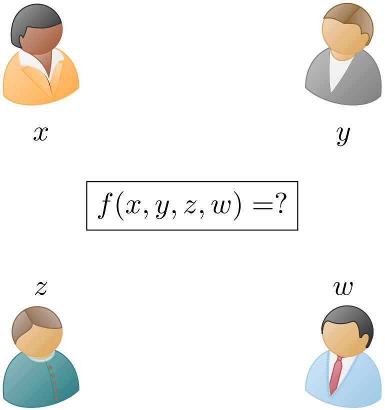
MD-ML: Super Fast Privacy-Preserving Machine Learning for Malicious Security with a Dishonest Majority

Boshi Yuan Shixuan Yang Yongxiang Zhang Ning Ding Dawu Gu Shi-Feng Sun

Shanghai Jiao Tong University

USENIX Security Symposium 2024

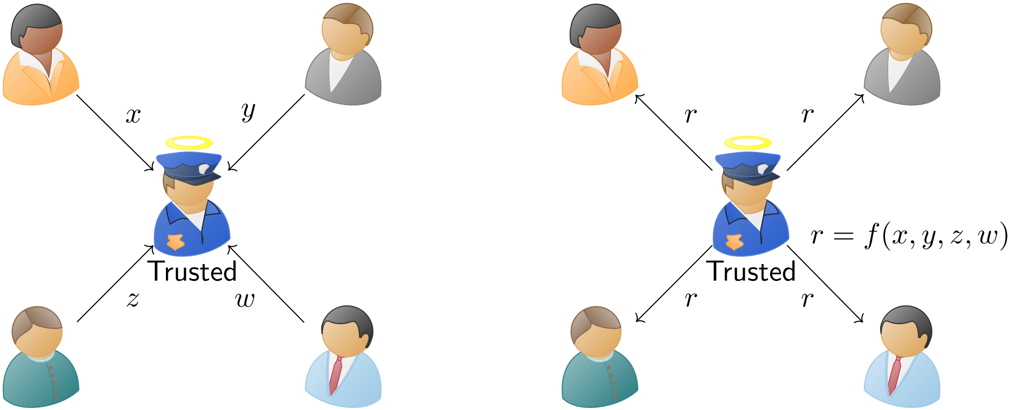
Introduction — Multi-party Computation (MPC)



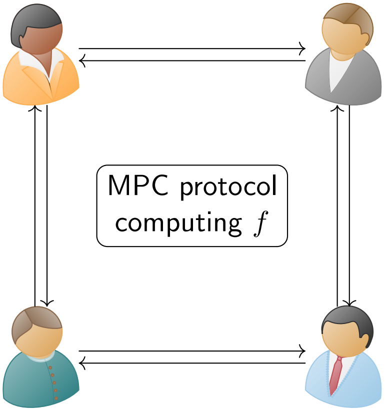
The Goal

- The inputs x, y, z, and w are private.
- The function f is public.

Introduction — Multi-party Computation (MPC)



Introduction — Multi-party Computation (MPC)



MPC

- MPC is a cryptographic protocol
- MPC ensures privacy and correctness
- When f is a machine learning model —
  - Privacy-Preserving Machine Learning (PPML)

Security Model in MPC

Adversary types:

- Semi-honest (passive)
- Malicious (active)

The number of corrupted parties t

(let n be the total number of parties):

- Honest majority ($t < n/2$)
- Dishonest majority ($t < n$)

This work: **M**aliciously secure **D**ishonest majority PP**ML** (MD-ML)

The Structure of PPML Protocols

PPML protocols consist of two parts:

An underlying MPC protocol for basic arithmetic circuits $(+, \times)$

Using existing protocols: SPDZ, SPD$\mathbb{Z}_{2^k}$, Rep3, etc.

Protocols for ML-specific operations

$\left.\begin{array}{l}\text{Truncation}\\ \text{Comparison}\\ \text{Vector dot product}\end{array}\right\}$ Make Improvements!

The State of the Art in PPML

In malicious security with dishonest majority model.

Damgård et al. (SP 2019)[1], we refer to as "SPD$\mathbb{Z}_{2^k}+$".

- They use SPD$\mathbb{Z}_{2^k}$ as the underlying MPC protocol.
- The first PPML protocol in this model.

Dalskov et al. (PETS 2020)[2]

- Quantized Neural Networks (QNN) evaluation (out of our scope).
- The underlying protocol is the same as SPD$\mathbb{Z}_{2^k}+$.

We mainly compare with SPD$\mathbb{Z}_{2^k}+$.

[1] Ivan Damgård et al. "New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning". In: *2019 IEEE Symposium on Security and Privacy (SP)*. 2019, pp. 1102–1120.

[2] Anders Dalskov, Daniel Escudero, and Marcel Keller. "Secure Evaluation of Quantized Neural Networks". In: *Proceedings on Privacy Enhancing Technologies Symposium* 2020.4 (2020), pp. 355–375.

Our Contributions

Efficiency $\uparrow$ Online communication $\downarrow$

Techiniques

- Circuit-dependent preprocessing with SPD$\mathbb{Z}_{2^k}$
- New truncation, comparison, and vector dot product protocols

In terms of online communication

- Truncation $+$ Multiplication $=$ Multiplication (Truncation is free)
- Vector dot product $=$ 1 element/party (regardless of vector length)

Implementation, benchmarks, and open-source $\checkmark$

SPD$\mathbb{Z}_{2^k}$ Protocol[3]

SPD$\mathbb{Z}_{2^k}$ secret-sharing $[x]$: additive, with authentication.

Addition

$[x] + [y] = [x + y]$ (computed locally).

Multiplication

Preprocessing: a multiplication triple $([a], [b], [c])$.
Online:

- Locally compute $[\delta_x] = [a] - [x]$, $[\delta_y] = [b] - [y]$
- Open δ_x, δ_y.
- Locally compute $[z] = [c] + \delta_x \cdot [b] + \delta_y \cdot [f] + e \cdot f$.

[3] Ronald Cramer et al. "SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Majority". In: *Advances in Cryptology – CRYPTO 2018*. Ed. by Hovav Shacham and Alexandra Boldyreva. Cham: Springer International Publishing, 2018, pp. 769–798.

Circuit-Dependent Preprocessing (CDP)[4]

Core Idea

Preprocessing: Every wire x in the circuit is associated with a value $[\lambda_x]$.
Online: Each party computes Δ_x where $\Delta_x = x + \lambda_x$.

Input

Preprocessing: Random $[\lambda_x]$.
Online: $[\Delta_x] = x + [\lambda_x]$ then open Δ_x.

Addition

Preprocessing: $[\lambda_z] = [\lambda_x] + [\lambda_y]$.
Online: $\Delta_z = \Delta_x + \Delta_y$.

---

[4] Aner Ben-Efraim, Michael Nielsen, and Eran Omri. "Turbospeedz: Double Your Online SPDZ! Improving SPDZ Using Function Dependent Preprocessing". In: *Applied Cryptography and Network Security*. Ed. by Robert H. Deng et al. Cham: Springer International Publishing, 2019, pp. 530–549.

Circuit-Dependent Preprocessing (CDP)

Multiplication with CDP

Preprocessing:

- Random $[\lambda_z]$.
- Multiplication triple $([a], [b], [c])$.
- Locally compute $[\delta_x] = [a] - [\lambda_x]$, $[\delta_y] = [b] - [\lambda_y]$.
- Open δ_x, δ_y.

Online:

- Locally compute $[\Delta_z] = (\Delta_x + \delta_x)(\Delta_y + \delta_y) - (\Delta_y + \delta_y)[a] - (\Delta_x + \delta_x)[b] + [c] + [\lambda_z]$.
- Open Δ_z.

Multiplication with CDP

Multiplication

Preprocessing:

- Multiplication triple $([a], [b], [c])$.

Online:

- Locally compute $[\delta_x]$, $[\delta_y]$
- Open δ_x, δ_y.
- Locally compute $[z]$.

Multiplication with CDP

Preprocessing:

- Multiplication triple $([a], [b], [c])$.
- Random $[\lambda_z]$
- Locally compute $[\delta_x]$, $[\delta_y]$.
- Open δ_x, δ_y.

Online:

- Locally compute $[\Delta_z]$.
- Open Δ_z.

Online Communication $2 \rightarrow 1$ elements/party.

Core Idea

- Previous work used CDP to improve multiplications
- We use CDP to improve vector dot product, truncation, comparison.

Vector Dot Product

Length-m vector dot product

$\vec{x} \cdot \vec{y} = \sum_{i=1}^{m} x[i]y[i]$ m invocations of multiplication?

Observations (with CDP)

$$\Delta_z = z + \lambda_z = \sum_{i=1}^{m} \vec{x}[i]\vec{y}[i] + \lambda_z$$

$$= \underbrace{\sum_{i=1}^{m} \Big((\overrightarrow{\Delta_x}[i] + \overrightarrow{\delta_x}[i])(\overrightarrow{\Delta_y}[i] + \overrightarrow{\delta_y}[i]) - (\overrightarrow{\Delta_y}[i] + \overrightarrow{\delta_y}[i])[a[i]] - (\overrightarrow{\Delta_x}[i] + \overrightarrow{\delta_x}[i])[b[i]] + [c[i]] \Big)}_{\text{Can be computed locally!}} + \lambda_z$$

Vector Dot Product

Vector Dot Product Protocol

Preprocessing:

- Random $[\lambda_z]$.
- Multiplication triples $([\vec{a}], [\vec{b}], [\vec{c}])$.
- Locally compute $[\vec{\delta_x}]$, $[\vec{\delta_y}]$.
- Open $\vec{\delta_x}$, $\vec{\delta_y}$.

Online:

- Locally compute $[\Delta_z]$.
- Open Δ_z.

Online communication: 1 element/party, regardless of length m.
Previous: $2m$ elements/party.

Truncation

Classical Truncation: $[z'] \mapsto [z]$ where $z = z'/2^d$

- Generate a truncation pair $[r']$, $[r]$ where $r = r'/2^d$.
- Locally compute $[c'] = [z'] + [r']$. Open c'.
- Compute $c = c'/2^d$.
- Compute $[z] = c - [r]$.

$\mathcal{F}_{\text{TruncPair}}$

- Random r'.
- $r = r'/2^d$
- Output $[r']$, $[r]$

Observations (with CDP)

- In CDP we already have $\Delta_{z'} = z' + \lambda_{z'}$.
- $\lambda_{z'}$ can be used as r'?
- In multiplication $z' = xy$, $\lambda_{z'}$ is random.
- Combine truncation with multiplication.
- Generate $\lambda_{z'}$, λ_z from $\mathcal{F}_{\text{TruncPair}}$.

Multiplication with Truncation

Multiplication with Truncation

Preprocessing:

- $([\lambda_{z'}], [\lambda_z]) \leftarrow \mathcal{F}_{\mathsf{TruncPair}}$.
- Multiplication triple $([a], [b], [c])$.
- Locally compute $[\delta_x]$, $[\delta_y]$.
- Open $\delta_{z'}$.

Online:

- Locally compute $[\Delta_{z'}] = \Delta_{z'} + \lambda_{z'}$.
- Open $\Delta_{z'}$.
- $\Delta_z = \Delta_{z'}/2^d$.

Online: 1 element in 1 round

Previous: 3 elements in 2 rounds

Implementation

- We implement the online phase of MD-ML in C++.
  - Open-source at `https://github.com/NemoYuan2008/MD-ML`.
- We benchmark the offline phase using MP-SPDZ[5].
- We focus on 2-party setting in the evaluation.
- We compare MD-ML with SPD$\mathbb{Z}_{2^k}$+[6].

[5] Marcel Keller. "MP-SPDZ: A Versatile Framework for Multi-Party Computation". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS '20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1575–1590.

[6] Ivan Damgård et al. "New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning". In: *2019 IEEE Symposium on Security and Privacy (SP)*. 2019, pp. 1102–1120.

Online Phase

Online phase benchmarks:

- AlexNet inference on CIFAR-10, Tiny ImageNet, ImageNet.

Dataset	LAN Time			WAN Time			Communication		
	Ours	SPD$\mathbb{Z}_{2^k}$+	Factor	Ours	SPD$\mathbb{Z}_{2^k}$+	Factor	Ours	SPD$\mathbb{Z}_{2^k}$+	Factor
CIFAR-10	0.82 s	6.80 s	8.3×	34.88 s	3254.7 s	93.3×	241.51 MB	2364.82 MB	9.8×
Tiny ImageNet	2.06 s	16.40 s	8.0×	53.89 s	6774.6 s	125.7×	405.00 MB	8274.95 MB	20.4×
ImageNet	7.38 s	81.35 s	11.0×	188.92 s	29785.2 s	157.7×	1319.31 MB	31447.70 MB	23.8×

- ResNet-18 on CIFAR-10

Model and Dataset	LAN	WAN	Communication
ResNet-18 on CIFAR-10	25.8 s	362.9 s	4.15 GB

Preprocessing Phase

Preprocessing phase benchmarks:

- Dot product of length 65536.
- MultTrunc and LTZ: 1024 values.

Operation	LAN time			WAN time			Communication		
	Ours	$SPD\mathbb{Z}_{2^k}+$	Factor	Ours	$SPD\mathbb{Z}_{2^k}+$	Factor	Ours	$SPD\mathbb{Z}_{2^k}+$	Factor
MultTrunc	2.191 s	2.189 s	0.999×	436.991 s	436.383 s	0.999×	162.294 MB	162.261 MB	1.0000×
LTZ	2.388 s	2.390 s	1.001×	435.234 s	434.636 s	0.999×	165.096 MB	165.079 MB	0.9999×
Dot prod.	8.065 s	6.246 s	0.775×	283.548 s	230.505 s	0.813×	1270.23 MB	1124.39 MB	0.8852×

Conclusion

- Malicious, dishonest majority model
- New protocols from CDP
  - truncation
  - vector dot product
  - comparison
- Implementation and benchmarks

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE | ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL | ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

`https://github.com/NemoYuan2008/MD-ML`

About me: Boshi Yuan
PhD student at SJTU

`nemoyuan2008@sjtu.edu.cn`

Thank you!