CSG399: Gems of Theoretical Computer Science — Lectures 13-14. Feb. 20-24, 2009
Instructor: Emanuele Viola — Scribe: Eric Miles

Cryptography in constant depth: II & III

1 Locally computable randomized encodings

In the last lecture, we saw that a randomized encoding of a one-way function is itself also one-way. In this lecture, we will exhibit randomized encodings which are "locally" computable, in the sense that each output bit depends on a (small) constant number of input bits. We start by recalling the definition.

Definition 1. *A* randomized encoding *of a function $f : \{0,1\}^n \to \{0,1\}^n$ with* blow-up t *is a function $f' : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^t$ for which the following conditions hold:*

1. $\forall\ x, x', r, r' :\ f(x) \neq f(x') \implies f'(x,r) \neq f'(x',r')$.
2. *There exists a distribution D on circuits C of size t such that for any fixed string x, the distribution $C(f(x))$ (for $C \in D$) is identical to the distribution $f'(x,r)$ (for $r \in U_t$).*

Without loss of generality, we will focus on randomized encodings of functions $f : \{0,1\}^n \to \{0,1\}$, whose output is a single bit; the encodings can be concatenated to handle the more general case.

1.1 An encoding via Barrington's theorem

Let $f : \{0,1\}^n \to \{0,1\}$ be computable by an $O(\log n)$-depth circuit. Then, as seen in Lecture 11, Barrington's theorem gives a group program $[(g_1^0, \ldots, g_\ell^0), (g_1^1, \ldots, g_\ell^1), (k_1, \ldots, k_\ell)] \in (S_5)^\ell \times (S_5)^\ell \times [n]^\ell$ which α-computes f. That is,

$$\forall x \in \{0,1\}^n :\ \prod_{i=1}^{\ell} g_i^{x_{k_i}} = \alpha^{f(x)}$$

Note that S_5 is the permutation group on five elements, and α is any cycle in S_5.

Our first attempt at transforming this group program into a randomized encoding of f is to define $f'(x) := (g_1^{x_{k_1}}, \ldots, g_l^{x_{k_l}})$. We can check if the necessary properties are satisfied:

- It's locally computable, because each element in the output depends on one bit of x.
- $f(x) \neq f(x') \Rightarrow f'(x) \neq f'(x')$
- Unfortunately, we cannot hope to find a distribution D of circuits C such that $C(f(x)) \equiv f'(x)$, because the input to each circuit is only a single bit.

Notice that in the preceding construction, the encoding function f' has no random input in addition to the string x. The problem we encountered can be fixed with the use of randomness. Let $r = (r_1, \ldots, r_{\ell-1}) \in (S_5)^{\ell-1}$ be a set of random group elements. Then, define the randomized encoding $f' : \{0,1\}^n \times (S_5)^{\ell-1} \to (S_5)^\ell$ as

$$f'(x, r) := \left(g_1^{x_{k_1}} \cdot r_1,\ \ (r_1)^{-1} \cdot g_2^{x_{k_2}} \cdot r_2,\ \ \ldots\ \ ,\ \ (r_{\ell-2})^{-1} \cdot g_{\ell-1}^{x_{k_{\ell-1}}} \cdot r_{\ell-1},\ \ (r_{\ell-1})^{-1} \cdot g_\ell^{x_{k_\ell}} \right)$$

We can again check the necessary properties, and this time they are all satisfied:

- It's locally computable, as each output element depends on a fixed-size portion of the input.
- For a fixed x and any $r \in (S_5)^{\ell-1}$, the group product of the elements in $f'(x, r)$ is equal to $\alpha^{f(x)}$ by construction. Thus, $f(x) \neq f(x') \Rightarrow \forall r, r' :\ f'(x, r) \neq f'(x', r')$.
- For a fixed x, the first $\ell - 1$ elements of $f'(x, r)$ are independently distributed over S_5 (because they use "fresh" randomness), and the last is dependent. So, we define a distribution D over circuits by again letting $r \in (S_5)^{\ell-1}$, and defining $C_r(f(x))$ to be the circuit which outputs $\left(r_1, \ldots, r_{\ell-1}, \left(\prod_{i=1}^{\ell-1} r_i \right)^{-1} \alpha^{f(x)} \right)$. It is easy to see that for every fixed x, the distribution $C_r(f(x))$ is identical to $f'(x, r)$.

One drawback of the above construction is that our randomized encoding f' does not operate on bits, as the size of S_5 is not a power of two. (In fact, no group with size which is a power of two will work for Barrington's theorem.) Next, we describe a randomized encoding which does operate on bits, and furthermore which begins with functions f computable by branching programs of width n (a less restrictive class than was considered in this section).

1.2 An encoding via matrices

Let $f : \{0,1\}^n \to \{0,1\}$ be a function computable by a branching program of width n and length $\text{poly}(n)$, and let $m + 1 = \text{poly}(n)$ be the number of nodes in the branching program. This program naturally defines a DAG on $m + 1$ nodes, denoted as G, with the following properties:

- there are two distinguished "start" and "accept" nodes
- each node is labeled with "x_i", for some $i \in [n]$
- each node has two outgoing edges labeled "0" and "1" (except those in the last layer of the branching program, which have none)

We fix a topological ordering φ on the $m + 1$ nodes such that $\varphi(\text{start}) = 0$, $\varphi(\text{accept}) = m$, and $u \rightsquigarrow v \Rightarrow \varphi(u) < \varphi(v)$. Given any string $x \in \{0,1\}^n$, define G_x to be the graph induced by x on G's nodes. Specifically, any node labeled x_i has only the outgoing edge labeled with the ith bit of x. Then, we can see that $f(x) = 1$ iff the accept node is reachable from the

start node in G_x. For technical reasons, we also define G_x to have a self-edge on every node.

Consider the adjacency matrix of G_x, which we denote as $M(x)$. With the nodes of G_x ordered according to φ, $M(x)$ is an upper-triangular matrix, with 1s along the diagonal, and at most one additional 1 per row (to the right of the diagonal):

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 1 & \cdots & 0 \\ 0 & 1 & 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 1 & 0 \\ \vdots & & & \ddots & & & \vdots \\ & & & & & 1 & 0 \\ 0 & 0 & & \cdots & & 0 & 1 \end{pmatrix}$$

Then, we define $L(x)$ to be $M(x)$ with the first column and last row removed:

$$\begin{pmatrix} 0 & 0 & \cdots & 1 & \cdots & 0 \\ 1 & 0 & 1 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 1 & 0 \\ \vdots & & \ddots & & & \vdots \\ 0 & & \cdots & & 1 & 0 \end{pmatrix}$$

Note that $L(x)$ is an $m \times m$ matrix. In a sense, $L(x)$ corresponds to taking one step from the start node: the column number of the 1 in the first row tells us to which node we stepped. In the remainder of this section, we will see how to transform $L(x)$ (through matrix multiplications) into a matrix which contains $f(x)$ in the upper right corner, and how this implies a randomized encoding of f. To this end, we define two families of matrices. All the matrices are over $\mathrm{GF}(2) = \{0, 1\}$, so addition is bit XOR.

$\mathbf{R}_1$: the family of upper triangular matrices with 1s along the diagonal.

$$\begin{pmatrix} 1 & * & \cdots & * \\ & 1 & \ddots & * \\ & & \ddots & \vdots \\ & 0 & & 1 & * \\ & & & & 1 \end{pmatrix}.$$

Multiplying a matrix M on the left with a matrix in R_1 corresponds to summing to each row of M a linear combination of the rows below it. For example,

$$\begin{pmatrix} 1 & a & b \\ 0 & 1 & c \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{pmatrix} = \begin{pmatrix} x_1 + ax_2 + bx_3 & y_1 + ay_2 + by_3 & z_1 + az_2 + bz_3 \\ x_2 + cx_3 & y_2 + cy_3 & z_2 + cz_3 \\ x_3 & y_3 & z_3 \end{pmatrix}.$$

$\mathbf{R}_2$: the family of matrices with 1s along the diagonal, and any other 1s in the last column.

$$\begin{pmatrix} 1 & 0 & \cdots & 0 & * \\ & 1 & \ddots & & * \\ & & \ddots & 0 & \vdots \\ & 0 & & 1 & * \\ & & & & 1 \end{pmatrix}.$$

Multiplying a matrix M on the right with a matrix in R_2 corresponds to summing to the last column of M a linear combination of the other columns. For example,

$$\begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{pmatrix} \begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} x_1 & y_1 & ax_1 + by_1 + z_1 \\ x_2 & y_2 & ax_2 + by_2 + z_2 \\ x_3 & y_3 & ax_3 + by_3 + z_3 \end{pmatrix}.$$

It is important to note that both R_1 and R_2 are algebraic groups, which is not difficult to see given the intuition of what types of operations multiplying by the matrices corresponds to. We now show how to transform $L(x)$ using these matrices.

Lemma 2. *For all $x \in \{0,1\}^n$, there exist matrices $r_1 \in R_1$ and $r_2 \in R_2$ such that*

$$r_1 \cdot L(x) \cdot r_2 = \begin{pmatrix} 0 & \cdots & 0 & f(x) \\ 1 & \ddots & & 0 \\ & \ddots & & \vdots \\ 0 & & 1 & 0 \end{pmatrix}$$

Proof. Let c be the column number of the 1 in the first row of $L(x)$ (numbering the columns 1 through m). Recall that this means c is the number of the node immediately following the start node in G_x. We will "push" this 1 to the right by repeatedly summing the lower rows to the first. To start, we sum row $c+1$ to row 1, which has two effects: **(i)** the entry at $(1, c)$ is zeroed, because we are working over GF(2) and row $c+1$ has a 1 in column c; **(ii)** the entry at $(1, d)$ is set to 1, where d is the node to which c points. We then sum row $d+1$ to row 1, and continue in this manner, ending when one of the following conditions holds:

- There is a 1 in entry $(1, m)$. This indicates that there is a path from the start node to the accept node in G_x, and thus that $f(x) = 1$.
- The first row consists of all 0s. This indicates that a non-accepting sink was reached, and thus that $f(x) = 0$.

A key observation here is that, in either case, the upper right corner has the value $f(x)$. Also, note that this process will always terminate because G_x is a DAG. We repeat the process for each of the lower rows, pushing the non-self-loop-1 to the right until it is in the rightmost

column or until the portion of the row to the right of the self-loop-1 is zeroed out. When we have done this for every row, we have a matrix in the following form:

$$\begin{pmatrix} 0 & \cdots & 0 & f(x) \\ 1 & \ddots & & * \\ & \ddots & & \vdots \\ & 0 & & * \\ & & 1 & 0 \end{pmatrix}$$

Each operation performed so far has been summing a row of $L(x)$ to a row above it; as previously mentioned, these operations can be performed by multiplying on the left by a matrix from R_1. Composing the multiplications then, we have shown that $\forall x \in \{0,1\}^n$: $\exists r_1 \in \mathrm{R}_1$ such that $r_1 \cdot L(x)$ has the above form. We now choose $r_2 \in \mathrm{R}_2$ to be the matrix which, when multiplied on the right, causes the first $m-1$ columns of $r_1 \cdot L(x)$ to be summed to the last column so as to zero out all entries below the top. That is, $r_2(i,m) := [r_1 \cdot L(x)]\,(i+1,m)$ for $1 \leq i < m$, and all other entries of r_2 are fixed from the definition of R_2. Then, $r_1 \cdot L(x) \cdot r_2$ has the form given in the statement of the lemma. □

This construction leads naturally to a randomized encoding. We note that exactly $m(m-1)/2$ bits and $m-1$ bits are needed to represent matrices in R_1 and R_2, respectively, and so we abuse notation by referring to matrices from these families as bit strings of the correct length.

Theorem 3. *Let $f : \{0,1\}^n \to \{0,1\}$ be computable by an $(m+1)$-size branching program, with $m = \mathrm{poly}(n)$. Then, the function $f' : \{0,1\}^n \times \{0,1\}^{m(m-1)/2} \times \{0,1\}^{m-1} \longrightarrow \{0,1\}^{m^2}$ defined by $f'(x, r_1, r_2) := r_1 \cdot L(x) \cdot r_2$ is a randomized encoding of f.*

Proof. We verify the two properties of a randomized encoding.

1. $f(x) \neq f(x') \Rightarrow f'(x, r_1, r_2) \neq f'(x', r_1', r_2')$.
   Fix x and x' such that $f(x) \neq f(x')$. Notice that each matrix from R_1 or R_2 has full rank. Therefore, due to Lemma 2 and the fact that a product of matrices has full rank iff each of the matrices do, we see that $L(x)$ has full rank iff $f(x) = 1$. Since exactly one of $L(x)$ and $L(x')$ has full rank, no choice of the r's can make the products equal.

2. *A distribution of circuits so that $C_{r_1,r_2}(f(x)) \equiv f'(x, r_1, r_2)$ for uniformly chosen r_1, r_2.*
   Fix a string x. Define $Z(x)$ to be the matrix on the right side of the equation in the statement of Lemma 2. That is, $Z(x)$ is the $m \times m$ identity matrix with the diagonal shifted "southwest" and $f(x)$ in the upper right corner. Define $C_{r_1,r_2}(f(x))$ to be the circuit which computes $r_1 \cdot Z(x) \cdot r_2$. Now, fix r_1, r_2; we need to give r_1', r_2' such that $C_{r_1',r_2'}(f(x)) = f'(x, r_1, r_2)$. Let $\hat{r_1}, \hat{r_2}$ be the matrices guaranteed by Lemma 2 such that $\hat{r_1} \cdot L(x) \cdot \hat{r_2} = Z(x)$. Then, using the fact that R_1 and R_2 are groups, the choices which satisfy the equation are $r_1' := r_1 \cdot (\hat{r_1})^{-1}$ and $r_2' := (\hat{r_2})^{-1} \cdot r_2$. The reverse case (fixing r_1', r_2' and finding r_1, r_2) is analogous.

□

1.3 A *locally computable* encoding via matrices

We now show how to transform the preceding randomized encoding into one in which each output bit depends on at most four input bits. Each bit of the randomized encoding from Theorem 3 is computable by a degree-3 polynomial, or more specifically, a sum of m^2 monomials of degree at most 3. So, applying the next theorem to each bit of Theorem 3's randomized encoding gives the desired result.

Theorem 4. *Let $f : \{0,1\}^n \to \{0,1\}$ be a degree-3 polynomial. Then, f has a randomized encoding with blow-up* $\mathrm{poly}(n)$ *in which each output bit depends on at most four input bits.*

Proof. We separate f into its degree-3 monomials: $f(x) := T_1(x) + \cdots + T_k(x)$, for some parameter $k = \mathrm{poly}(n)$. Let r, r' be randomly chosen bit strings with $|r| = k$, $|r'| = k-1$. We define a randomized encoding similarly to how Barrington's group program was randomized:

$$f' : \{0,1\}^n \times \{0,1\}^k \times \{0,1\}^{k-1} \longrightarrow \{0,1\}^{2k}$$

$$\begin{aligned} f'(x,r,r') \quad := \quad & (T_1(x) + r_1, \quad T_2(x) + r_2, \quad \ldots, \quad T_{k-1}(x) + r_{k-1}, \quad T_k(x) + r_k, \\ & \quad r_1 + r'_1, \quad r'_1 + r_2 + r'_2, \quad \ldots, \quad r'_{k-2} + r_{k-1} + r'_{k-1}, \quad r'_{k-1} + r_k) \end{aligned}$$

If we sum all the bits of $f'(x,r,r')$, each bit of r and r' appears exactly twice, and each $T_i(x)$ appears exactly once. Therefore, because we're working over GF(2), summing the bits of $f'(x,r,r')$ will give back $f(x)$ for any r, r', and so the first condition of a randomized encoding holds. For the second condition, note that for any fixed x, the first $2k-1$ bits of $f'(x,r,r')$ are uniformly distributed (for uniform r, r') and the last bit is the one which makes the parity of the entire output equal to $f(x)$. Thus, our distribution is over circuits C which output $2k-1$ bits at random, and then one final bit to ensure the parity is equal to $f(x)$; recall that we can do this because $f(x)$ is the sole input to C. Finally, each output bit of f' depends on at most four input bits because each $T_i(x)$ is a degree-3 monomial. □

2 Notation and Terminology

Here we briefly review the standard terminology for the classes of circuits we have seen in the last several lectures.

- $\mathbf{NC}^k$: the class of circuits with fan-in 2, depth $O(\log^k n)$ and size $\mathrm{poly}(n)$
- $\mathbf{AC}^k$: the class of circuits with unbounded fan-in, depth $O(\log^k n)$ and size $\mathrm{poly}(n)$

$\mathbf{NC}^0$ is the class of fan-in 2 constant-depth circuits. We have just seen that there exist one-way functions computable by this class if there exist one-way functions computable by polynomial-size branching programs. In particular if factoring is hard, $\mathbf{NC}^0$ contains one-way functions, because multiplication is computable by log-depth circuits (and thus by polynomial-size BPs). $\mathbf{AC}^0$ is the class of polynomial-size constant-depth circuits with unbounded fan-in. We have seen previously that parity cannot be computed by circuits in $\mathbf{AC}^0$.

For any $k \geq 0$, the following relationship holds:

$$\mathbf{NC}^k \subseteq \mathbf{AC}^k \subseteq \mathbf{NC}^{k+1}$$

Strict separations are only known in two instances. It is easy to see that $\mathbf{NC}^0 \subsetneq \mathbf{AC}^0$, because $\mathbf{NC}^0$ circuits are severely restricted by the fact that each output bit can only depend on a constant number of input bits. (For instance, an n-way OR is not computable by $\mathbf{NC}^0$ circuits as n goes to infinity.) We also know that $\mathbf{AC}^0 \subsetneq \mathbf{NC}^1$, because parity is easily computable by an $\mathbf{NC}^1$ circuit. Additionally, the class of functions computable by constant-width polynomial-length branching programs is sandwiched between $\mathbf{NC}^1$ and $\mathbf{AC}^1$; Barrington's theorem showed that $\mathbf{NC}^1$ functions are computable by these BPs, and any such BP can be transformed into an $\mathbf{AC}^1$ circuit. In general, moving from right to left along the chain of inclusions results in functions which are faster and more parallelizable.

Finally, we note that the circuits discussed have been of the *non-uniform* variety. A non-uniform family of circuits is one in which the form of each circuit can vary arbitrarily based on the input length. On the other hand, a *uniform* family of circuits is one for which there exists a single algorithm that, when given 1^n as input, constructs the circuit which operates on inputs of length n. The space and time requirements on this construction algorithm can vary to give different flavors of uniformity; some common choices are poly-time and log-space.