COS 533: Advanced Cryptography — Princeton University
Lecture 19 (April 13, 2021)
Lecturer: Mark Zhandry — Scribe: Alan Chung

Notes for Lecture 19

1 Recap from Last Lecture

Last time, we saw how to convert any function in NC^1 to any matrix branching program. We recall the definition below:

Definition 1 (Matrix Branching Program) *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. The program takes some function* $inp : [l] \to [n]$, *where* n *is the number of input bits. Then, the program is evaluated as*

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

where x *is some input bit string, and we want to know whether this product is* $0 \pmod q$. *The input function will determine which matrix we choose from each column, which is represented by a bit in* x.

2 Re-randomization

Suppose that we have l columns of matrices $\{A_{ib}\}_{i\in[l],b\in\{0,1\}}$, and each matrix is in $\mathbb{R}^{w\times w}$. As in the definition above, suppose that the beginning vector is s^T, and the end vector is t. Then, the scheme is as follows:

1. Choose $l+1$ random invertible matrices in $\mathbb{Z}_q^{w\times w}$. Note that for large q, a random matrix is invertible with high probability. To see this, look at the determinant of a random matrix, and it will be nonzero with overwhelming probability.
2. Map $(\{A_{ib}\}, s, t, inp) \to (\{R_{i-1}^{-1} A_{ib} R_i\}, s^T R_0, R_l^{-1} t, inp)$.

Intuitively, we generate these random matrices to "mask" the original matrices. We note that in the product, the R's will cancel out, so that the iterated product is unchanged, which implies that the functionality of the program will be unchanged.

For now, we will assume that *inp* is independent of the program, but it will depend on the program size, number of input bits, etc. Actually, this can be assumed WLOG by inserting "dummy columns" of identity matrices. Considering the following: suppose we have some Matrix Branching scheme with 4 columns, and $ind(1) = 1, ind(2) = 2, ind(3) = 4, ind(4) = 2$, meaning that for the first column, to choose which matrix we use, take $x_1 \in \{0, 1\}$. Instead of *ind*, we can define some othe function ind' so that $ind'(1) = 1$, $ind'(2) = 2$, $ind'(3) = 3$, $ind'(4) = 4$, $ind'(5) = 1$, $ind'(6) = 2$, $ind'(7) = 3$, $ind'(8) = 4$. In other words, we have that the original subsequence 1242 as a substring of the new sequence 12341234, with new numbers inserted at indices 3, 5, 7, 8, where indexing starts at 1. Then we can convert the old Matrix Branching Process into a new one, where at for the columns 3, 5, 7, 8, we can just let the matrices be the identity matrix. This will not change the functionality of the program, but the point is that the *ind* function does not need to depend on the specific program, since we can modify the program accordingly to have the same functionality.

3 Why is Re-Randomization Useful?

It turns out that this gives a garbled circuit for NC^1 computations. We can write the labels as

$$L_{ib} = (A_{jb} \text{ for } j \text{ with inp(j) =i}), s, t.$$

Then for $x \in \{0, 1\}^n$, $\{L_{ix_i}\}$ allows for computing BP(x). Because of the multiplication by random matrices, it turns out that $\{L_{i,x_i}\}, s, t$ are uniformly random, conditioned that the product

$$s^T \prod_{i=1}^{l} L_{i,x_{inp(i)}} \cdot t = \begin{cases} 0 & \text{if } BP(x) = 1 \\ 1 & \text{if } BP(x) = 0 \end{cases}$$

This actually gives an information-theoretic garbled circuit. This is sort of like a 1-time secure obfuscation, which we will show implies a many-time secure scheme.

4 One-time ⇒ Many-time

Question: What if we just give out all the labels? Suppose we converted the original program into a Matrix Branching program, and possibly added "dummy matrices" as above to hide any information that the *ind* function gave. Then, we do this randomization. Then, you could compute the program on all inputs, but the question is: what kind of security could it provide? The answer is, it wouldn't provide full security, but there are only a couple of ways we know how to attack it.

Consider some reasonable attacks. Define $\hat{A}_{ib} = R_{i-1}^{-1} A_{ib} R_i$. Then we can do the following:

1. Consider $\hat{A}_{i0}\hat{A}_{i1}^{-1} = R_{i-1}^{-1} A_{i0} A_{i1}^{-1} R_i$, so this LHS is a matrix that is similar to $A_{i0}A_{i1}^{-1}$, and similar matrices have the same eigenvalues. So now if A_{i0} and A_{i1} are dummy matrices, then $A_{i0}A_{i1}^{-1} = I$. Otherwise, if A_{i0}, A_{i1} were not dummy matrices, then $A_{i0}A_{i1}^{-1} \neq I$ with high probability. In particular, for the matrices given by Barrington's Theorem, the matrix will be a permutation matrix, and its eigenvalues will be the primitive roots of unity. Then, we can figure out where the dummy matrices are, which could reveal information about the program.

2. "Mixed input attacks:" Suppose that we have some Matrix Branching program with start vector s^T, end t, and matrices M_{ib} for $i \in [4]$. Now suppose that the input string is 1242, meaning that we are supposed to multiply the matrices M_{1x_1}, M_{2x_2}, M_{3x_4}, M_{4x_2}, where $x = 0011$. But suppose we cheat, and instead take M_{10}, M_{20}, M_{31}, M_{41} (the point is that we were supposed to choose the index 0 matrices for rows 2 and 4, but we choose the 0 and 1 matrices instead). But if we do this, we are learning something about the program that doesn't exactly correspond to a way that you are supposed to evaluate the program, which can yield some information.

It turns out that our techniques to block these attacks will essentially block all attacks.

5 Multilinear Maps

We consider a generalization of bilinear maps. Suppose that we have some value l, which we call the "multilinearity." Then $\forall S \in [l]$, we have some group $\mathbb{G}_S$, with some generator g_s, so $g_s^a g_s^b = g_s^{a+b}$. Now suppose we have some map

$$e : \mathbb{G}_S \times \mathbb{G}_T \to G_{S \cup T}, \quad e(g_S^a, g_T^b) = g_{S\cup T}^{ab},$$

where above $S \cap T = \emptyset$. We call this an asymmetric multilinear map. Previously, we had $l = 2$, corresponding to a bilinear maps, and we had some pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}' = \mathbb{G}_{1,2}$.

5.1 First attempt using Multilinear Maps for Obfuscation

So suppose we have a matrix branching program, $\{A_{ib}\}_{i\in[l],b\in\{0,1\}}$, where we assume that this has been re-randomized and inserted with dummy matrices to hide information about *ind*. Now for each column, we associate column i with the group $\mathbb{G}_i$ defined as above, and for this group, suppose we choose some generator g_i. Now, we

replace the existing matrices with matrix exponentiation. Concretely, for column i, we have matrices A_{i0} and A_{i1}, then we replace then with $g^{A_{i0}}$ and $g^{A_{i1}}$, where

$$g_s^A = \begin{bmatrix} g_S^{A_{11}} & g_s^{A_{12}} & \dots & g_S^{A_{1n}} \\ g_S^{A_{21}} & g_s^{A_{22}} & \dots & g_S^{A_{2n}} \\ \vdots & \vdots & \dots & \vdots \\ g_S^{A_{n1}} & g_s^{A_{n2}} & \dots & g_S^{A_{nn}} \end{bmatrix}.$$

Now, the idea is that the pairing operation allows us to compute $g_{[l]}^{s \cdot \prod_j A_{jx_{\text{inp}(j)}} \cdot t}$. Then we can test whether $s \cdot \prod_j A_{jx_{\text{inp}(j)}} \cdot t = 0$ or not by seeing whether the result is the identity $1 \in \mathbb{G}_{[l]}$.

Then for the attacks mentioned above, it seems that for attack (1), we can still test if $A_{i0} = A_{i1}$, and we can still (2) perform mixed-input attacks. But actually, it turns out that we can do some more sophisticated things. For attack (1), consider computing

$$g_j^{\alpha_{jb} R_{j-1}^{-1} A_{jb} R_j},$$

i.e. we introduce this random scalar α_{jb}. It turns out that this prevents (1), because now the top and bottom get different output α_{jb}, so we can't just cmpare the matrices in the exponent to tell them apart. However, multiplying by a scalar doesn't change whether the iterated product is 0.

To block (2), there are some different approaches. One idea is to use more sophisticated index sets for the matrices. Rather than doing

$$g_j^{\hat{A}_{jb}} \to g_{S_{jb}}^{\hat{A}_{jb}},$$

so we change j into a more "complex" index set S_{jb}. The point is that these sets prevent mixing input. A second idea is to take the matrices A_{jb}, and prior to performing the rerandomization, you turn them into a "Block Diagonal" of the form

$$\begin{bmatrix} A_{jb} & 0 \\ 0 & B_{jb} \end{bmatrix}.$$

Then the matrix product is the product of the individual blocks, and we can use the B to "enforce consistency."

Theorem 2 *If the multilinear map is "ideal," meaning the only allowed operations are the allowed operations by the multilinear map, then one can instantiate the system so that the scheme is secure in the iO sense.*

Actually, we get VBB security, which would imply that ideal multilinear maps don't exist.

6 Can we build Multilinear maps?

For the rest of this lecture, we will consider this long-open problem. We recall that actually elliptic curves give a way to construct bilinear maps ($l = 2$). It turns out that there is no obvious generalization. What we do have is what's referred to as "noisy" multilinear maps, which are based on lattice ideas. The rough idea is that we start from an FHE scheme. Note that we could encrypt all of the matrices $\hat{A}_{ib}$ under the public key, so

$$Enc(pk, \hat{A}_{ib}) \Rightarrow Enc(pk, s \cdot \prod A_{j,x_{\text{inp}(j)}} \cdot t),$$

i.e. if we can encrypt all of the individual matrices, we can compute the encryption of the iterated product by using the fully homomorphic properties of the encryption scheme. However, we can't evaluate whether the product is 0 or not, since it is encrypted. But, we can try to "break" it in a way to do zero testing, i.e. find out if the product is 0. However, a problem is that it turns out that there are a bunch of nontrivial attacks on these schemes. The current status is that we've figured out how to handle the attacks, but it is pretty unsatisfactory right now; we have some understanding on how to defend the scheme, but we don't have any great security proofs.

7 Another approach to Program Obfuscation

It turns out that FE $\Rightarrow iO$. The idea is that we can give out an encryption $Enc(pk, C)$, where C is the program you want to evaluate. Then, we can give out the secret keys $sk_{f_{i,b}}$ for the functions $f_{i,b}(C) = Enc(pk, C(b, \cdot))$ for $b = 0, 1$ (so the bit b is hardcoded into the circuit). What this allows you to do is to go from $Enc(pk, C) \rightarrow Enc(pk, C(x))$ by gradually modifying the string x. But now, at the end, you can have the value under f in the clear.

This leads to a problem, which is how to construct the FE in the first place. One solution is to use iO (note this is circular). The idea is that you can start with a PKE, and to give out the secret key for a particular function, you would create the program that decrypts the ciphertext and then applies the function to the plaintext and then you would obfuscate the program and output that as your secret key. It turns out that using an l-degree multilinear map, we can directly construct FE for "degree l computations." Then in this FE to IO transformation, it is basically enough to have FE for PRGs in order to do this bootstrapping strategy to compile FE into an IO scheme.

So you can show that if you have a degree l multilinear map, and PRGs that are degree l, you can get iO. The problem, of course, is that for PRGs, we have conjectured PRGs

of degree 5, but it is impossible with lower degree, so we can use 5-linear maps to get iO. There is a very recent breakthrough work that showed that LWE + constant depth PRGs+ pairings + LPN implies iO.