Advanced Tools from Modern Cryptography

Lecture 6
Secure Multi-Party Computation without Honest Majority: “GMW” Protocol
MPC without Honest-Majority

- Plan (Still sticking with passive corruption):
  - Two protocols, that are secure computationally
    - The “passive-GMW” protocol for any number of parties
    - A 2-party protocol using Yao’s Garbled Circuits
    - Both rely on a computational primitive called Oblivious Transfer
  - Today: OT and Passive-GMW
Oblivious Transfer

Pick one out of two, without revealing which.

Intuitive property: transfer partial information "obliviously"

\[ x_0, x_1 \rightarrow b \rightarrow x_b \]

If we had a trusted third party
Is OT Possible?

- No information theoretically secure 2-party protocol for OT
- Because OT can be used to carry out information-theoretically secure 2-party AND (coming up)
- Computationally secure OT protocols exist under various computational hardness assumptions
- Will define computational security of MPC later, comparing the protocol to the ideal functionality
An OT Protocol (against passive corruption)

Using *(a special)* public-key encryption
- In which one can sample a public-key without knowing secret-key
- $c_{1-b}$ inscrutable to a passive corrupt receiver
- Sender learns nothing about $b$

$$
\begin{align*}
(SK_b, PK_b) &\leftarrow \text{KeyGen} \\
\text{Sample } PK_{1-b} \\
\end{align*}
$$

$$
\begin{align*}
&x_0, x_1 \\
&b \\
&PK_0, PK_1 \\
&c_0, c_1 \\
&x_b = \text{Dec}(c_b; SK_b) \\
\end{align*}
$$

$$
\begin{align*}
&x_0, x_1 \\
&b \\
&c_0 = \text{Enc}(x_0; PK_0) \\
&c_1 = \text{Enc}(x_1; PK_1) \\
&x_b = \text{Dec}(c_b; SK_b) \\
\end{align*}
$$
Why is OT Useful?

Naïve 2PC from OT

- Say Alice’s input $x$, Bob’s input $y$, and only Bob should learn $f(x,y)$
- Alice (who knows $x$, but not $y$) prepares a table for $f(x,\cdot)$ with $D = 2^{|y|}$ entries (one for each $y$)
- Bob uses $y$ to decide which entry in the table to pick up using 1-out-of-$D$ OT (without learning the other entries)

Bob learns only $f(x,y)$ (in addition to $y$). Alice learns nothing beyond $x$.

OT captures the essence of MPC:

Secure computation of any function $f$ can be **reduced** to OT

Problem: $D$ is exponentially large in $|y|$

Plan: somehow exploit efficient computation (e.g., circuit) of $f$
Passive secure MPC based on OT, without any other computational assumptions

Will assume that a trusted party is available to carry out OT between any pair of parties (replaced by a cryptographic protocol, later)

Tolerates any number of corrupt parties

Idea: Computing on additively secret-shared values

For a variable (wire value) s, will write \([s]_i\) to denote its share held by the \(i^{th}\) party

Computing on Shares: 2 Parties

Let gates be $+ \& \times$ (XOR & AND for Boolean circuits)

Plan: Similar to BGW: shares of each wire value will be computed, with Alice holding one share and Bob the other. At the end, Alice sends her share of output wire to Bob.

$w = u + v$ : Each one locally computes $[w]_i = [u]_i + [v]_i$
What about $w = u \times v$?

$[w]_1 + [w]_2 = ( [u]_1 + [u]_2 ) \times ( [v]_1 + [v]_2 )$

Alice picks $[w]_1$ and lets Bob compute $[w]_2$ using the naive (proof-of-concept) protocol.

Note: Bob’s input is $([u]_2, [v]_2)$. Over the binary field, this requires a single 1-out-of-4 OT.
Passive GMW

Secure?

View of Alice:
- Input $x$ and random values it picks throughout the protocol

View of Bob:
- Input $y$ and random values it picks throughout the protocol
- A random value (picked via OT) for each wire out of a $\times$ gate
- $f(x,y)$ - own share, for the output wire
- This distribution is the same for $x, x'$ if $f(x,y)=f(x',y)$

Exercise: What goes wrong in the above claim if Alice reuses $[w]_1$ for two $\times$ gates?
m-way sharing: \( s = [s]_1 + \ldots + [s]_m \)

Addition, local as before

Multiplication: For \( w = u \times v \)
\[
[w]_1 + \ldots + [w]_m = ([u]_1 + \ldots + [u]_m) \times ([v]_1 + \ldots + [v]_m)
\]

Party i computes \([u]_i[v]_i\)

For every pair \((i,j), i \neq j\), Party i picks random \( a_{ij} \) and lets Party j securely compute \( b_{ij} \) s.t. \( a_{ij} + b_{ij} = [u]_i[v]_j \) using the naive protocol (a single 1-out-of-2 OT)

Party i sets \([w]_i = [u]_i[v]_i + \sum_j (a_{ij} + b_{ji})\)
Computing on Shares: m Parties
Arithmetic Version

- m-way sharing: \( s = [s]_1 + ... + [s]_m \)
- Addition, local as before
- Multiplication: For \( w = u \times v \)
  
  \[
  [w]_1 + ... + [w]_m = ( [u]_1 + ... + [u]_m ) \times ( [v]_1 + ... + [v]_m )
  \]

- Party i computes \([u]_i [v]_i\)
- For every pair \((i,j), i \neq j\), Party i picks random \(a_{ij}\) and lets
  Party j securely compute \(b_{ij}\) s.t. \(a_{ij} + b_{ij} = [u]_i [v]_j\) using
  Oblivious Linear-function Evaluation (OLE)
- Party i sets \([w]_i = [u]_i [v]_i + \sum_j (a_{ij} + b_{ji})\)
MPC for Passive Corruption

Story so far:
- For honest-majority: Information-theoretically secure protocol, using Shamir secret-sharing [BGW]
- Without honest-majority: Using Oblivious Transfer (OT), using additive secret-sharing [GMW]

Up next
- A 2-party protocol (so no honest-majority) using Oblivious Transfer and Yao’s Garbled Circuits
  - Uses additional computational primitives and is limited to arithmetic circuits over small fields (e.g., boolean circuits)
  - Needs just one round of interaction