CPSC 467: Cryptography and Computer Security

Michael J. Fischer

Lecture 22
November 30, 2015
Mutual Privacy-Preserving Protocols

Coin-Flipping

Locked Boxes
   Locked Box Paradigm
   Locked Box Implementation
   Oblivious Transfer

The Millionaires’ Problem

Appendix: Privacy-Preserving Boolean Function Evaluation
   Boolean circuits
   Implementation Using Value Shares
   Implementation Using Garbled Circuits
Mutual Privacy-Preserving Protocols
Privacy

We have looked at several protocols that are intended to keep Alice’s information secret, both from Bob and from a malicious adversary masquerading as Bob.

We now look at other protocols whose goal is to control the release of partial information about Alice’s secret. Just enough information should be released to carry out the purpose of the protocol but no more.

We’ll see several examples in the following sections.
Coin-Flipping
Flipping a common coin

Alice and Bob are in the process of getting a divorce and are trying to decide who gets custody of their pet cat, Fluffy.

They both want the cat, so they agree to decide by flipping a coin: heads Alice wins; tails Bob wins.

Bob has already moved out and does not wish to be in the same room with Alice.

The feeling is mutual, so Alice proposes that she flip the coin and telephone Bob with the result.

This proposal of course is not acceptable to Bob since he has no way of knowing whether Alice is telling the truth when she says that the coin landed heads.
Making it fair

“Look Alice,” he says, “to be fair, we both have to be involved in flipping the coin.”

“We’ll each flip a private coin and XOR our two coins together to determine who gets Fluffy.”

“You should be happy with this arrangement since even if you don’t trust me to flip fairly, your own fair coin is sufficient to ensure that the XOR is unbiased.”
A proposed protocol

This sounds reasonable to Alice, so she lets him propose the protocol below, where 1 means “heads” and 0 means “tails”.

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Choose random bit $b_A \in {0, 1}$</td>
<td>$b_A \rightarrow$</td>
</tr>
<tr>
<td>2. $b_B \leftarrow b_B \in {0, 1}$</td>
<td>Choose random bit $b_B \in {0, 1}$</td>
</tr>
<tr>
<td>3. Coin outcome is $b = b_A \oplus b_B$</td>
<td>Coin outcome is $b = b_A \oplus b_B$</td>
</tr>
</tbody>
</table>

Alice considers this for awhile, then objects.

“This isn’t fair. You get to see my coin before I see yours, so now you have complete control over the outcome.”
Alice’s counter proposal

She suggests that she would be happy if the first two steps were reversed, so that Bob flips his coin first, but Bob balks at that suggestion.

They then both remember last week’s lecture and decide to use blobs to prevent either party from controlling the outcome. They agree on the following protocol.
A mutually acceptable protocol

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Choose random ( k_A, s_A \in \mathcal{K}_A ).</td>
<td>( k_A, k_B )</td>
</tr>
<tr>
<td>2. Choose random bit ( b_A \in {0, 1} ).</td>
<td>Choose random bit ( b_B \in {0, 1} ).</td>
</tr>
<tr>
<td>( c_A = \text{enclose}(s_A, k_B, b_A) ).</td>
<td>( c_B = \text{enclose}(s_B, k_A, b_B) ).</td>
</tr>
<tr>
<td>3. Send ( s_A ).</td>
<td>Send ( s_B ).</td>
</tr>
<tr>
<td>4. ( b_B = \text{reveal}(s_B, k_A, c_B) ).</td>
<td>( b_A = \text{reveal}(s_A, k_B, c_A) ).</td>
</tr>
<tr>
<td>Coin outcome is ( b = b_A \oplus b_B ).</td>
<td>Coin outcome is ( b = b_A \oplus b_B ).</td>
</tr>
</tbody>
</table>

At the completion of step 2, both Alice and Bob have each others’ commitment (something they failed to achieve in the past, which is why they’re in the middle of a divorce now), but neither knows the other’s private bit.

They learn each other’s bit at the completion of steps 3 and 4.
Remaining asymmetry

While this protocol appears to be completely symmetric, it really isn’t quite, for one of the parties completes step 3 before the other one does.

Say Alice receives $s_B$ before sending $s_A$.

At that point, she can compute $b_B$ and hence know the coin outcome $b$.

If it turns out that she lost, she might decide to stop the protocol and refuse to complete her part of step 3.
Premature termination

What happens if one party quits in the middle or detects the other party cheating?

So far, we’ve only considered the possibility of undetected cheating.

But in any real situation, one party might feel that he or she stands to gain by cheating, *even if the cheating is detected.*
Responses to cheating

Detected cheating raises complicated questions as to what happens next.

- Does a third party Carol become involved?
- If so, can Bob prove to Carol that Alice cheated?
- What if Alice refuses to talk to Carol?

Think about Bob’s recourse in similar real-life situations and consider the reasons why such situations rarely arise.

For example, what happens if someone

- fails to follow the provisions of a contract?
- ignores a summons to appear in court?
A copycat attack

There is a subtle problem with the previous coin-flipping protocol.

Suppose Bob sends his message before Alice sends hers in each of steps 1, 2, and 3.

Then Alice can choose $k_A = k_B$, $c_A = c_B$, and $s_A = s_B$ rather than following her proper protocol, so

$$\text{reveal}(s_A, k_B, c_A) = \text{reveal}(s_B, k_A, c_B).$$

In step 4, Bob will compute $b_A = b_B$ and won’t detect that anything is wrong. The coin outcome is $b = b_A \oplus b_A = 0$.

Hence, Alice can force outcome 0 simply by playing copycat.
Preventing a copycat attack

This problem is not so easy to overcome.

One possibility is for both Alice and Bob to check that $k_A \neq k_B$ after step 1.

That way, if Alice, say, chooses $c_A = c_B = c$ and $s_A = s_B = s$ on steps 2 and 3, there still might be a good chance that

$$b_A = \text{reveal}(s, k_B, c) \neq \text{reveal}(s, k_A, c) = b_B.$$ 

However, depending on the bit commitment scheme, a difference in only one bit in $k_A$ and $k_B$ might not be enough to ensure that different bits are revealed.

In any case, it’s not enough that $b_A$ and $b_B$ sometimes differ. For the outcome to be unbiased, we need $\Pr[b_A \neq b_B] = 1/2$. 
A better idea

A better idea might be to both check that $k_A \neq k_B$ after step 1 and then to use $h(k_A)$ and $h(k_B)$ in place of $k_A$ and $k_B$, respectively, in the remainder of the protocol, where $h$ is a hash function.

That way, even a single bit difference in $k_A$ and $k_B$ is likely to be magnified to a large difference in the strings $h(k_A)$ and $h(k_B)$.

This should lead to the bits $\text{reveal}(s_A, h(k_B), c_A)$ and $\text{reveal}(s_B, h(k_A), c_B)$ being uncorrelated, even if $s_A = s_B$ and $c_A = c_B$. 
A possible problem

The general properties for \texttt{reveal} given in \texttt{lecture 21} do not seem strong enough to imply that \texttt{enclose}(k_A, k_B, b) has a strong enough dependence on \(k_B\) to prevent cheating by an Alice who knows a commitment \(c\) that she can open to reveal either 0 or 1.

For example, suppose \texttt{enclose}(k_A, k_B, b) does not depend on \(k_B\).

If Alice knows \(s_0, k_0, s_1, k_1\) such that

\[
\texttt{enclose}(s_0, k_0, 0) = \texttt{enclose}(s_1, k_1, 1) = c,
\]

it follows that

\[
\texttt{enclose}(s_0, k_B, 0) = \texttt{enclose}(s_1, k_B, 1) = c
\]

for any \(k_B\). Hence,

\[
\texttt{reveal}(s_0, k_B, c) = 0 \quad \text{and} \quad \texttt{reveal}(s_1, k_B, c) = 1.
\]

The bit-commitment implementations given \texttt{lecture 21} all do strongly depend on Bob’s random value \(k_B\).
Locked Boxes
Locked boxes

Protocols for coin flipping and for dealing a poker hand from a deck of cards can be based on the intuitive notion of locked boxes. This idea in turn can be implemented using commutative-key cryptosystems.

We first present a coin-flipping protocol using locked boxes.
Preparing the boxes

Imagine two sturdy boxes with hinged lids that can be locked with a padlock.

Alice writes “heads” on a slip of paper and “tails” on another.

“heads”, signed Alice  “tails”, signed Alice

She places one of these slips in each box.
Alice locks the boxes

Alice puts a padlock on each box for which she holds the only key.

She then gives both locked boxes to Bob, in some random order.
Bob adds his lock

Bob cannot open the boxes and does not know which box contains “heads” and which contains “tails”.

He chooses one of the boxes and locks it with his own padlock, for which he has the only key.

He gives the doubly-locked box back to Alice.
Alice removes her lock

Alice gets

She removes her lock.

and returns the box to Bob.
Bob opens the box

Bob gets

He removes his lock

opens the box, and removes the slip of paper from inside.

“heads”, signed Alice

He gives the slip to Alice.
Alice checks that Bob didn’t cheat

At this point, both Alice and Bob know the outcome of the coin toss.

Alice verifies that the slip of paper is one of the two that she prepared at the beginning, with her handwriting on it.

She sends her key to Bob.
Bob checks that Alice didn’t cheat

Bob still has the other box.

He removes Alice’s lock,

opens the box, and removes the slip of paper from inside.

“tails”, signed Alice

He checks that it contains the other coin value.
Locked Box Implementation
Commutative-key cryptosystems

Alice and Bob can carry out this protocol electronically using any *commutative-key* cryptosystem, that is, one in which $E_A \circ E_B = E_B \circ E_A$.\(^1\)

RSA is commutative for keys $A$ and $B$ with a common modulus $n$, so we can use RSA in an unconventional way. Rather than making the encryption exponent public and keeping the factorization of $n$ private, we turn things around.

\(^1\)Recall the related notion of “commutative cryptosystem” of [lecture 11](#) in which the encryption and decryption functions for the *same* key commuted.
### RSA as a commutative-key cryptosystem

Alice and Bob jointly chose primes $p$ and $q$, and both compute $n = pq$.

Alice chooses an RSA key pair $A = ((e_A, n), (d_A, n))$, which she can do since she knows the factorization of $n$.

Similarly, Bob chooses an RSA key pair $B = ((e_B, n), (d_B, n))$ using the same $n$.

Alice and Bob both keep their key pairs private (until the end of the protocol, when they reveal them to each other to verify that there was no cheating).
Security remark

We note that this scheme may have completely different security properties from usual RSA.

In RSA, there are three different secrets involved with the key: the factorization of $n$, the encryption exponent $e$, and the decryption exponent $d$.

We have seen previously that knowing $n$ and any two of these three pieces of information allows the third to be reconstructed.

Thus, knowing the factorization of $n$ and $e$ lets one compute $d$. It is also possible to factor $n$ given both $e$ and $d$.

The way RSA is usually used, only $e$ is public, and it is believed to be hard to find the other two secrets.
A new use for RSA

Here we propose making the factorization of $n$ public but keeping $e$ and $d$ private.

It may indeed be hard to find $e$ and $d$, even knowing the factorization of $n$, but if it is, that fact is not going to follow from the difficulty of factoring $n$.

Of course, for security, we need more than just that it is hard to find $e$ and $d$.

We also need it to be hard to find $m$ given $c = m^e \mod n$.

This is reminiscent of the discrete log problem, but of course $n$ is not prime in this case.
Coin-flipping using commutative-key cryptosystems

We now implement the locked box protocol using RSA.

Here we assume that Alice and Bob initially know large primes $p$ and $q$.

In step (2), Alice chooses a random number $r$ such that $r < (n - 1)/2$.

This ensures that $m_0$ and $m_1$ are both in $\mathbb{Z}_n$.

Note that $i$ and $r$ can be efficiently recovered from $m_i$ since $i$ is just the low-order bit of $m_i$ and $r = (m_i - i)/2$. 
### Locked Box Implementation

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Choose RSA key pair $A$ with modulus $n = pq$.</td>
<td>Choose RSA key pair $B$ with modulus $n = pq$.</td>
</tr>
<tr>
<td>2. Choose random $r \in \mathbb{Z}_{(n-1)/2}$. Let $m_i = 2r + i$, for $i \in {0, 1}$. Let $c_i = E_A(m_i)$ for $i \in {0, 1}$. Let $C = {c_0, c_1}$.</td>
<td>$\xrightarrow{C}$ Choose $c_a \in C$.</td>
</tr>
<tr>
<td>3. ( \xleftarrow{c_{ab}} ) Let $c_{ab} = E_B(c_a)$.</td>
<td></td>
</tr>
<tr>
<td>4. Let $c_b = D_A(c_{ab})$.</td>
<td>$\xrightarrow{c_b}$</td>
</tr>
<tr>
<td>5. Let $m = D_B(c_b)$. Let $i = m \mod 2$. Let $r = (m - i)/2$. If $i = 0$ then “tails”. If $i = 1$ then “heads”.</td>
<td>( \xleftarrow{B} )</td>
</tr>
</tbody>
</table>
6. Let $m = D_B(c_b)$. Check $m \in \{m_0, m_1\}$. If $m = m_0$ then “tails”. If $m = m_1$ then “heads”.

7. Let $c'_a = C - \{c_a\}$. Let $m' = D_A(c'_a)$. Let $i' = m' \mod 2$. Let $r' = (m' - i')/2$. Check $i' \neq i$ and $r' = r$. 
Correctness when Alice and Bob are honest

When both Alice and Bob are honest, Bob computes
\[ c_{ab} = E_B(E_A(m_j)) \] for some \( j \in \{0, 1\} \).

In step 4, Alice computes \( c_b \).

By the commutativity of \( E_A \) and \( E_B \),
\[ c_b = D_A(E_B(E_A(m_j))) = E_B(m_j). \]

Hence, in step 5, \( m = m_j \) is one of Alice’s strings from step 2.
A dishonest Bob

A dishonest Bob can control the outcome of the coin toss if he can find two keys $B$ and $B'$ such that $E_B(c_a) = E_{B'}(c'_a)$, where $C = \{c_a, c'_a\}$ is the set received from Alice in step 2.

In this case, $c_{ab} = E_B(E_A(m_j)) = E_{B'}(E_A(m_{1-j}))$ for some $j$. Then in step 4, $c_b = D_A(c_{ab}) = E_B(m_j) = E_{B'}(m_{1-j})$.

Hence, $m_j = D_B(c_b)$ and $m_{1-j} = D_{B'}(c_b)$, so Bob can obtain both of Alice’s messages and then send $B$ or $B'$ in step 5 to force the outcome to be as he pleases.

To find such $B$ and $B'$, Bob would need to solve the equation

$$c_a^e \equiv c_a'^{e'} \pmod{n}$$

for $e$ and $e'$. Not clear how to do this, even knowing the factorization of $n$. 

---

Locked Box Implementation
Card dealing using locked boxes

The same locked box paradigm can be used for dealing a 5-card poker hand from a deck of cards.

Alice takes a deck of cards, places each card in a separate box, and locks each box with her lock.

She arranges the boxes in random order and ships them off to Bob.

Bob picks five boxes, locks each with his lock, and sends them back.

Alice removes her locks from those five boxes and returns them to Bob.

Bob unlocks them and obtains the five cards of his poker hand.

Further details are left to the reader.
Generalization of coin-flipping protocol

In the locked box coin-flipping protocol, Alice has two messages $m_0$ and $m_1$.

Bob gets one of them.

Alice doesn’t know which (until Bob tells her).

Bob can’t cheat to get both messages.

Alice can’t cheat to learn which message Bob got.

The *oblivious transfer problem* abstracts these properties from particular applications such as coin flipping and card dealing,
One-of-two oblivious transfer

In one-of-two oblivious transfer (OT$_1^2$), Alice has two secrets, $s_0$ and $s_1$.

Bob always gets exactly one of the secrets, each with probability 1/2.

Alice does not know which one Bob gets.

The locked box protocol is one way to implement one-of-two oblivious transfer.
Another 1-of-2 OT protocol using blinding $^2$

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Secrets $s_0$ and $s_1$. Choose RSA key $(n, e, d)$. Let $y_i = E_e(s_i)$, $i \in {0, 1}$.</td>
<td>$n, e, y_0, y_1$ $\rightarrow$</td>
</tr>
<tr>
<td>2.</td>
<td>Choose random $b \in {0, 1}$. Choose random $r \in \mathbb{Z}_n^*$. $\leftarrow c$ Compute $c = y_b E_e(r) \mod n$.</td>
</tr>
<tr>
<td>3. Let $c' = D_d(c) \equiv s_b r \pmod{n}$.</td>
<td>$c'$ $\rightarrow$</td>
</tr>
<tr>
<td>4.</td>
<td>Output $c'r^{-1} \mod n = s_b$.</td>
</tr>
</tbody>
</table>

$^2$This protocol is adapted from notes by David Wagner, U.C. Berkeley, CS276, lecture 29, May 2006.
Analysis

- In step 1, Alice sends Bob encryptions of both secrets.
- In step 2, Bob chooses one of Alice’s encryptions, blinds it, and returns the result to Alice.
- In step 3, Alice decrypts whatever Bob sends her, which allows Bob to unblind the decryption and recover the secret he chose in step 2.

Alice’s other secret is safe assuming semi-honest parties (see lecture 21) as long as RSA is secure under a limited chosen ciphertext attack (since that is what Alice permits in step 3).

Bob’s blinding prevents Alice from knowing which secret he learned.
Blinding

*Blinding* is a powerful concept in privacy-preserving protocols.

Bob computes a *blinding factor* $B$ by encrypting a random $r \in \mathbb{Z}_n^*$, so $B = E_e(r) \mod n$ is also a random element of $\mathbb{Z}_n^*$.

Alice gets no information about $b$ from $c = y_b B \mod n$ in step 2 since $c$ is just another random element in $\mathbb{Z}_n^*$.

Because Alice is using RSA, when she decrypts $c$, the resulting $c'$ is also random.

$$c' = D_d(c) \equiv c^d \equiv (y_b B)^d \equiv (s_b r^e)^d \equiv (s_b r)^{ed} \equiv s_b r \pmod{n}.$$  

Bob removes the blinding factor by computing $s_b = c' r^{-1} \mod n$. 
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
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

\[
\begin{align*}
T[0, 0] &= a_w \oplus g(a_u, a_v) \\
T[0, 1] &= a_w \oplus g(a_u, a_v \oplus 1) \\
T[1, 0] &= a_w \oplus g(a_u \oplus 1, a_v) \\
T[1, 1] &= a_w \oplus g(a_u \oplus 1, a_v \oplus 1)
\end{align*}
\]

(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 $\text{OT}_4^4$ 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 Bobs shares for $w$ are both independent of $\sigma_w$.
   - Alices share is chosen uniformly at random.
   - Bobs 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 OT$^2_1$ to distribute the shares for Bobs inputs, and one OT$^4_1$ for each gate.$^3$

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 OT protocols.

$^3$Note: The $n_y$ executions of OT$^2_1$ 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 $\text{OT}_1^2$ 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 $\text{OT}^2_1$ 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 $\text{OT}^2_1$ 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.