COS 597C: Recent Developments in Program Obfuscation — Princeton University
Lecture 17 (11/17/16)
Lecturer: Mark Zhandry — Scribe: Fermi Ma

Notes for Lecture 17

Today: We will finish the proof of VBB security.

Last time we saw an obfuscator that seemed to block the strategies of attack we have on multilinear maps. But now we actually need to prove that it does this.

Just as before, the ideal scenario is that we can make some concrete assumption on multilinear maps and reduce solving that problem to breaking our obfuscator. Unfortunately, we don't know how to do that.

Instead, we'll prove security in a generic model that seems to capture known attack strategies.

When we obfuscate a program, we produce a bunch of plaintext elements we want to encode. So think of the obfuscation plaintexts as $a_1, \ldots, a_t$. These will be encoded at levels $s_1, \ldots, s_t$. The model is going to choose at random $r_1, \ldots, r_t \leftarrow R$ where R is a ring.

We can think of the model as keeping track of a table with columns for a_i, s_i, r_i and labels ℓ_i. Label ℓ_i will represent a_i, r_i at level s_i, so addition of $\ell_1 + \ell_2$ looks like $a_1 + a_2, r_1 + r_2$. Multiplication of $\ell_1 \times \ell_2$ is $a_1 a_2, a_1 r_1 + a_2 r_1 + r_1 r_2$.

The adversary interacts with the model as follows. The adversary first receives labels $\{\ell_i\}$. The adversary makes $+, \times$ requests on these labels and receives the labels of the outputs of these requests.

The adversary then makes isZero queries on labels, and these queries get responded to with a true/false based on whether or not the corresponding a_i and r_i's evaluate to 0.

We also have that at the end, the adversary can query on a polynomial $Q(\{m_i\})$, and if $Q(\{r_i\}) = 0$, the adversary wins.

If we want to prove VBB security, we need to be able to simulate the view of the adversary.

We call this model the MMap world. We can consider an alternate world called the Simulated World, where the adversary interacts with a simulator S that handles the same queries. S doesn't get to see any of the a_i's or r_i's. S just gets to interact with some program P where it submits x and receives $P(x)$.

We need to construct this simulator. The requirement is that this simulated world is indistinguishable from the honest Mmap world case to the adversary.

Suppose we have a branching program. We left and right multiply so we have $\alpha_{i,b_0,b_1} R_1^{-1} A'_{i,b_0,b_1} R_{i+1}$.

We are going to divide the integers from 1 to k into chunks. We have a straddling set system for each chunk which means that to union anything, you have to union an entire chunk.

We look at the top bit and the bottom bit from $\alpha_{i,b_0,b_1} R_1^{-1} A'_{i,b_0,b_1} R_{i+1}$ and those will correspond to bits in a chunk.

Two classes ago when we were discussing this construction, we were saying that the adversary can only really run the program on inputs and zero tests. What these sets force us to do is that if we add two elements come from inconsistent inputs, we'll have to fill out an entire chunk for that input.

Doing $+, \times$ is easy. If we do IsZero, we do $\ell_i \to C(\{a_i\})$. We can associate any label with a polynomial. So the difference is that in the original Mmap world, we associate ring elements. Here we associate formal variables a_i.

There is a lemma which we won't prove that says there is an efficient procedure that maps C into $T \subseteq \{0,1\}^n$. Also $C = \sum_{x \in T}$. $|T|$ will be polynomial sized and C_x are polynomials in variables consistent with x. It will be consistent for all x for which those two bits match b_0, b_1. So this breaks C into polynomially many smaller polynomials C_x.

It turns out these polynomials may not correspond to an honest execution of the program. But we will show in that case that they will not be useful to the adversary.

We can rewrite c_x as $\prod_i \alpha_{i,x_{inp_0(i)},x_{inp_1(1)}} c'_x$. So c'_x is polynomial in $(R_i^{-1} A_{i,b_0,b_1}, R_{i+1})$.

One change we make now is we change $R_i{-}1$ to $R_i^{adjoint}$. This is just the inverse with no determinant factored out and randomness means that we don't have to worry about this.

Suppose $C = 0$ with non negligible probability. By the schwartz Zippel lemma it must be that C is actually equivalent to 0 when I think of C as a polynomial over $\alpha_{i,b_0,b_1} R_i$.

If C is 0 then every term in the sum is identically zero since terms cannot cancel out since the $\alpha_{i,x_{inp_0(i)},x_{inp_1(1)}}$ makes things linearly independent.

So to see whether or not C evaluates to 0 on the obfuscated randomness, it suffices to determine if C'_x is identically 0 for all $x \in T$.

The BGKPS approach for doing this is to look at the case of a single input branching program.

So we have $R_0^{-1} A_{0b} R_1, R_1^{-1} A_{1b} R_2, \ldots,$ for $b = 0, 1$.

When i consider C'_x, I'm considering one path through these matrices, so just one matrix from each column. After I consider the first i columns, we have something times R_i and the R_i masks everything.

Let $B_{i,x_{inp(i)}}$ be the whole block containing $A_{i,x_{inp(i)}}$. Then $B_{i,x_{inp(i)}}$ is uniformly random conditioned on $\hat{s}\prod B_{i,x_{inp(i)}}\hat{t} = s\prod A_{i,x_{inp(i)}}t$.

(See board to fill in details here).

But this proof will not completely work. It is not sufficient to just answer isZero queries. We need to make sure they correspond to honest executions.

What I need is that if C_x is not an honest program evaluation, then $C_x \neq 0$ with overwhelming probability.

Suppose that $s\prod_{i=1}^{\ell'<\ell} A_{i,x_{inp_0(i)},x_{inp_1(i)}} = 0$.

I need to impose a technical condition that all these partial products are nonzero. We need something nonshortcutting, so if we take a product minus the bookends it needs to be nonzero. For all x, we need $s\prod_{i=1}^{\ell} A'_{i,x_{inp_0(i)},x_{inp_1(i)}} \neq 0$ and $\prod_{i=1}^{\ell} A'_{i,x_{inp_0(i)},x_{inp_1(i)}}t \neq 0$.

We claim that if the matrices are full rank, this implies non-shortcutting.

To get these requirements we can append things to the diagonal (see board pictures). We need S_{i,b_0,b_1} to be a random matrix and not a scalar so the branching program doesn't get annihilated.

Proof Sketch.

C'_x is expanded out into a giant sum of monomials. Each monomial in C'_x involves exactly 1 element from each $B_{i,x_{inp_0(i)},x_{inp_1(i)}}$.

We're going to consider a generic C_x. This means we think of each C_x as $\sum \beta_m m$ where m is summed over the monomials.

Now I expand the monomials in terms of $\{R_i\}$.

Now $C_x = \sum B_x poly(R_i)$.

Basically what we can show is that C_x is not identically zero unless this polynomial corresponds to the correct matrix branching product. We can solve a linear system for the B_x and we see that what we have is an honest execution of the branching program.

We assume $C_x = 0$. Linear system in B_x.

To test if C_x is equivalently 0, first query on $P(x)$. If $P(x) = 0$, non-zero. Regardless of what the adversary does, any polynomial is going to be nonzero. Else, test $C(x)$ on random values such that their product is 0.

If $C(x) = 0$, it is zero. Else, non-zero.

This along with what we showed last time tells us that we can't annihilate this program.