Qualification Exam

A Domain-Specific Programming Language
for
Secure Multiparty Computation

Janus Dam Nielsen
Creating tools with strong security guarantees which exploits the benefits obtained by combining confidential information without compromising it, is feasible and useful.

Report on SMCL
Overview

• Secure Multiparty Computation
• SMCL Concepts
• An example
• Security - what, why
• Efficiency
• Future Work
• Conclusion
Secure Multiparty Computation

- n parties P1,...,Pn wish to jointly compute the computable function: \( f(x_1,\ldots,x_n) \)

- Party Pi only knows the input value \( x_i \) which must be kept secret from the other parties.

- Even if some adversary has power to corrupt some subset of the parties
The Millionaire’s Example
SMC Solves Problems

- Auctions
- Distributed Voting
- Matchmaking
- Benchmarking
Overview

• Secure Multiparty Computation

• SMCL Concepts

• An example

• Security - what, why

• Efficiency

• Future Work

• Conclusion
Conceptual Model
Values

Clients:
Private values
- Booleans
- Integers
- Records

Server:
Public & Secret values
- Booleans
- Secret booleans
- Integers
- Secret integers
- Records
- Client identity
- Secret client identity
Communication

Clients:
Tunnels:
- Asynchronous
- put and get functions
- Primitive types only
- Data encrypted
- Secret data - shared and encrypted

Functions:
- Synchronous
- Primitive types only
- Invoked by server

Server:
Tunnels:
- Accessed via client identity
- put and get functions
Client Identity

Clients:

Server:

Groups of clients:
- A set of clients
- All of the same kind
- Iterated using a for loop
- Uniform treatment of clients
- Secrecy of client identity
- Specified externally
Overview

• Secure Multiparty Computation
• SMCL Concepts
• An example
• Security - what, why
• Efficiency
• Future Work
• Conclusion
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) {
            if (netWorth >= max) {
                max = netWorth;
                rich = c;
            }
        }

        foreach (client c in mills) {
            c.tell(open(c==rich|rich));
        }
    }

    function void ask() {
        netWorth = c.netWorth.take();
        if (netWorth >= max) {
            max = netWorth;
            rich = c;
        }
    }
Overview

- Secure Multiparty Computation
- SMCL Concepts
- An example
- Security - what, why
- Efficiency
- Future Work
- Conclusion
Security

- Identity property
- Commutative property

Adversary may:
- Observe physical state of the server
- Not observe private and secret values
Adversary Traces

- A sequence of states of an entire computation
- Secret values are masked out
- Private state of clients not available
- No declassification
Identity Property

- $p' = p''$ - Low equiv.
- Traces must be identical
- Prevents attacks which are a function of the trace (e.g. timing)
- Requires basic operations independent of arguments
Commutative Property

- Soundness of secret representation
Ensuring Security

- Carefully crafted semantics
- Static analysis of well-typed SMCL programs
Semantics

- Conditionals are a source of differences in trace
- Execute both branches
- Termination

\[
x = b \times y + (1-b) \times z
\]
\[
G \vdash \langle C_2, S \rangle \rightarrow \text{COM}_sv \langle C'_2, S' \rangle \\
G \vdash \langle \text{if} [\square] \{ \} \text{ else } \{ C_2 \}, U\text{then}, S \rangle \rightarrow \text{COM}_sv \langle \text{if} [\square] \{ \} \text{ else } \{ C'_2 \}, S', U\text{then}, S \rangle
\]

\[\text{(if-sbool-else)}\]

\[
\sigma . S' = S[x \rightarrow \square \ast U\text{then}(x) + (1 - \square) \ast U\text{else}(x)] \\
\forall x \in S | U\text{then}(x) = v = U\text{else}(x) \vee U\text{then}(x) = v' \land U\text{else}(x) = v''
\]

\[G \vdash \langle \text{if} [\square] \{ \} \text{ else } \{ \}, U\text{else}, U\text{then}, S \rangle \rightarrow \text{COM}_sv \langle \sigma . S' \rangle\]

\[\text{(if-sbool-phi)}\]
Hoistability

- Branches must agree on public side-effects
  - Assignment to public variables
  - Communication
  - Function calls
- While loops and recursion with secret condition - not allowed
\[
\begin{array}{c}
\Gamma_t \vdash e : (\text{bool}, S, v, \iota_0)-\text{exp} \\
\Gamma_t \vdash C_1 : (\text{PL}, \text{NIO})\text{-cmd} \quad \Gamma_t \vdash C_2 : (\text{PL}, \text{NIO})\text{-cmd} \\
\hline
\Gamma_t \vdash \text{if} (e) \{C_1\} \text{ else } \{C_2\} : (\text{PL} \hat{\land} v, \iota_0)-\text{cmd} \\
(\text{TIP-SECRET})
\end{array}
\]

\[
\begin{array}{c}
\Gamma_t \vdash e : (\text{bool}, P, v_0, \iota_0)-\text{exp} \\
\Gamma_t \vdash C_1 : (v_1, \iota_0_1)\text{-cmd} \quad \Gamma_t \vdash C_2 : (v_2, \iota_0_2)\text{-cmd} \\
\hline
\Gamma_t \vdash \text{if} (e) \{C_1\} \text{ else } \{C_2\} : (\frac{2}{\land} v_i, \frac{2}{\lor} \iota_{0_i})\text{-cmd} \\
(\text{TIP-PUBLIC})
\end{array}
\]
Semantic Security

- Ideal computations are inefficient
- Prove that a pragmatic version reveals same information as the ideal version
- Assist the programmer

Ideal computation

open(e|x,y,z)
Overview

- Secure Multiparty Computation
- SMCL Concepts
- An example
- Security - what, why
- Efficiency
- Future Work
- Conclusion
Efficiency

<table>
<thead>
<tr>
<th>(parties, threshold)</th>
<th>ideal</th>
<th>pragmatic</th>
<th>public</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3,1)</td>
<td>12 sec</td>
<td>30 ms</td>
<td>&lt; 1 ms</td>
</tr>
<tr>
<td>(5,2)</td>
<td>17 sec</td>
<td>65 ms</td>
<td>&lt; 1 ms</td>
</tr>
<tr>
<td>(7,3)</td>
<td>30 sec</td>
<td>132 ms</td>
<td>&lt; 1 ms</td>
</tr>
</tbody>
</table>

Ideal

sint x = 17;
sint a = 42;
sint b = -5;
sint c = 87;
sint p = a*(x*x) + b*x +c
sint sign = 0;
int output;
if (p<0) sign = -1;
if (p>0) sign = 1;
output = open(sign|p);

Pragmatic

int x = 17;
sint a = 42;
sint b = -5;
sint c = 87;
int p = open(a*(x*x) + b*x +c|a,b,c)
int sign = 0;
int output;
if (p<0) sign = -1;
if (p>0) sign = 1;
output = sign;
Future Work

- SMCL
  - Formalize Adversary traces
  - Dynamic groups
  - Secret compound datatypes
  - More elaborate examples
- SecRas
- SVM, SPL...
Conclusion

- A DSL for SMC
- High-level abstractions
- Strong security guarantees
- Useful in practice
Questions?