CS 5410 - Computer and Network Security: Secure Multiparty Computation

Professor Patrick Traynor
Fall 2017
Announcements

• No class next week.

• Posters (and all of the accompanying parts) are due by the beginning of class time on December 6th.

• Reminder: Course Evaluations - Please get this done!
Cloud computing

• Cloud services provide tons of new and convenient functionality

• Many services require storage and processing of sensitive data

• Can we trust the cloud?
Secure Computation

• Goal: to compute a shared result between untrusting parties.

• Example: Millionaire’s problem

• Simple solution: Trusted Third Party

• Can we get the same security based on cryptographic assumptions instead?

• What computations could you run in this setting?
What’s the point?

- Real-world applications:
  - Data mining/anonymizing
  - Electronic voting
  - Location-based mobile mapping
  - Private search
Garbled Circuits

• Developed by Andrew Yao in 1982
• First construction for SMC

• Basic concept:
  • Turn a function into a logical circuit
  • Turn logical gates into “garbled” truth tables
  • Based on the input wire “label”, decrypt a single output wire “label”

• Necessary interactivity: one party constructs the circuit while the other evaluates it
An example

Alice = 11
Bob = 10
Garbling

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</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>0</td>
</tr>
</tbody>
</table>

\[
A_0 = 01100101
\]

\[
A_1 = 11110001
\]

\[
B_0 = 11011100
\]

\[
B_1 = 10100101
\]

\[
A_0 \| B_0, C_1
\]

\[
A_0 \| B_1, C_0
\]

\[
A_1 \| B_0, C_0
\]

\[
A_1 \| B_1, C_1
\]

\[
C_0 = 11101000
\]

\[
C_1 = 00011001
\]
Garbling

\[
\begin{array}{c|c|c}
D & E & F \\
\hline
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c|c}
D & E & F & D & E & F & D & E & F \\
\hline
D_0 & E_0 & Enc(D_0||E_0, F_1) & D_0 & E_1 & Enc(D_0||E_1, F_0) & D_1 & E_0 & Enc(D_1||E_0, F_0) \\
D_1 & E_1 & Enc(D_1||E_1, F_1) & & & & & & \\
\end{array}
\]

D_0 = 00001111
D_1 = 01000001
E_0 = 11001100
E_1 = 11000001

F_0 = 10101011
F_1 = 00011111
Garbling

<table>
<thead>
<tr>
<th>C</th>
<th>F</th>
<th>G</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>

\[ C_0 = 11101000 \]
\[ C_1 = 00011001 \]
\[ F_0 = 10101011 \]
\[ F_1 = 00011111 \]

\[ G_0 = 00000001 \]
\[ G_1 = 01111111 \]
Evaluating
Evaluating

\[
\begin{align*}
11110001 & \quad 01000001 \\
10100101 & \quad 10100101 \\
Enc(A_0||B_0, C_1) & \quad Enc(D_0||E_0, F_1) \\
Enc(A_0||B_1, C_0) & \quad Enc(D_0||E_1, F_0) \\
Enc(A_1||B_0, C_0) & \quad Enc(D_1||E_0, F_0) \\
Enc(A_1||B_1, C_1) & \quad Enc(D_1||E_1, F_1) \\
000000001 = 0
\end{align*}
\]
Oblivious Transfer

- 1-out-of-2 OT:
  - One party offers two items
  - The other party chooses one
  - The first party doesn’t know which item was chosen, while the second party doesn’t know what the other item is

- Necessary for exchanging input wire labels
- Typically require expensive crypto
A simple OT example

• An RSA-based scheme by Even, Goldreich, and Lempel (See Wikipedia article on “Oblivious Transfer”)

• Steps for 1-out-of-2 OT:
  • Sender chooses two random messages
  • Chooser picks one and blinds it with a random encryption
  • Sender removes both random messages and decrypts to create two blinds for both selection messages
  • Chooser removes the only random blind he possesses
Security

TOUCHING WIRES CAUSES INSTANT DEATH

$200 FINE

Newcastle Tramway Authority
• An adversary should “learn nothing” from a secure protocol beyond the output of computation

• How do we quantify “learning”?

• Simulation security
Simulation proof for $f(x,y)$

Bad guy
Input private “a”

Result: $f(a,b)$

Good guy
Input private “b”
Simulation proof for $f(x,y)$

Bad guy
Input private “a”

Sim
Input ???
knows $f(a,b)$

Result: $f(a,b)$

Can Alice distinguish between the two?
The Fairplay Implementation

• Compiling the logical circuit:
  • SFDL => SHDL
  • Security requirements (e.g., loop unrolling)
  • Compiler optimizations

• Evaluating the circuit:
  • Encryption function: SHA-1 of key information XORed with plaintext
  • Diffie-Hellman based OT
Practical issues

• Size of circuits
  • On the order of GB
• OT constructions require very costly group operations
• Hundreds of circuit copies evaluated for malicious security
  • $2^{-k}$ where $k =$ number of circuits generated
Improving speed

- Pipelining
- Parallelization
- Smarter Circuit construction
  - Free XOR
  - Circuit templates
- Outsourcing
How far have we come?

- Millionaire’s problem [MNPS2004]:
  - 2.4 GHz core
  - 254 gates
  - 1.25 seconds
  - ~200 gates/sec

- Edit distance 4095 [sS13]:
  - 2.7 GHz core
  - 5.9B gates
  - 9042 seconds
  - ~650,000 gates/sec
Conclusion

- Secure Multiparty Computation allows for oblivious computation over encrypted data
- Garbled circuits are the most efficient (2-party) technique for SMC
- Still working towards a truly practical implementation
- What’s the “killer app”?