Optimizing Secure Multiparty Computation Programs with Private Conditions

Peeter Laud
joint work with Alisa Pankova

07.02.2015
Secure multiparty computation

- $n$ parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, with inputs $x_1, \ldots, x_n$.
- Want to compute $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$, where $\mathcal{P}_i$ learns $y_i$.
- $\mathcal{P}_i$ should not learn anything beyond $x_i, y_i$. More generally, for any $I \subseteq \{1, \ldots, n\}$, $|I| < t$, the coalition $\{\mathcal{P}_i\}_{i \in I}$ should not learn anything beyond $(x_i, y_i)_{i \in I}$.
- $\mathcal{P}_1, \ldots, \mathcal{P}_n$ run a cryptographic protocol for that end.
The Sharemind model for Secure Computation

Three computing parties

Passive security against one party

Additive sharing of values:
\[ [v] = ([v]_1, [v]_2, [v]_3) \]
\[ [v]_1 + [v]_2 + [v]_3 = v \ (\text{in } \mathbb{Z}_N) \]

Some other sharing mechanisms also available

\( CP_1, CP_2, CP_3 \) run protocols to compute sharings of outputs from the sharings of inputs

At least as secure as ...
Ideal functionality $F_{ABB}$

- receives inputs from $IP_A^i$ and stores them;
- fulfills orders from $CP_A^1$, $CP_A^2$, $CP_A^3$ to perform computation operations on stored values;
- $CP_A^i$ refer to these values through handles only
- sends values to $OP_A^i$, if ordered by $CP_A^1$, $CP_A^2$, $CP_A^3$.

Hence $CP_A^i$ remain oblivious to the actual values
Building large applications

```plaintext
private int a[], b, c, x[], s
public int e, i
b = read_private_int()
a = read_private_vector()
e = read_public_int()
c = 2 * b + (a.len * e - 3)
s = 0
for i = 1 to a.len do
  s = s + a[i]
  x[i] = a[i] - c * a[i-1]
done
e = declassify(s) / a.len
```

07.02.2015
Building large applications

private int $a[]$, $b$, $c$, $x[]$, $s$
public int $e$, $i$

$b = \text{read\_private\_int}()$
$a = \text{read\_private\_vector}()$
$e = \text{read\_public\_int}()$
$c = 2 \times b + (a\text{.len} \times e - 3)$
$s = 0$
for $i = 1$ to $a\text{.len}$ do
    $s = s + a[i]$
    $x[i] = a[i] - c \times a[i-1]$
done
e = \text{declassify}(s) / a\text{.len}$
Conditional statements

- No attempt to hide the control flow

<table>
<thead>
<tr>
<th>if b &gt; 5 then</th>
<th>else</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = b / c + d</td>
<td>b = c * z</td>
</tr>
<tr>
<td>c = b * e</td>
<td>if b &gt; a</td>
</tr>
<tr>
<td>if a &gt; x</td>
<td>else</td>
</tr>
<tr>
<td>b = 10</td>
<td>c = a / y</td>
</tr>
<tr>
<td>b = c * y</td>
<td>c = d</td>
</tr>
</tbody>
</table>

This is compiled into...
<table>
<thead>
<tr>
<th>if ( b &gt; 5 ) then</th>
<th>else</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a = b / c + d )</td>
<td>( b = c * z )</td>
</tr>
<tr>
<td>( c = b * e )</td>
<td>( if \ b &gt; a )</td>
</tr>
<tr>
<td>( if \ a &gt; x )</td>
<td>( else )</td>
</tr>
<tr>
<td>( b = 10 )</td>
<td>( c = a / y )</td>
</tr>
<tr>
<td>( b = c * y )</td>
<td>( c = d )</td>
</tr>
</tbody>
</table>

\[ C = (b > 5) \]
\[ t1 = b / c \]
\[ a1 = t1 + d \]
\[ c1 = b * e \]
\[ C1 = (a1 > x) \]
\[ b11 = 10 \]
\[ b12 = c1 * y \]
\[ b1 = C1 ? b11 : b12 \]

\[ b2 = c * z \]
\[ C2 = (b2 > a) \]
\[ c21 = a / y \]
\[ c22 = d \]
\[ c2 = C2 ? c21 : c22 \]
\[ a = C ? a1 : a \]
\[ b = C ? b1 : b2 \]
\[ c = C ? c1 : c2 \]

**Oblivious choice**

\( c ? x : y \) is realized as \( c * (x - y) + y \)
if \( b > 5 \) then
\[
\begin{align*}
    a &= b / c + d \\
    c &= b * e \\
    \text{if } a > x \text{ then } b &= 10 \text{ else } b &= c * y
\end{align*}
\]
else
\[
\begin{align*}
    b &= c * z \\
    \text{if } b > a \text{ then } c &= a / y \text{ else } c &= d
\end{align*}
\]

\[ C = (b > 5) \]
\[ \text{arg1} = C \ ? \ b \ : \ a \]
\[ \text{arg2} = C \ ? \ c \ : \ y \]
\[ \text{res} = \text{arg1} / \text{arg2} \]
\[ a1 = \text{res} + d \]
\[ c1 = b * e \]
\[ C1 = (a1 > x) \]
\[ b11 = 10 \]
\[ b12 = c1 * y \]
\[ b1 = C1 \ ? \ b11 \ : \ b12 \]

\[ \text{We try to merge some operations together in the converted program} \]

\[ \text{b2} = c * z \]
\[ C2 = (b2 > a) \]
\[ c22 = d \]
\[ c2 = C2 \ ? \ \text{res} \ : \ c22 \]
\[ a = C \ ? \ a1 \ : \ a \]
\[ b = C \ ? \ b1 \ : \ b2 \]
\[ c = C \ ? \ c1 \ : \ c2 \]
A combinatorial optimization task

- Let the converted program be represented as an arithmetic circuit
- Our goal then is to merge some gates together
A combinatorial optimization task

- Let the converted program be represented as an arithmetic circuit.
- Our goal then is to merge some gates together.
- Choosing which gates should be merged is not straightforward.

These two mergings are mutually exclusive.

- Which merging is the best possible?
Hiding the structure of the circuit

- There are some gates
- The results of some of them have to be routed to the inputs of some other gates
- The actual routing is private
- We have an instance of private function evaluation
- There are several possible operations to do the routing
  - Oblivious extended permutations
  - Oblivious choices
- It turned out that oblivious choices perform better
  - ... because most of the structure of the circuit is still known
A combinatorial optimization task

To solve the task, let us use
Integer Linear Programming

- A task of the form:

  \[
  \text{minimize } c^T \cdot x, \text{ subject to } Ax \leq b, x_i \in \mathbb{N}
  \]

- The task is in general NP-hard.
- However, there exist good solvers for it.
The Cost

- Let $C_i$ be a set of merged gates.

For each $C_i$, we eventually get one gate.

- The total cost is $\sum_i \text{cost}(C_i)$.

- The cost of $C_i$ is equal to the cost of the operation performed there.
Putting Gates Together

For $i \in \mathcal{J}$ let $g_i$ be the gates, and $C_i$ the sets.

$\text{Excl} = \{(i, j) \mid \text{gates } g_i \text{ and } g_j \text{ are mutually exclusive}\}$.

Let $g^j_i \in \{0, 1\}$, $g^j_i = 1$ iff $g_i \in C_j$. 
Constraints (I)

1. Non-mutually-exclusive gates are not merged. 
   \[ g^j_i + g^j_k \leq 1 \text{ for } i, k \in J, \ i \neq k, \ (i, k) \notin Excl. \]
2. Each gate belongs to exactly one set. 
   \[ \sum_{j=1}^{|J|} g^j_i = 1 \text{ for all } i \in J. \]
3. If \( C_j \neq \emptyset \), then \( g_j \in C_j \). This breaks symmetries. 
   \[ g^j_j \geq g^j_i \text{ for all } i \in J, \ j \in J. \]
4. The set and gate types should match. 
   \[ g^j_i = 0 \text{ if } op(g_i) \neq op(C_j) := op(g_j). \]
Auxiliary constructions of constraints

- We define auxiliary constraints that will be used multiple times:
  
  \[(\exists \geq k i \in [n] : x_i = 1) \implies x = 1 \quad (n + 1 - k)x - \sum_{i \in [n]} x_i \geq (1 - k)\]
  
  \[-(\exists \geq k i \in [n] : x_i = 1) \implies x = 0 \quad kx - \sum_{i \in [n]} x_i \leq 0\]

- This allows us for example to express the and operation.

  \[x = x_1 \land x_2:\]
  
  \[(\exists \geq 2 i \in [2] : x_i = 1) \implies x = 1.\]

Remarks

- These constructions are valid for variables with values in \{0, 1\}.
- \([n]\) denotes the set \{1, 2, \ldots, n\}
Avoiding cycles

- Merging only mutually exclusive gates is not sufficient.

\[ \text{Pred} = \{(k, \ell) \mid \text{gate } g_k \text{ is a predecessor of } g_\ell\} \]

- Introduce variables \( l_i \) and \( m_j \) indicating the level of each gate and set.
  - \( l_i \) and \( m_j \) belong to set \( \{0, 1, \ldots, M\} \) for some large \( M \)
  - Constraints: \( l_k < l_\ell \) for \((k, \ell) \in \text{Pred}\)
Constraints (II)

All gates in the same set $C_j$ have the same level $m_j$

$g^j_i = 1 \Rightarrow l_i = m_j$

- Consider the constraint $l_i - m_j + M(1 - g^j_i) \geq 0$
- This expresses $g^j_i = 1 \Rightarrow l_i \geq m_j$
Cost of oblivious choices

Oblivious choice in general form

- Bits $b_1, \ldots, b_n$, values $v_1, \ldots, v_n$. Select $v_i$, where $b_i = 1$
- Compute $b_1 v_1 + \cdots + b_n v_n = b_1(v_1 - v_n) + \cdots + b_{n-1}(v_{n-1} - v_n) + v_n$
- Cost: $(n - 1)$ multiplications

- Count the number of distinct $\ell$-th inputs of $C_j$.
- Let $x_k$ for $k \in K$ be an input of some gate $g_i$.
- $K = \{(k, k') \mid \text{the input } x_k \text{ is an output of a gate } g_{k'}\}$. 

07.02.2015
Constraints (III)

1. $f_{x_{jk}} = 1$ iff $x_k$ is used as the $\ell$-th input of some gate of $C_j$: 
   
   $(\exists i \in J, x_k$ is the $\ell$-th input of $g_i : g_i^j = 1) \iff f_{x_{jk}} = 1$).

2. $e_{i \ell} = 1$ iff $x_k$ is used as the $\ell$-th input by $C_j$, and $g_{k'} \in C_i$:
   
   $f_{x_{jk}} \land g_{i \ell} = e_{i \ell}$ for all $k \in K, (k, k') \in K$.

3. $f_{c_{ji}} = 1$ iff $C_i$ has at least one $x_k$ as the $\ell$-th input of $C_j$:
   
   $(\exists k \in K : e_{i \ell} = 1) \iff (f_{c_{ji}} = 1)$ for all $i \in J$.

4. $s_{j \ell}$ counts distinct $\ell$-th inputs after merging gates of $C_j$:
   
   $s_{j \ell} = \sum_{k \in K} f_{x_{jk}} + \sum_{i \in J} f_{c_{ji}}$

Cost: $\sum_{j \in J} g_j^j \cdot \text{cost}(g_j) + \sum_{j \in J} (s_{j \ell} - 1) \cdot \text{cost}(\text{mul})$.
Sharemind-specific: One-time Operations

- In some multiplication protocols, if $a \times b$ and $b \times c$ are already computed, then $a \times c$ is free.
- A special operation replication is applied to each input.
- The multiplication itself is then free.

![Diagram showing the replication process](image)
Include replications in the cost

- For each gate’s output, we find whether it should be replicated
  - Is it an input to an oblivious choice, or to multiplication?
- Oblivious choices are needed only if there are at least two possible inputs
- Cost of oblivious choices and multiplications — zero
  - Cost of replication — something non-zero
What we have got

Take the initial program with private conditions.

Transform it to a flow graph with final oblivious choices.

minimize $c^T x$
subject to $A x \leq b$
$x \in \{0, 1\}$

Construct an integer linear programming task.

Get a runnable bytecode based on the graph.

Reconstruct the graph according to the solution.

Solve it using an existing solver (such as glpk), get the optimal solution.
Implementation

- Constraint generator, circuit transformer implemented in SWI-Prolog
- GLPK used to solve ILP
- Toy examples (30 lines) run in a few seconds
Conclusion

- We have presented an optimization for privacy-preserving programs with private conditions.
- We have implemented the optimization in SWI-Prolog and tested it on small simple imperative programs.
- Bigger examples probably require some new ideas, though