CSC 5930/9010 – Cloud S & P: Secure Multiparty Computation

Professor Henry Carter
Fall 2016
Recap

• Symmetric (shared) key systems
  ‣ Efficient (Many MB/sec throughput)
  ‣ Difficult key management

• Asymmetric (public) key systems
  ‣ Slow algorithms (so far …)
  ‣ Easy (easier) key management
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?

- Data mining/anonymizing
- Electronic voting
- Location-based mobile mapping
- Private search

Real-world applications:
## What’s the difference?

<table>
<thead>
<tr>
<th>FHE</th>
<th>Secret Sharing</th>
<th>Garbled Circuits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic circuits</td>
<td>Arithmetic circuits</td>
<td>Logical circuits</td>
</tr>
<tr>
<td>Non-interactive protocols</td>
<td>Interactive protocols</td>
<td>Interactive protocols</td>
</tr>
<tr>
<td>Expensive computation</td>
<td>Amortize costs for large groups</td>
<td>Optimal for two parties</td>
</tr>
</tbody>
</table>
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>1</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
A_0 &= 01100101 \\
A_1 &= 11110001 \\
B_0 &= 11011100 \\
B_1 &= 10100101
\end{align*}
\]

\[
\begin{align*}
A_0 \| B_0 &= Enc(A_0 \| B_0, C_1) \\
A_0 \| B_1 &= Enc(A_0 \| B_1, C_0) \\
A_1 \| B_0 &= Enc(A_1 \| B_0, C_0) \\
A_1 \| B_1 &= Enc(A_1 \| B_1, C_1)
\end{align*}
\]

\[
\begin{align*}
C_0 &= 11101000 \\
C_1 &= 00011001
\end{align*}
\]
Garbling

<table>
<thead>
<tr>
<th>D</th>
<th>E</th>
<th>F</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>1</td>
</tr>
</tbody>
</table>

$D_0 = 00001111$
$D_1 = 01000001$

$E_0 = 11001100$
$E_1 = 11000001$

$F_0 = 10101011$
$F_1 = 00011111$

$D_0 \| E_0 \| F_1$

$D_0 \| E_1 \| F_0$

$D_1 \| E_0 \| F_0$

$D_1 \| E_1 \| F_1$
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>

C₀ = 11101000
C₁ = 00011001
F₀ = 10101011
F₁ = 00011111

<table>
<thead>
<tr>
<th>C</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>C₀</td>
<td>F₀</td>
<td>Enc(C₀</td>
</tr>
<tr>
<td>C₀</td>
<td>F₁</td>
<td>Enc(C₀</td>
</tr>
<tr>
<td>C₁</td>
<td>F₀</td>
<td>Enc(C₁</td>
</tr>
<tr>
<td>C₁</td>
<td>F₁</td>
<td>Enc(C₁</td>
</tr>
</tbody>
</table>

G₀ = 00000001
G₁ = 01111111
Evaluating
Evaluating

\[
\begin{align*}
11110001 & \quad 10100101 \\
\overline{Enc(A_0\|B_0, C_1)} & \quad \overline{Enc(A_0\|B_1, C_0)} \\
\overline{Enc(A_1\|B_0, C_0)} & \quad \overline{Enc(A_1\|B_1, C_1)} \\
\overline{Enc(C_0\|F_0, G_0)} & \quad \overline{Enc(C_0\|F_1, G_0)} \\
\overline{Enc(C_1\|F_0, G_0)} & \quad \overline{Enc(C_1\|F_1, G_0)} \\
\overline{Enc(C_1\|F_1, G_1)} & \quad \overline{Enc(D_0\|E_0, F_1)} \\
\overline{Enc(D_0\|E_0, F_1)} & \quad \overline{Enc(D_0\|E_1, F_0)} \\
00000001 & = 0
\end{align*}
\]
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
Security

- 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
Secret Sharing

• First scheme: GMW 1987

• Basic concept:
  › Secret share all parties’ inputs
  › Perform (boolean or arithmetic) gate operations on shares
  › Combine shares to recover results

• Retains interactivity requirement

• Security again based on simulator proofs
Secret Sharing

- Based on the concept of a one-time pad
- Allows a value to be divided into $n$ random values called “shares”
- Given some number of shares, reconstruct the original value
- Example: XOR secret shares
Secret Shares: XOR
Secret Shares: AND triples

- Generate random shares with the property:
  \[(c_1 \oplus c_2) = (a_1 \oplus a_2) \land (b_1 \oplus b_2)\]

- Consume a single share for each AND gate

- Allows for fast protocol computation at the cost of precomputation
Secret Shares: AND triples

- \( d_i = x_i \oplus a_i, e_i = y_i \oplus b_i \)

- \( d = d_1 \oplus d_2, e = e_1 \oplus e_2 \)

- \( z_1 = (d \land e) \oplus (b_1 \land d) \oplus (a_1 \land e) \oplus c_1 \)

- \( z_2 = (b_2 \land d) \oplus (a_2 \land e) \oplus c_2 \)
Pros and Cons

• For large groups, the ability to do some computation locally saves on communication

• Requires more rounds of communication than garbled circuits
  ‣ Depends on the number of AND gates

• Can be combined with other techniques (use OT-based AND or homomorphic triple generation)
Homomorphic Encryption

• First fully homomorphic scheme: Gentry 2009
  ‣ Idea dates back to RSA

• Unique from previous techniques in that no interaction is needed
  ‣ Security proofs look very different

• Major performance challenges
  ‣ Costly in terms of time and space
Homomorphic Encryption

• An encryption scheme where operations on two ciphertexts have a predictable effect on the underlying messages

• Allow both boolean or arithmetic operations

• Example: RSA

• A FULLY homomorphic encryption scheme permits all functions
  ‣ i.e., allows an arbitrary number of AND and XOR operations
Example Scheme

• Keygen: choose a random value $p$

• Encrypt: choose a random value $q$ and a random $m' = m \mod 2$ and compute $c = m' + pq$

• Decrypt: output $m = (c \mod p) \mod 2$, where $c \mod p$ is the distance from $c$ to the nearest multiple of $p$

• XOR: $c + c$

• AND: $c \times c$
Problem: Noise

- As more multiplications and additions are performed, the number of bits in the noise $m'$ grows.
- If it grows too much, it will no longer round to the correct nearest multiple of $p$.
- This scheme is called somewhat homomorphic encryption.
Bootstrapping

- Decrypting the message strips away the noise buildup
- Double encrypt then remove inner encryption homomorphically
- SHE scheme must be able to perform decryption in addition to some operations
Problems

• Notice ciphertext expansion

• Notice mathematical operations cost much more than symmetric-key operations

• Hardness assumptions are relatively new
  ‣ Learning with errors and shortest vector problem are not as well understood as DL or prime factorization
Comparisons

• Garbled circuits
  ‣ Pro: Fastest (for two parties)
  ‣ Con: Doesn’t scale for more participants

• Secret sharing
  ‣ Pro: Fast for many parties
  ‣ Con: High number of communication rounds

• Fully homomorphic encryption
  ‣ Pro: Non-interactive
  ‣ Con: Very inefficient
Recap

• 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”? 
Next Time...

• Networking Basics
  ‣ Remember, you need to read it BEFORE you come to class!

• Homework:
  ‣ Project ideas (1 week)