SIMAP

Secure Information Management and Processing

Martin Geisler, Mikkel Krøigaard,
Janus Dam Nielsen, and Tomas Toft

BRICS Research School
University of Aarhus
SMCL

The Secure Multiparty Computation Language

Work in progress

VAM1-Bristol’07
Outline

- Motivation
- Secure Multiparty Computation
- SMCL
- Why and What?
- SMCR (How?)
Motivation

The Millionaire's problem, Yao 1982
n parties P1,...,Pn wish to jointly compute the computable function: f(x₁,...,xn)

Party Pi only knows the input value xi which must be kept secret from the other parties.

Even if some adversary has power to corrupt some subset of the parties
SMCL - Why?

- Writing SMC programs is tedious and error-prone

DSL:
- Important concepts up front (concise)
- Efficiency
- Management
- Analyze (security)
SMCL – What

- High level domain specific language
- Language support for fundamental concepts
- Parties are separated into clients and servers
SMCL
The Millionaire's Example

Declare Client Millionaires:

Tunnel of sint netWorth;

function void main(int[] args) {
    ask();
}

function void ask() {
    netWorth.put(readInt());
}

function void tell(bool b) {
    if (b) {
        display("You are the richest!");
    } else {
        display("Make more money!");
    }
}


Declare Server Max:

Group of Millionaires mills;

function void main(int[] args) {
    sint max = 0;
    sclient rich;

    foreach (client c in mills) {
        sint netWorth = c.netWorth.take();
        if (netWorth >= max) {
            max = netWorth;
            rich = c;
        }
    }

    max = b*netWorth+(1-b)*max

    foreach (client c in mills) {
        c.tell(open(c==rich|rich));
    }
}
The server is the Trusted Third Party

Alice

Server

Bob
Concepts

Clients:
Public values
- (Bools, Ints, Records)
Fields
Tunnels
Functions
- callable from server

Server:
Public & secret values
- (Bools, Ints, Records, clients)
Fields
Groups of clients
Functions
Security
at the language level

- Preventing covert channels:
  - Direct and indirect information flow
  - Timing and termination leaks
  - Open and responsibilities
  - etc.
```c
sbool h = ...;
sint i = 0;
int l = 0;
if (h) {
    i = 7 * h;
    l = 7;
} else {
    l = 42;
}
open(i|h);
```
SMCLc

- Compiler: SMCLc (alpha)
- Available from www.BRICS.dk/SMCL/
SMCR
The Secure Multiparty Computation Runtime
Overview of the Runtime

- Implements an ideal functionality
- Provides the primitives used by the compiler:
  - Secret sharing input
  - Opening sharings
  - Arithmetic (addition and multiplication)
  - Comparison
- Security against passive adversaries
Design of the Runtime System

- Decoupled from the language (thin interface to compiler)
- Modularity
- Ability to exchange implementation of primitives
Primitives: Sharing and Opening

- Input is secret shared using an additively homomorphic secret sharing system over $\mathbb{Z}_p$
- Basic shares are standard Shamir-sharing
  - Other techniques for sharing used in special cases (e.g. PRSS)
- Output is reconstructed by opening shares when enough parties agree
Primitives: Addition

- Add shares together
- Requires no communication, free in our complexity model
- Corollary: arbitrary linear combinations are free
Primitives: Multiplication

- Standard GRR: multiply shares, reshare result
- Requires a round of communication
- Basic unit of complexity
Primitives: Comparison

- Complex protocol using the other arithmetic primitives
- Seen as a primitive by the compiler
- Most expensive operation: 10-12 communication rounds
- Number of multiplications: linear in bitlength
- With preprocessing: ~2 communication rounds
- Faster special cases: equality, public result, comparison of public and secret integers, etc.
Primitives: Comparison

Some ideas for computing “a>b?”:

- Compute $c = 2^l + a - b$, extract the $l$’th bit of $c$ (e.g. compute $c \mod 2^l$)
- Extract the bit of $a$ at the most significant bit position where $a$ and $b$ differ (assuming bit sharings are available)
Possibilities for Optimization

- Multiplications require a round of communication
  
  - Run independent multiplications in parallel!

- Do the same for comparison

- Tradeoff between round complexity and number of multiplications
Future Work

- Explore possibilities for better primitives
- Construct and implement applications: e.g. simplex
- Intermediate language for writing complex primitives for thin runtime system.
- Security against active adversaries and self-trust
Questions?