Secure function evaluation using garbled circuits built from propositional formulae.

Timo Petmanson
Oleg Šelajev
Agenda

- Secure evaluation existence
- Boolean circuits
  - Custom xml representation
- Function $\Rightarrow$ boolean formulae
- Garbling
- Encryption
  - AES scheme
- Garbled evaluation
- Results and conclusion
Secure evaluation existence

- Data is private
- Decisions are based on data
- Collaboration of parties
- Secure evaluation
There is a function $f : ({0, 1}^\ell)_n \rightarrow ({0, 1}^\ell)_n$.

There are $n$ parties, $P_1, \ldots, P_n$.

Party $P_i$ has the bit-string $x_i \in \{0, 1\}^\ell$.

They want to evaluate a function $f$ and learn the output.

In our case:

Number of parties = 2
One of them constructs the circuit
The second one evaluates it using inputs from both
Boolean circuits

- Andrew Chi-Chih Yao. *Protocols for secure computations*. 1982

- Circuit model
  - Wires
  - Gates

- Directed acyclic graph
  - edges must be ordered!
Boolean circuits 2

- XML representation:
  - schema
Boolean circuit XML - wire

<xs:element name="edge">
  <xs:complexType>
    <xs:attribute name="srcIndex" type="xs:integer" use="required" />
    <xs:attribute name="dstIndex" type="xs:integer" use="required" />
    <xs:attribute name="src" type="xs:integer" />
    <xs:attribute name="dst" type="xs:integer" />
  </xs:complexType>
</xs:element>
Wire and Gate

value = \{0, 1\}
Boolean circuit XML - gate

<xs:element name="node">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="table" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="numInputs" type="xs:string" />
    <xs:attribute name="numOutputs" type="xs:string" />
    <xs:attribute name="id" type="xs:integer" />
  </xs:complexType>
</xs:element>
Circuit

Two bit numbers equality circuit
<xs:element name="circuit">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="node" maxOccurs="unbounded" />
      <xs:element ref="edge" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="numNodes" type="xs:integer" />
    <xs:attribute name="numEdges" type="xs:integer" />
    <xs:attribute name="id" type="xs:string" />
  </xs:complexType>
</xs:element>
<circuit id="eq2" numNodes="3" numEdges="7">
  <node id="0" numInputs="2" numOutputs="2">
    <table>1,0,0,1</table>
  </node>
  <node id="1" numInputs="2" numOutputs="2">
    <table>1,0,0,1</table>
  </node>
  <node id="2" numInputs="2" numOutputs="1">
    <table>0,0,0,1</table>
  </node>
  <edge src="-1" dst="0" srcIndex="0" dstIndex="0" />
  <edge src="-1" dst="0" srcIndex="1" dstIndex="1" />
  <edge src="-1" dst="1" srcIndex="2" dstIndex="0" />
  <edge src="-1" dst="1" srcIndex="3" dstIndex="1" />
  <edge src="0" dst="2" srcIndex="0" dstIndex="0" />
  <edge src="1" dst="2" srcIndex="0" dstIndex="1" />
  <edge src="2" dst="-1" srcIndex="0" dstIndex="0" />
</circuit>
Circuit evaluation

Evaluation of a circuit
XML circuit

Now we are able:
- to pass circuit between parties
- to evaluate for given input on a wires

It's time to construct a circuit.
Boolean circuit example

\[ f((X_1, X_2, X_3), (Y_1, Y_2, Y_3)) = (X == Y, X >= Y) \]
Using propositional formulae to construct a Boolean circuit

- We can construct a DAG step by step
  - $E_2 = X_1 \leftrightarrow Y_2$
  - $E_3 = X_1 \leftrightarrow Y_3$
  - $G_1 = \neg E_1 \land (X_1 \rightarrow Y_1)$
  - $G_2 = \neg E_2 \land (X_2 \rightarrow Y_2)$
  - $G_3 = \neg E_3 \land (X_3 \rightarrow Y_3)$
  - $Z_1 = E_1 \land E_2 \land E_3$
  - $Z_2 = G_1 \lor E_1 \land G_2 \lor E_1 \land E_2 \land G_3$

- Every formula has a tree structure
- Thus it is easy to combine them into a DAG
What about secrecy?

- We need to modify the contents of the circuit such, that it would allow **secure function evaluation**.
  - We use method called „**garbling**“ to hide the actual contents of the circuit
Hiding the contents of a gate

- Let's look at *truth tables* of implication and equivalence gates

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- It is trivial to determine the function of a gate just by seeing it's function table.
- Let's flip the output values of gates and permute the order of cells in *truth tables*
Hiding the contents of a gate

- To do that, let's toss a fair coin for every wire in the circuit. Depending on the result, let's apply a rewrite rule to that wire.
  - If a wire has a rewrite rule, flip the value it is transmitting

- To make use of it, we decode these rewriting rules into the truth tables of the gates
Hiding the contents of a gate

- First modify the gates that receive input from wires with rewrite rules

<table>
<thead>
<tr>
<th></th>
<th>$e_1$</th>
<th>$e_2$</th>
<th>$e_3$</th>
<th>$e_4$</th>
<th>$X_1 \leftrightarrow Y_1$</th>
<th>$X_1 \rightarrow Y_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$e_1$</th>
<th>$e_2$</th>
<th>$e_3$</th>
<th>$e_4$</th>
<th>$X_1 \leftrightarrow Y_1$</th>
<th>$X_1 \rightarrow Y_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Hiding the contents of a gate

- Then modify the gates having these wires as output gates
  - We may need to **extend** the truth tables
Hiding the contents of a gate

- \( f((X_1,X_2,X_3), (Y_1,Y_2,Y_3)) = (X==Y, X >= Y) \)
Hiding the contents of a gate

- As a result we have modified the original Boolean circuit such that it is not trivial to determine the function of the gates
  - We permuted the ordering of cells in the truth tables
  - We flipped some output values of some gates
- Note that also input wires and output wires of a gate may have rewrite rules
  - A circuit with inputs X and Y, where Y has a rewrite rule, must instead giving \((X = 0, Y = 0)\) be given \((X = 0, Y = 1)\) instead.
    - if an output wire with a rewrite rule outputs value 0, it should be interpreted as 1.
- Still, this is not enough
Garbling the circuit

- Every gate in a Boolean circuit has only fixed output value per outgoing wire given some input values.
- So when evaluating, the evaluator actually needs to see only contents of one cell per gate.
- We can use symmetric encryption to hide the contents of all other irrelevant cells.
Garbling the circuit

- We generate two symmetric encryption keys for every wire in the circuit
  - One key for value 0 and another for value 1
- We accompany these keys with the original values of the gate
Garbling the circuit

- New we encrypt every cell of the truth table with corresponding keys from input gates
Garbling the circuit

- New we encrypt every cell of the truth table with corresponding keys from input gates.
Garbling the circuit

- New we encrypt every cell of the truth table with corresponding keys from input gates.
Garbling the circuit

- New we encrypt every cell of the truth table with corresponding keys from input gates.
Garbling the circuit

- \( f((X1,X2,X3), (Y1,Y2,Y3)) = (X==Y, X >= Y) \)
Garbling the circuit

- Thus, for every gate there is only one combination of input (bit, key) pairs available and the evaluator can only open the relevant cell with these inputs.
- Having done this for all gates in the circuit, we have:
  - Hidden the functionality of the circuit
  - Secured it even more by allowing the evaluator to see only one relevant cell per gate
- If we generate such a circuit every time again, we need to compute a function $f$, we achieve security.
Input specification

Each party needs to transfer own input value to the evaluator:

- confidentially
- which means that nobody will understand the value.

Oblivious Transfer Protocol will help us do that.
Garbled circuit evaluation

- each wire has now value:
  - ${\{0,1\}, \text{key}}$
  - bit, key
- obtain table line from these bits

$\text{enc(bit)}_{\text{key}}$ is a AES stretch.

- $[\text{enc(bit)}_{\text{key}}, \text{key}]_1 \text{ XOR } [\text{enc(bit)}_{\text{key}}, \text{key}]_2 \text{ XOR } ... \text{ XOR } [\text{bit}, \text{key}]_n \text{ XOR table_line}.$
- split result into $numOutputs$ values $[\text{bit}, \text{key}]$
Garbled evaluation continued

- as an final output we have now value [key, bit]
  - but it could mean any int value

- Only the constructor of circuit is able to interpret the output
  - by applying the knowledge of garbling procedure
Practical results

- It works!
- 8bit and 32bit greater equal comparison implemented
- the system is able to compare two 32 bit integers represented as 2 bitstrings: $\geq$

- Client is happy and everything!
Performance

32 bit >= circuit
  • creation - 66 ms
  • evaluation - 48 ms

8 bit >= circuit
  • creation - 37 ms
  • evaluation - 32 ms

Future task: try processor with AES support
Any questions?