MP-SPDZ Bootcamp

Christopher Smith

August 28, 2026

Overview

Today we get our hands dirty with MP-SPDZ. We will build the MP-SPDZ engine, write and execute circuits using said engine, and discuss MPC along the way.

Outline

1. Setup
2. Theory
   - Secret Sharing
   - MPC
3. MP-SPDZ
4. References

System Requirements

A computer with an ssh client. We will do everything on a single Ubuntu 22.04 VM. Remote SSH with VSCode is a nice experience.

SSH

1. Locate an existing ssh key pair with `ls ~/.ssh`, or generate a new ssh key pair with `ssh-keygen`
2. Copy the entire public key and email it to me.
3. Add this to `~/.ssh/config` :

```
Host mpc-tutorial
    Hostname <IP>
    User cryptographer
```

4. `ssh mpc-tutorial` (or do it from VSCode)
5. Create a directory for yourself. E.g., `mkdir chris && cd chris`

MP-SPDZ Hello World

1. `cp -r ../MP-SPDZ ./ && cd MP-SPDZ`
2. `python3 Programs/Source/hello.py`
3. `Scripts/shamir.sh hello`

Outline

1 Setup

2 Theory
- Secret Sharing
- MPC

3 MP-SPDZ

4 References

$(t+1, n)$ Secret Sharing

- $\text{Share}(m) \mapsto [m]$, where $[m] = ([m]_1, \ldots, [m]_n)$
- $\text{Rec}([m]_\Theta) \mapsto m'$ for any $\Theta \subseteq \{1, \ldots, n\}$
- Correctness: $\text{Rec}(\text{Share}(m)_\Theta) = m$ for any $|\Theta| > t$.
- Perfect Privacy: $\mathbf{SD}(\text{Share}(m)_\Theta, \text{Share}(m')_\Theta) = 0$ for any $|\Theta| \leq t$.

$\text{Share}(m)$: // Example: Shamir's secret sharing

1. $p(X) := m + r_1 X + \cdots + r_t X^t$ where $r_1, \ldots, r_t \xleftarrow{\$} \mathbb{F}$
2. Output $[m]_i = p(\alpha_i)$ for some $\{\alpha_i\}_{1 \leq i \leq n} \subset \mathbb{F}$, $\alpha_i \neq 0$ for all i.

Linear Secret Sharing

A $(t+1, n)$ secret sharing scheme $(\mathsf{Share}, \mathsf{Rec})$ over a finite field $\mathbb{F}$ is **linear** if Rec is a linear function: $\mathsf{Rec}([m]_\Theta) := \sum_{i \in \Theta} \lambda_i [m]_i$ for some fixed $\{\lambda_i\}$ dependent on Θ [Bei96].

Equivalently, if $[m]_i := \mathsf{Share}(m; r)_i$ is a linear combination of m and randomness r [Bei96].

Upshot: Easy for party P_i to locally compute linear functions over secret-shared data.

- $\alpha \cdot [a]_i = [\alpha \cdot a]_i$
- $[a]_i + [b]_i = [a + b]_i$

Can we do similar for multiplication?

Multiplicative Secret Sharing

A $(t+1, n)$ scheme $(\mathsf{Share}, \mathsf{Rec})$ over $\mathbb{F}$ is **multiplicative** if there exists a function mult such that $ab = \sum_i \mathsf{mult}(i, [a]_i, [b]_i)$ [CDM00, BIW10].

Upshot: a multiplication protocol where P_i inputs shares $([a]_i, [b]_i)$ and gets $[ab]_i$ [CDM00]:

1. P_i computes $[c_i] \leftarrow \mathsf{Share}(c_i)$ for $c_i = \mathsf{mult}(i, [a]_i[b]_i)$. Send $[c_i]_j$ to P_j for all j.
2. After receiving $[c_j]_i$ from each P_j, P_i computes $[ab]_i$ as $\sum_j [c_j]_i$.

Correctness follows as $\sum_j [c_j]_i = \left[\sum_j c_j\right]_i = \left[\sum_j \mathsf{mult}(j, [a]_j, [b]_j)\right]_i = [ab]_i$

For Shamir's secret sharing, $\mathsf{mult}(i, [a]_i, [b]_i) \coloneqq \lambda_i [a]_i [b]_i$, where λ_i is a Lagrange interpolating coefficient for $q(0)$, where q is the unique degree $2t$ polynomial determined by $\{[a]_i[b]_i\}_{i \in \Theta}$ for any $|\Theta| > 2t$. Specifically, $\lambda_i \coloneqq \prod_{j \neq i} \alpha_j / (\alpha_j - \alpha_i)$. Note this requires $n \geq 2t + 1$.

Linear vs. Multiplicative

- A multiplicative secret sharing scheme can be generically constructed from any linear secret sharing scheme, with at most a 2x overhead in the total number of field elements distributed amongst parties [CDM00].
- A priori, a "d-multiplicative" scheme need not be linear [BIW10].
- The [CDM00] definition of a multiplicative scheme as a monotone span program (MSP) with a multiplicative property does imply linearity by virtue of the MSP.
- Going forward, we always assume Shamir's scheme with $n = 2t + 1$.

Semi-Honest MPC with Honest Majority

Let $\mathcal{C} : \mathbb{F}^n \to \mathbb{F}^n$ be an arithmetic circuit. Parties $P_1, \ldots, P_n$ can securely evaluate $\mathcal{C}$ under an honest majority using the following semi-honest protocol:

1. Each P_i secret shares input x_i and sends $[x_i]_j$ to P_j.
2. Parties execute $\mathcal{C}$ gate-by-gate with secret-shared inputs and outputs. Addition gates can be computed locally using linearity of shares. Multiplication gates are computed using the multiplication protocol.
3. At the end, each P_i holds secret-shared final outputs $\{[y_j]_i\}$. P_i sends $[y_j]_i$ to P_j.
4. P_i reconstructs its output y_i using the received $\{[y_i]_j\}$.

Malicious Security

Malicious security is defined in the real/ideal simulation paradigm. Many finer details and modeling assumptions hide in the descriptions of the real and ideal computational models.

- *Static vs. Mobile Adversary*: A static adversary chooses which parties to corrupt before protocol begins, and may not corrupt more parties during execution.
- *Secure Channels*: Parties have perfectly secure point-to-point communication channels.
- *Synchronicity*: Synchronous protocols assume communication proceeds in rounds, and each message sent in a round is delivered in that round (by the adversary).
- *Security with abort*: Adversary may learn prescribed outputs and abort the protocol, leaving honest parties with nothing.
- *Fairness*: Everyone receives output or nobody does.
- *Guaranteed Output Delivery*: Honest parties always receive outputs.

A Semi-Honest to Malicious Compiler under Honest Majority

- CCS'17 Yehuda Lindell and Ariel Nof [LN17][1]
- Template for semi-honest to malicious MPC compiler under honest majority.
- Concretely efficient: 0.5 seconds for 3PC of depth 20 arithmetic circuit with 1mil multiplication gates. 29 seconds for 50 parties; 1min for 90 parties.
- Entirely information-theoretic, only uses secret sharing.
- Main idea: (0) Precompute some triples; (1) Check correctness of shares on input wires; (2) run rest of semi-honest protocol; (3) use triples to do a batch verification of all multiplication gates at the end.

[1] A follow-up work by Chida *et al.* [CHI+23] significantly outperforms this compiler if the semi-honest multiplication protocol is maliciously secure up to additive attacks (this is true for Shamir's secret sharing). We present the [LN17] compiler for pedagogical purposes as it follows the popular triple preprocessing template.

Preliminaries

- A sharing $[v]$ is *correct* if it defines a single secret.
- Let open($[v]$) be a protocol guaranteeing:
  1. If $[v]$ is correct, each party outputs v (or aborts).
  2. Else, all honest parties abort.

  Let $\text{val}([v])_\Theta$ denote value output by set of parties Θ, $|\Theta| > t$ running open($[v]$).
- Let $\mathcal{F}_{\text{RAND}}$ be ideal functionality that outputs sharing $[r]$ for $r \xleftarrow{\$} \mathbb{F}$.
- Let $\mathcal{F}_{\text{COIN}}$ be ideal functionality that outputs $r \xleftarrow{\$} \mathbb{F} \setminus \{0\}$ to all.
- A *triple* is a tuple of shares $([a], [b], [c])$ s.t. $c = ab$. A triple is random if $a, b \xleftarrow{\$} \mathbb{F}$.

An Important Lemma

Let $[u]$ be an incorrect sharing, and $[v]$ any sharing. Then the probability $[w] = \alpha[u] + [v]$ is correct for $\alpha \xleftarrow{\$} \mathbb{F} \setminus \{0\}$ is at most $\frac{1}{|\mathbb{F}|-1}$.

Proof Sketch. Analyze several cases. Most instructive case is where $[u]$ and $[v]$ are both incorrect. Let J_1, J_2 be size $t+1$ subsets of honest parties. Let $u_1, u_2, v_1, v_2 \in \mathbb{F}$ such that $\text{val}([u])_{J_1} = u_1 \neq u_2 = \text{val}([u])_{J_2}$, and $\text{val}([v])_{J_1} = v_1 \neq v_2 = \text{val}([v])_{J_2}$. The event $\text{val}([w])_{J_1} = \text{val}([w])_{J_2}$ (which is necessary for $[w]$ to be correct) is equivalent to $\alpha u_1 + v_1 = \alpha u_2 + v_2$. This happens iff $\alpha = (v_2 - v_1)/(u_1 - u_2)$, which happens with probability $\frac{1}{|\mathbb{F}|-1}$.

Batch Correctness Check of Shares

PROTOCOL 3.1 (Batch Correctness Check of Shares).

- **Inputs:** The parties hold m shares $[x_1], \ldots, [x_m]$.
- **The protocol:**
  (1) The parties call $\mathcal{F}_{\text{coin}}$ to receive random elements $\rho_1, \ldots, \rho_m \in \mathbb{F} \setminus \{0\}$.
  (2) The parties call $\mathcal{F}_{\text{rand}}$ and obtain a sharing $[r]$.
  (3) The parties locally compute
  $$[v] = \rho_1 \cdot [x_1] + \ldots + \rho_m \cdot [x_m] + [r]$$
  (4) The parties run open($[v]$).
  (5) If no abort message was received, then the parties output accept.

Claim: any incorrect shares result in abort w.h.p. Proof: Spam lemma m times.

Triple Verification

PROTOCOL 3.4 (Triple Verification Using Open).

- **Inputs:** The parties hold a triple $([x], [y], [z])$ to verify and an additional random triple $([a], [b], [c])$.
- **The protocol:**
  (1) The parties call $\mathcal{F}_{\text{coin}}$ to generate a random $\alpha \in \mathbb{F} \setminus \{0\}$.
  (2) Each party locally computes $[\rho] = \alpha \cdot [x] + [a]$ and $[\sigma] = [y] + [b]$.
  (3) The parties run open($[\rho]$) and open($[\sigma]$), as defined in Section 2, to receive ρ and σ. If a party receives $\perp$ in an opening, then it sends $\perp$ to all the other parties and aborts.
  (4) Each party locally computes
  $$[v] = \alpha[z] - [c] + \sigma \cdot [a] + \rho \cdot [b] - \rho \cdot \sigma.$$
  (5) The parties run the open($[v]$) procedure to receive v. If a party receives $\perp$ in the opening, then it sends $\perp$ to all the other parties and aborts.
  (6) Each party checks that $v = 0$. If not, then it sends $\perp$ to the other parties and aborts.
  (7) If no abort messages are received, then the parties output accept.

- Correctness: convince yourself $v = 0$ if all shares and triples are correct.
- Security: Analyze probability of correct shares/triples in several cases. Apply lemma in one of these cases.

Outline

1 Setup

2 Theory
- Secret Sharing
- MPC

3 MP-SPDZ

4 References

What is MP-SPDZ?

- A "framework for multi-party computation"... whatever that means.
- A front end *domain specific language* (DSL) lets you write programs in a high-level language and compile them to arithmetic circuits.
- A back end *MPC engine* (*i.e.*, the binary executable run by each party) lets you securely evaluate those circuits with the chosen MPC protocol.
- In more detail, parties securely execute a *virtual machine* that takes as input the arithmetic circuit $\mathcal{C}$ and party inputs $x_1, \ldots, x_n$, and outputs $C(x_1, \ldots, x_n)$. Note the arithmetic circuit is encoded as an assembly program in the VM's instruction set architecture (ISA).

The MP-SPDZ DSL

- Technically, a DSL program is just a Python program importing special "runtime" data types (representing wire-values) and other helpful objects from the MP-SPDZ `Compiler` package.
- It is important to understand the distinction between "compile-time" data types (*e.g.*, the usual Python types `int`, `list`, `str`) and runtime data types (*e.g.*, `sgf2n`, `cgf2n`, `sint`, `cint`).
- Maybe best to just try it yourself instead of explaining further...

Exercises

1. Univariate Polynomial Evaluation (`Programs/Source/poly.py`)
2. Yao's Millionaires Problem (`Programs/Source/yao.py`)
3. Dot product preimage sampling (`Programs/Source/preimage.py`)

Help each other, ask me for help, ask codex for help.

MPC in the Real World

- Berkeley maintains watchlist for MPC deployments `https://mpc.cs.berkeley.edu/`
- Wallet-as-a-service. Use MPC for threshold signing and proactive share refresh. Coinbase and Fireblocks.
- Secure private auctions in blockchains (participants constantly bidding to have their transactions executed)
- Secure analytics: hospitals, government agencies, fraud detection.
- Privacy-preserving ML: MP-SPDZ has circuits for AlexNet training and Bert inference.

Outline

1. Setup
2. Theory
   - Secret Sharing
   - MPC
3. MP-SPDZ
4. References

Amos Beimel.
Secure schemes for secret sharing and key distribution.
PhD thesis, Israel Institute of Technology, Technion, 1996.

Omer Barkol, Yuval Ishai, and Enav Weinreb.
On *d*-multiplicative secret sharing.
J. Cryptol., 23(4):580–593, 2010.

Ronald Cramer, Ivan Damgård, and Ueli M. Maurer.
General secure multi-party computation from any linear secret-sharing scheme.
In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 316–334. Springer, 2000.


Koji Chida, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Daniel Genkin, Yehuda Lindell, and Ariel Nof.
Fast large-scale honest-majority mpc for malicious adversaries.
Journal of Cryptology, 36(3):15, 2023.


Yehuda Lindell and Ariel Nof.
A framework for constructing fast mpc over arithmetic circuits with malicious adversaries and an honest-majority.
In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 259–276, 2017.