Notes for Lecture 16

1 Review of Function Encryption

Functional encryption is like a regular encryption scheme, except that you can give out secret keys for functions f, where the user has a secret key for function f only learns f of the message m, rather than the message m itself. That is, you give slightly functional secret keys, that don't allow decrypting completely.

Last lecture, we gave a distinguishability-based definition which was a natural generalization of the definitions we've seen for Identity Based Encryption and other primitives. In the homework, you showed that it's actually unsatisfactory for some applications.

But yet, the the better sort of simulation based notion isn't feasible. We hinted at how they work last lecture, and the homework will take you through this in a bit more detail.

Because of the impossibility, in general, of achieving this ideal notion, the literature is broken into two or three categories:

- Single secret key case (1 collision) with one-bit outputs. This circumvents the impossibility of the simulation definition.
- Focus on indistinguishability definition (maybe for many keys). There is no impossibility known for indistinguishability, but the definition is slightly lacking for some cases.
- Focus on specific functionality. The impossibility is due to functions that evaluate specific crypto functionality, such as a PRF. Limiting the scope of possible functions to more specific functions (like membership in a hyperplane, for example) circumvents the impossibility of the simulation notion.

In this lecture, we're mainly going to focus on the single secret key case, because it turns out here that we can actually do something interesting. While it is possible to get the stronger definition, for simplicity, we will focus on the indistinguishability experiment from last lecture. We won't use the stronger simulation based notion, because defining it is a bit more complex.

Starting next week, we will cover program obfuscation, and we'll see that program obfuscation actually lets us come back and solve the many-key setting.

In order to solve the single key case first, we will need to introduce and define a new crypto tool called garbled circuits.

2 Garbled Circuits

The definition of garbled circuits predates functional encryption by quite a lot. They were first discussed and defined in the 80s in a series of unpublished works, and they were used for multi party computation (a topic that hasn't been covered in this course). We will see that they're useful for functional encryption as well as in other places throughout cryptography.

Definition 1. *A Garbled Circuit consists of two algorithms,* Garble*, and* Eval*.* Garble *takes in a circuit $C : \{0,1\}^n \to \{0,1\}$, and security parameter λ.*

$$\left(\hat{C}, \{L_{ib}\}_{i\in[n],b\in\{0,1\}}\right) \leftarrow \mathsf{Garble}\left(C, 1^\lambda\right)$$

Here, $\{L_{ib}\}_{i\in[n],b\in\{0,1\}}$ are called the labels. There are two labels for every for every input bit to the circuit, one label corresponding to whether that input bit is zero and another corresponding to whether that input bit is one. $\hat{C}$ is called the garbled circuit.

Eval *takes in the garbled circuit, an input $x \in \{0,1\}^n$, and a set of n labels $\{L_{ix_i}\}_{i\in[n]}$ corresponding to the bits of the input x, and outputs a bit $z \in \{0,1\}$.*

$$z \leftarrow \mathsf{Eval}\left(\hat{C}, x, \{L_{ix_i}\}_{i\in[n]}\right)$$

Correctness: $z = C(x)$. That is, you get the correct evaluation of the circuit.

Security: There exists an algorithm Sim that takes in an input to the circuit $x \in \{0,1\}^n$ and a possible output $z \in \{0,1\}$, as well as n, the size of the circuit $|C|$, and security parameter λ. It then outputs a simulated garbled circuit and simulated label set for x.

$$(\hat{C}', \{L'_{ix_i}\}_{i\in[n]}) \leftarrow \mathsf{Sim}\left(x, z, 1^n, 1^{|C|}, 1^\lambda\right)$$

such that it is computationally indistinguishable from the output of Garble (after filtering the labels for input x, and using output $C(x)$). That is,

$$\mathsf{Sim}\left(x, C(x), 1^n, 1^{|C|}, 1^\lambda\right) \approx_c \left(\hat{C}, \{L_{ix_i}\}_{i\in[n]}\right)$$

Intuitively what's going on is that someone has a circuit, C, that they garble to produce a garbled circuit $\hat{C}$ and a set of labels $\{L_{ib}\}_{i\in[n],b\in\{0,1\}}$. Then, if they want to give someone else the ability to evaluate the circuit on some input x, they send the garbled circuit and the set of labels corresponding to x.

The *correctness requirement* says that from the garbled circuit $\hat{C}$ and the set of labels $\{L_{ix_i}\}_{i\in[n]}$ for an input x, you actually learn the correct evaluation of the circuit.

The *security* says that you learn nothing else. Sim only takes as input an input to the circuit x, the supposed output $C(x)$, and the parameters of the circuit n, $|C|$, and λ. Yet somehow, even though it doesn't know the circuit and only knows the outputs of the circuit on the on that single particular input, it is somehow able to simulate the view of the Eval procedure. It is somehow able to simulate the garbled circuit on half of the labels.

Note that it certainly can't simulate all of the labels, because if you had all the labels, you could evaluate the circuit on any input, but Sim is only given the evaluation of the circuit on a single input. But it can nevertheless simulate, the half of the labels corresponding to the single input x that it was given.

The simulation that we require here is one that the output is computationally indistinguishable from the real distribution. That is, the actual distributions may be technically different, but no computationally bounded adversary should be able to tell the difference. This is denoted with a $\approx_c$ above.

Next, to give some intuition, we will go over one of the original applications of garbled circuits.

3 Two Party Function Evaluation

This discussion of 2 party function evaluation won't be complete because we won't cover something called oblivious transfer which is necessary for the protocol, but that we haven't really formally defined. Hopefully it still gives some intuition for why garbled circuits are useful.

Setup:

- Alice gets a circuit $C : \{0,1\}^n \to \{0,1\}$.
- Bob gets an input $x \in \{0,1\}^n$.

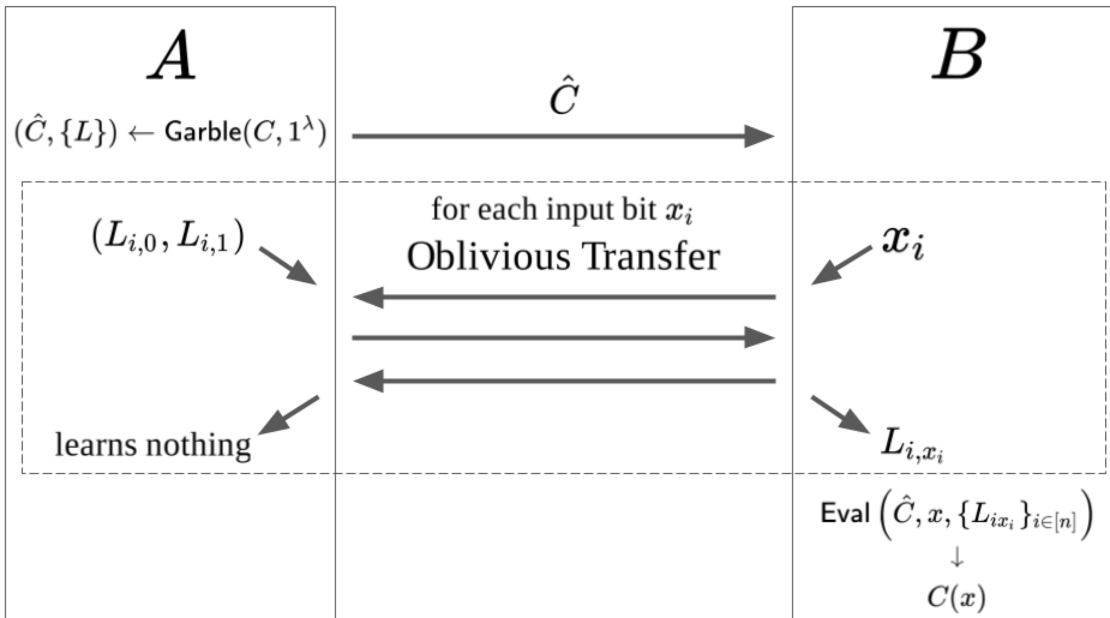
Goal:

- Alice learns nothing.

- Bob learns $C(x)$, but nothing else.

This is achieved as follows:

1. First Alice garbles her circuit C to get a garbled circuit, $\hat{C}$, which she sends to Bob, and a set of labels $\{L_{ib}\}_{i\in[n],b\in\{0,1\}}$.
2. Alice then sends the labels, $\{L_{ix_i}\}_{i\in[n]}$, corresponding to Bob's input x without learning what x is. This uses a process called *oblivious transfer*, which takes in a pair of labels (or in general any two values), L_0 and L_1, from Alice, and a bit b from Bob. At the end of the oblivious transfer process, Bob learns L_b and nothing else, while Alice learns nothing (in particular, she doesn't find out which label Bob has learned).
3. Finally, Bob uses the garbled circuit and the labels from the oblivious transfer to evaluate the circuit on his input.



The notion of learning nothing is again formalized by a simulation condition. That is, it requires that Bob's entire view can be simulated just knowing $C(x)$ and not knowing anything else about the circuit other than its size. Therefore this implies that Bob learns nothing about Alice's circuit except $C(x)$. At the same time, Alice learns nothing because the oblivious transfer never gives her any information about Bob's input x.

4 Building functional encryption from garbled circuits

We're going to assume just a generic public key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. We are also going to assume an upper bound T on the description size of the functions we would like to support in the functional encryption scheme (can think of T as a bound of the circuit size).

Construct the functional encryption scheme as follows:

$\mathsf{Gen}_{\mathrm{FE}}()$:

- **Master public key mpk:** A set of $2T$ public keys $\{\mathsf{pk}_{ib}\}_{i\in[T],b\in\{0,1\}}$
- **Master secret key msk:** The corresponding set of $2T$ secret keys $\{\mathsf{sk}_{ib}\}_{i\in[T],b\in\{0,1\}}$

$\mathsf{Extract}_{\mathrm{FE}}(\mathsf{msk}, f)$:

- Write f as a bit string in $\{0,1\}^T$
- Use the bit string for f to select a subset of the secret keys $\mathsf{sk}_f = \{\mathsf{sk}_{i\,f_i}\}_{i\in[T]}$

$\mathsf{Enc}_{\mathrm{FE}}(\mathsf{mpk}, m)$:

- Let $U(\cdot,\ \cdot)$ be a universal circuit. That is, $U(f,m) = f(m)$, where U gets a bit string description of f, interprets it as a circuit (or some other model), and evaluates it on m. Such universal circuits can be shown to exist, but we will not cover them in this class and just assume them.
- Let $C_m = U(\cdot,\ m)$. That is, C_m is a circuit that takes as input a description of a function and evaluates that function on m.
- Compute a garbled version of C_m $\left(\hat{C}_m,\ \{L_{ib}\}\right) \leftarrow \mathsf{Garble}\left(C_m,\ 1^\lambda\right)$
- Output the garbled circuit along with the encrypted labels $\left(\hat{C}_m,\ \{\mathsf{Enc}\left(\mathsf{pk}_{ib},\ L_{ib}\right)\}_{ib}\right)$

$\mathsf{Dec}_{\mathrm{FE}}\left(\mathsf{sk}_f,\ \ c = \left(\hat{C},\ \{d_{i\,b}\}\right)\right)$:

- Use the T to decrypt the relevant part of the ciphertext corresponding to the bit description of the function f to get the corresponding T labels. Use that to evaluate the garbled circuit.
$\mathsf{Eval}\left(\hat{C},\ f,\ \{\mathsf{Dec}\left(\mathsf{sk}_{f_i},\ d_{i\ f_i}\right)\}\right)$

- By the correctness of the garbled circuit procedure, this will output $C_m(f) = f(m)$ as desired.

Security Intuition

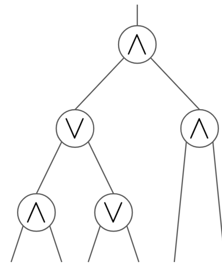
1. If you only know sk_f and you learn $L_{i\ f_i}$ and other labels remain hidden, you can replace the other labels with "junk" (when generating the challenge ciphertext) as the adversary doesn't have the secret key for those and so cannot tell the difference.

2. Simulate the remaining labels just given $f(m)$ and the size of the function $|f| = T$.

This actually gives the stronger simulation security notion that we've alluded to. That's because all you need in order to simulate the ciphertext is the output of the secret keys, without needing to know the secret message.

5 Constructing a Garbled Circuit

Suppose we start with some circuit made of AND and OR gates



Assign to each wire w two labels, $L_{w,0}$ and $L_{w,1}$.

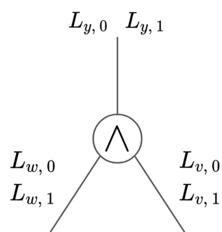
- The labels on the input wires are going to be the labels outputted by Garble for the input bits. These are outputted in the clear.

- The internal wires will be used to construct $\hat{C}$, but not outputted in the clear.

Invariant: As we run $\mathsf{Eval}\left(\hat{C},\ x,\ \{L_{i\,x_i}\}\right)$, imagine running $C(x)$. If wire w has value b, Eval learns $L_{w,b}$, but not $L_{w,1-b}$.

In other words, as Eval goes gate by gate through the circuit (without knowing what the circuit actually is, of course, besides its topology), for every wire in the circuit, it will learn the label corresponding to the the bit that that wire takes in the correct evaluation, but not the bit itself.

To do this let's imagine that we have a gate and let's say without loss of generality that it's an AND gate. Call the two input wires w and v, and call the output wire y. Assign two labels for each of those wires:



We want it to be possible to learn the appropriate label of the output wire, but it should not be possible to learn anything else about the other label for that wire, which should remain completely hidden.

More specifically, if the values of the two input wires are bits a and b, respectively, then by the invariant, we have both $L_{w,\,a}$ and $L_{v,\,b}$. By using those two labels, we should only learn the corresponding $L_{y,\,c}$ where $c = a \wedge b$.

$$\begin{aligned}
L_{w,\,0},\ L_{v,\,0} &\longrightarrow L_{y,\,0}\\
L_{w,\,0},\ L_{v,\,1} &\longrightarrow L_{y,\,0}\\
L_{w,\,1},\ L_{v,\,0} &\longrightarrow L_{y,\,0}\\
L_{w,\,1},\ L_{v,\,1} &\longrightarrow L_{y,\,1}
\end{aligned}$$

To do this we encrypt $L_{y,\,0}$ in such a way that knowing any of the first three pairs of labels, allows you to learn $L_{y,\,0}$, whereas knowing the last pair doesn't allow you to learn anything about $L_{y,\,0}$.

A natural way to do this is to interpret $L_{w,\,a}$ and $L_{v,\,b}$ as secret keys for a secret key encryption, and for every pair of bits (a, b), take the label $L_{y,\,a\wedge b}$ that would be the result of the gate on those two inputs, and encrypt it first under $L_{v,\,b}$, and then under $L_{w,\,a}$.

$$c_{a,\,b} = \mathsf{Enc}\left(L_{w,\,a},\ \mathsf{Enc}\left(L_{v,\,b},\ L_{v,\,a\wedge b}\right)\right) \quad \forall\, a,\, b \in \{0, 1\}$$

If you know only one label for each input wire, then there's only one of these double encrypted values that you are able to decrypt. Therefore you only learn the appropriate output label.

In order to make sure that you don't know which of the $c_{a,b}$'s you were able to decrypt (and therefore learn a and b), we have to shuffle the four $c_{a,b}$ ciphertexts randomly. You therefore need to try decrypting all four ciphertexts, which means that we need a signal indicating when decryption succeeds.

We want the encryption to abort if you use the wrong keys, but at the same time to not accidentally reveal everything else. For example, let's assume that all ciphertexts are "decryptable" by any secret key, but that when the secret key is incorrect it decrypts to nonsense. We can do this, for example by padding a preset number of 0's to the encrypted values, so that only a correct encryption will reproduce those zeros.

While we have described this for a 2-input AND gate, we can actually do this for any gate, and just as easily for gates on more than 2-inputs. So this can be done for each of the gates in the circuit.

Finally, for the output wire, include both $L_{\text{out},0}$ and $L_{\text{out},1}$ in plaintext in the garbled circuit. These labels are never used to encrypt anything (only the lower wires' labels are used to encrypt), so it doesn't break the security of any of the encryptions. When evaluating the garbled circuit, just compare the final label you get out of the last gate with these two labels to reveal the output bit of the circuit.

The one remaining issue is that the Eval procedure must know the circuit topology, as it evaluates each gate of the circuit one by one. That is, it reveals which gates are present and where, and which gates connect to which other, but not what type of gate each is (AND, OR, NOT, etc).

Of course the circuit topology is more information than what we want to give our simulator. We want our simulator to receive just the circuit size but know nothing at all about the topology. The way to do this, of course, is to compile the scheme into one that hides the topology using Universal Circuits. That is, instead of garbling C, we garble

$$\mathsf{Garble}\left(U(C,\ \cdot)\right)$$

which is the universal circuit applied to a bit encoding of C. The topology of the circuit being garbled is therefore fixed, and is just the topology of the universal circuit. The circuit itself only affects the instantiation of the individual gates.

6 Known Results in Functional Encryption

- PKE $\Longrightarrow$ 1-ciphertext FE of bounded size functions (what we did here)
  $|\text{ciphertexts}| \geq |\text{function}|$

This can sometimes be undesirable, because if, say, f is a spam filter, the ciphertext will grow with the size of the spam filter rather than the length of the message.

- LWE $\Longrightarrow$ 1-ciphertext FE with small ciphertexts. That is, the size of the ciphertexts is independent of the size of the functions (called "succinct").
- Specific cases (for example, pairings $\Longrightarrow$ subspace membership, even for many functions/secret keys), but they cannot do arbitrary computations.
- Obfuscation $\Longleftrightarrow$ many-time (many secret keys for several functions) FE for arbitrary functions. [These notions are essentially equivalent.]