Notes for Lecture 18

We are going to explore a weaker notion of obfuscation than the black box definition (which is proven to be impossible).

1 Indistinguishability Obfuscation (iO)

Security Definition

$$\forall \text{ programs } P_0, P_1 s.t. \forall x, P_0(x) = P_1(x)$$
$$\mathsf{Obf}(P_0, 1^\lambda) \approx_c \mathsf{Obf}(P_1, 1^\lambda)$$

where we require the two programs to be identical on all inputs because if we know it defers on some input x, then we can trivially distinguish the two programs just by running them on the input and seeing that they differ.

Question

What if the two programs are identical on majority of inputs and differ on a negligible amount?

Answer

iO is not the strongest security definition. Informally there is a variant of iO definition Differing inputs obfuscation (diO) which says

$$\mathsf{Obf}(P_0, 1^\lambda) \approx_c \mathsf{Obf}(P_1, 1^\lambda)$$
$$\forall P_0, P_1 s.t. \text{ differing inputs are hard to find}$$

Unfortunately diO is suspected to be impossible in general. The intuition is that one could give out some program Q which knows differing input x, meaning

$$Q(P) \text{ runs } P(x) \text{ and decides which program is given}$$

if Q is sufficiently obfuscated, then we might hope that x is hidden, which means P_0, P_1 will be two programs that you cannot find the differing input. However, one

can clearly distinguish any code for them by running Q. The point is that there appears to be some sort of barrier which leads to the suspicion of diO's non-existence.

One special case is when P_0, P_1 differ on poly many inputs, then iO implies diO for P_0, P_1.

Question

What about the time that takes to run the programs, for example when will they halt if at all?

Answer

There is some subtleties if we consider a Turing Machine. We want to consider the runtime/space of the turing machine as part of its output, but most of the time we are focusing on circuits and hence we will ignore these subtleties.

2 Why might iO be possible

Focusing on circuits.

$$\text{if } P = NP \implies \text{iO for circuits exists trivially}$$

Simply let

$$\mathsf{Obf}(C) = \text{ minimal circuit} \hat{C} \text{ for } C$$

this can be computed in poly-time if $P = NP$. This puts iO in another category than other regular crypto schemes since it is more likely to exists if the complexity class collapses. On the other hand this version of iO is uninteresting, since if $P = NP$ we do not have any other crypto schemes. Hence, we really want iO to exist in the world where $NP \neq P$.

Another equivalent definition is similar to VBB but with a computationally unbounded simulator.

Definition 1

$$\forall PPT\ A, \exists S, negl\ \epsilon,\ s.t.$$
$$\forall C, |\Pr[A(\mathsf{Obf}(C, 1^\lambda)) = 1] - \Pr[S^C(1^{|C|}, 1^\lambda) = 1]| < \epsilon(\lambda)$$

The point is that if the simulator is unbounded, then it can query the entire circuit for itself and learn what the function is, making it easy to simulate the view of the adversary.

For impossibility from last time, S can query C on all points and then learn α, β, γ. This indicate that there is no obvious impossibilities for the existence of iO.

3 What Use is iO

It is unclear what the use of iO is in terms of software protection. Specifically, in order to invoke the security definition of iO, we need to exhibit an equivalent program. If all equivalent programs actually reveals the information we are trying to hide, then iO does not give us anything. Nevertheless, we can show that iO is the "best possible" obfuscation.

Theorem 2 *(Informal) if there exists some obfuscator* Obf *satisfying some security, then iO should as well.*

Let iO denote the actual algorithm that computes the obfuscation.

$$iO(C, 1^\lambda) \approx_c iO(\mathsf{Obf}(C, 1^\lambda), 1^\lambda)$$

simply by definition. Note that on the RHS iO is post-processing of Obf and should not reduce security. A slight caveat is that $\mathsf{Obf}(C, 1^\lambda)$ and C might not have the same size, therefore, we must pad C in some trivial way on the LHS to be as large as $\mathsf{Obf}(C, 1^\lambda)$ since iO requires same-sized circuit.

The challenging question then is then understanding what circuits can be VBB obfuscated.

4 Application

Despite the fact that iO might seem useless since we need to come up with an equivalent circuit, there is something interesting that we can do with iO.

Definition 3 *public key encryption from OWF to iO.*

$$sk = s \in \{0,1\}^n$$

$$pk = G(s) \in \{0,1\}^{2n}$$

$$\mathsf{Enc}(pk, m) : P_{pk,m}(s') = \begin{cases} m, & \text{if } G(s') = pk \\ 0, & \text{else} \end{cases}$$

$$\text{Output } iO(P_{pk,m})$$

$$\mathsf{Dec}(sk, c) : \quad \text{output } C(sk)$$

Security(informal): hybrid experiment

- $H_0(pk = G(s))$, $c = iO(P_{pk,m_0})$
- $H_1 : pk \leftarrow$ random, $c = iO(P_{pk,m_0})$
- $H_2 : pk \leftarrow$ random, $c = iO(P_{pk,m_1})$
- $H_3 : pk = G(s)$, $c = iO(P_{pk,m_1})$

Indistinguishability between H_0, H_1 is simply due to pseudorandomness(PRG security). Indistinguishability between H_1, H_2 is due to the fact that if pk is random, then with high probability that there is no s such that $G(s) = pk$. Therefore w.h.p, P_{pk,m_0} is equivalent to P_{pk,m_1}(both would output 0 everywhere). Then by iO, H_1 and H_2 are indistinguishable.

5 Construction (cont. next time)

Definition 4 *A matrix branching program consists of* $\{B_{i,b}\}_{i\in[l],b\in\{0,1\}}$, $B \in \mathbb{Z}_q^{w\times w}$, $s \in \mathbb{Z}_q^{1\times w}$, $t \in \mathbb{Z}_q^{w\times 1}$. *We call* l *the length of the program and* w *the width of the program.* $inp : [l] \to [n]$ *where* n *is the number of input bits.*

$$Eval(x) := s \cdot \Pi_{i=1}^{l} B_{i,x_{inp(i)}} \cdot t \stackrel{?}{=} 0$$

Note that for a constant w, we can evaluate a branching program in NC^1(log depth). Specifically, we can first do a local computation to select the $B_{i,b}$'s to multiply, and then compute the multiplication of these matrices by doing a binary tree of multiplication (recursively multiplying the right half and left half). Note that each matrix multiplication is constant size since we assumed the width is constant. We now give Barrington's theorem which essentially states that the converse is true as well.

Theorem 5 *(Barrington's theorem) Any circuit can be computed by a branching program of* $w = 5$ *and* $l = 4^{depth}$. *Note that* l *is polynomial for* NC^1.

Sketch of proof:

Let

$$B_{i,b} \in \{5 \times 5 \text{ permutation matrices } \}$$

for example,

$$(12345) \iff \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Construct BP where

$$\Pi_i B_{i,x_{inp(i)}} = \begin{cases} Id, & \text{if circuit } C(x) = 0 \\ (12345), & \text{if } C(x) = 1 \end{cases}$$

$$s = (0,1,0,0,0)\ t = (1,0,0,0,0)^T$$

$$s \cdot M \cdot t \text{ is entry at } (2,1)$$

We now do a recursive construction. Suppose that the last gate of C is a NOT gate, then we construct a BP' for C'

$$BP'(x) = \begin{cases} Id, & \text{if } C'(x) = 0 (C(x) = 1) \\ (12345), & \text{if } C'(x) = 1 (C(x) = 0) \end{cases}$$

$$(12345)^{-1} \cdot BP'(x) = \begin{cases} Id, & \text{if } C(x) = 0 \\ (12345)^{-1}, & \text{if } C(x) = 1 \end{cases}$$

We then use the fact that the permutation matrices of (12345) and $(12345)^{-1}$ are similar. This means that

$$(12345) = U \cdot (12345)^{-1} \cdot U^{-1}$$

we can then compute

$$U \cdot (12345)^{-1} \cdot BP'(x) \cdot U^{-1}$$

this gives us the right thing in both cases. Namely, when $C'(x)$ is 0, the U and U^{-1} cancel out and gives the identity, and when $C'(x) = 0$, the inverses gives us (12345). Hence we only need to absorb $U \cdot (12345)^{-1}$ and U^{-1} into the matrices of BP' and we finished the construction.

Now suppose that we have an AND gate in the end, namely, $C(x) = C_0(x) \wedge C_1(x)$, we then recursively construct BP_0 and BP_1 that compute the corresponding C_0 and C_1. By similarity transformations,

$$BP_0 = \begin{cases} Id, & \text{if circuit } C_0(x) = 0 \\ (12345), & \text{if } C_0(x) = 1 \end{cases}$$

$$BP_1 = \begin{cases} Id, & \text{if circuit } C_1(x) = 0 \\ V, & \text{if } C_1(x) = 1 \end{cases}$$

where V is some other 5-cycle. We now have

$$BP_0 \cdot BP_1 \cdot BP_0^{-1} \cdot BP_1^{-1}$$

Now if either $C_0(x)$ or $C_1(x)$ is 0, then the product is the identity by definition. We now just need to consider the case where $C_0(x) = C_1(x) = 1$, in this case the above product is $(12345)V(12345)^{-1}V^{-1}$, which is in commutator notations

$$[(12345), V]$$

We want this to map to (12345). We can do this as long as $[(12345), V]$ is similar to (12345), meaning that the commutator is a 5-cycle. This only exists for $w = 5$ and up due to solvability of the permutation groups.

6 Next Time

We will give a formula for iO:

- apply Barington
- plug BP into crypto tools which will allow for evaluating the program while hiding everything else
- bootstrapping to move from NC^1 to all circuits.