Notes for Lecture 2

Recap of last time

Last time, we started defining obfuscation. For Turing Machines, recall that we assume the machine outputs its running time in addition to whatever output it is supposed to produce.

Definition 1. *An obfuscator is a probabilistic program* Obf *that takes as input a program* P *(generally either a Turing Machine or Circuit), and a security parameter* λ*, and produces another program* $\hat{P}$*, such that* Obf *is:*

- **Functionality Preserving:** $\hat{P}$ *is "equivalent" to* P*, meaning* $\hat{P}(x) = P(x)$ *for all inputs* x*.*
- **Efficient:** *The running time of* Obf *is polynomial in* λ *and in the size of* P*.*
- **Polynomial Slowdown:** *The running time of* $\hat{P}$ *is polynomial in* λ *and in the running time of* P

The remaining piece to fully specify obfuscation is security. Last time, we saw two definitions:

Definition 2. *An obfuscator* Obf *is Virtual Black Box (VBB) secure if, for any PPT adversary* A*, there exists a simulator* S *and negligible function* negl *such that for any program* P*,*

$$\left|\Pr[A(\mathsf{Obf}(P,\lambda)) = 1] - \Pr[S^P(|P|,\lambda) = 1]\right| < \mathsf{negl}(\lambda)$$

Here, the running time of S *is polynomial in* $|P|$ *and* λ*, and* S *takes unit time to make an oracle query to* P*.*

Definition 3. *An obfuscator* Obf *is indistinguishability secure if, for any PPT adversary* A *and any two equivalent programs* P_0 *and* P_1 *of the same size, there exists a negligible function* negl *such that*

$$|\Pr[A(\mathsf{Obf}(P_0,\lambda)) = 1] - \Pr[A(\mathsf{Obf}(P_1,\lambda)) = 1]| < \mathsf{negl}(\lambda)$$

1 Impossibility for VBB Obfuscation

Here, we show that VBB obfuscation is impossible in general. The proofs are due to Barak et al. [BGI+01]. If we strengthen VBB obfuscation to allow the adversary to produce arbitrary strings instead of a single bit, and require the simulator to simulate the string with just black box access, the impossibility is straightforward. Indeed, there is one thing that A can do that S cannot: output the obfuscated program $\hat{P}$. $\hat{P}$ is a polynomial-sized program that computes the functionality of P. An adversary given just black-box access to P cannot in general produce such a program.

Now, we turn to the harder case where the adversary only outputs a single bit. We first prove impossibility for Turing Machines, and then prove impossibility for circuits.

1.1 Impossibility of VBB Obfuscating Turing Machines

Now that we only allow A to output a single bit, what can A possibly do with the code for $\hat{P}$ that she cannot do using black-box access to its functionality? She could try to output a single bit of the program description. However, it is plausible that the obfuscator makes sure that each individual bit is 0 or 1 with 50% probability. What else can A do with $\hat{P}$?

One thing she can do is run $\hat{P}$ on itself. While the simulator can learn the output of P on arbitrary inputs of its choice, it does not know any code for P. This prevents it from choosing $\hat{P}$ — or any program with the same functionality as P — as an input. If P does something special on $\hat{P}$ (say, output some secret value), then A can potentially do something with $\hat{P}$ that S cannot.

Our goal, roughly, is then to devise a program P such that, for any code $\hat{P}$ with the same functionality as P, $\hat{P}(\hat{P})$ outputs some secret (we will not quite get this, but will get something close that is good enough). Moreover, this secret should not be learnable just given black-box access to P.

The starting point will be the following unlearnable function

$$C_{\alpha,\beta}(x) = \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{if } x \neq \alpha \end{cases}$$

$C_{\alpha,\beta}$ is just a point function that outputs β on input α, and outputs 0 elsewhere. Here, $\alpha, \beta \in \{0,1\}^\lambda$. This function will not be enough for us, since $C_{\alpha,\beta}(P) = 0$ for almost any program P that implements $C_{\alpha,\beta}$ — in particular, it does nothing special when run on itself.

Instead, we will give a second program $D_{\alpha,\beta}$ that does do something different on

programs that implement $C_{\alpha,\beta}$. Namely,

$$D_{\alpha,\beta,\gamma}(C) = \begin{cases} \gamma \text{if } C(\alpha) = \beta \\ 0 \text{if } C(\alpha) \neq \beta \end{cases}$$

If A is given any code $\hat{C}, \hat{D}$ for both $C_{\alpha,\beta}$ and $D_{\alpha,\beta,\gamma}$, then it can easily learn γ by running $\hat{D}(\hat{C})$. Moreover, any simulator S given oracle access to $C_{\alpha,\beta}$ and $D_{\alpha,\beta,\gamma}$ cannot find an accepting input for either $C_{\alpha,\beta}$ or $D_{\alpha,\beta,\gamma}$. Therefore, these programs demonstrate an impossibility for a 2-program version of VBB obfuscation.

To get an impossibility for a 1-program version, we do the following. Let $E_{\alpha,\beta,\gamma}(b, x)$ for a bit b be the program that, if $b = 0$ runs $C_{\alpha,\beta}(x)$, and if $b = 1$ runs $D_{\alpha,\beta,\gamma}(x)$.

Consider the following adversary A. On input $\hat{E}$, A computes the program $\hat{C}$ obtained by fixing the input $b = 0$, and the program $\hat{D}$ obtained by fixing the input $b = 1$. Then A runs and outputs the result of $\hat{D}(\hat{C})$. If $\hat{E}$ implements $E_{\alpha,\beta,\gamma}$, then $A(\hat{E}) = \gamma$.

However, a simulator S given black-box access to $E_{\alpha,\beta,\gamma}$ can never find an input that causes $E_{\alpha,\beta,\gamma}$ to give anything but 0. Hence, S cannot output γ.

Remark 4. *In the above, we have ignored the efficiency of the various functions. In particular, the program $D_{\alpha,\beta,\gamma}$ and the adversary A will not halt on all inputs (let alone halt in polynomial time). This is easily remedied by having the program and adversary run for a certain polynomial number of steps, and output 0 if the computation is not finished.*

This gives us the following theorem:

Theorem 5. *There do not exist VBB obfuscators for Turing Machines.*

Remark 6. *The proof above actually shows something much stronger: not only does any obfuscator fail to VBB obfuscate some functions, but there are functions that are unobfuscateable by* any *VBB obfuscator, even ones that take arbitrary time to obfuscate.*

1.2 Impossibility for Circuits

Here, we prove the following:

Theorem 7. *VBB obfuscation for circuits does not exist*

The proof in the Turing machine setting roughly required the ability to run an obfuscated program on itself. For Turing machines, this is okay, since Turing machines operate on unbounded inputs. For circuits, however, the input is typically much

smaller than the description size, precluding the possibility of running a circuit on itself.

Instead, we will use an approach which uses Fully Homomorphic Encryption (FHE).

Definition 8. *A (secret key) fully homomorphic encryption scheme is a quadruple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *such that:*

- $\mathsf{Gen}(\lambda)$ *outputs a secret key* k *and two polynomial time binary operators* $\oplus$ *and* $\otimes$*.*
- **Correctness of decryption:** *For all* λ *and all messages* $m \in \{0,1\}$, $\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m$*,*
$$\Pr[\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m : (k, \oplus, \otimes) \leftarrow \mathsf{Gen}(\lambda)] = 1$$
- **Correctness of homomorphic operations:** *For all* λ*, all message pairs* $m_0, m_1 \in \{0,1\}$*, the following holds:*
$$\Pr[\mathsf{Dec}(k, \mathsf{Enc}(k, m_0) \oplus \mathsf{Enc}(k, m_1)) = m_0 + m_1 : (k, \oplus, \otimes) \leftarrow \mathsf{Gen}(\lambda)] = 1$$
$$\Pr[\mathsf{Dec}(k, \mathsf{Enc}(k, m_0) \otimes \mathsf{Enc}(k, m_1)) = m_0 \times m_1 : (k, \oplus, \otimes) \leftarrow \mathsf{Gen}(\lambda)] = 1$$
Here, $+$ *is carried out mod 2.*
- **CPA security:** *Consider the following game between an adversary and challenger. The challenger is parameterized by the security parameter* λ *and a bit* b*.*
  - *The challenger generates* $(k, \oplus, \otimes) \leftarrow \mathsf{Gen}(\lambda)$*. It gives* $\oplus, \otimes$ *to the adversary.*
  - *The adversary comes up with two sequences of messages* $m_1^{(0)}, \ldots, m_t^{(0)}, m_1^{(1)}, \ldots, m_t^{(1)} \in \{0,1\}$*, which it sends to the challenger.*
  - *The challenger encrypts the sequence of messages corresponding to bit* b*. That is, it computes* $c_1, \ldots, c_t$ *where* $c_i \leftarrow \mathsf{Enc}(k, m_i^{(b)})$*. It then gives the sequence* $\{c_i\}_{i \in [t]}$ *to the adversary.*
  - *The adversary makes a guess* b' *for* b*.*

  Let $W_b(A, \lambda)$ *be the probability* $b' = 1$ *when the challenger is uses bit* b *and security parameter* λ*, and the adversary is* A*. CPA security means that for all PPT adversaries* A*, there exists a negligible function* negl *such that*
$$|\Pr[W_0(A, \lambda) = 1] - \Pr[W_1(A, \lambda) = 1]| < \mathsf{negl}(\lambda)$$

Lemma 9. *If CPA-Secure FHE exists, then VBB obfuscation for circuits does not.*

Proof. Let $(k, \oplus, \otimes)$ be the output of $\mathsf{Gen}(\lambda)$, let α, β be random in $\{0,1\}^\lambda$, $\gamma \in \{0,1\}$, and let $c = \mathsf{Enc}(k, \alpha)$ (here, encryption is done bit-wise).

Let $C_{k,\oplus,\otimes,\alpha,\beta,\gamma,c}$ the the following circuit:

$$C_{k,\oplus,\otimes,\alpha,\beta,\gamma,c}(i,x) = \begin{cases} (\oplus, \otimes, c) & \text{if } i = 0 \\ \beta & \text{if } i = 1 \text{ and } x = \alpha \\ 0 & \text{if } i = 1 \text{ and } x \neq \alpha \\ \gamma & \text{if } i = 2 \text{ and } \mathsf{Dec}(k,x) = \beta \\ 0 & \text{if } i = 2 \text{ and } \mathsf{Dec}(k,x) \neq \beta \end{cases}$$

Now consider the following adversary A, that receives some obfuscated circuit $\hat{C}$. A first runs $\hat{C}(0,0)$ to get $\oplus, \otimes, c$. Next, let $\hat{D}(x) = \hat{C}(1,x)$ be the circuit with $i = 1$ hardwired. Treat the circuit $\hat{D}$ as an arithmetic circuit of $+$ and $\times$ gates. Let $\tilde{D}$ be the circuit with $+, \times$ replaced with $\oplus, \otimes$. Feed c into $\tilde{D}$, by feeding c_i into the ith input. Run $\tilde{D}(c)$, obtaining output d. Finally, run $\hat{C}(2,d)$, and output the result.

If $\hat{C}$ is a circuit that implements $C_{k,\oplus,\otimes,\alpha,\beta,\gamma,c}$ for c as above, then the output of $\tilde{D}$, namely d, will be an encryption of β. This is because $\tilde{D}$ homomorphically evaluates the point function $C_{\alpha,\beta}$ defined in the Turing machine proof, and the input is an encryption of α.

Then running $\hat{C}(2,d)$ will give γ.

Meanwhile, consider a simulator S given just black box access to $C_{k,\oplus,\otimes,\alpha,\beta,\gamma,c}$. Effectively, S has three oracles: the oracle E that always outputs $(\oplus, \otimes, c)$, the point function $C_{\alpha,\beta}$, and the oracle

$$F_{k,\beta,\gamma}(x) = \begin{cases} \gamma & \text{if } \mathsf{Dec}(k,x) = \beta \\ 0 & \text{if } \mathsf{Dec}(k,x) \neq \beta \end{cases}$$

We wish to show that S cannot find γ. Suppose towards contradiction that S can output γ with probability $1 - \mathsf{negl}(\lambda)$. Since the only oracle that depends on γ is $F_{k,\beta,\gamma}$, this means that with probability $1 - \mathsf{negl}(\lambda)$, at some point S queries $F_{k,\beta,\gamma}$ on a ciphertext d such that $\mathsf{Dec}(k,d) = \beta$. Consider the first such query d to $F_{k,\beta,\gamma}$. For all prior queries to $F_{k,\beta,\gamma}$, F outputs 0.

Next, prior to the query on d, the view of S is independent of β, unless it queries $C_{\alpha,\beta}$ on α. Since the query d is an encryption of β, this means it must have queried $C_{\alpha,\beta}$ on α prior to querying $F_{k,\beta,\gamma}$ on d (or got lucky, and guessed β, which can only happend with probability $2^{-\lambda}$). Therefore, up to the point where S queries $C_{\alpha,\beta}$ on α, the oracles $C_{\alpha,\beta}$ and $F_{k,\beta,\gamma}$ output 0, except with exponentially small probability.

Now consider replacing c with encryptions of 0. In this case, the view of S is completely independent of α until it makes a query to $C_{\alpha,\beta}$ on α. Therefore, S can only

query $C_{\alpha,\beta}$ on α with probability $q \times 2^{-\lambda}$, where q is the total number of queries made by S. Since q is a polynomial, this probability is exponentially small. Since S never makes a query to C on α, the view of S is independent of β until it makes a query to F on an encryption of β. Again, since the total number of queries is polynomial, this means that S can only make such a query with exponentially small probability. Therefore, with overwhelming probability, all of S's queries to C or F return 0 in this case.

This means we can effectively distinguish encryptions of 0 from encryptions of α. Indeed, consider the following adversary B for the FHE scheme. B chooses a random α, and sends α to its challenger. In return, it receives $\oplus, \otimes$ as well as c, which is either $\mathsf{Enc}(k, \alpha)$ or $\mathsf{Enc}(k, 0)$.

B now runs S as a subroutine. When S makes a call to E, B responds with $(\oplus, \otimes, c)$. When S makes a call to F, B responds with 0. When S makes a call to C, B does one of two things:

- If the oracle query is on α, then B aborts and outputs 1.
- Otherwise, B responds to the oracle query with 0.

If S finishes and B never aborts, then B outputs 0. In the case where c is an encryption of α, then with overwhelming probability B correctly answers all oracle queries up until the point where S queries C on α, which happens with overwhelming probability by the arguments above. Therefore, B outputs 1 with overwhelming probability.

In the case where c is an encryption of 0, then with overwhelming probability B answers all queries correctly. Since S does not query on α in this case, B finishes without aborting and outputs 0. Therefore, B distinguishes encryptions of α from encryptions of 0 with high probability, contradicting the security of the FHE scheme. □

Next, we see that assuming FHE is unnecessary for the impossibility, and instead we can assume only one-way functions. One-way functions are the most basic object in cryptography, meaning this assumption is very minimal.

Definition 10. *A one-way function is a polynomial-time computable function* $\mathsf{OWF} : \{0,1\}^* \to \{0,1\}^*$ *such that the following holds. For any PPT adversary A, there exists a negligible function* negl *such that*

$$\Pr_{x \leftarrow \{0,1\}^\lambda}[f(\ A(\ f(x)\ )\ ) = f(x)] < \mathsf{negl}(\lambda)$$

The idea is to use VBB obfuscation to build an FHE scheme. Thus, if VBB obfuscation exists, FHE exists, and therefore VBB obfuscation does not exist.

Lemma 11. *If one-way functions and VBB obfuscation for circuits exist, then so does FHE*

You will prove this lemma in your homework.

The last piece is the following:

Lemma 12. *If VBB obfuscation for circuits exists, then so do one-way functions.*

Proof. Let Obf be a VBB obfuscator. Let $F(\alpha, \beta, r) = \mathsf{Obf}(C_{\alpha,\beta}; r)$. Here, $\alpha \in \{0,1\}^\lambda$ and $\beta \in \{0,1\}$. $C_{\alpha,\beta}$ is a circuit computing the point function

$$C_{\alpha,\beta}(x) = \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{if } x \neq \alpha \end{cases}$$

The notation $\mathsf{Obf}(C; r)$ means to obfuscate the circuit C, and whenever Obf needs to flip a coin, look a the next bit of r and use that to determine the coin toss. Thus, while $\mathsf{Obf}(C)$ may be randomize, $\mathsf{Obf}(C; r)$ is a deterministic function.

Why is this function one-way? Suppose there was an adversary B that inverted F with non-negligible probability ϵ. We will construct the following adversary A for Obf. A, on input a circuit $\hat{C}$, runs $B(\hat{C})$ to obtain α, β, r. A then checks if $F(\alpha, \beta, r) = \hat{C}$. If not, A outputs a random bit. If so, then A outputs β. Then, if $\hat{C}$ is an obfuscation of $C_{\alpha,\beta}$ for random α, β (that is, $\hat{C} = \mathsf{Obf}(C_{\alpha,\beta}; r) = F(\alpha, \beta, r)$ for random α, β, r), A outputs the correct answer for β with probability $\frac{1}{2} + \frac{\epsilon}{2}$.

Meanwhile, consider any simulator S that is given oracle access to $C_{\alpha,\beta}$. Unless S is able to query on α (which occurs with exponentially small probability), S can only guess β with probability $\frac{1}{2}$. Therefore, the simulator is unable to simulate the output of A, violating VBB security.

Hence, if we assume VBB security, the adversary B could not have existed, completing the proof. □

References

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (Im)possibility of obfuscating programs. In *Advances in Cryptology — CRYPTO 2001*, number Im, 2001. 2