Multi-Party Computation
Part 1

Claudio Orlandi, Aarhus University
Plan for the next 3 hours...

- **Part 1: Secure Computation with a Trusted Dealer**
  - Warmup: One-Time Truth Tables
  - Evaluating Circuits with Beaver’s trick
  - MAC-then-Compute for Active Security

- **Part 2: Oblivious Transfer**
  - OT: Definitions and Applications
  - Passive Secure OT Extension
  - OT Protocols from DDH (Naor-Pinkas/PVW)

- **Part 3: Garbled Circuits**
  - GC: Definitions and Applications
  - Garbling gate-by-gate: Basic and optimizations
  - Active security 101: simple-cut-and choose, dual-execution
Want more?

• Cryptographic Computing – Foundations
  – http://orlandi.dk/crycom
  – Programming & Theory Exercises
  – Will be happy to answer questions by mail!

...also the reason why I cannot stay here longer 😞

• These slides (+ references & pointers)
  – http://orlandi.dk/ecrypt
Secure Computation

- Privacy
- Correctness
- Input independence
- ...

\( f(x, y) \)

\( x \rightarrow 8dx2r ru3d0fW2TS \)
\( \text{muv6tbWg32flqlo} \)
\( \text{s1e4xq13O}TzoJc \)
What kind of Secure Computation?

• **Dishonest majority**
  – The adversary can corrupt up to n-1 participants (n=2).

• **Static Corruptions**
  – The adversary chooses which party is corrupted before the protocol starts.

• **Passive & Active Corruptions**
  – Adversary follows the protocol vs. (aka semi-honest, honest-but-curious)
  – Adversary can behave arbitrarily (aka malicious, byzantine)

• **No guarantees of fairness or termination**
  – Security with abort
(r_A, r_B) \leftarrow D

f(x, y)
Online Phase

\[ r_A \]

\[ r_B \]

\[ f(x,y) \]

\[ x \]

\[ y \]

- Independent of \( x,y \)
- Typically only depends on size of \( f \)
- Uses public key crypto technology *(slower)*

- Uses only information theoretic tools *(order of magn. faster)*
Part 1: Secure Computation with a Trusted Dealer

- **Warmup**: One-Time Truth Tables
- Evaluating Circuits with Beaver’s trick
- MAC-then-Compute for Active Security
“The simplest 2PC protocol ever”

\[(r_A, r_B) \leftarrow D\]
“The simplest 2PC protocol ever” OTTT (Preprocessing phase)

1) Write the truth table of the function $F$ you want to compute

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
“The simplest 2PC protocol ever” OTTT
(Preprocessing phase)

2) Pick random \((r, s)\), rotate rows and columns

\[
\begin{array}{c|cccc}
& 0 & 1 & 2 & 3 \\
\hline
0 & 1 & 4 & 4 & 1 \\
1 & 2 & 2 & 2 & 3 \\
2 & 0 & 0 & 4 & 3 \\
3 & 0 & 0 & 4 & 1 \\
\end{array}
\]
“The simplest 2PC protocol ever” OTTT (Preprocessing phase)

3) Secret share the truth table i.e.,

Pick at random, and let

\[
\begin{array}{cccc}
T1 & 1 & 4 & 4 & 1 \\
T2 & 2 & 2 & 2 & 3 \\
& 0 & 0 & 4 & 3 \\
& 0 & 0 & 4 & 1 \\
\end{array}
\]
"The simplest 2PC protocol ever"

$u = x + r$

$v = y + s$

output $f(x,y) = T1[u,v] + T2[u,v]$

"Privacy": inputs masked w/uniform random values

Correctness: by construction
What about active security?

\[ u = x + r \]

\[ v = y + s + e_1 \]

\[ T_2[u,v] + e_2 \]
Is this cheating?

- \( v = y + s + e_1 = (y + e_1) + s = y' + s \)
  - Input substitution, not cheating according to the definition!

- \( M_2[u, v] + e_2 \)
  - Changes output to \( z' = f(x, y) + e_2 \)
  - Example: \( f(x, y) = 1 \) iff \( x = y \) (e.g. pwd check)
  - \( e_2 = 1 \) the output is 1 whp (login without pwd!)
    - Clearly breach of security!
Force Bob to send the right value

- **Problem**: Bob can send the wrong shares
- **Solution**: use MACs
  - e.g. $m = ax + b$ with $(a, b) \leftarrow F$

Abort if $m' \neq ax' + b$
output \( f(x,y) = T1[u,v] + T2[u,v] \)
else
abort

Statistical security vs. malicious Bob w.p. \( 1 - 1/|F| \)
“The simplest 2PC protocol ever” OTTT

- Optimal communication complexity 😊
- Storage exponential in input size 😞

👉 Represent function using circuit instead of truth table!
Part 1: Secure Computation with a Trusted Dealer

- Warmup: One-Time Truth Tables
- Evaluating Circuits with Beaver’s trick
- MAC-then-Compute for Active Security
Circuit based computation
Invariant

• For each \textit{wire} x in the circuit we have
  
  – [x] := (x_A, x_B) \quad \text{// read “x in a box”}
  
  – Where Alice holds x_A
  
  – Bob holds x_B
  
  – Such that x_A+x_B=x

• Notation overload:
  
  – x is both the r-value and the l-value of x
  
  – use n(x) for name of x and v(x) for value of x when in doubt.
  
  – Then [n(x)] = (x_A, x_B) such that x_A+x_B=v(x)
Circuit Evaluation
(Online phase)

1) \( [x] \leftarrow \text{Input}(A,x) : \)
   - chooses random \( x_B \) and send it to Bob
   - set \( x_A = x + x_B \) \hspace{1cm} // symmetric for Bob

   Alice only sends a random bit! “Clearly” secure

2) \( z \leftarrow \text{Open}(A,[z]) : \)
   - Bob sends \( z_B \)
   - Alice outputs \( z = z_A + z_B \) \hspace{1cm} // symmetric for Bob

   Alice should learn \( z \) anyway! “Clearly” secure
Circuit Evaluation (Online phase)

2) \([z] \leftarrow \text{Add}([x],[y])\) \quad \text{// at the end } z = x + y

- Alice computes \(z_A = x_A + y_A\)
- Bob computes \(z_B = x_B + y_B\)

- We write \([z] = [x] + [y]\)

No interaction! “Clearly” secure

“for free” : only a local addition!
Circuit Evaluation
(Online phase)

2a) \([z] \leftarrow \text{Mul}(a, [x])\)  
   \[
   \begin{align*}
   - \text{Alice computes } z_A &= a \times x_A \\
   - \text{Bob computes } z_B &= a \times x_B 
   \end{align*}
   \]
   // at the end \(z = a \times x\)

2c) \([z] \leftarrow \text{Add}(a, [x])\)  
   \[
   \begin{align*}
   - \text{Alice computes } z_A &= a + x_A \\
   - \text{Bob computes } z_B &= x_B 
   \end{align*}
   \]
   // at the end \(z = a + x\)
Circuit Evaluation
(Online phase)

3) Multiplication?

How to compute \([z]=[xy]\) ?

Alice, Bob should compute

\[ z_A + z_B = (x_A + x_B)(y_A + y_B) \]

\[ = x_A y_A + x_B y_A + x_A y_B + x_B y_B \]

Alice can compute this

Bob can compute this

How do we compute this?
Circuit Evaluation
(Online phase)

3) \([z] \leftarrow \text{Mul}([x],[y])\):

1. Get \([a],[b],[c]\) with \(c=ab\) from trusted dealer

2. \(e=\text{Open}([a]+[x])\)

3. \(d=\text{Open}([b]+[y])\)

4. Compute \([z] = [c] + e[y] + d[x] - ed\)
   
   \[ab + (ay+xy) + (bx+xy) - (ab+ay+bx+xy)\]
Part 1: Secure Computation with a Trusted Dealer

- Warmup: One-Time Truth Tables
- Evaluating Circuits with Beaver’s trick
- MAC-then-Compute for Active Security
Secure Computation

\[
\begin{align*}
&\text{z}^* \\
&\text{[w] [w+e]} \\
&\text{[w]} \\
&\text{[w]} \\
&\text{[y_5]} \\
&\text{[x_5]} \\
&\text{[y_4]} \\
&\text{[x_4]} \\
&\text{[y_3]} \\
&\text{[x_3]} \\
&\text{[y_2]} \\
&\text{[x_2]} \\
&\text{[y_1]} \\
&\text{[x_1]} \\
\end{align*}
\]
Active Security?

- “Privacy?”
  - even a malicious Bob does not learn anything 😊

- “Correctness?”
  - a corrupted Bob can change his share during any “Open” (both final result or during multiplication) leading the final output to be incorrect 😞
Problem

2) $z \leftarrow \text{Open}(A,[z])$:

- Bob sends $z_B + e$
- Alice outputs $z = z_A + z_B + e$  
  \hspace{1cm}  // error change output distribution in way that cannot be simulated by input substitution
Solution: add MACs

2) $z \leftarrow \text{Open}(A,[z])$:

- Bob sends $z_B, m_B$
- Alice outputs
  
  - $z=z_A+z_B$ if $m_B = z_B \Delta_A + k_A$
  
  - “abort” otherwise

**Solution:** Enhance representation $[x]$

- $[x] = ( (x_A, k_A, m_A), (x_B, k_B, m_B) )$ s.t.
- $m_B = x_B \Delta_A + k_A$ (symmetric for $m_A$)
- $\Delta_A, \Delta_B$ is the same for all wires.
Linear representation

• Given
  – \([x] = ( (x_A, k_{Ax}, m_{Ax}) , (y_B, k_{Bx}, m_{Bx}) )\)
  – \([y] = ( (y_A, k_{Ay}, m_{Ay}) , (y_B, k_{By}, m_{By}) )\)
  – Compute \([z] = (\)
    \((z_A=x_A+y_A , \quad k_{Az}=k_{Ax}+k_{Ay} , \quad m_{Az}=m_{Ax}+m_{Ay} )\),
    \((z_B=x_B+y_B , \quad k_{Bz}=k_{Bx}+k_{By} , \quad m_{Bz}=m_{Bx}+m_{By} )\),\)

• And \([z]\) is in the right format since...
  \[ m_{Bz} = (m_{Bz}+m_{By}) = (k_{Ax} + x_B\Delta_A ) + (k_{Ay} + y_B\Delta_A ) \]
  \[ = (k_{Ax} + k_{Ay} ) + (x_B+y_B)\Delta_A = k_{Az} + z_B\Delta_A \]
Recap

1. Output Gates:
   – Exchange shares and MACs
   – Abort if MAC does not verify

2. Input Gates:
   – Get a random \([r]\) from trusted dealer
   – \(r \leftarrow \text{Open}(A,[r])\)
   – Alice sends Bob \(d=x-r\),
   – Compute \([x]=[r]+d\)

Allows simulator to extract \(x^* = r+d^*\)
1. Addition Gates:
   - Use linearity of representation to compute
     \[ z = x + y \]

2. Multiplication gates:
   - Get a random triple \([a][b][c]\) with \(c = ab\) from
   - \(e \leftarrow \text{Open}([a]+[x]), \ d \leftarrow \text{Open}([b]+[y])\)
   - Compute \([z] = [c] + a[y] + b[x] - ed\)
Final remarks

- Size of MACs
- Lazy MAC checks
Size of MACs

1. Each party must store a mac/key pair for each other party
   – quadratic complexity! 😞
   – SPDZ for linear complexity.

2. MAC is only as hard as guessing key!
   \( k \) MACs in parallel give security \( 1/|F|^k \)
   – In TinyOT \( F=\mathbb{Z}_2 \), then MACs/Keys are \( k \)-bit strings
   – MiniMACs for constant overhead
Lazy MAC Check

\[ z^* \]

\[ +e \]
Lazy MAC Check

1) $z \leftarrow \text{PartialOpen}(A,[z])$:
   1. Bob sends $z_B$
   2. Bob runs $\text{OutMAC.append}(m_B)$
   3. Alice runs $\text{InMAC.append}(k_A + z_B \Delta_A)$
   4. Alice outputs $z = z_A + z_B$

2) $z \leftarrow \text{FinalOpen}(A,[z])$:
   1. Steps 1-3 as before
   2. Bob sends $u = H(\text{OutMAC})$ to Alice
   3. Alice outputs $z = z_A + z_B$ if $u = H(\text{InMAC})$
   4. “abort” otherwise
Recap of Part 1

• Two protocols “in the trusted dealer model”
  – One Time-Truth Table
    • Storage $\exp(\text{input size})$
    • Communication $O(\text{input size})$
    • 1 round
  – (SPDZ)/BeDOZa/TinyOT online phase
    • Storage linear $\#\text{number of AND gates}$
    • Communication linear $\#\text{number of AND gates}$
    • $\#\text{rounds} = \text{depth of the circuit}$
  – ...and add enough MACs to get active security
Recap of Part 1

• To do secure computation is enough to precompute enough random multiplications!

• If no semi-trusted party is available, we can use cryptographic assumption (next)