Cryptography in NC^0

(EXTENDED ABSTRACT)*

Benny Applebaum Yuval Ishai Eyal Kushilevitz

Computer Science Department, Technion
{abenny,yuvali,eyalk}@cs.technion.ac.il

Abstract

We study the parallel time-complexity of basic cryptographic primitives such as one-way functions (OWFs) and pseudorandom generators (PRGs). Specifically, we study the possibility of computing instances of these primitives by NC^0 circuits, in which each output bit depends on a constant number of input bits. Despite previous efforts in this direction, there has been no significant theoretical evidence supporting this possibility, which was posed as an open question in several previous works.

We essentially settle this question by providing overwhelming positive evidence for the possibility of cryptography in NC^0. Our main result is that every "moderately easy" OWF (resp., PRG), say computable in NC^1, can be compiled into a corresponding OWF (resp., low-stretch PRG) in NC^0_4, i.e. whose output bits each depend on at most 4 input bits. The existence of OWF and PRG in NC^1 is a relatively mild assumption, implied by most number-theoretic or algebraic intractability assumptions commonly used in cryptography. Hence, the existence of OWF and PRG in NC^0 follows from a variety of standard assumptions. A similar compiler can also be obtained for other cryptographic primitives such as one-way permutations, encryption, commitment, and collision-resistant hashing.

The above results leave a small gap between the possibility of cryptography in NC^0_4 and the known impossibility of implementing even OWF in NC^0_2. We partially close this gap by providing evidence for the existence of OWF in NC^0_3.

Finally, our techniques can also be applied to obtain unconditionally provable constructions of non-cryptographic PRGs. In particular, we obtain ϵ-biased generators in NC^0_3, resolving an open question posed by Mossel et al. [25], as well as a PRG for logspace in NC^0.

Our results make use of the machinery of randomizing polynomials *[19], which was originally motivated by questions in the domain of information-theoretic secure multiparty computation.*

* Supported by grant no. 36/03 from the Israel Science Foundation.

1. Introduction

The efficiency of cryptographic primitives is of both theoretical and practical interest. In this work, we consider the question of minimizing the *parallel time-complexity* of basic cryptographic primitives such as one-way functions (OWFs) and pseudorandom generators (PRGs) [7, 33]. Taking this question to an extreme, it is natural to ask if there are instances of these primitives that can be computed in *constant* parallel time. Specifically, the following fundamental question was posed in several previous works (e.g., [15, 11, 9, 23, 25]):

> Are there one-way functions, or even pseudorandom generators, in NC^0?

Recall that NC^0 is the class of functions which can be computed by (a uniform family of) constant-depth circuits with bounded fan-in. In an NC^0 function each bit of the output depends on a constant number of input bits. We refer to this constant as the *output locality* of the function and denote by NC^0_c the class of NC^0 functions with locality c.

The above question is qualitatively interesting, since one might be tempted to conjecture that cryptographic hardness requires some output bits to depend on many input bits. Indeed, this view is advocated by Cryan and Miltersen [9], whereas Goldreich [11] takes an opposite view and suggests a concrete candidate for OWF in NC^0. However, despite previous efforts, there has been no significant theoretical evidence supporting either a positive or a negative resolution of this question.

1.1. Previous Work

Linial et al. show that pseudorandom *functions* cannot be computed even in AC^0 [24]. However, no such impossibility result is known for PRGs. The existence of PRGs in NC^0 has been recently studied in [9, 25]. Cryan and Miltersen [9] observe that there is no PRG in NC^0_2, and prove that there is no PRG in NC^0_3 achieving a superlinear stretch;

namely, one that stretches n bits to $n + \omega(n)$ bits.[1] Mossel et al. [25] extend this impossibility to NC^0_4. Viola [31] shows that an AC^0 PRG with superlinear stretch cannot be obtained from a OWF via non-adaptive black-box constructions. Negative results for other restricted computation models appear in [10, 35].

On the positive side, Impagliazzo and Naor [18] construct a (sublinear-stretch) PRG in AC^0, relying on an intractability assumption related to the subset-sum problem. PRG candidates in NC^1 (or even TC^0) are more abundant, and can be based on a variety of standard cryptographic assumptions including ones related to the intractability of factoring [29, 13, 21], discrete logarithms [7, 33, 27] and lattice problems [2, 16].[2]

Unlike the case of pseudorandom generators, the question of one-way functions in NC^0 is relatively unexplored. The impossibility of OWFs in NC^0_2 follows from the easiness of 2-SAT [11, 9]. Håstad [15] constructed a family of permutations in NC^0 whose inverses are P-hard to compute. Cryan and Miltersen [9], improving on [1], presented a circuit family in NC^0_3 whose range decision problem is NP-complete. This, however, gives no evidence of cryptographic strength. Since any PRG is also a OWF, all PRG candidates cited above are also OWF candidates. (In fact, the one-wayness of an NC^1 function often serves as the underlying cryptographic *assumption*.) Finally, Goldreich [11] suggested a candidate OWF in NC^0, whose conjectured security does not follow from any well-known assumption.

1.2. Our Results

As indicated above, the possibility of implementing most cryptographic primitives in NC^0 was left wide open. We present a positive answer to this basic question, showing that surprisingly many cryptographic tasks can be performed in constant parallel time.

Since the existence of cryptographic primitives implies that $\mathrm{P} \neq \mathrm{NP}$, we cannot expect unconditional results and have to rely on some unproven assumptions.[3] However, we avoid relying on *specific* intractability assumptions. Instead, we assume the existence of cryptographic primitives in a relatively "high" complexity class and transform them to the seemingly degenerate complexity class NC^0 without substantial loss of their cryptographic strength. These transformations are inherently non-black-box, thus providing further evidence for the usefulness of non-black-box techniques in cryptography.

1 From here on, we use a crude classification of PRGs into ones having sublinear, linear, or superlinear additive stretch. Note that a PRG stretching its seed by just one bit can be invoked *in parallel* to yield a PRG stretching its seed by $n^{1-\epsilon}$ bits, for an arbitrary $\epsilon > 0$.

2 In some of these constructions it seems necessary to allow a *collection* of NC^1 PRGs, and use polynomial-time preprocessing to pick (once and for all) a random instance from this collection. This is similar to the more standard notion of OWF collection (cf. [12], Section 2.4.2).

3 This is not the case for non-cryptographic PRGs such as ϵ-biased or logspace generators, for which we do obtain unconditional results.

An overview of the main ideas used for obtaining these results appears in Section 2. The reader might want to skip to that section before moving on to the following, more detailed, account of results.

A GENERAL COMPILER. Our main result is that any OWF (resp., PRG) in a relatively high complexity class, containing uniform NC^1 and even $\oplus\mathrm{L}/poly$, can be efficiently "compiled" into a corresponding OWF (resp., PRG) in NC^0_4. (The class $\oplus\mathrm{L}/poly$ contains $\mathrm{L}/poly$ and NC^1 and is contained in NC^2. In a non-uniform setting it also contains $\mathrm{NL}/poly$ [32].) The existence of OWF and PRG in this class is a mild assumption, implied in particular by most number-theoretic or algebraic intractability assumptions commonly used in cryptography. Hence, the existence of OWF and PRG in NC^0 follows from a variety of standard assumptions and is not affected by the potential weakness of a particular algebraic structure. A similar compiler can also be obtained for other cryptographic primitives including one-way permutations, encryption, signatures, commitment, and collision-resistant hashing (see Section 7).

It is important to note that the NC^0_4 PRG produced by our compiler will generally have a sublinear additive stretch even if the original PRG has a large stretch. However, one cannot do much better, as there is no PRG with superlinear stretch in NC^0_4 [25].

OWF WITH OPTIMAL LOCALITY. The above results leave a small gap between the possibility of cryptography in NC^0_4 and the known impossibility of implementing even OWF in NC^0_2. We partially close this gap by providing positive evidence for the existence of OWF in NC^0_3. Specifically, we construct such OWF based on either: (1) the intractability of decoding a random linear code; or (2) the existence of a moderately-easy OWF (say, in NC^1) that enjoys a certain strong "robustness" property. We show that a seemingly conservative variant of a OWF candidate suggested by Goldreich [11] provably satisfies this property, assuming that it is indeed a OWF. Further details are omitted from this extended abstract and will appear in the full version.

NON-CRYPTOGRAPHIC GENERATORS. Our techniques can also be applied to obtain unconditional constructions of non-cryptographic PRGs. In particular, building on an ϵ-biased generator in NC^0_5 constructed by Mossel et al. [25], we obtain a linear-stretch ϵ-biased generator in NC^0_3. This generator has optimal locality, answering an open question posed in [25]. (It is also essentially optimal with respect to stretch, since locality 3 does not allow for a superlinear stretch [9].) Our techniques apply also to other types of non-cryptographic PRGs such as generators for logspace [4, 28], yielding the first such generators in NC^0.

2. Overview of Techniques

Our key observation is that instead of computing a given "cryptographic" function $f(x)$, it might suffice to compute a function $\hat{f}(x, r)$ having the following relation to f:

1. For every fixed input x and a uniformly random choice of r, the output distribution $\hat{f}(x,r)$ forms a "randomized encoding" of $f(x)$, from which $f(x)$ can be decoded. That is, if $f(x) \neq f(x')$ then the random variables $\hat{f}(x,r)$ and $\hat{f}(x',r')$, induced by a uniform choice of r,r', should have disjoint supports.
2. The distribution of this randomized encoding depends only on the encoded value $f(x)$ and does not further depend on x. That is, if $f(x) = f(x')$ then the random variables $\hat{f}(x,r)$ and $\hat{f}(x',r')$ should be identically distributed. Furthermore, we require that the randomized encoding of an output value y be efficiently samplable given y. Intuitively, this means that the output distribution of $\hat{f}$ on input x reveals no information about x except what follows from $f(x)$.

Each of these requirements alone can be satisfied by a trivial function $\hat{f}$ (e.g., $\hat{f}(x,r) = x$ and $\hat{f}(x,r) = 0$, respectively). However, their combination can be viewed as a non-trivial natural relaxation of the usual notion of computing. In a sense, the function $\hat{f}$ defines an "information-theoretically equivalent" representation of f. In the following, we refer to $\hat{f}$ as a *randomized encoding* of f.

For this approach to be useful in our context, two conditions should be met. First, we need to argue that a randomized encoding $\hat{f}$ can be *securely* used as a substitute for f. Second, we hope that this relaxation is sufficiently *liberal*, in the sense that it allows to efficiently encode relatively complex functions f by functions $\hat{f}$ in NC^0. These two issues are addressed in the following subsections.

2.1. Security of Randomized Encodings

To illustrate how a randomized encoding $\hat{f}$ can inherit the security features of f, consider the case where f is a OWF. We argue that the hardness of inverting $\hat{f}$ reduces to the hardness of inverting f. Indeed, a successful algorithm A for inverting $\hat{f}$ can be used to successfully invert f as follows: given an output y of f, apply the efficient sampling algorithm guaranteed by requirement 2 to obtain a random encoding $\hat{y}$ of y. Then, use A to obtain a preimage (x,r) of $\hat{y}$ under $\hat{f}$, and output x. It follows from requirement 1 that x is indeed a preimage of y. Moreover, if y is the image of a uniformly random x, then $\hat{y}$ is the image of a uniformly random pair (x,r). Hence, the success probability of inverting f is the same as that of inverting $\hat{f}$.

The above argument can tolerate some relaxations to the notion of randomized encoding. In particular, one can relax the second requirement to allow a small statistical variation of the output distribution. On the other hand, to maintain the security of other cryptographic primitives, it may be required to further strengthen this notion. For instance, when f is a PRG, the above requirements do not guarantee that the output of $\hat{f}$ is pseudo-random, or even that its output is longer than its input. However, by imposing suitable "regularity" requirements on the output encoding defined by $\hat{f}$, it can be guaranteed that if f is a PRG then so is $\hat{f}$. Thus, different security requirements suggest different variations of the above notion of randomized encoding.

2.2. Complexity of Randomized Encodings

It remains to address the second issue: how can we encode a complex function f by an NC^0 function $\hat{f}$? Our best solutions to this problem rely on the machinery of *randomizing polynomials,* described below. But first, we outline a simple alternative approach[4] based on Barrington's theorem [5], combined with a randomization technique of Kilian [22].

Suppose f is a boolean function in NC^1. (Non-boolean functions are handled by repeating the following procedure for each bit of the output.) By Barrington's theorem, evaluating $f(x)$ reduces to computing an iterated product of polynomially many elements $s_1, \ldots, s_m$ from the symmetric group S_5, where each s_i is determined by a single bit of x. Now, let $\hat{f}(x,r) = (s_1 r_1,\, r_1^{-1} s_2 r_2, \ldots,\, r_{m-2}^{-1} s_{m-1} r_{m-1},\, r_{m-1}^{-1} s_m)$, where the random inputs r_i are picked uniformly and independently from S_5. It is not hard to verify that the output $(t_1, \ldots, t_m)$ of $\hat{f}$ is random subject to the constraint that $t_1 t_2 \cdots t_m = s_1 s_2 \cdots s_m$, where the latter product is in one-to-one correspondence to $f(x)$. It follows that $\hat{f}$ is a randomized encoding of f. Moreover, $\hat{f}$ has constant locality when viewed as a function over the alphabet S_5, and thus yields the qualitative result we are after. Still, this construction falls short of providing a randomized encoding in NC^0, since it is impossible to sample a uniform element of S_5 in NC^0 (even up to a negligible statistical distance). Also, this $\hat{f}$ does not satisfy the properties required by more "sensitive" primitives such as PRGs or one-way permutations. The solutions presented next avoid these disadvantages and, at the same time, apply to a higher complexity class than NC^1 and achieve a very small constant locality.

RANDOMIZING POLYNOMIALS. The concept of randomizing polynomials was introduced in [19] as a representation of functions by vectors of low-degree multivariate polynomials. (Interestingly, this concept was motivated by questions in the area of *information-theoretic* secure multiparty computation, which seems unrelated to the current context.) Randomizing polynomials capture the above encoding question within an algebraic framework. Specifically, a representation of $f(x)$ by randomizing polynomials is a randomized encoding $\hat{f}(x,r)$ as defined above, in which x and r are viewed as vectors over a finite field $\mathcal{F}$ and the outputs of $\hat{f}$ as multivariate polynomials in the variables x,r. In this work, we will always let $\mathcal{F} = \mathrm{GF}(2)$.

4 In fact, a modified version of this approach has been applied for constructing randomizing polynomials in [8].

The most crucial parameter of a randomizing polynomials representation is its algebraic *degree*, defined as the maximal (total) degree of the outputs as a function of the input variables x, r. (Note that both x and r count towards the degree.) Its *complexity* is measured as the total number of inputs and outputs. Quite surprisingly, it is shown in [19, 20] that every boolean function $f : \{0,1\}^n \to \{0,1\}$ admits a representation by *degree-3* randomizing polynomials whose complexity is at most quadratic in its branching program size.[5] (Moreover, this degree bound is tight in the sense that most boolean functions do not admit a degree-2 representation.) Note that a representation of a non-boolean function can be obtained by concatenating representations of its output bits, using independent blocks of random inputs. This concatenation leaves the degree unchanged.

The above positive result implies that functions whose output bits can be computed in the complexity class $\oplus\mathrm{L}/poly$ admit an efficient representation by degree-3 randomizing polynomials. This also holds if one requires the most stringent notion of representation required by our applications. We note, however, that different constructions from the literature [19, 20, 8] are incomparable in terms of their exact efficiency and the security-preserving features they satisfy. Hence, different constructions may be suitable for different applications. These issues are discussed in Section 4.

DEGREE VS. LOCALITY. Combining our general methodology with the above results on randomizing polynomials already brings us close to our goal, as it enables "degree-3 cryptography". Taking on from here, we show that any function $f : \{0,1\}^n \to \{0,1\}^m$ of algebraic degree d admits an efficient randomized encoding $\hat{f}$ of degree d and locality $d+1$. That is, each output bit of $\hat{f}$ can be computed by a degree-d polynomial over $\mathrm{GF}(2)$ depending on at most $d+1$ inputs and random inputs. Combined with the previous results, this allows us to make the final step from degree 3 to locality 4.

Paper organization. Following some preliminaries (Section 3), in Section 4 we formally define our notion of randomized encoding and discuss some of its variants, properties, and constructions. In Section 5 we apply randomized encodings to construct OWFs in NC^0 and in Section 6 we do the same for cryptographic and non-cryptographic PRGs. Finally, in Section 7 we discuss extensions to other cryptographic primitives, and in Section 8 we conclude with some further research directions. For lack of space, some proofs were omitted from this version.

5 By default, "branching programs" refer here to mod-2 branching programs, which output the parity of the number of accepting paths. See Section 3.

3. Preliminaries

Probability notation. Let U_n denote a random variable that is uniformly distributed over $\{0,1\}^n$. Different occurrences of U_n are independent. The *statistical distance* between discrete probability distributions Y and Y' is defined as $\mathrm{SD}(Y, Y') \stackrel{\text{def}}{=} \frac{1}{2}\sum_y |\Pr[Y = y] - \Pr[Y' = y]|$. A function $\varepsilon(\cdot)$ is said to be *negligible* if $\varepsilon(n) < n^{-c}$ for any $c > 0$ and sufficiently large n. For two distribution ensembles $Y = \{Y_n\}$ and $Y' = \{Y'_n\}$, we write $Y \equiv Y'$ if Y_n and Y'_n are identically distributed, and $Y \stackrel{s}{\approx} Y'$ if the two ensembles are statistically indistinguishable, namely $\mathrm{SD}(Y_n, Y'_n)$ is negligible in n.

Branching programs. A branching program (BP) is defined by a tuple $BP = (G, \phi, s, t)$, where $G = (V, E)$ is a directed acyclic graph, ϕ is a labeling function assigning each edge a a positive literal x_i, a negative literal $\bar{x}_i$ or the constant 1, and s, t are two distinguished nodes of G. The *size* of *BP* is the number of nodes in G. Each input assignment $w = (w_1, \ldots, w_n)$ naturally induces an unlabeled subgraph G_w, whose edges include all $e \in E$ such that $\phi(e)$ is satisfied by w. BPs may be assigned different semantics: in a *non-deterministic* BP, an input w is accepted if G_w contains at least one path from s to t; in a *mod-p* BP, w is accepted if the number of such paths is nonzero modulo p. In this work, we will mostly be interested in mod-2 BPs.

Function families and representations. We associate with a function $f : \{0,1\}^* \to \{0,1\}^*$ a function family $\{f_n\}_{n \in \mathbb{N}}$, where f_n is the restriction of f to n-bit inputs. We assume all functions to be length regular, namely their output length depends only on their input length. Hence, we may write $f_n : \{0,1\}^n \to \{0,1\}^{l(n)}$. We will represent functions f by families of circuits, branching programs, or polynomial vectors. Whenever f is taken from a uniform class, we assume that its representation is uniform as well. That is, the representation of f_n is generated in time $\mathrm{poly}(n)$ and in particular is of polynomial size. We will often abuse notation and write f instead of f_n even when referring to a function on n bits.

Locality and degree. We say that f is c-local if each of its output bits depends on at most c input bits. The non-uniform class NC^0_c includes all c-local functions. We will sometimes view the binary alphabet as the finite field $\mathcal{F} = \mathrm{GF}(2)$, and say that a function f has degree d if each of its outputs can be expressed as a multivariate polynomial of degree (at most) d in the inputs.

Complexity classes. For brevity, we assume all complexity classes to be polynomial-time uniform by default. For instance, NC^0 refers to the class of functions admitting uniform NC^0 circuits. We let $\mathrm{NL}/poly$ (resp., $\oplus\mathrm{L}/poly$) denote the class of boolean functions computed by a uniform family of nondeterministic (resp., modulo-2) BPs. Equivalently, these are the classes of functions computed by NL (resp., $\oplus$L) Turing machines taking a uniform advice. We extend boolean complexity classes, such as $\mathrm{NL}/poly$ and $\oplus\mathrm{L}/poly$, to include non-boolean functions by letting the representation include $l(n)$ branching programs, one for each output. Uniformity requires that the $l(n)$ branching programs be all generated in time $\mathrm{poly}(n)$.

4. Randomized Encodings of Functions

We now formally introduce our notion of randomized encoding, discuss some of its variants and properties, and present constructions of randomized encodings in NC^0.

4.1. Definitions

Definition 4.1 (Randomized encoding) *Let $f : \{0,1\}^n \to \{0,1\}^l$ be a function. We say that a function $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$ is a δ-correct, ε-private* randomized encoding *of f, if it satisfies the following:*

- **δ-correctness.** *There exists a (possibly randomized) algorithm C, called a* decoder*, such that for any input $x \in \{0,1\}^n$, $\Pr[C(\hat{f}(x, U_m)) \neq f(x)] \leq \delta$.*
- **ε-privacy.** *There exists a randomized algorithm S, called a* simulator*, such that for any $x \in \{0,1\}^n$, $\mathrm{SD}(S(f(x)), \hat{f}(x, U_m)) \leq \varepsilon$.*

We refer to the second input of $\hat{f}$ as its random input.

On uniform randomized encodings. The above definition naturally extends to functions $f : \{0,1\}^* \to \{0,1\}^*$. In this case, the parameters $l, m, s, \delta, \varepsilon$ are all viewed as functions of the input length n, and the algorithms C, S receive 1^n as an additional input. In our default uniform setting, we require that $\hat{f}_n$, the encoding of f_n, be computable in time $\mathrm{poly}(n)$ (given $x \in \{0,1\}^n$ and $r \in \{0,1\}^{m(n)}$). Thus, in this setting both $m(n)$ and $s(n)$ are polynomial. We also require both the decoder and the simulator to run in probabilistic polynomial time. (This is not needed by some of the applications, but is a feature of our constructions.) Finally, we will sometimes view $\hat{f}$ as a function of a single input of length $n + m(n)$ (e.g., when using it as OWF or PRG). In this case, we require $m(\cdot)$ to be monotone (so that $n + m(n)$ uniquely determines n), and apply a standard padding technique for defining $\hat{f}$ on inputs whose length is not of the form $n + m(n)$. Specifically, if $n + m(n) + k < (n+1) + m(n+1)$ we define $\hat{f}$ on inputs of length $n + m(n) + k$ by padding $\hat{f}_n$ with k additional input bits and adding these bits to the output of $\hat{f}_n$. The above conventions will be implicit in the following.

We move on to discuss some variants of the basic definition. Correctness (resp., privacy) can be either *perfect*, when $\delta = 0$ (resp. $\varepsilon = 0$), or *statistical*, when $\delta(n)$ (resp. $\varepsilon(n)$) is negligible. While for some of the primitives (such as OWF) statistical privacy and correctness will do, others require even stronger properties than perfect correctness and privacy. We say that an encoding is *balanced* if it admits a perfectly private simulator S such that $S(U_l) \equiv U_s$. Such S will be referred to as a *balanced simulator*. We say that the encoding is *stretch preserving* if $\hat{f}$ has the same additive stretch as f; namely, $s - (n+m) = l - n$ or equivalently $s = l + m$. We are now ready to define our two main variants of randomized encoding.

Definition 4.2 (Statistical randomized encoding) *A statistical randomized encoding is a randomized encoding which is statistically correct and private.*

Definition 4.3 (Perfect randomized encoding) *A perfect randomized encoding is a randomized encoding which is perfectly correct and private, balanced, and stretch-preserving.*

A perfect randomized encoding guarantees the existence of a perfect simulator S whose 2^l output distributions form a perfect tiling of the space $\{0,1\}^s$ by tiles of size 2^m.

Finally, we define two complexity classes that capture the power of randomized encodings in NC^0.

Definition 4.4 (The classes SREN, PREN) *The class $\mathcal{SREN}$ (resp., $\mathcal{PREN}$) is the class of functions admitting statistical (resp., perfect) randomized encoding in NC^0.*

4.2. Basic Properties

We now put forward some useful properties of randomized encodings, which are stated here without a proof. We first argue that an encoding of a non-boolean function can be obtained by concatenating encodings of its output bits, using an independent random input for each bit. The resulting encoding inherits all the features of the concatenated encodings. Thus, the following lemma applies to both the statistical and the perfect cases.

Lemma 4.5 (Concatenation) *Let $f^{(i)} : \{0,1\}^n \to \{0,1\}$, $1 \leq i \leq l$, be the boolean functions computing the output bits of $f : \{0,1\}^n \to \{0,1\}^l$. If $\hat{f}^{(i)}(x, r^{(i)})$ is a randomized encoding of $f^{(i)}(x)$, then the concatenation $\hat{f}(x, (r^{(1)}, \ldots, r^{(l)})) \stackrel{def}{=} (\hat{f}^{(1)}(x, r^{(1)}), \ldots, \hat{f}^{(l)}(x, r^{(l)}))$ is a randomized encoding of f.*

When applying the above lemma in a uniform setting, we assume that $l(n) = \mathrm{poly}(n)$ and that the family $\hat{f}_n^{(i)}$ is uniform both in n and i.

Another useful feature of randomized encodings is the following intuitive composition property: suppose we encode f by g, and then view g as a deterministic function and encode it again. Then, the resulting function (parsed appropriately) is a randomized encoding of f. Again, the following lemma applies to all variants of randomized encoding.

Lemma 4.6 (Composition) *Let $g(x, r)$ be a randomized encoding of $f(x)$ and $h((x, r), r')$ a randomized encoding of $g(x, r)$. Then, h is a randomized encoding of f whose random inputs are (r, r').*

Finally, we state two useful features of a *perfect* encoding.

Lemma 4.7 (Unique randomness) *Suppose $\hat{f}$ is a perfect randomized encoding of f. Then, $\hat{f}$ satisfies the following* unique randomness *property: for any input x, the function $\hat{f}(x,\cdot)$ is injective, namely there are no distinct r, r' such that $\hat{f}(x,r) = \hat{f}(x,r')$. Moreover, if f is a permutation then so is $\hat{f}$.*

4.3. Constructions

In this section we construct randomized encodings in NC^0. We first review a construction from [20] of degree-3 randomizing polynomials based on mod-2 branching programs and analyze some of its properties. Then, we apply a general locality reduction technique, allowing to transform a degree-d encoding to a $(d+1)$-local encoding.

DEGREE-3 RANDOMIZING POLYNOMIALS FROM MOD-2 BRANCHING PROGRAMS [20]. Let $BP = (G, \phi, s, t)$ be a mod-2 BP of size ℓ, computing a boolean function $f : \{0,1\}^n \to \{0,1\}$. Fix some topological ordering of the vertices of G, where the source vertex s is labeled 1 and the terminal vertex t is labeled ℓ. For any input x, let A_x be the $\ell \times \ell$ adjacency matrix of G_x, viewed as a matrix over GF(2). Define $L(x)$ as the submatrix of $A_x - I$ obtained by deleting column s and row t (i.e., the first column and the last row). Each entry of $L(x)$ is a degree-1 polynomial in a single input variable x_i; moreover, $L(x)$ contains the constant -1 in each entry of its second diagonal (the one below the main diagonal) and the constant 0 below this diagonal.

Fact 4.8 ([20]) $f(x) = \det(L(x))$.

Let $r^{(1)}$ and $r^{(2)}$ be vectors over GF(2) of length $\binom{\ell-1}{2}$ and $\ell-2$ respectively. Let $R_1(r^{(1)})$ be an $(\ell-1)\times(\ell-1)$ matrix with 1's on the main diagonal, 0's below it, and $r^{(1)}$'s elements in the remaining $\binom{\ell-1}{2}$ entries above the diagonal (a unique element of $r^{(1)}$ is assigned to each matrix entry). Let $R_2(r^{(2)})$ be an $(\ell-1)\times(\ell-1)$ matrix with 1's on the main diagonal, $r^{(2)}$'s elements in the rightmost column, and 0's in each of the remaining entries.

Fact 4.9 ([20]) *Let M, M' be $(\ell-1)\times(\ell-1)$ matrices that contain the constant -1 in each entry of their second diagonal and the constant 0 below this diagonal. Then, $\det(M_1) = \det(M_2)$ if and only if there exist $r^{(1)}$ and $r^{(2)}$ such that $R_1(r^{(1)})MR_2(r^{(2)}) = M'$.*

Lemma 4.10 (implicit in [20]) *Let BP and f be as above. Define a degree-3 function $\hat{f}(x,(r^{(1)},r^{(2)}))$ whose outputs contain the $\binom{\ell}{2}$ entries on or above the main diagonal of the matrix $R_1(r^{(1)})L(x)R_2(r^{(2)})$. Then, $\hat{f}$ is a perfect randomized encoding of f.*

Proof: We start by describing the simulator and decoder algorithms. Given an output of $\hat{f}$, representing a matrix M, the decoder C simply outputs $\det(M)$. (Note that the entries below the main diagonal of this matrix are constants and therefore are not included in the output of $\hat{f}$.) The simulator S, on input $y \in \{0,1\}$, outputs the $\binom{\ell}{2}$ entries on and above the main diagonal of the matrix $R_1(r^{(1)})H_yR_2(r^{(2)})$, where $r^{(1)}$, $r^{(2)}$ are randomly chosen, and H_y is the $(\ell-1)\times(\ell-1)$ matrix that contains -1's in its second diagonal, y in its top-right entry, and 0's elsewhere. The perfectness of the C, S follows from Facts 4.8, 4.9; for a detailed proof the reader is referred to [20].

We now prove the other properties of a perfect encoding that are not explicit in [20]. The length of the random input of $\hat{f}$ is $m = \binom{\ell-1}{2} + \ell - 2 = \binom{\ell}{2} - 1$ and its output length is $s = \binom{\ell}{2}$. Thus we have $s = m+1$, and since f is a boolean function its encoding $\hat{f}$ preserves its stretch.

It remains to show that $\hat{f}$ is balanced. It follows from Fact 4.9 and the description of S that the support of $S(b)$, $b \in \{0,1\}$, includes all strings in $\{0,1\}^s$ representing matrices with determinant b. Hence, $S(0)$ and $S(1)$ cover the entire space $\{0,1\}^s$. Since we have already shown $\hat{f}$ to be stretch-preserving, the simulator S must be balanced. ■

REDUCING THE LOCALITY. It remains to convert the degree-3 encoding into one in NC^0. To this end, we show how to construct for any degree-d function (where d is constant) a $(d+1)$-local perfect encoding. Using the composition lemma, we can obtain an NC^0 encoding of a function by first encoding it as a constant-degree function, and then applying the locality construction.

The idea for the locality construction is to represent a degree-d polynomial as a sum of monomials, each having locality d, and randomize this sum using a variant of the method for randomizing group product, described in Section 2.2. (A direct use of the latter method over the group Z_2 gives a $(d+2)$-local encoding instead of the $(d+1)$-local one obtained here.)

Construction 4.11 (Locality construction) *Let $f(x) = T_1(x) + \ldots + T_k(x)$, where summation is over* GF(2). *The local encoding $\hat{f}$ is defined by:*

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

Lemma 4.12 (Locality lemma) *Let f and $\hat{f}$ be as in Construction 4.11. Then, $\hat{f}$ is a perfect randomized encoding of f. In particular, if f is a degree-d polynomial written as the sum of monomials, then $\hat{f}$ is a perfect encoding of f with degree d and locality* $\max(d+1,3)$.

Proof: Since $m = 2k-1$ and $s = 2k$, $\hat{f}$ is stretch preserving. Moreover, it is easy to verify that the outputs add up to $f(x)$. It thus suffices to show that the outputs of $\hat{f}(x)$

are uniformly distributed subject to the constraint that they add up to $f(x)$. This follows by observing that, for any x and any assignment $y \in \{0,1\}^{2k-1}$ to the first $2k-1$ outputs of $\hat{f}(x)$, there is a unique way to set the random inputs r_i, r'_i so that the output of $\hat{f}(x,(r,r'))$ is consistent with y. Indeed, for $1 \le i \le k$, the values of x, y_i uniquely determine r_i. For $1 \le i \le k-1$, the values y_{k+i}, r_i, r'_{i-1} determine r'_i (where $r'_0 \stackrel{\text{def}}{=} 0$). ■

Combining the degree-3 construction of Lemma 4.10 together with the locality lemma (4.12), composition lemma (4.6), and concatenation lemma (4.5), we get the main theorem of this section.

Theorem 4.13 $\oplus\mathrm{L}/poly \subseteq \mathcal{PREN}$. *Moreover, any* $f \in \mathcal{PREN}$ *admits a perfect randomized encoding in* NC^0_4.

Remark 4.14 A more direct approach for perfect randomized encodings in NC^0 is possible using a randomizing polynomials construction from [20], which is based on an information-theoretic variant of Yao's garbled circuit technique [34]. This construction directly gives an encoding with (large) constant locality for functions in NC^1.

There are variants of the above construction that can handle non-deterministic branching programs as well, at the expense of losing perfectness [19, 20]. Thus, we get the following theorem, whose proof is deferred to the full version.

Theorem 4.15 $\mathrm{NL}/poly \subseteq \mathcal{SREN}$. *Moreover, any* $f \in \mathcal{SREN}$ *admits a statistical randomized encoding in* NC^0_4.

5. One-Way Functions in NC^0

A *one-way function* (OWF) $f : \{0,1\}^* \to \{0,1\}^*$ is a polynomial-time computable function that is hard to invert; namely, every polynomial time algorithm that tries to invert f on $f(x)$, where x is picked from U_n, succeeds with a negligible probability. In the following, we show that a randomized encoding $\hat{f}$ of a OWF f is also a OWF. The idea, as described in Section 2.1, is to argue that the hardness of inverting $\hat{f}$ reduces to the hardness of inverting f. Here, we will further formalize this claim and slightly strengthen it. We start with a technical claim.

Claim 5.1 *Let* $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$ *be a perfectly private (resp., statistically private) randomized encoding of* $f : \{0,1\}^n \to \{0,1\}^l$, *and let* S *be its perfect (resp., statistical) simulator. Then* $S(f(U_n)) \equiv \hat{f}(U_n, U_{m(n)})$ *(resp.,* $S(f(U_n)) \stackrel{s}{\approx} \hat{f}(U_n, U_{m(n)})$*).*

Lemma 5.2 *Suppose that* $f : \{0,1\}^* \to \{0,1\}^*$ *is hard to invert and* $\hat{f}(x,r)$ *is a perfectly-correct, statistically-private (uniform) encoding of* f. *Then* $\hat{f}$, *viewed as a deterministic function, is also hard to invert.*

Proof: Let $s = s(n), m = m(n)$ be the lengths of the output and random input of $\hat{f}$ respectively. We prove that $\hat{f}$ is as "hard to invert" as f. Assume, towards a contradiction, that there is an efficient algorithm B inverting $\hat{f}_n(x,r)$ with success probability $\phi(n+m) > \frac{1}{q(n+m)}$ for some polynomial $q(\cdot)$ and infinitely many n's. We use B to construct an efficient algorithm A that inverts f with similar success. On input $(1^n, y = f(U_n))$, the algorithm A runs S, the statistical simulator of $\hat{f}_n$, on the input y and gets a string $\hat{y}$ as S's output. A proceeds by running the inverter B on the input $(1^{n+m}, \hat{y})$, getting (x,r) as B's output (i.e., B "claims" that $\hat{f}_n(x,r) = \hat{y}$). A terminates with output x.

COMPLEXITY: since S and B are both polynomial-time algorithms, and since $m(n)$ is polynomially bounded, it follows that A is also a polynomial-time algorithm.

CORRECTNESS: Observe that, by perfect correctness, if $f(x) \neq f(x')$ then the sets $\hat{f}(x, U_m)$ and $\hat{f}(x', U_m)$ are disjoint. Hence, if B succeeds (that is, indeed $\hat{y} = \hat{f}_n(x,r)$) then so does A (namely, $f(x) = y$). Next, observe that by Claim 5.1 the input $\hat{y}$ on which A runs B is $\varepsilon(n)$-close to $\hat{f}_n(U_n, U_{m(n)})$, and therefore B succeeds with probability $\ge \phi(n+m) - \varepsilon(n)$. Formally, we can write:

$$\begin{aligned}
&\Pr_{x \in U_n}[A(1^n, f(x)) \in f^{-1}(f(x))] \\
&= \Pr_{x \in U_n, \hat{y} \in S(f(x))}[B(1^{n+m}, \hat{y}) \in \hat{f}^{-1}(\hat{y})] \\
&\ge \Pr_{x \in U_n, r \in U_{m(n)}}[B(1^{n+m}, \hat{f}_n(x,r)) \in \hat{f}^{-1}(\hat{f}(x,r))] - \varepsilon(n) \\
&\ge \phi(n+m) - \varepsilon(n) > \frac{1}{q(n+m)} - \varepsilon(n) > \frac{1}{q'(n)},
\end{aligned}$$

where $q'(n)$ is a polynomial. It follows that f is not a one-way function, in contradiction to the hypothesis. ■

The perfect correctness of $\hat{f}$ is essential for Lemma 5.2 to hold. In the full version we show that even if $\hat{f}$ is only statistically correct, it is still *distributionally* one-way [17]. In this case, one can apply a standard transformation (cf. [12], p. 96) to convert a distributionally OWF $\hat{f}$ in NC^0 to a OWF $\hat{f}'$ in NC^1, and then encode the latter by a OWF in NC^0. Based on the above, we get:

Theorem 5.3 *A OWF in* $\mathcal{SREN}$ *(in particular, in* $\oplus\mathrm{L}/poly$ *or* $\mathrm{NL}/poly$*) implies a OWF in* NC^0_4.

Combining Lemma 5.2 and Lemma 4.7, we get a similar result for one-way permutations.

Theorem 5.4 *A one-way permutation in* $\mathcal{PREN}$ *(in particular, in* $\oplus\mathrm{L}/poly$*) implies one in* NC^0_4.

A NOTE CONCERNING EFFICIENCY. Loosely speaking, the main security loss in the reduction follows from the expansion of the input. (The simulator's running time has a minor effect on the security, since it is added to the overall running-time of the adversary.) Thus, to achieve a similar level of security to that achieved by applying f on n-bit inputs, one would need to apply $\hat{f}$ on $n + m(n)$ bits (the random input part of the encoding does not contribute to the security). Going through our constructions (bit-by-bit encoding of the output, based on some size-$\ell(n)$ BPs, followed by the locality reduction), we get $m(n) = l(n) \cdot \mathrm{poly}(\ell)$, where $l(n)$ is the output length of f. Some more efficient alternatives will be discussed in the full version.

6. Pseudorandom Generators in NC^0

A *pseudorandom generator* is an efficiently computable function $G : \{0,1\}^n \to \{0,1\}^{l(n)}$ such that: (1) G has a positive stretch, namely $l(n) > n$; (2) any "computationally bounded" algorithm D, called a *distinguisher*, has a negligible advantage in distinguishing $G(U_n)$ from $U_{l(n)}$. That is, $|\Pr[D(1^n, G(U_n)) = 1] - \Pr[D(1^n, U_{l(n)}) = 1]|$ is negligible in n.

Different notions of PRGs differ mainly in the computational bound imposed on D. In the default case of *cryptographic* PRGs, D can be any probabilistic polynomial-time algorithm (alternatively, polynomial-size circuit family). In the case of ϵ*-biased* generators, D can only compute a linear function of the output bits, namely the exclusive-or of some subset of the bits. Other types of PRGs, e.g. for logspace computation, have also been considered.

We show that a *perfect* randomized encoding of a PRG is also a PRG. We start by proving this claim for cryptographic PRGs and then obtain a similar result for ϵ-biased generators. The discussion of generators for logspace is deferred to the full version.

6.1. Cryptographic Generators

Lemma 6.1 *If $G : \{0,1\}^n \to \{0,1\}^l$ is a PRG and $\hat{G} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$ is a (uniform) perfect randomized encoding of G, then $\hat{G}$ is also a PRG.*

Proof sketch: Since $\hat{G}$ has the same additive stretch as G, it is guaranteed to expand its seed. To prove the pseudorandomness of its output, we again use a reducibility argument. Given a distinguisher $\hat{D}$ between U_s and $\hat{G}(U_n, U_m)$, we obtain a distinguisher D between U_l and $G(U_n)$ as follows. On input $y \in \{0,1\}^l$, run the balanced simulator of $\hat{G}$ on y, and invoke $\hat{D}$ on the result $\hat{y}$. If y is taken from U_l then the simulator, being balanced, outputs $\hat{y}$ that is distributed as U_s; if y is taken from $G(U_n)$ then, by Claim 5.1, the output of the simulator is distributed as $\hat{G}(U_n, U_m)$. Thus, the distinguisher D we get for G has the same advantage as the distinguisher $\hat{D}$ for $\hat{G}$. Since $m(n)$ is polynomial in n, this advantage is negligible also in $n + m$. ■

Thus, we get:

Theorem 6.2 *A pseudorandom generator in $\mathcal{PREN}$ (in particular, in $\oplus\mathrm{L}/poly$) implies one in NC^0_4.*

We stress that the NC^0_4 PRG $\hat{G}$ one gets from our construction has a sublinear stretch even if G has a large stretch. This follows from the fact that the length $m(n)$ of the random input is superlinear in the input length n.

Remark 6.3 The transformation of OWF to PRG from [16] (Construction 7.1) involves only the computation of universal hash functions and hard-core bits in the case that the "entropy" of the OWF is known (e.g., if the OWF is regular). In this case, an NC^1 OWF can be transformed into an NC^1 PRG.[6] Combined with Theorems 5.3, 6.2, this yields a PRG in NC^0_4 based on regular OWF in $\mathcal{SREN}$ (alternatively, a PRG in nonuniform-NC^0_4 from any OWF in $\mathcal{SREN}$).

6.2. ε-Biased Generators

The proof of Lemma 6.1 uses the balanced simulator to transform a challenge for G into a challenge for $\hat{G}$. If this transformation can be made linear, then the security reduction goes through also in the case of ε-biased generators.

Lemma 6.4 *Let G be an ε-biased generator and $\hat{G}$ a perfect randomized encoding of G. Assume that the balanced simulator of $\hat{G}$ is* linear *in the sense that it outputs a randomized linear transformation of $G(x)$ (which is not necessarily a linear function of the simulator's randomness). Then, $\hat{G}$ is also an ε-biased generator.*

Proof sketch: The proof is similar to that of Lemma 6.1. By an averaging argument and by the linearity of the simulator, it follows that a linear distinguisher for $\hat{G}$ can be transformed into a (nonuniform) linear distinguisher for G. ■

Mossel et al. present an ε-biased generator in nonuniform NC^0_5 with degree 2 and a linear stretch ([25], Theorem 14). Since this generator is already in NC^0, applying the locality reduction keeps the stretch linear. Using Lemmas 4.12,6.4 we thus get:

Theorem 6.5 *There is a linear-stretch ε-biased generator in nonuniform NC^0_3.*

One can also apply the locality reduction to get a uniform NC^0_3 generator from the ϵ-biased generator $G(x_1, \ldots, x_{2n}) = (x_1, \ldots, x_{2n}, x_1x_2 + \ldots + x_{2n-1}x_{2n})$ (cf. [30]). However, the resulting generator will have sublinear stretch. Using our general encoding machinery, one can transform an arbitrary uniform NC^0 generator with *linear* stretch (if such exists) into one in NC^0_4.

7. Other Cryptographic Primitives

We now outline some extensions of our results to other cryptographic primitives. Aiming at NC^0 implementations, we can use our machinery in two different ways: (1) compile a primitive in a relatively high complexity class (say NC^1) into its randomized encoding and show that the encoding inherits the security properties of this primitive; (2) use known reductions between cryptographic primitives together with NC^0 primitives we construct (e.g., OWF or

6 Viola [31] obtains a similar result for AC^0. Our techniques allow to further reduce the complexity of this reduction to NC^0.

PRG) to obtain new NC^0 primitives. We mainly adopt the first approach, since most of the known reductions between primitives are not in NC^0. Moreover, using the first approach, we can start by reducing one primitive to another and *then* apply our machinery. (Still, below we give an example for the usefulness of the second approach.)

We first consider the case of collision-resistant hashing. Suppose that a collection of functions h is collision-resistant, and let $\hat{h}$ be a perfect randomized encoding of h. Then, $\hat{h}$ is also collision-resistant since any collision $(x, r), (x', r')$ under $\hat{h}$ (that is, $(x, r) \neq (x', r')$ and $\hat{h}(x, r) = \hat{h}(x', r')$), can be trivially translated into a collision x, x' under h. Perfect correctness ensures that $h(x) = h(x')$ and unique-randomness (see Lemma 4.7) ensures that $x \neq x'$; also, since h and $\hat{h}$ have the same additive stretch, $\hat{h}$ shrinks its input.

A slightly different argument is used for encryption schemes. Suppose that $\mathcal{E} = (G, E, D)$ is a public-key encryption scheme, where G is a key-generation algorithm, the encryption function $E(e, m, r)$ encrypts the message m using the key e and randomness r, and $D(d, y)$ decrypts the cipher y using the decryption key d. As usual, the functions G, E, D are polynomial-time computable, and the scheme provides correct decryption and satisfies indistinguishability of encryptions [14]. Let $\hat{E}$ be a randomized encoding of E, and let $\hat{D}(d, \hat{y}) \stackrel{\text{def}}{=} D(d, C(\hat{y}))$ be the composition of D with the decoder C of the encoding $\hat{E}$. We argue that the scheme $\mathcal{E}' \stackrel{\text{def}}{=} (G, \hat{E}, \hat{D})$ is also a public-key encryption scheme. The efficiency and correctness of $\mathcal{E}'$ are guaranteed by the uniformity of the encoding and its correctness. Using the efficient simulator of the encoded function $\hat{E}$, we can reduce the security of $\mathcal{E}'$ to the security of $\mathcal{E}$; if some efficient adversary A' can break $\mathcal{E}'$ by distinguishing encryptions of m_1 and m_2, then we can construct an efficient adversary A that breaks the original scheme $\mathcal{E}$ by using the simulator to transform original ciphers into "new" ciphers, and then invoke A'.

Similar constructions can be used for commitments, signatures and MACs. In all these cases, we can replace the sender (i.e., the encrypting party, committing party or signer, according to the case) with its randomized encoding and let the receiver (the decrypting party or verifier) use the decoding algorithm to translate the output of the new sender to an output of the original one. The security of the resulting scheme reduces to the security of the original one by using the efficient simulator. Note that these transformations can be used to construct an NC^0 sender but they do not promise anything regarding the parallel complexity of the receiver.[7] The second approach mentioned above can be used to get a symmetric encryption scheme in which both encryption and decryption are in NC^0 by using the output of an NC^0 PRG to mask the plaintext. However, the resulting scheme is severely limited by the low stretch of our PRGs.

An interesting feature of the case of commitment is that we can also improve the complexity at the receiver's end; indeed, the sender can decommit by sending its random coins, and the receiver needs only to emulate the computation of the sender and compare it with the message it received in the commit stage. Thus, the receiver can be implemented as an NC^0 circuit with a single unbounded fan-in AND gate (we denote such a circuit as $\mathrm{NC}^0[\mathrm{AND}]$). Such a commitment scheme can then be used to implement coin flipping over the phone [6] between an NC^0 circuit and an $\mathrm{NC}^0[\mathrm{AND}]$ circuit. Moreover, such commitments can also be used to construct zero-knowledge proof-systems where both the prover and the verifier are highly parallelized.

7 Actually, it can be proved that some of these schemes cannot be secure if the receiver is in NC^0.

THE CASE OF PRFS. It is natural to ask why our machinery cannot be applied to pseudorandom functions (PRFs), as follows from the impossibility results of Linial et al. [24]. In our constructions of randomized encodings, the output $\hat{f}(x, r)$ together with the randomness r allows to recover x; i.e., the encoding loses its privacy. Now, suppose that a PRF family $f_k(x) = f(k, x)$ is encoded as the family $\hat{f}_k(x, r) = \hat{f}(k, x, r)$. The adversary can recover k by observing a point (x, r) along with the value of $\hat{f}_k$ at this point. More generally, our methodology works well for cryptographic primitives which employ fresh secret randomness for each invocation. PRFs do not fit into this category: while the key contains secret randomness, it is not freshly picked at each invocation.

COMPUTATIONALLY-PRIVATE ENCODINGS. For the purpose of most applications discussed above, it suffices to use a randomized encoding which offers *computational privacy* rather than a statistical or a perfect one. It turns out that, assuming the existence of a PRG in $\mathcal{PREN}$, it is possible to get a such a randomized encoding in NC^0 for arbitrary (polynomial-time computable) functions. This can be done by combining a variant of Yao's *garbled circuit* construction [34] with a PRG in NC^0. Computationally-private randomized encodings maintain the security of cryptographic primitives such as public-key encryption, signatures, and variants of commitments and zero knowledge proofs. Thus, given arbitrary (polynomial-time) implementations of these primitives, and assuming that there is a PRG in $\mathcal{PREN}$, we get implementations of these primitives in NC^0. Further details and additional applications will appear in [3].

8. Conclusions and Open Problems

Our results provide overwhelming evidence for the possibility of cryptography in NC^0. They are also close to optimal in terms of the exact locality that can be achieved. Still, several questions are left for further study. In particular:

- What are the minimal assumptions required for cryptography in NC^0? For instance, does the existence of an arbitrary OWF imply the existence of OWF in NC^0?
- Is there a PRG with linear stretch or even superlinear stretch in NC^0? In particular, is there a PRG with linear stretch in NC^0_4? (The possibility of PRG with superlinear stretch in NC^0_4 is ruled out in [25].)
- Can the existence of OWF (or PRG) in NC^0_3 be based on more general assumptions?
- Can our paradigm for achieving better parallelism be of any practical use?

The above questions motivate a closer study of the complexity of randomized encodings, which so far was only motivated by questions in the domain of secure multiparty computation.

Acknowledgments. We are grateful to Oded Goldreich for many useful suggestions and comments that helped improve this writeup. We also thank Emanuele Viola for sending us an early manuscript of [31] and for sharing with us some of his insights about constructing PRGs from OWFs.

References

[1] M. Agrawal, E. Allender, and S. Rudich. Reductions in circuit complexity: An isomorphism theorem and a gap theorem. *J. Comput. Syst. Sci.*, 57(2):127–143, 1998.

[2] M. Ajtai. Generating hard instances of lattice problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 3(7), 1996. Preliminary version in STOC '96.

[3] B. Applebaum, Y. Ishai, and E. Kushilevitz. Manuscript in preparation.

[4] L. Babai, N. Nisan, and M. Szegedy. Multiparty protocols and logspace-hard pseudorandom sequences. In *Proc. 21st STOC*, pp. 1–11, 1989.

[5] D.A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150-164, 1989. Preliminary version in STOC '86.

[6] M. Blum. Coin flipping by telephone: A protocol for solving impossible problems. *SIGACT News*, 15(1):23–27, 1983.

[7] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. on Computing,* Vol. 13, 1984, pp. 850-864, 1984. Preliminary version in FOCS 82.

[8] R. Cramer, S. Fehr, Y. Ishai, and E. Kushilevitz. Efficient multi-party computation over rings. In *Proc. EUROCRYPT '03*, pp. 596–613, 2003. Full version on ePrint Archives.

[9] M. Cryan and P. B. Miltersen. On pseudorandom generators in NC^0. In *Proc. 26th MFCS*, pp. 272–284, 2001.

[10] A. V. Goldberg, M. Kharitonov, and M. Yung. Lower bounds for pseudorandom number generators. In *Proc. 30th FOCS*, pp. 242–247, 1989.

[11] O. Goldreich. Candidate one-way functions based on expander graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 7(090), 2000.

[12] O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001.

[13] O. Goldreich and L.A. Levin. Hard-core predicate for any one-way function. In *Proc. 21st STOC*, pp. 25–32, 1989.

[14] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, 1984. Preliminary version in STOC '82.

[15] J. Håstad. One-way permutations in NC^0. *Information Processing Letters*, 26:153–155, 1987.

[16] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[17] R. Impagliazzo and M. Luby. One-way functions are essential for complexity based cryptography. In *Proc. of the 30th FOCS* pp. 230–235, 1989.

[18] R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of Cryptology*, 9:199–216, 1996. Preliminary version in FOCS '89.

[19] Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *Proc. 41st FOCS*, pp. 294–304, 2000.

[20] Y. Ishai and E. Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *Proc. 29th ICALP*, pp. 244–256, 2002.

[21] M. Kharitonov. Cryptographic hardness of distribution-specific learning. In *Proc. 25th STOC*, pp. 372–381, 1993.

[22] J. Kilian. Founding cryptography on oblivious transfer. In *Proc. of 20th STOC*, pp. 20–31, 1988.

[23] M. Krause and S. Lucks. On the minimal hardware complexity of pseudorandom function generators (extended abstract). In *Proc. 18th STACS*, *LNCS* 2010, pp. 419–430, 2001.

[24] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, Fourier transform, and learnability. *J. ACM*, 40(3):607–620, 1993. Preliminary version in FOCS '89.

[25] E. Mossel, A. Shpilka, and L. Trevisan. On ϵ-biased generators in NC^0. In *Proc. 44th FOCS*, pp. 136–145, 2003.

[26] J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993. Preliminary version in Proc. STOC '90.

[27] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, 2004. Preliminary version in Proc. FOCS '97.

[28] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[29] M.O. Rabin. Digitalized signatures and public key functions as intractable as factoring. TR-212, LCS, MIT, 1979.

[30] P. Savicky. On the bent functions that are symmetric. *European J. of Combinatorics*, 15:407–410, 1994.

[31] E. Viola. On parallel pseudorandom generators. Manuscript, 2004. To be posted on ECCC.

[32] A. Wigderson. $\mathrm{NL}/poly \subseteq \oplus\mathrm{L}/poly$. In *Proc. 9th Complexity Theory Conference*, pp. 59–62, 1994.

[33] A. C. Yao. Theory and application of trapdoor functions. In *Proc. 23rd FOCS*, pp. 80–91, 1982.

[34] A. C. Yao. How to generate and exchange secrets. In *Proc. 27th FOCS*, pp. 162–167, 1986.

[35] X. Yu and M. Yung. Space lower-bounds for pseudorandom-generators. In *Proc. 9th Complexity Theory Conference*, pp. 186–197, 1994.