SMCompiler

SMCompiler in a nutshell

High-level tasks:

- Implement an SMC protocol
- Evaluate its performance
- Use in an application

Deliverables: 2-page report and code
Deadline: **March 28th, 2025** at 23:59
Form groups (https://forms.gle/3LANqQrQCf2dTLef7) before **February 28**

Implementation Goal*: Convenient Python library for SMC

*very simplified

```
# Define secrets
alice_secret = Secret()
bob_secret = Secret()

# Define arithmetic circuit (=“expression”)
expr = alice_secret + bob_secret

# Alice runs protocol, communicating with Bob and third parties
run_protocol(expr, value_dict={alice_secret: 5})

# Bob runs protocol, communicating with Alice and third parties
run_protocol(expr, value_dict={bob_secret: 12})
```

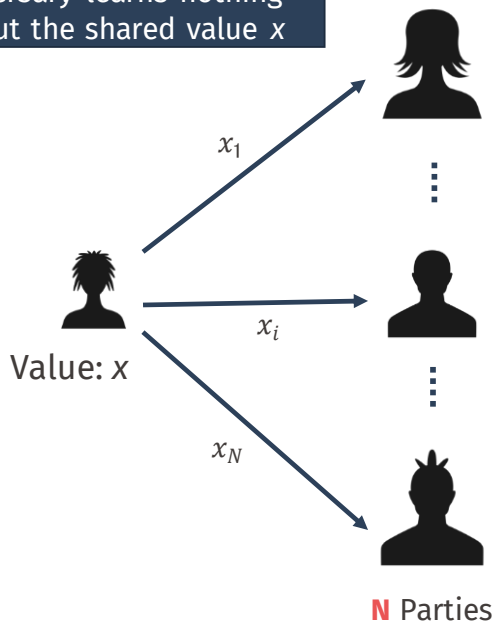
1. Implementation: SMC with Secret Sharing

Building Block
Additive Secret Shares

Privacy Property: given at most *N – 1* shares, an adversary learns nothing about the shared value *x*

Operate over a field $\mathbb{F}$ (for example, integers modulo a prime *p*)

- **Share:** given a value $x \in \mathbb{F}$ we compute shares $x_1, \cdots, x_N$:
  - Sample $x_2, \cdots, x_N$ uniformly at random from $\mathbb{F}$
  - Set $x_1 = x - \sigma_{\{i=2\}}^{N} x_i$ (over $\mathbb{F}$)
  - We denote $[x] = \{x_1, \cdots, x_N\}$ the sharing of x
- **Reconstruction:** given a sharing $[x] = \{x_1, \cdots, x_N\}$ output $x = \sigma_{\{i=1\}}^{N} x_i$



1. Implementation: Secret Sharing and Addition

Computing on shares
Addition (Add-Protocol)

General Structure/Invariant: the parties in the protocol hold secret shares of the circuit wire values.

Here: Party *i* holds secret shares s_i, v_i such that: $s = \sum_i s_i$ and $v = \sum_i v_i$.

Goal: Each party must obtain t_i such that $t = \sum_i t_i = s + v$ or in other words $[t] = [s + v]$

Algorithm:

- Each party (locally!) sets $t_i = s_i + v_i$

1. Implementation: Multiplication using Beaver triplets

Computing on shares
Multiplication (Mul-Protocol)

Here: Party *i* holds secret shares s_i, v_i such that: $s = \sum_i s_i$ and $v = \sum_i v_i$ as well as shares a_i, b_i, c_i, for a *fresh* Beaver Triplet (a, b, c)

Goal: Each party must obtain t_i such that $t = \sum_i t_i = sv$ or in other words $[t] = [sv]$

A useful identity:

$$\begin{aligned} sv &= (s - a + a)(v - b + b) \\ &= (\mathrm{d} + \mathrm{a})(\mathrm{e} + \mathrm{b}) \\ &= de + db + ae + ab \\ &= de + db + ae + c \end{aligned}$$

Algorithm:

1. Each party locally computes a share of *[d] = [s – a]* and broadcasts it. As a result, everybody learns *d*
2. Each party locally computes a share of *[e] = [v – b]* and broadcasts it. As a result, everybody learns *e*
3. Locally compute a share of:
   $[sv] = de + d[b] + e[a] + [c]$
   (note that this requires only additions and multiplications by constants)

1. Implementation – we provide:

- Skeleton of the implementation in Python
- Test suite that your implementation has to satisfy (`test_integration.py`)
- Code handling networking and communication

github.com/spring-epfl/CS-523-public

1. Implementation - Overview of the skeleton

Your implementation should normally reside in these files:

- `secret_sharing.py`—Secret sharing scheme
- `expression.py`—Tools for defining arithmetic circuits (=“expressions”)
- `ttp.py`—Trusted parameter generator for the Beaver multiplication scheme
- `smc_party.py`—SMC party implementation

Some code that will help you out:

- `protocol.py`—Specification of SMC protocol
- `communication.py`—SMC party-side of communication
- `server.py`—Trusted server to exchange information between SMC parties

Tests

- `test_integration.py`—Integration test suite. **Your implementation must pass these.**
- Some templates of test files for you to start from

1. Implementation - Communication

- `send_private_message(receiver, label, message)`
- `retrieve_private_message(label)`

- `publish_message(label, message)`
- `retrieve_public_message(sender_identifier, label)`

- `retrieve_beaver_triplet_shares(operation_identifier)`

2. Evaluation

Measure costs = runtime and communication

- Effect of the number of parties on costs
- Effect of the number of additions on costs
- Effect of the number of multiplication on costs

3. Application

Requirements:

- Involves multiple parties
- Uses all types of operations

Implementation: Test the correctness of your implementation of the circuit

Analysis:

- Motivation for your application
- Threat model
- Privacy properties: SMC guarantees that parties learn nothing but the output, but the output itself may leak private information

Deliverables

Report:

- Use the report template provided in the repository
- In the introduction, clearly state what each group member did

SMCompiler in a nutshell

High-level tasks:

- Implement an SMC protocol
- Evaluate its performance
- Use in an application

Deliverables: 2-page report and code
Deadline: **March 28th, 2025** at 23:59
Form groups (https://forms.gle/3LANqQrQCf2dTLef7) before **February 28**