance cost.

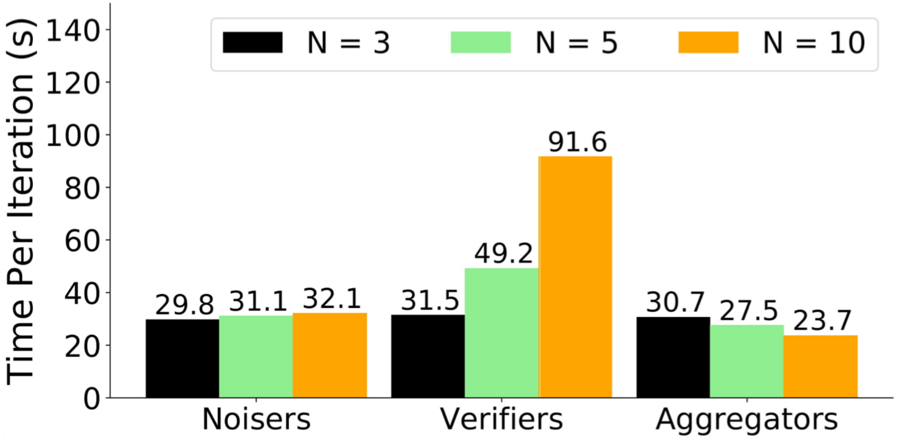
Evaluation - Component-wise Performance Breakdown



Biscotti scales up to 100 nodes. As the number of peers increases, Biscotti's verification overhead increases.

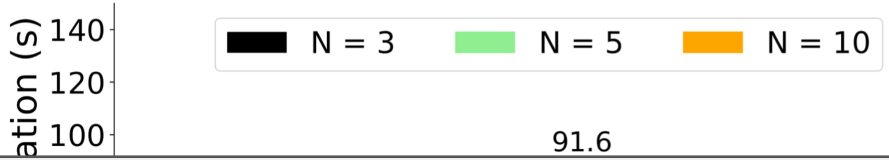
- Time per iteration increases exponentially with the number of nodes
- Verification of updates is the most costly operation.
- Noising and aggregation have negligible overhead on the total performance cost.

Evaluation - Performance as the size of the VRF set varies (50 peers total)



- Time per iteration increases exponentially with more verifiers due to expensive RONI operation
- Time per block remains constant with increase in the noiser set as it only adds few RTT's per iteration
- Aggregation time decreases owing to sufficient shares able to be quickly collected and non-participation of aggregators in generating updates.

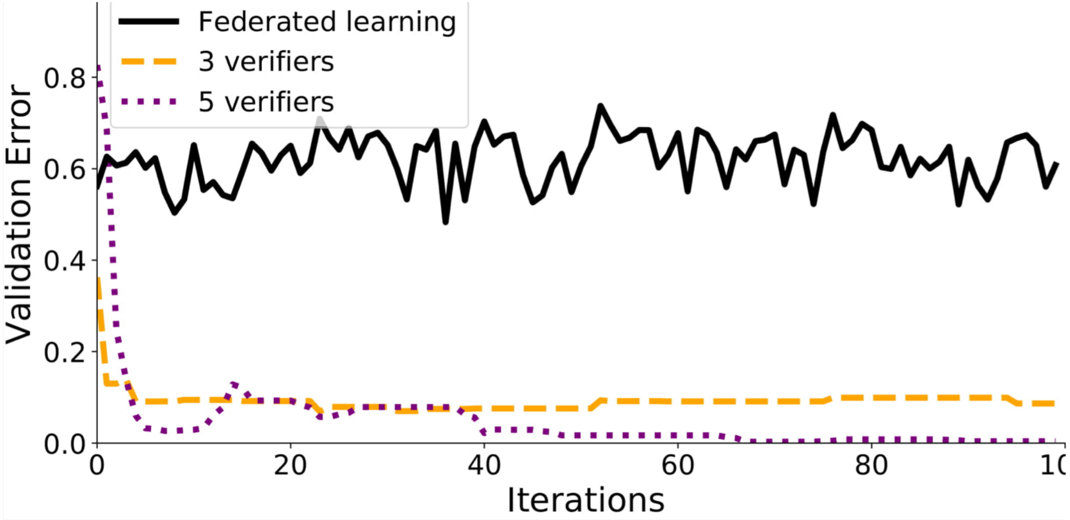
Evaluation - Performance as the size of the VRF set varies



Biscotti's performance can scale to accommodate more noisers or aggregators.

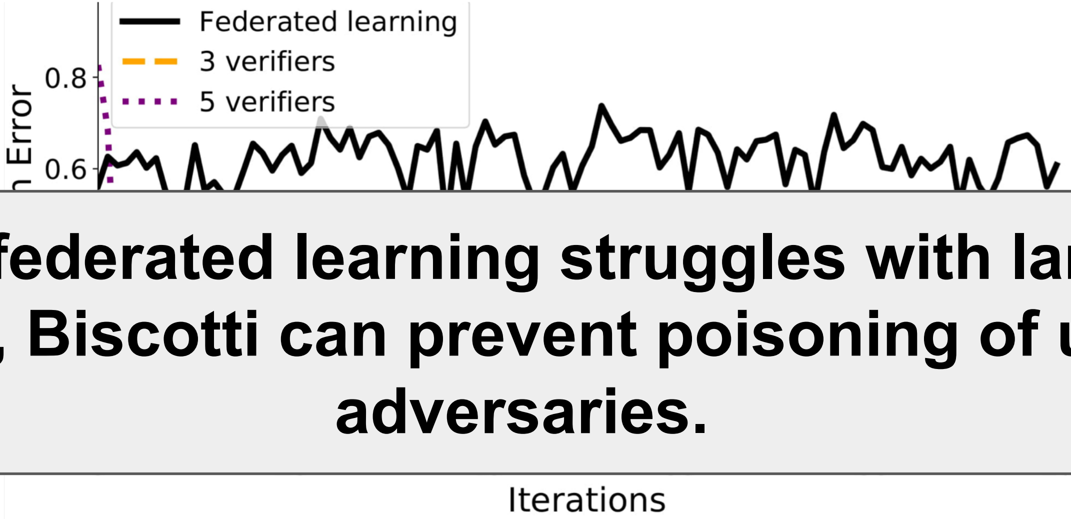
- Time per iteration increases exponentially with more verifiers due to expensive RONI operation
- Time per block remains constant with increase in the noiser set as it only adds few RTT's per iteration
- Aggregation time decreases owing to sufficient shares able to be quickly collected and non-participation of aggregators in generating updates.

Evaluation - Poisoning attack



- Poisoning attack on the credit card dataset evaluated with 49% poisoners in the system.
- Federated learning unable to defend against such attacks.
- Biscotti able to converge faster if it has sufficient number of verifiers in the VRF set.

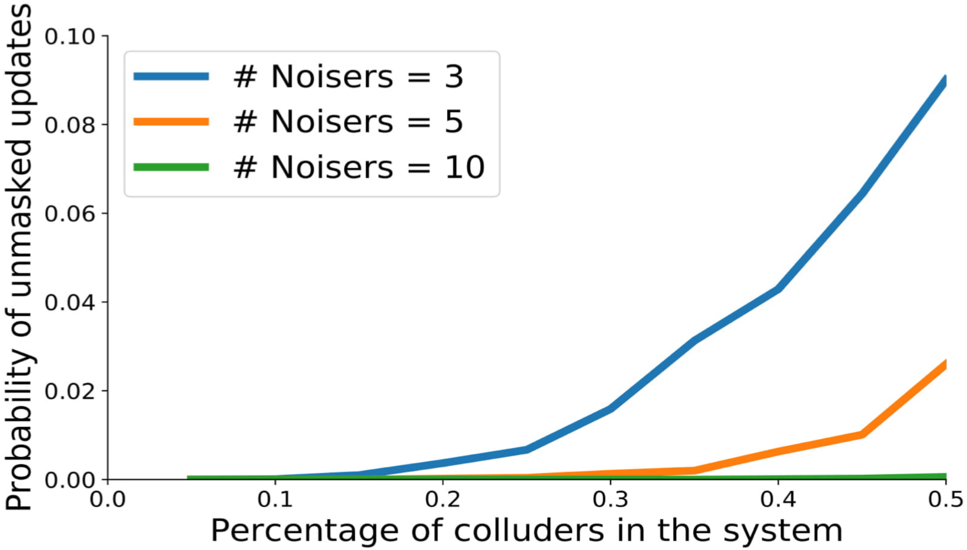
Evaluation - Poisoning attack



Whereas federated learning struggles with large scale poisoning, Biscotti can prevent poisoning of up to 49% adversaries.

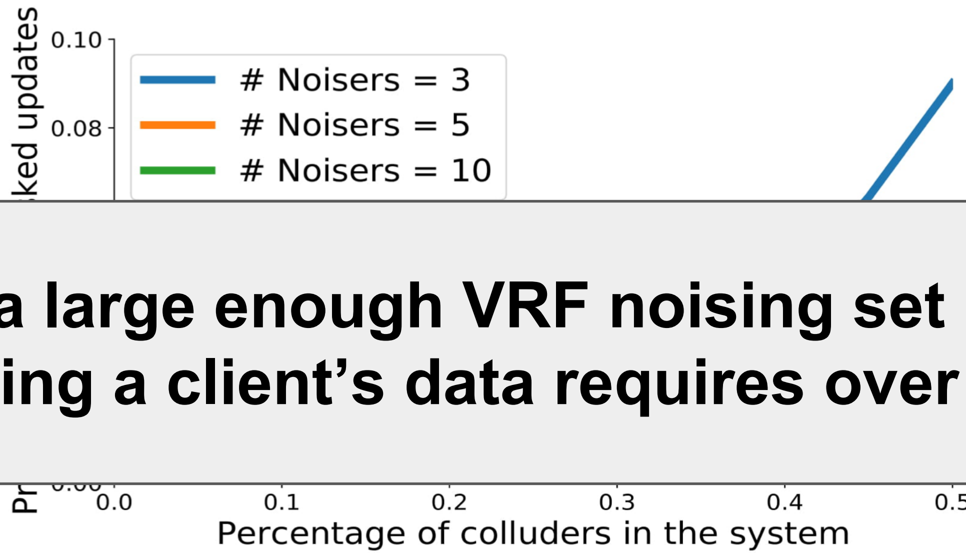
- Poisoning attack on the credit card dataset evaluated with 49% poisoners in the system.
- Federated learning unable to defend against such attacks.
- Biscotti able to converge faster if it has sufficient number of verifiers in the VRF set.

Evaluation - Sybil attack on privacy (noisers that don't add any noise)



- Extremely low probability of being able to unmask a client's updates
- Probability gets close to zero with sufficient number of noisers in the system.
- Stake distribution uniform.

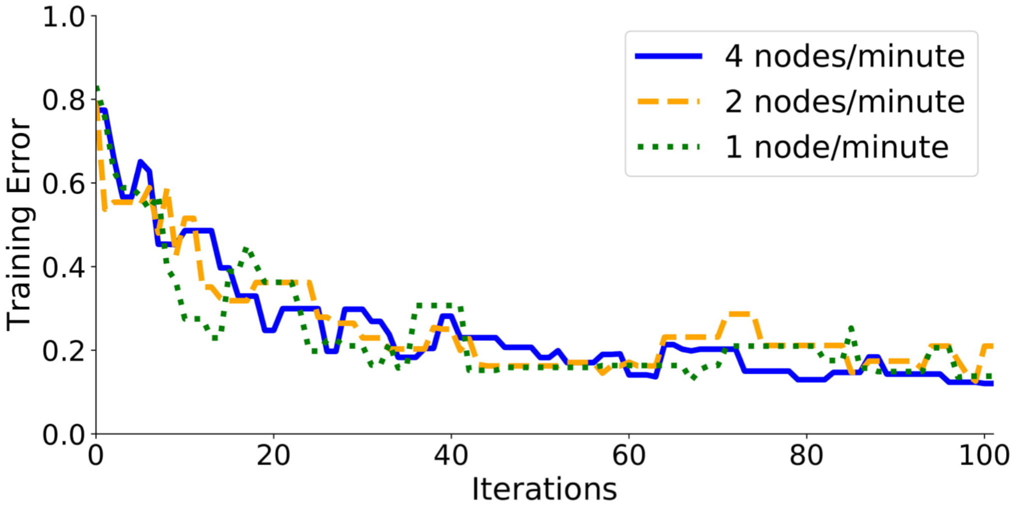
Evaluation - Sybil attack on privacy



With a large enough VRF noising set (N=10), deanonymizing a client's data requires over 50% of stake.

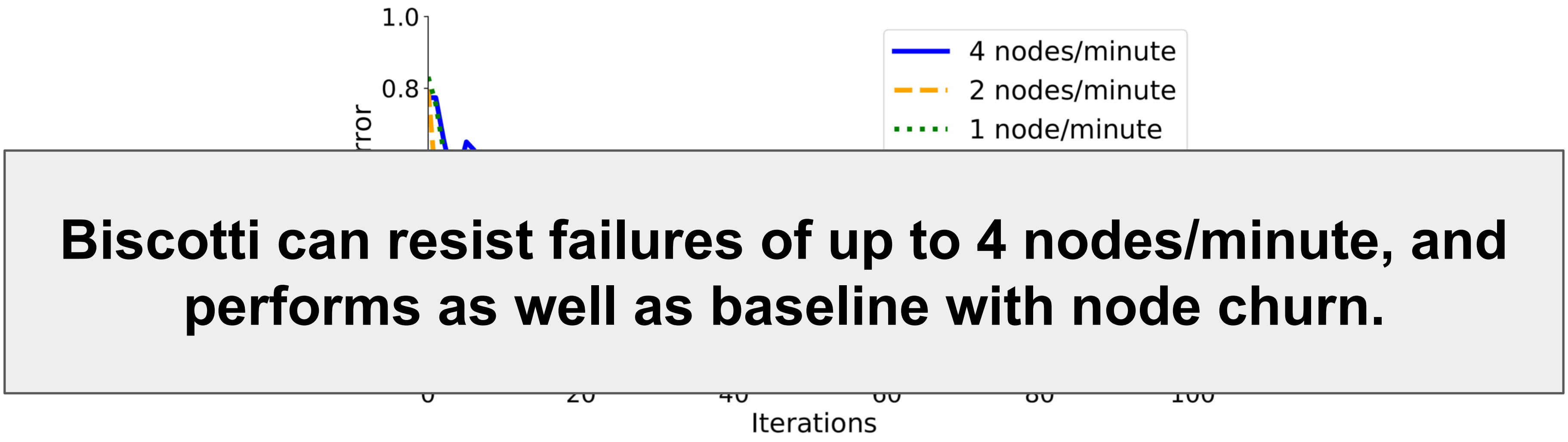
- Extremely low probability of being able to unmask unmasking a client's updates
- Probability goes close to zero with sufficient number of noisers in the system.
- Stake distribution uniform.

Evaluation - Fault Tolerance



- Biscotti was able to resist node churn of up to 4 nodes/minute with negligible effect on convergence.
- Training error reaches expected value after 100 iterations.

Evaluation - Fault Tolerance



- Biscotti was able to resist node churn of up to 4 nodes/minute with negligible effect on convergence.
- Training error reaches expected value after 100 iterations.

Limitations / Future Work

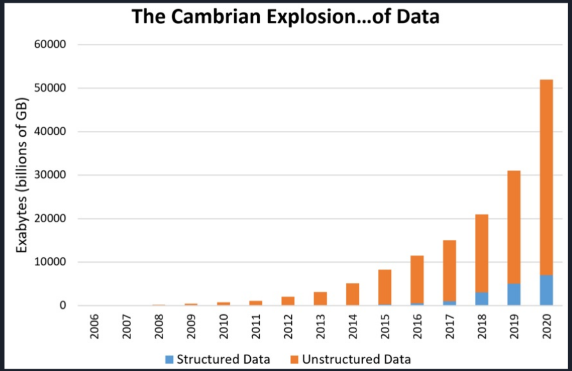
- Relies on an assumption of stake
  - Proof of X: should be something hard to fake and inherent to the system
  - Training data comes to mind, but has privacy concerns
- Can’t handle full range of poisoning attacks (only class level):
  - Adversarial examples, backdoors, targeted poisoning
- Better use of the blockchain
  - The blockchain provides a provenance record for the training process.
  - Audit trail could be leveraged to re-train the model by omitting certain poisoned updates after it is detected.

Contributions

- The first **peer to peer** system to empower collaborative ML training:
  - **Preserving privacy** with **noisy verification** and **secure aggregation**
  - Defending against **poisoning attacks** with **RONI**
  - A novel design that combines **blockchain** primitives with cryptography
    - **Mitigates sybils with** verifiable random functions and client stake

- Biscotti is able to produce models similar to federated learning:
  - At a wall clock overhead of 6X, but **similar iteration overhead**
  - While **scaling up to 100 nodes**, and with **tunable** parameters for each stage
  - While being robust to node **churns up to 4 nodes/minute**.

Private ML in the cloud : review

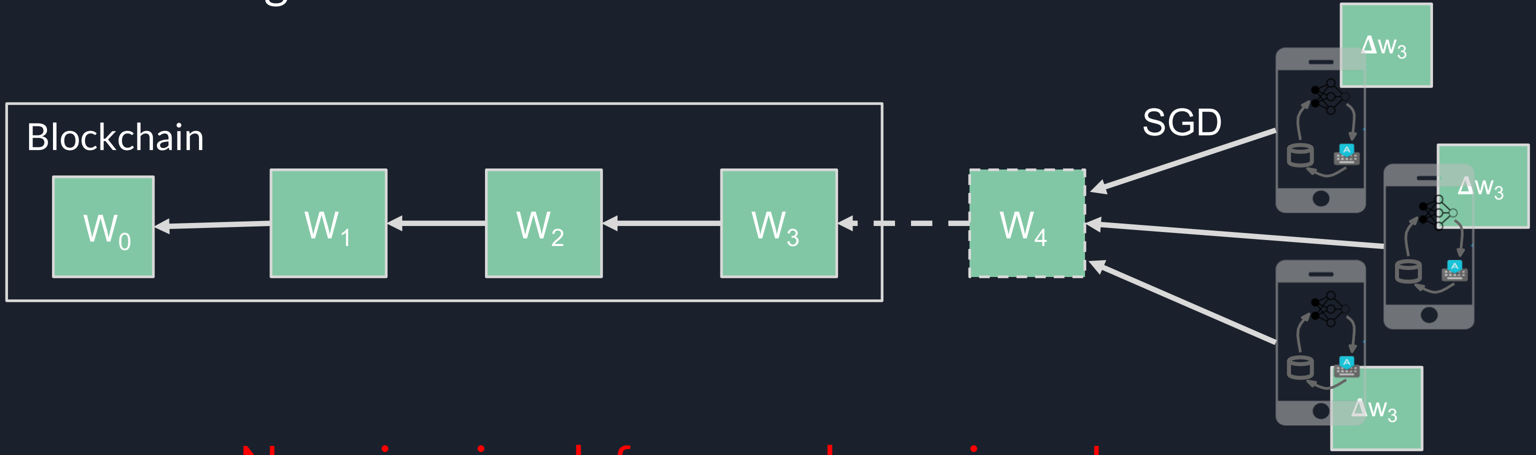


+

- Cloud ML today: centralize all the things
- Federated learning: an alternative, but actively involves clients (sybils)
  - FoolsGold: detect sybils in targeted sybil poisoning
- P2P ML: can we forego centralization altogether?
  - Biscotti: a solution based on blockchain, diff priv, and crypto

The easy part: SGD + Blockchain

- Each block stores a set of SGD updates from multiple peers
  - Each peer computes SGD using their blockchain state
  - With each block, the set of updates is added, updating the global model



No poisoning defense and no privacy!