CPSC 467: Cryptography and Computer Security

Michael J. Fischer

Preview Lecture 24
December 4, 2017
The Millionaires’ Problem

Appendix: Privacy-Preserving Boolean Function Evaluation

Boolean circuits
Implementation Using Value Shares
Implementation Using Garbled Circuits
The Millionaires’ Problem
The Millionaires’ Problem

The Millionaires’ problem, introduced by Andy Yao in 1982, began the study of *privacy-preserving multiparty computation*. Alice and Bob want to know who is the richer without revealing how much they are actually worth.

Alice is worth $I$ million dollars; Bob is worth $J$ million dollars.

They want to determine whether or not $I \geq J$, but at the end of the protocol, neither should have learned any more about the other person’s wealth than is implied by the truth value of the predicate $I \geq J$. 
Privacy-preserving multiparty computation

Another example is vote-counting.

Each voter has an input $v_i \in \{0, 1\}$ indicating their no/yes vote on an issue.

The goal is to collectively compute $\sum v_i$ while maintaining the privacy of the individual $v_i$. 
Appendix: Privacy-Preserving Boolean Function Evaluation
Boolean functions computed by circuits

Let $\bar{z} = f(\bar{x}, \bar{y})$, where $\bar{x}$, $\bar{y}$, and $\bar{z}$ are bit strings of lengths $n_x$, $n_y$, and $n_z$, respectively, and $f$ is a Boolean function computed by a polynomial size Boolean circuit $C_f$ with $n_x + n_y$ input wires and $n_z$ output wires.

In a *private evaluation* of $C_f$, Alice furnishes the (private) input data to the first $n_x$ input wires, and Bob furnishes the (private) input data for the remaining $n_y$ input wires. The $n_z$ output wires should contain the result $\bar{z} = f(\bar{x}, \bar{y})$. The corresponding functionality is

$$F(\bar{x}, \bar{y}) = (\bar{z}, \bar{z}).$$

Alice and Bob should learn nothing about each others inputs or the intermediate values of the circuit, other than what is implied by their own inputs and the output values $\bar{z}$. 
Circuit evaluation

An evaluation of a circuit assigns a Boolean value $\sigma_w$ to each wire of the circuit. The input wires are assigned the corresponding input values.

Let $G$ be a gate with input wires $u$ and $v$ and output wire $w$ that computes the Boolean function $g(x, y)$. In a correct assignment, $\sigma_w = g(\sigma_u, \sigma_v)$.

A complete evaluation of the circuit first assigns values to the input wires and then works its way down the circuit, assigning a value to the output wire of any gate whose inputs have already received values.
A Boolean circuit

Alice

Bob

\[ \sigma_1 \]

\[ \sigma_2 \]

\[ \sigma_3 \]

\[ \sigma_4 \]

\[ \sigma_5 \]

\[ \sigma_6 \]

\[ \sigma_7 \]
Private circuit evaluation

In a *private circuit evaluation*,

- Both Alice and Bob learn the output values of the circuit;
- Neither Alice nor Bob gain any information about each others private input values except for whatever is implied by their own input values and the circuit output.

We present two different schemes for privately evaluating a circuit:

- Value shares;
- Garbled circuits.
Implementation Using Value Shares
Value shares

In a private evaluation using value shares, we split each value $\sigma_w$ into two random shares $a_w$ and $b_w$ such that $\sigma_w = a_w \oplus b_w$.

- Alice knows $a_w$; Bob knows $b_w$.
- Neither share alone gives any information about $\sigma_w$, but together they allow $\sigma_w$ to be computed.

After all shares have been computed for all wires, Alice and Bob exchange their shares $a_w$ and $b_w$ for each output wire $w$.

They are both then able to compute the circuit output.
Obtaining the shares

We now describe how Alice and Bob obtain their shares while maintaining the desired privacy.

There are three cases, depending on whether \( w \) is

1. An input wire controlled by Alice;
2. An input wire controlled by Bob;
3. The output wire of a gate \( G \).
Alices input wires

1. Input wire controlled by Alice:

Alice knows $\sigma_w$.

She generates a random share $a_w \in \{0, 1\}$ for herself and sends Bob his share $b_w = a_w \oplus \sigma_w$. 

Bobs input wires

2. Input wire controlled by Bob:

Bob knows $\sigma_w$.

Alice chooses a random share $a_w \in \{0, 1\}$ for herself.

She prepares a table $T$:

<table>
<thead>
<tr>
<th>$\sigma$</th>
<th>$T[\sigma]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$a_w$</td>
</tr>
<tr>
<td>1</td>
<td>$a_w \oplus 1$</td>
</tr>
</tbody>
</table>

Bob requests $T[\sigma_w]$ from Alice via $\text{OT}_1^2$ and takes his share to be $b_w = T[\sigma_w] = a_w \oplus \sigma_w$. 
Obtaining shares for gate output wires

3. Output wire of a gate $G$:

Let $G$ have input wires $u, v$ and compute function $g(x, y)$. Alice chooses random share $a_w \in \{0, 1\}$ for herself. She computes the table

\[
T[0, 0] = a_w \oplus g(a_u, a_v) \\
T[0, 1] = a_w \oplus g(a_u, a_v \oplus 1) \\
T[1, 0] = a_w \oplus g(a_u \oplus 1, a_v) \\
T[1, 1] = a_w \oplus g(a_u \oplus 1, a_v \oplus 1)
\]

(Equivalently, $T[r, s] = a_w \oplus g(a_u \oplus r, a_v \oplus s)$.)

Bob requests $T[b_u, b_v]$ from Alice via OT$_1^4$ and takes his share to be $b_w = T[b_u, b_v] = a_w \oplus g(\sigma_u, \sigma_v)$. 
Remarks

1. Alice and Bob’s shares for $w$ are both independent of $\sigma_w$.
   ▶ Alices share is chosen uniformly at random.
   ▶ Bob’s share is always the XOR of Alices random bit $a_w$ with something independent of $a_w$.
2. This protocol requires $n_y$ executions of $\text{OT}_1^2$ to distribute the shares for Bob’s inputs, and one $\text{OT}_1^4$ for each gate.\(^1\)
3. This protocol assumes semi-honest parties.
4. This protocol generalizes readily from 2 to $m$ parties.
5. Bob does not even need to know what function each gate $G$ computes. He only uses his private inputs or shares to request the right line of the table in each of the several $\text{OT}$ protocols.

\(^1\)Note: The $n_y$ executions of $\text{OT}_1^2$ can be eliminated by having Bob produce the shares for his input wires just as Alice does for hers. Our approach has the advantage of being more uniform since Alice is in charge of distributing the shares for all wires.
Implementation Using Garbled Circuits
Garbled circuits

A very different approach to private circuit evaluation is the use of *garbled circuits*.

The idea here is that Alice prepares a garbled circuit in which each wire has associated with it a tag corresponding to 0 and a tag corresponding to 1.

Associated with each gate is a template that allows the tag that represent the correct output value to be computed from the tags representing the input values.

This is all done in a way that keeps hidden the actual values that the tags represent.
A sketch of the protocol

After creating the circuit, Alice, who knows all of the tags, uses $OT^2_1$ to send Bob the tags corresponding to values on the input wires that he controls.

She also sends him the tags corresponding to the values on the input wires that she controls.

Bob then evaluates the circuit all by himself, computing the output tag for each gate from the tags on the input wires.
Finishing up

At the end, he knows the tags corresponding to the output wires. Alice knows which Boolean values those tags represent, which she sends to Bob (either before or after he has evaluated the circuit). In this way, Bob learns the output of the circuit, which he then sends to Alice.
Role of the tags

The scrambled gate is a 4-line table giving the output tag corresponding to each of the possible 4 input values.

Each line of the table is encrypted differently.

The input tags to the gate allow the corresponding table item to be decrypted.

Evaluating the circuit then amounts to decrypting ones way though the circuit, gate by gate, until getting the output tag.
Remarks

1. The $\text{OT}_1^2$ protocol steps used to distribute the tags for the wires that Bob controls keeps his inputs private from Alice. The privacy of Alice’s inputs and intermediate circuit values from Bob relies on the encryption function used to hide the association between tags and values.

2. The security of the protocol relies on properties of the encryption function that we have not stated.

3. This protocol requires only $n_y$ executions of $\text{OT}_1^2$ and hence should be considerably faster to implement than the share-based protocol.

4. This protocol also assumes semi-honest parties.

5. Doesn’t easily generalize to more than two parties.

6. Bob doesn’t need to know the function each gate computes. He only needs the associated templates.