

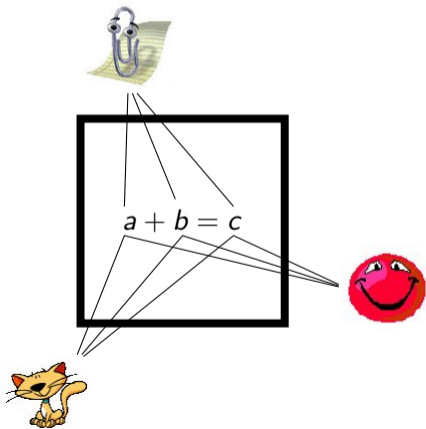
MP-SPDZ: Six Years of Scalable MPC

Marcel Keller

CSIRO's Data61

10 July 2025

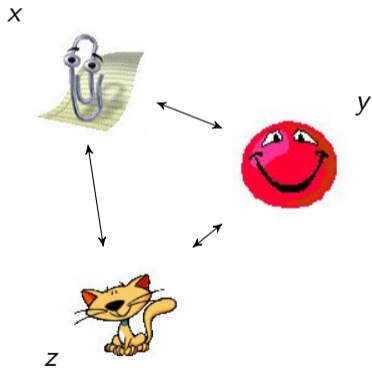
Imagine a Magic Black Box Between a Set of Parties



Parties

- ▶ Have handles to values
- ▶ Don't know the values
- ▶ Can input values
- ▶ Can agree on computations creating new values
- ▶ Can agree on outputting values

Secure Multiparty Computation: Black Box as Protocol



Wanted: $f(x, y, z)$

- ▶ Computation on secret inputs
- ▶ Replace black box
- ▶ Central questions in MPC
 - ▶ How many honest parties?
 - ▶ Dishonest parties still follow the protocol?
- ▶ MP-SPDZ supports > 40 protocol variants across all properties

I/O Parallelization

Core Property of MPC

Multiplication requires communication, addition does not

$$z = x \cdot y$$

$$u = z \cdot w$$

$$z = x \cdot y$$

$$u = v \cdot w$$

I/O Parallelization

Core Property of MPC

Multiplication requires communication, addition does not

$$z = x \cdot y$$

$$u = z \cdot w$$

1. Compute z
2. Compute u

$$z = x \cdot y$$

$$u = v \cdot w$$

1. Compute z and u

Goal: Automate I/O Parallelization

Manual parallelization is tedious:

$$x_{10} = x_2 \cdot x_3$$

$$x_{11} = x_8 + x_4$$

$$x_{12} = x_{10} \cdot x_1$$

$$x_{13} = x_7 + x_9$$

$$x_{14} = x_7 \cdot x_1$$

$$x_{15} = x_9 + x_{12}$$

$$x_{16} = x_{13} \cdot x_{14}$$

$$x_{17} = x_0 + x_{11}$$

$$x_{18} = x_{11} \cdot x_{15}$$

$$x_{19} = x_{13} \cdot x_7$$

$$x_{20} = x_4 + x_6$$

$$x_{21} = x_{16} + x_2$$

$$x_{22} = x_0 + x_{12}$$

$$x_{23} = x_{22} + x_{14}$$

$$x_{24} = x_{11} + x_{19}$$

$$x_{25} = x_4 \cdot x_{19}$$

$$x_{26} = x_{23} \cdot x_9$$

$$x_{27} = x_7 \cdot x_5$$

$$x_{28} = x_{13} + x_{21}$$

$$x_{29} = x_{14} + x_{27}$$

$$x_{30} = x_{19} \cdot x_1$$

$$x_{31} = x_{16} + x_{26}$$

$$x_{32} = x_0 \cdot x_{10}$$

$$x_{33} = x_{26} + x_{32}$$

$$x_{34} = x_7 + x_3$$

$$x_{35} = x_9 \cdot x_{29}$$

$$x_{36} = x_{33} + x_{22}$$

$$x_{37} = x_{29} \cdot x_{24}$$

$$x_{38} = x_{16} + x_{23}$$

$$x_{39} = x_{15} + x_{37}$$

$$x_{40} = x_{12} \cdot x_{39}$$

$$x_{41} = x_{34} + x_7$$

$$x_{42} = x_{32} + x_5$$

$$x_{43} = x_{12} + x_{26}$$

$$x_{44} = x_{43} \cdot x_{38}$$

$$x_{45} = x_{38} + x_{14}$$

$$x_{46} = x_{44} \cdot x_{27}$$

$$x_{47} = x_{22} + x_{24}$$

$$x_{48} = x_{39} \cdot x_{38}$$

$$x_{49} = x_{21} \cdot x_3$$

$$x_{50} = x_{28} + x_{16}$$

$$x_{51} = x_{15} + x_{38}$$

$$x_{52} = x_{50} \cdot x_{46}$$

$$x_{53} = x_{19} + x_2$$

$$x_{54} = x_{20} \cdot x_{13}$$

$$x_{55} = x_{21} + x_{22}$$

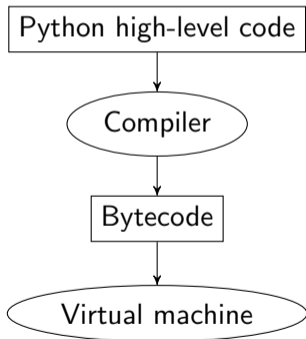
$$x_{56} = x_{19} \cdot x_6$$

$$x_{57} = x_{46} + x_1$$

$$x_{58} = x_{38} \cdot x_{55}$$

$$x_{59} = x_{47} + x_{29}$$

Toolchain Overview



Compiler

- ▶ Implemented in Python
- ▶ Optimization (reduce network rounds)
- ▶ Library for various arithmetic: integer, fractional, mathematical
- ▶ Machine learning functionality

Virtual machine

- ▶ One per protocol
- ▶ Implemented in C++ for speed

Compilation with Budget (HyCC)

What to do with loops in MPC?

Consecutive execution slow due to cost of communication rounds

Complete optimization has limits: compilation time, RAM/disk usage

Use a budget!

1. Unroll loop until budget exceeded
2. Optimize unrolled loop
3. Use sequential execution on top of unrolled loop

Example

Need 1000 repetitions: optimize 100, call 10 times

Bytecode Reusability

Mathematical Building Blocks

- ▶ Comparison, exponentiation, logarithm. . .
- ▶ In C/C++: simple call of function or CPU instruction
- ▶ In MPC: want parallelization so no simple function calls

MP-SPDZ

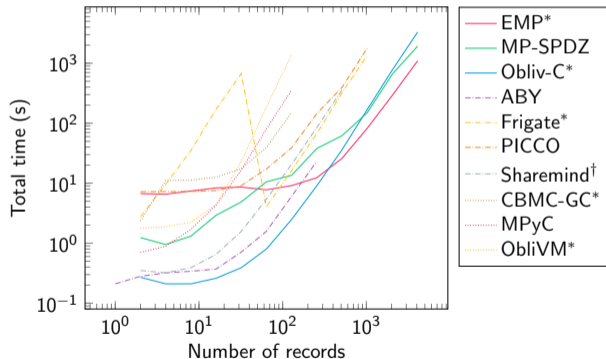
- ▶ Find and combine parallelizable invocations of building blocks
- ▶ Compile functionality once depending on protocol options
- ▶ Reuse bytecode object (faster compilation, lower disk usage)
- ▶ Extra benefit: calling high-level code from C++ (new in version 0.4.0)

Crosstabulation Example

```
for i in range(n):  
    for j in range(n):  
        if x[i] == y[j]:  
            sums[cat[i]] += val[j]
```

- ▶ Example used by Hastings et al. (S&P'19) to evaluate MPC compilers
- ▶ Not necessarily optimal but good test case
 - ▶ Nested loops
 - ▶ Quadratic run-time
- ▶ Relevant building block: equality test
 - Small n Parallelize across both loops
 - Large n Parallelize inner loop only

Crosstabulation Benchmarks



Time to compile and run cross-tabulation with different MPC frameworks.

* denotes a garbled circuit implementation, and † denotes an emulation.

Section 1

Function-Dependent Preprocessing

Function-Independent Preprocessing

Most protocols in MP-SPDZ

- ▶ Function-independent preprocessing:
“Commodity” like Beaver multiplication triples, random bits etc.
- ▶ Symmetric secret-sharing:
Every share “looks” the same
- ▶ Symmetric communication:
Everyone sends to everyone or via a (possibly rotating) “king”
- ▶ One-pass execution possible:
Generate enough preprocessing ahead of time or on-demand
- ▶ Protocols go back to early days of (practical) MPC

Honest-Majority Function-Dependent Preprocessing

Newer line of work

- ▶ Preprocessing depends on functionality:
Information matching an exact multiplication in relation to other computation
- ▶ Asymmetric secret-sharing:
Different shares with different meanings for different parties
- ▶ Asymmetric communication:
Some parties only send but never receive
- ▶ Only two-pass execution:
Preprocessing follows function, no bulk preprocessing
- ▶ First paper in 2019

Function-Independent vs Function-Dependent Preprocessing

We show

- ▶ Cost of two-pass execution underestimated by prior work
- ▶ Same for cost of storage between phases
- ▶ Asymmetric communication also possible with function-*independent* preprocessing
- ▶ **Simplicity wins**

ImageNet Training Benchmark

| Protocol | Phase | Communication | Time | Phase transmission |
|----------|---------------|---------------|-----------------|--------------------|
| Rep3 | | 513.11 GB | 804.87 seconds | |
| Astra | Preprocessing | 233.22 GB | 4254.35 seconds | |
| | Online | 354.28 GB | 868.01 seconds | 820.69 GB |
| | Total | 587.50 GB | 5122.36 seconds | |
| Trio | Preprocessing | 233.22 GB | 679.95 seconds | |
| | Online | 354.28 GB | 866.71 seconds | 820.69 GB |
| | Total | 587.50 GB | 1546.66 seconds | |

Section 2

Secure Shuffling

Secure Shuffling

Have Secret-shared list $[x_1], \dots, [x_n]$

- Want
- ▶ Secret-shared list $[x_{\pi(1)}], \dots, [x_{\pi(n)}]$
 - ▶ π random and secret permutation
 - ▶ Ability to repeat for both π and π^{-1}

Approaches

- ▶ Switching networks (Waksman)
- ▶ Permute by maximally unqualified set (honest majority only)
- ▶ Dual execution for malicious security (akin SPDZ-wise)
- ▶ Recent advances for 2PC using MPC-friendly PRFs (no available implementation)

Application of Secure Shuffling: Sorting

Secret Permutation Operation

Have $[x_1], \dots, [x_n]$ such that $x_1 = \pi(1), \dots, x_n = \pi(n)$
 $[y_1], \dots, [y_n]$

Want $[y_{\pi(1)}], \dots, [y_{\pi(n)}]$ without revealing π

1. Reveal secret shuffle ρ of $[x_1], \dots, [x_n] \Rightarrow$ public description of $\rho \circ \pi$
2. Apply $\rho \circ \pi$ to $[y_1], \dots, [y_n]$
3. Revert secret shuffle ρ on $[y_{(\rho \circ \pi)(1)}], \dots, [y_{(\rho \circ \pi)(n)}]$

Radix Sorting

Build permutation needed to sort bit by bit, then apply to data

Application of Secure Shuffling: Decision Tree Training

Problem

- ▶ Decision trees are trained level by level where samples are sorted into nodes
- ▶ Naive MPC has complexity $O(\#nodes \cdot \#samples)$

Solution by Hamada et al., PETS'22

- ▶ Sort samples by node at every level and use secret node markers
- ▶ Heavily relies on shuffling

Outlook

- ▶ MP-SPDZ remains popular (citations/GitHub issues)
- ▶ Unmatched protocol variety
- ▶ Unmatched programmability?
- ▶ Governance?

Links

<https://eprint.iacr.org/2024/1990>

<https://eprint.iacr.org/2025/919>

<https://github.com/data61/MP-SPDZ>

<https://mp-spdz.readthedocs.io>