Lecture Notes 22

58 Simple Application of Oblivious Transfer

Consider the problem of privately evaluating a Boolean function $f(x, y)$, where x is private to Alice and y is private to Bob. This corresponds to privately computing the functionality

$$F(x, y) = (f(x, y), f(x, y)).$$

Here's the protocol.

1. Alice, with private input $x \in \{0, 1\}$, prepares a table T:

y	$f(x, y)$
0	$f(x, 0)$
1	$f(x, 1)$

   She doesn't know y, but she does know that the correct value $f(x, y)$ is in her table. It's either $f(x, 0)$ or $f(x, 1)$.

2. Bob, with private input y, obtains line y of the table using OT_1^2. Bob outputs $f(x, y)$ without learning x.

3. Bob sends $f(x, y)$ to Alice, who also outputs it.

While this functionality seems almost too trivial to be interesting, it's really not. For example, if $f(x, y) = x \wedge y$ and Alice knows $x = 0$, then the answer $f(x, y)$ does not tell her Bob's value y, so it's important that the protocol also not leak y in this case. Similarly, when Bob requests the value corresponding to row 0, he gets no information about x when the result $f(x, 0) = 0$ comes back. (In fact he knew that already before getting row 0 from Alice.)

59 Private Circuit Evaluation using Shares

We now generalize the example of section 58 to any function $\bar{z} = f(\bar{x}, \bar{y})$, where $\bar{x}$, $\bar{y}$, and $\bar{z}$ are bit strings of lengths n_x, n_y, and n_z, respectively, and $f(\bar{x}, \bar{y})$ is computed by a polynomial size Boolean circuit with $n_x + n_y$ input wires and n_z output wires. The corresponding functionality is

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

Alice furnishes the (private) input data to the first n_x input wires. Bob furnishes the input data for the remaining n_y input wires. Alice and Bob should learning nothing about each other's inputs or the intermediate values of the circuit, other than what is implied by their own inputs and the n_z output values.

A non-private evaluation of the circuit associates a Boolean value σ_w with each wire of the circuit. The input wires are associated with the corresponding input values. Let G be a gate with

input wires u and v and output wire w that computes the Boolean function $g(x, y)$. If σ_u is the value on wire u and σ_v the value on wire v, then the value on wire w is $g(\sigma_u, \sigma_v)$. 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.

To carry out the evaluation privately, we split the value σ_w on each wire w into two random shares a_w and b_w. Neither share alone gives any information about σ_w, but together they allow σ_w to be computed as $a_w \oplus b_w$.

We now describe how Alice and Bob obtain their shares while maintaining the desired privacy. There are three cases, depending on whether w is an input wire controlled by Alice, an input wire controlled by Bob, or the output wire of a gate G.

Input wire controlled by Alice: Alice knows σ_w. She generates a random share $a_w \in \{0, 1\}$ for herself and sends Bob his share $b_w = a_w \oplus \sigma_w$.

Input wire controlled by Bob: Alice chooses a random share $a_w \in \{0, 1\}$ for herself. She prepares a table T such that $T[0] = a_w$ and $T[1] = a_w \oplus 1$. She uses OT_1^2 to send $T[\sigma_w]$ to Bob, who takes his share to be $b_w = T[\sigma_w]$.

Output wire of a gate G: Let the input wires to G be u and v as before, and let $g(x, y)$ be the function computed by G. Alice chooses a random share $a_w \in \{0, 1\}$ for herself. She computes the table

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

(Equivalently, $T[r, s] = a_w \oplus g(a_u \oplus r, a_v \oplus s)$ for $r, s \in \{0, 1\}$.) Alice sends $T[b_u, b_v] = a_w \oplus g(\sigma_u, \sigma_v)$ to Bob using OT_1^4, who takes his share to be $b_w = T[b_u, b_v]$.

After having computed shares for all of the wires, Alice and Bob exchange their shares a_w and b_w for each output wire w.

Remarks:

1. Alice and Bob's shares for w are both independent of σ_w. This follows because Alice's share is always chosen uniformly and independently at random, and Bob's share is always the XOR of Alice's random bit a_w with something independent of of a_w.

2. This protocol requires n_y executions of OT_1^2 to distribute the shares for Bob's inputs, and one OT_1^4 for each gate in the circuit.[1]

3. We emphasize that this protocol assumes semi-honest parties.

4. This protocol generalizes readily from 2 to m parties. (The textbook in fact presents only the m-party version, which, although similar, is considerably more difficult to understand.)

5. Bob does not even need to know what function each gate G computes. All he has to do is to use his private inputs or shares to request the right line of the table in each of the several OT protocols.

[1] We remark that the n_y executions of 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.

60 Private Circuit Evaluation using 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 are templates that allow 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.

After creating the circuit, Alice, who knows all of the tags, uses 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. 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.

In greater detail, for each wire w, Alice generates two n-bit *tags* s_w^0 and s_w^1. Tag s_w^0 consists of an $(n-1)$-bit random string followed by 0, and tag s_w^1 consists of an $(n-1)$-bit random string followed by 1. The last bit of each tag is called its *type*. Alice also generates a random isomorphism $\tau_w : \{s_w^0, s_w^1\} \to \{0,1\}$ that gives the correspondence between the pair of tags on a wire and the pair of Boolean values that they represent.

The tags are used as the keys for an encryption function $E()$. Let $G(u,v)$ be a gate with input wires u and v and output wire w that computes the Boolean function $g(x,y)$. With each pair of tag types $i,j \in \{0,1\}$, we associate a *template* $T_G[i,j] = E_{s_u^i}(E_{s_v^j}(s_w))$, where $s_w \in \{s_w^0, s_w^1\}$. The type of s_w, that is, whether $s_w = s_w^0$ or $s_w = s_w^1$, is determined by the Boolean values $\sigma_u = \tau_u(s_u^i)$ and $\sigma_v = \tau_v(s_v^j)$. Namely, s_w is chosen so that $\tau_w(s_w) = g(\sigma_u, \sigma_v)$. Thus, template $T_G[i,j]$ represents the row in the truth table for $g(\sigma_u, \sigma_v)$, but the correspondences between i and σ_u and between j and σ_v are both random and can't be determined without knowing τ_u and τ_v. The garbled circuit attaches to each gate the four templates $T_G[0,0], T_G[0,1], T_G[1,0], T_G[1,1]$.

Now, suppose Bob knows tags $s_u \in \{s_u^0, s_u^1\}$ and $s_v \in \{s_v^0, s_v^1\}$ corresponding to input wires u and v of gate G. By looking at the last bit of s_u and s_v, he can determine their types i and j, respectively. He next looks up the template $T_G[i,j] = E_{s_u}(E_{s_v}(s_w))$. Since he knows s_u and s_v, he is able to decrypt this to obtain s_w. In this way, he is able to recover the tag corresponding to the output value of G. He still doesn't know the Boolean value that s_w represents (which is $\tau(s_w)$), but he is able to proceed with the garbled circuit evaluation. Assuming Alice has published τ_w for each output wire w, then Bob is able to interpret the tag he computes for each output wire and learn its value.

There remains the problem of how Bob gets the tags corresponding to the input wires. For the wires controlled by Alice, she just sends the appropriate tags to Bob (but not the values that they represent). Thus, if Alice controls input wire u and its value is x, she sends Bob $\tau_u^{-1}(x)$. For the wires that Bob controls, Alice uses OT_1^2 to send him the appropriate tags. Thus, if Bob controls input wire v and its value is y, Alice prepares a table $T[0] = \tau_v^{-1}(0)$ and $T[1] = \tau_v^{-1}(1)$ and uses OT_1^2 to send $T[y]$ to Bob.

Remarks:

1. The fact that Alice doesn't learn anything about Bob's private inputs follows from the fact that the only communication from Bob to Alice is in the OT_1^2 protocol steps used to distribute the tags for the wires that Bob controls. The fact that Bob doesn't learn anything about Alice's private inputs or the intermediate values of the circuit follows from the fact (not so easily proved) that Bob doesn't know the correspondence between tags and Boolean values and also

that Bob is only able to decrypt one of the four templates associated with each gate.

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 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. This protocol does not obviously generalize to more than two parties.

6. As with the share-based protocol, Bob does not need to know what function each gate G computes. All he needs to carry out his end of the protocol is the list of templates associated with each gate.

61 Homomorphic Encryption

An encryption function $E(\cdot)$ is said to be *homomorphic* with respect to an operator $\odot$ if one can compute $E(x \odot y)$ from $E(x)$ and $E(y)$ without decrypting either ciphertext.

Several well-known cryptosystems have a homomorphic property.

RSA $E(x \cdot y) = (xy)^e \bmod n = x^e \cdot y^e \bmod n = E(x) \cdot E(y) \bmod n$.

ElGamal

$$\begin{aligned} E(xy) &= (g^{r_x+r_y}, (xy)h^{r_x+r_y}) \\ &= (g^{r_x}, xh^{r_x}) \cdot (g^{r_y}, yh^{r_y}) \\ &= E(x) \cdot E(y), \end{aligned}$$

where $\cdot$ on pairs means componentwise multiplication.

Goldwasser-Micali Public key is $n = pq$, $y \in \mathrm{QNR}_n$, $E(b) = r^2 y^b \bmod n$ for random r.

$$\begin{aligned} E(b_1) \cdot E(b_2) \bmod n &= (r_1^2 y^{b_1})(r_2^2 y^{b_2}) \bmod n \\ &= (r_1 r_2)^2 y^{b_1+b_2} \bmod n \end{aligned}$$

While this is not equal to $E(b_1 \oplus b_2) = (r_1 r_2)^2 y^{b_1 \oplus b_2}$, is equal to $r^2 y^{b_1 \oplus b_2}$ for some possibly different choice of r. Hence, $E(b_1) \cdot E(b_2)$ is a valid encryption of $b_1 \oplus b_2$, as desired.

Benaloh This generalizes the Goldwasser-Micali scheme to give

$$E(\sum_{i=1}^{k} b_i) = \prod_{i=1}^{k} E(b_i)$$

As with Goldwasser-Micali, this is a randomized encryption scheme, so equality means only that the product is one of the possible encryptions of the sum of the b_i's.

One application of homomorphic encryption is to verifiable secret ballot elections. Each voter i has a vote b_i. To cast the vote, the voter computes $c_i = E(b_i)$ using the public encryption function of the voting authority and submits c_i. Here we assume the Benaloh scheme. The voting authority

publishes the c_i's for all of the voters, gives the tally $t = \sum b_i$, and gives the random string that shows $E(t) = \prod_{i=1}^{k} E(b_i)$. Any voter can check that her own vote appears and can check that this equation holds, but she cannot determine anyone else's votes. This makes sense in the situation where the voting authority is trusted to respect the privacy of votes but is not trusted to count the votes correctly. Much more is needed to turn this idea into a plausible voting system.

62 Flipping a Coin into a Well

The goal here is to compute the functionality $F(1^n, 1^n) = (b, b)$, where $b \in \{0, 1\}$ is uniformly distributed and n is a security parameter. If aborts are possible, we also have to allow the possibility of $(b, \perp)$.

Figure 62.1 gives a very simple protocol based on bit-commitment.

	Alice		Bob
1.	Chooses $b_1 \in \{0,1\}$, $s \in \{0,1\}^n$, $c_1 = C_s(b_1)$.	$\xrightarrow{c_1}$	
2.		$\xleftarrow{b_1}$	Chooses $b_2 \in \{0,1\}$.
3.	Outputs $b_1 \oplus b_2$.	$\xrightarrow{(b_1, s)}$	
4.			If $c_1 = C_s(b_1)$ outputs $b_1 \oplus b_2$ else outputs $\perp$.

Figure 62.1: Flipping a coin into a well.

This is just a sample of a protocol that, like zero-knowledge proof systems, is designed to work against a malicious adversary. How to prove the correctness of such protocols in general could be the topic of another course.