COS 597C: Recent Developments in Program Obfuscation — Princeton University
Lecture 21 (12/6/16)
Lecturer: Mark Zhandry — Scribe: Fermi Ma

Notes for Lecture 21

1 Overview

Today we'll talk about the difficulty of building obfuscation from other tools.

We'll have some idealized random model like the random oracle model or a random permutation oracle or constant degree multilinear maps.

We'll try to get a black box construction of iO from that object, and then we'll use that to get iO without that object. We'll also construct approximately correct iO. We can also build approximately correct, approximately secure PKE.

It turns out that with constant degree multilinear maps, you can construct iO, but it isn't done in a black box way. Generally, non-black box constructions are not as efficient. So black box impossibility results can be interpreted as it being difficult to construct something.

We also know that statistically secure approximate iO is impossible unless OWFs don't exist or PH collapses.

2 The Random Oracle Model

We'll have our iO scheme take as input some circuit C and then make queries to a random oracle in the sky. iO^O outputs some obfuscated circuit $\hat{C}^O$ that can make oracle queries.

We construct another obfuscation scheme iO' that takes as input C. Note that this scheme won't explicitly output a program, but rather a string that can be fed into a separate *Eval* program that will be used for evaluation. In its head, it's going to concoct a random oracle. But instead of actually writing down the oracle, it's going to come up with its oracle query answers on the fly. Then it records its given answer to make sure it's consistent with future queries. It runs the original obfuscator $C \to \mathsf{iO}^O$, and we just simulate O to handle oracle queries. Now we pick some polynomial-size parameter t that is sufficiently large. For $x = 1, \ldots, t$, it samples a random x_i from the domain of C. It runs $\hat{C}^O(x_i)$ and sets $L_i = \{(r_i, s_i)\}$ to be the query response set generated during this computation. Let $L = \cup L_i$, and finally output $(\hat{C}, L)$.

Let's see why this gives indistinguishability. Suppose $C_0 \equiv C_1$ and there is a PPT adversary A that distinguishes $(\hat{C_0}, L_0)$ from $(\hat{C_1}, L_1)$. We can use this to come up with a PPT B^O that breaks security of our original obfuscator, iO^O.

$B^O(\hat{C})$ simply does the following. For $i = 1, \dots, T$, it chooses a random x_i in the domain of $\hat{C}$, and then runs $\hat{C}^O(x_i)$ and sets $L_i = \{(r_i, s_i)\}$. Let $L = \cup_i L_i$. It outputs $A(\hat{C}, L)$ and breaks the security of the original obfuscator.

Evaluation is done as follows. $Eval(\hat{C}, L, x)$ runs $\hat{C}(x)$. On query r, if $(r, s) \in L$, we respond with s (in this case we're lucky). Otherwise, we answer randomly.

Assume $\hat{C}(x)$ makes unique queries (if we make a query a second time, we don't need to query again).

We can prove that this construction is a p approximately correct iO scheme. This just means that $Pr[iO(C)(x) = C(x)] \geq 1 - p$ for $x \leftarrow domain(C)$.

We prove correctness as follows. Consider running $\hat{C}(x)$ on a random x from the domain of C. Let Q be the set of queries made during obfuscation. We'll look at a query r in two possible cases. First, the easy case is to suppose $r \notin Q$. Let O be a random oracle consistent with Q. In our (the simulator's) view, O is actually drawn from this distribution. So $O(r)$ is just random, which corresponds to how the evaluation algorithm answers. The harder case is when $r \in Q$. Let q be the probability that r is queried by $\hat{C}(x)$ on a random x. If q is tiny (much smaller than t), then it doesn't matter if I don't have the answer, since with high probability I will be fine. If q is large, then there is a good enough probability that I picked it during the simulation phase of iO', so long as we set t large enough.

3 Constant Degree Multilinear Map Case

Now suppose we have an iO scheme that queries a multilinear map M. This obfuscator places random values α_i in M. It obfuscates a circuit C and produces $\hat{C}$ that can query M on degree d polynomials P, to which M responds with whether or not $P(\alpha_i) = 0$.

We can hope to apply the same strategy that we used in the random oracle case, but the problem is that now there is more structure. We'll start with that idea first. The construction is initially the same if we replace the O's with M's. The one thing we change is that we keep all the P_j such that $P_j(\overline{\alpha}) = 0$. Let L consist of the union of all the P_j's saved during the obfuscation. We output $(\hat{C}, L)$.

Security (indistinguishability) goes through in the same way as it did before.

Evaluation must be different, however, since we note the following example: if we query on the sum of two zero polynomials, it should be zero, but this sum polynomial likely won't have been stored from before.

We'll think of the vector space V of all degree d polynomials. The set of all polynomials W that evaluate to 0 will be a subspace. So to answer a query, we check whether or not the query lies in this subspace. Let W' be the space spanned by the polynomials in L. Given any other polynomial, we test if $p \in W'$. We claim that the dimension of W is polynomial. This is because we have constant degree d polynomials.

The problem case occurs when we evaluate on a random input in W but not in W', but if this happens with high enough probability, then W' would have had a higher dimension.

Note that this strongly relies on d being constant, since if it were allowed to grow all these arguments about parameters being polynomial would fail.

There are some limits of black box separations. $O(1)$ degree multilinear maps can be used to construct Compact Functional Encryption in a black box way. This is just functional encryption where the cyphertexts aren't too large. This is not trivial at all. We can use this to get iO in a not black box way.