Secure Multiparty Computation

Muhammad Naveed
Please Interrupt
Millionaire's Problem
Millionaire's Problem
Millionaire's Problem
Millionaire's Problem

Let's see who is richer
OK, tell me your wealth.
Millionaire's Problem

Millionaire's Problem

NO, tell me your wealth.
Trusted Third Party

Let's use trusted third party
Trusted Third Party

Let's use trusted third party
trusted third party

Let's use trusted third party

$1 Billion

$66 Billion
Trusted Third Party

Let's use trusted third party
trusted third party

Let’s use trusted third party

Bill is Richer

Third Party

Bill is Richer

Bill is Richer
Trusted Third Party

Let's use trusted third party
Trusted Third Party

Let’s use trusted third party

Forbes
Let's use trusted third party
Secure Multiparty Computation

- Yao’s Garbled Circuits [Yao1982]

- solves Millionaire's Problem

- first secure multiparty computation scheme

- can compute any function securely

- doesn’t leak anything about inputs, other than what output leaks

- security only in honest but curious model

Guys, you don’t need third party.

Andy Yao
Secure Multiparty Computation

- Yao’s Garbled Circuits [Yao1982]
  - solves Millionaire's Problem
  - first secure multiparty computation scheme
  - can compute any function securely
  - doesn’t leak anything about inputs, other than what output leaks
  - security only in honest but curious model

Andy Yao
APPLICATIONS

- Auctions
- Electronic Voting
- Genomic Computation
Applications

- Auctions
- Electronic Voting
- Genomic Computation
- Space Security
Applications

- Auctions
- Electronic Voting
- Genomic Computation
- Space Security

Sharing information between satellites to avoid collision but not sharing trajectories
[http://sharemind.cyber.ee/]
Yao’s Garbled Circuits

First convert circuit into boolean circuit

Alice’s inputs

Bob’s inputs

Truth table:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Truth table:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Slide adapted from Vitaly Shmatikov Slides
YaO’s Protocol

- Consider a two input AND gate
  - same idea extends to larger circuits
- Alice have bit $b_A$ and Bob with bit $b_B$ wants to compute $b_A \text{ AND } b_B$
- Two parties:
  - Generator generates the circuit
  - Evaluator evaluate the circuit
- Any party can generate the circuit and the other party evaluates the circuit
Without loss of generality, suppose Alice generates the circuit

Alice will pick two random keys for all wires of the gate

Slide adapted from Vitaly Shmatikov Slides
Garbling the circuit

Alice encrypts each row of the truth table with encrypting the output wire key with the corresponding input wire keys.

Original truth table:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Encrypted truth table:

- $E_{k_0x}(E_{k_0y}(k_{0z}))$
- $E_{k_0x}(E_{k_1y}(k_{0z}))$
- $E_{k_1x}(E_{k_0y}(k_{0z}))$
- $E_{k_1x}(E_{k_1y}(k_{1z}))$

Slide adapted from Vitaly Shmatikov Slides
Send Garbled Circuit to Bob

- Alice randomly permute the garbled truth table
- And send it to Bob

Alice

\[ E_{k_0x}(E_{k_0y}(k_{0z})) \]
\[ E_{k_0x}(E_{k_1y}(k_{0z})) \]
\[ E_{k_1x}(E_{k_0y}(k_{0z})) \]
\[ E_{k_1x}(E_{k_1y}(k_{1z})) \]

Bob

\[ E_{k_1x}(E_{k_0y}(k_{0z})) \]
\[ E_{k_0x}(E_{k_1y}(k_{0z})) \]
\[ E_{k_1x}(E_{k_1y}(k_{1z})) \]
\[ E_{k_0x}(E_{k_0y}(k_{0z})) \]

Doesn’t know which row of garbled truth table corresponds to rows in original truth table

Garbled truth table:

Slide adapted from Vitaly Shmatikov Slides
Alice send its keys corresponding to its inputs to Bob

Garbled truth table:

\[
\begin{align*}
& E_{k_1x}(E_{k_0y}(k_{0z})) \\
& E_{k_0x}(E_{k_1y}(k_{0z})) \\
& E_{k_1x}(E_{k_1y}(k_{1z})) \\
& E_{k_0x}(E_{k_0y}(k_{0z}))
\end{align*}
\]

Bob will learn $K_{bx}$, but not the bit $b$. Why?

If Alice’s bit is 1, she simply sends $k_{1x}$ to Bob; if 0, she sends $k_{0x}$

Slide adapted from Vitaly Shmatikov Slides
**Bob Get His Keys Using OT**

- OT stands for oblivious transfer. Suppose,
  - 1st party has \( k_0 \) and \( k_1 \)
  - 2nd party input is a bit \( b = 0 \) or \( 1 \) and wants to learn \( k_b \)
  - Using OT, second party will learn \( k_b \), while first party will not learn \( b \)

\[
\begin{align*}
\text{Alice} & \quad \text{AND} \quad \text{Bob} \\
\text{Alice's input: } & k_0x, k_1x \\
\text{Bob's input: } & k_0z, k_1z \\
\text{Garbled truth table: } & E_{k_{1x}}(E_{k_{0y}}(k_0z)), E_{k_{0x}}(E_{k_{1y}}(k_0z)), E_{k_{1x}}(E_{k_{1y}}(k_1z)), E_{k_{0x}}(E_{k_{0y}}(k_0z)) \\
\text{Run oblivious transfer} & \\
\text{Alice's input: } & k_0y, k_1y \\
\text{Bob's input: } & \text{his bit } b \\
\text{Bob learns } & k_{by} \\
\text{What does Alice learn?} & 
\end{align*}
\]

Slide adapted from Vitaly Shmatikov Slides
Using the two keys, bob will be able to decrypt only one entry in the truth table and will get output wire key.

Bob does not learn if the output wire key corresponds to 0 or 1.

This is the only row Bob can decrypt. He learns $K_{0z}$.

---

Slide adapted from Vitaly Shmatikov Slides
In the same way, Bob evaluates the entire garbled circuit.

For each wire, Bob learns one key.

But Bob doesn’t know whether the key corresponds to 0 or 1.

i.e. Bob doesn’t know intermediate values.

Bob tells Alice the key for the final output.

She tells him whether it corresponds to 0 or 1.

Bob will not tell Alice the intermediate values.

Slide adapted from Vitaly Shmatikov Slides.
Yao’s garbled circuit was proposed as a theoretical construction

Real implementation is memory intensive

Many improvements to make it more efficient and scalable

Garbling XOR gates for free

Pipelining
Reading Paper


- Circuit Level Optimization
  - minimize bid-width
  - exploit free XOR garbling, convert as much gates to XOR as possible
    - MultiInput/MultiOutput gates

- Program Level
  - exploit local computation

Circuit Level Optimization

<table>
<thead>
<tr>
<th></th>
<th>Hamming Distance (900 bits)</th>
<th>Levenshtein Distance</th>
<th>AES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Online Time</td>
<td>Overall Time</td>
<td>Overall Time</td>
</tr>
<tr>
<td>Best Previous</td>
<td>0.310 s [26]</td>
<td>213 s [26]</td>
<td>92.4 s</td>
</tr>
<tr>
<td>Our Results</td>
<td>0.019 s</td>
<td>0.051 s</td>
<td>4.1 s</td>
</tr>
<tr>
<td>Speedup</td>
<td>16.3</td>
<td>4176</td>
<td>22.5</td>
</tr>
</tbody>
</table>

Table 1: Performance comparisons for several privacy-preserving applications.

† Inputs are 100-character strings over an 8-bit alphabet. The best previous protocol is the circuit-based protocol of [16].
‡ Inputs are 200-character strings over an 8-bit alphabet. The best previous protocol is the main protocol of [16].

MultiInput/MultiOutput gates

Program Level

exploit local computation
SMC guarantees that nothing will be leaked about the inputs, other than the leakage from output of computation.

E.g. Alice has 3 and Bob has 5 and they want to compute SUM(3, 5) = 8.

Alice’s learns Bob’s input and Bob’s learns Alice’s input.

It’s still perfectly secure SMC.
Conclusion

- Yao’s garbled circuits enable computation of any function without revealing inputs
- A constant round protocol
- Secure only against honest but curious adversaries
- State of the art SMC techniques are practically useful
- Other solutions for SMC