Cryptography in NC^0 [*]

Benny Applebaum Yuval Ishai Eyal Kushilevitz

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

September 27, 2006

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 implementing instances of these primitives by NC^0 functions, namely by functions in which each output bit depends on a constant number of input bits. Despite previous efforts in this direction, there has been no convincing theoretical evidence supporting this possibility, which was posed as an open question in several previous works.

We essentially settle this question by providing strong 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 which each output bit depends 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. A similar compiler can also be obtained for other cryptographic primitives such as one-way permutations, encryption, signatures, commitment, and collision-resistant hashing.

Our techniques can also be applied to obtain (unconditional) constructions of "non-cryptographic" PRGs. In particular, we obtain ϵ-biased generators and a PRG for space-bounded computation in which each output bit depends on only 3 input bits.

Our results make use of the machinery of *randomizing polynomials* (Ishai and Kushilevitz, *41st FOCS*, 2000), which was originally motivated by questions in the domain of information-theoretic secure multiparty computation.

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) [11, 52]. 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., [32, 22, 16, 41, 43]):

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

Recall that NC^0 is the class of functions that 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.

[*]A preliminary version of this paper appeared in the proceedings of FOCS 2004. Research supported by grant no. 36/03 from the Israel Science Foundation.

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 [16], whereas Goldreich [22] takes an opposite view and suggests a concrete candidate for OWF in NC^0. However, despite previous efforts, there has been no convincing 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 [42]. However, no such impossibility result is known for PRGs. The existence of PRGs in NC^0 has been recently studied in [16, 43]. Cryan and Miltersen [16] 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. [43] extend this impossibility to NC^0_4. Viola [50] shows that a PRG in AC^0 with superlinear stretch cannot be obtained from a OWF via non-adaptive black-box constructions. Negative results for other restricted computation models appear in [20, 54].

On the positive side, Impagliazzo and Naor [36] 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 [39, 44], discrete logarithms [11, 52, 44] and lattice problems [2, 33] (see Remark 6.6).[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 [22, 16]. Håstad [32] constructs a family of permutations in NC^0 whose inverses are P-hard to compute. Cryan and Miltersen [16], improving on [1], present 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 [22] suggests 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.

We now give a more detailed account of our 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., sublinear-stretch PRG) in NC^0_4. (The class $\oplus\mathrm{L}/poly$ contains the classes $\mathrm{L}/poly$ and NC^1 and is contained in NC^2. In a

[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* (on seeds of length n^ϵ) 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. [23], Section 2.4.2). See Appendix A for further discussion of this slightly relaxed notion of PRG.

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

non-uniform setting it also contains the class $\mathrm{NL}/poly$ [51].) 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 sublinear-stretch 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.

It is important to note that the 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 when insisting on an NC^0_4 PRG, as there is no PRG with superlinear stretch in NC^0_4 [43].

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. In particular, we construct such OWF based on the intractability of decoding a random linear code.

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. [43], we obtain a linear-stretch ϵ-biased generator in NC^0_3. This generator has optimal locality, answering an open question posed in [43]. It is also essentially optimal with respect to stretch, since locality 3 does not allow for a superlinear stretch [16]. Our techniques apply also to other types of non-cryptographic PRGs such as generators for space-bounded computation [6, 45], yielding such generators (with sublinear stretch) in NC^0_3.

1.3 Organization

In Section 2 we provide an overview of our techniques, which evolve around the notion of "randomized encoding" introduced in this work. 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. We then apply randomized encodings to obtain NC^0 implementations of different primitives: OWFs (Section 5), cryptographic and non-cryptographic PRGs (Section 6), and other cryptographic primitives (Section 7). In Section 8 we construct OWF with optimal locality based on specific intractability assumptions. We conclude in Section 9 with some further research directions and open problems. We also call the reader's attention to Appendix A which discusses *collections* of cryptographic primitives and how they fit in the context of the current work.

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, the combination of the two requirements 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 under f. 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: 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 [7], combined with a randomization technique of Kilian [40].

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 $f(x)$, for such a function f, 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 (i.e., for every i there exists j such that s_i is a function of x_j). 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.

However, the above 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).[5] Also, this $\hat{f}$ does not satisfy the extra "regularity" 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.

[4]In fact, a modified version of this approach has been applied for constructing randomizing polynomials in [15].

[5]Barrington's theorem generalizes to apply over arbitrary non-solvable groups. Unfortunately, there are no such groups whose order is a power of two.

RANDOMIZING POLYNOMIALS. The concept of randomizing polynomials was introduced by Ishai and Kushilevitz [37] 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 and r. In this work, we will always let $\mathcal{F} = \mathrm{GF}(2)$.

The most crucial parameter of a randomizing polynomials representation is its algebraic *degree*, defined as the maximal (total) degree of the outputs (i.e., the output multivariate polynomials) as a function of the input variables in x and r. (Note that both x and r count towards the degree.) Quite surprisingly, it is shown in [37, 38] that every boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ admits a representation by *degree-3* randomizing polynomials whose number of inputs and outputs is at most *quadratic* in its branching program size.[6] (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 [37, 38, 15] 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 $f : \{0,1\}^n \rightarrow \{0,1\}^m$ 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.

3 Preliminaries

Probability notation. Let U_n denote a random variable that is uniformly distributed over $\{0,1\}^n$. Different occurrences of U_n in the same statement refer to the same random variable (rather than independent ones). If X is a probability distribution, we write $x \leftarrow X$ to indicate that x is a sample taken from X. If S is a set, we write $x \in_R S$ to indicate that x is uniformly selected selected from S. The *statistical distance* between discrete probability distributions X and Y is defined as $\|X - Y\| \stackrel{\text{def}}{=} \frac{1}{2}\sum_z |\Pr[X = z] - \Pr[Y = z]|$. Equivalently, the statistical distance between X and Y may be defined as the maximum, over all boolean functions T, of the *distinguishing advantage* $|\Pr[T(X) = 1] - \Pr[T(Y) = 1]|$. 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 $X = \{X_n\}$ and $Y = \{Y_n\}$, we write $X \equiv Y$ if X_n and Y_n are identically distributed, and $X \stackrel{s}{\approx} Y$ if the two ensembles are *statistically indistinguishable*; namely, $\|X_n - Y_n\|$ is negligible in n.

We will rely on the following standard properties of statistical distance.

Fact 3.1 *For every distributions X, Y, Z we have $\|X - Z\| \leq \|X - Y\| + \|Y - Z\|$.*

[6]By default, the notion of "branching programs" refers here to mod-2 branching programs, which output the parity of the number of accepting paths. See Section 3.

Fact 3.2 *For every distributions X, X', Y, Y' we have $\|(X \times X') - (Y \times Y')\| \leq \|X - Y\| + \|X' - Y'\|$, where $A \times B$ denotes the product distribution of A, B, i.e., the joint distribution of independent samples from A and B.*

Fact 3.3 *For every distributions X, Y and every function f we have $\|f(X) - f(Y)\| \leq \|X - Y\|$.*

Fact 3.4 *Let $\{X_z\}_{z \in \mathcal{Z}}$, $\{Y_z\}_{z \in \mathcal{Z}}$ be distribution ensembles. Then, for every distribution Z over $\mathcal{Z}$, we have $\|(Z, X_Z) - (Z, Y_Z)\| = E_{z \leftarrow Z}[\|X_z - Y_z\|]$. In particular, if $\|X_z - Y_z\| \leq \varepsilon$ for every $z \in \mathcal{Z}$, then $\|(Z, X_Z) - (Z, Y_Z)\| \leq \varepsilon$.*

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 either 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 (e.g., an edge labeled x_i is satisfied by w if $w_i = 1$). 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 (*counting*) *mod-p* BP, the BP computes the number of paths from s to t modulo p. In this work, we will mostly be interested in mod-2 BPs. An example of a mod-2 BP is given in Figure 3.1.

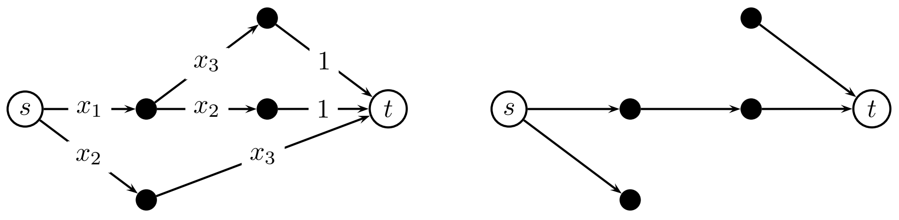

Figure 3.1: A mod-2 branching program computing the majority of three bits (left side), along with the graph G_{110} induced by the assignment 110 (right side).

Function families and representations. We associate with a function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ 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 $f_n : \{0,1\}^n \rightarrow \{0,1\}^{l(n)}$. We will represent functions f by families of circuits, branching programs, or vectors of polynomials (where each polynomial is represented by a formal sum of monomials). 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.[7] For a constant c, 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 $f : \mathcal{F}^n \rightarrow \mathcal{F}^{l(n)}$ has degree d if each of its outputs can be expressed as a multivariate polynomial of degree (at most) d in the inputs.

[7]A boolean function depends on the i^{th} input bit if there exists an assignment such that flipping the i^{th} input bit changes the value of the function.

Complexity classes. For brevity, we use the (somewhat nonstandard) convention that all complexity classes are polynomial-time uniform unless otherwise stated. For instance, NC^0 refers to the class of functions admitting uniform NC^0 circuits, whereas *non-uniform* NC^0 refers to the class of functions admitting non-uniform NC^0 circuits. We let $\mathrm{NL}/poly$ (resp., $\oplus\mathrm{L}/poly$) denote the class of boolean functions computed by a polynomial-time uniform family of nondeterministic (resp., modulo-2) BPs. (Recall that in a uniform family of circuits or branching programs computing f, it should be possible to generate the circuit or branching program computing f_n in time $\mathrm{poly}(n)$.) Equivalently, the class $\mathrm{NL}/poly$ (resp., $\oplus\mathrm{L}/poly$) is the class of functions computed by NL (resp., $\oplus$L) Turing machines taking a uniform advice. (The class $\oplus\mathrm{L}/poly$ contains the classes $\mathrm{L}/poly$ and NC^1 and is contained in NC^2. In a non-uniform setting it also contains the class $\mathrm{NL}/poly$ [51].) 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 Encoding of Functions

In this section we formally introduce our notion of randomized encoding. In Section 4.1 we introduce several variants of randomized encoding and in Section 4.2 we prove some of their useful properties. Finally, in Section 4.3 we construct NC^0_4 encodings for branching programs, building on [37, 38].

4.1 Definitions

We start by defining a randomized encoding of a finite function f. This definition will be later extended to a (uniform) family of functions.

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 deterministic*[8] *algorithm C, called a* decoder, *such that for every 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 every $x \in \{0,1\}^n$, $\|S(f(x)) - \hat{f}(x, U_m)\| \leq \varepsilon$.*

We refer to the second input of $\hat{f}$ as its random input *and to m and s as the* randomness complexity *and* output complexity *of $\hat{f}$, respectively.*

Note that the above definition only refers to the *information* about x revealed by $\hat{f}(x,r)$ and does not consider the complexity of the decoder and the simulator. Intuitively, the function $\hat{f}$ defines an "information-theoretically equivalent" representation of f. The correctness property guarantees that from $\hat{y} = \hat{f}(x,r)$ it is possible to reconstruct $f(x)$ (with high probability), whereas the privacy property guarantees that by seeing $\hat{y}$ one cannot learn too much about x (in addition to $f(x)$). The encoding is δ-correct (resp. ε-private), if it correct (resp. private) up to an "error" of δ (resp., ε). This is illustrated by the next example.

Example 4.2 Consider the function $f(x_1,\ldots,x_n) = x_1 \vee x_2 \vee \ldots \vee x_n$. We define a randomized encoding $\hat{f} : \{0,1\}^n \times \{0,1\}^{ns} \to \{0,1\}^s$ by $\hat{f}(x,r) = (\sum_{i=1}^n x_i r_{i,1}, \ldots, \sum_{i=1}^n x_i r_{i,s})$, where $x = (x_1,\ldots,x_n)$, $r = (r_{i,j})$ for $1 \leq i \leq n, 1 \leq j \leq s$, and addition is over GF(2). First, observe that the distribution of $\hat{f}(x, U_{ns})$ depends only on the value of $f(x)$. Specifically, let S be a simulator that outputs an s-tuple of zeroes if $f(x) = 0$, and a uniformly

[8]We restrict the decoder to be deterministic for simplicity. This restriction does not compromise generality, in the sense that one can transform a randomized decoder to a deterministic one by incorporating the coins of the former in the encoding itself.

chosen string in $\{0,1\}^s$ if $f(x) = 1$. It is easy to verify that $S(f(x))$ is distributed the same as $\hat{f}(x, U_{ns})$ for any $x \in \{0,1\}^n$. It follows that this randomized encoding is 0-private. Also, one can obtain an efficient decoder C that given a sample y from the distribution $\hat{f}(x, U_{ns})$ outputs 0 if $y = 0^s$ and otherwise outputs 1. Such an algorithm will err with probability 2^{-s}, thus $\hat{f}$ is 2^{-s}-correct.

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 polynomially bounded. We also require both the decoder and the simulator to be efficient. (This is not needed by some of the applications, but is a feature of our constructions.) We formalize these requirements below.

Definition 4.3 (Uniform randomized encoding) *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function and $l(n)$ an output length function such that $|f(x)| = l(|x|)$ for every $x \in \{0,1\}^*$. We say that $\hat{f} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ is a $\delta(n)$-correct $\epsilon(n)$-private uniform randomized encoding of f, if the following holds:*

- **Length regularity.** *There exist polynomially-bounded and efficiently computable length functions $m(n), s(n)$ such that for every $x \in \{0,1\}^n$ and $r \in \{0,1\}^{m(n)}$, we have $|\hat{f}(x,r)| = s(n)$.*
- **Efficient evaluation.** *There exists a polynomial-time* evaluation algorithm *that, given $x \in \{0,1\}^*$ and $r \in \{0,1\}^{m(|x|)}$, outputs $\hat{f}(x,r)$.*
- **δ-correctness.** *There exists a polynomial-time* decoder *C, such that for every $x \in \{0,1\}^n$ we have* $\Pr[C(1^n, \hat{f}(x, U_{m(n)})) \neq f(x)] \leq \delta(n)$.
- **ε-privacy.** *There exists a probabilistic polynomial-time* simulator *S, such that for every $x \in \{0,1\}^n$ we have $\|S(1^n, f(x)) - \hat{f}(x, U_{m(n)})\| \leq \varepsilon(n)$.*

When saying that a uniform encoding $\hat{f}$ is in a (uniform) circuit complexity class, we mean that its evaluation algorithm can be implemented by circuits in this class. For instance, we say that $\hat{f}$ is in NC^0_d if there exists a polynomial-time circuit generator G such that $G(1^n)$ outputs a d-local circuit computing $\hat{f}(x,r)$ on all $x \in \{0,1\}^n$ and $r \in \{0,1\}^{m(n)}$.

From here on, a randomized encoding of an efficiently computable function is assumed to be uniform by default. Moreover, we will freely extend the above definition to apply to a uniform collection of functions $\mathcal{F} = \{f_z\}_{z \in Z}$, for some index set $Z \subseteq \{0,1\}^*$. In such a case it is required that the encoded collection $\hat{\mathcal{F}} = \{\hat{f}_z\}_{z \in Z}$ is also uniform, in the sense that the same efficient evaluation algorithm, decoder, and simulator should apply to the entire collection when given z as an additional input. (See Appendix A for a more detailed discussion of *collections* of functions and cryptographic primitives.) Finally, for the sake of simplicity we will sometimes formulate our definitions, claims and proofs using finite functions, under the implicit understanding that they naturally extend to the uniform setting.

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. In fact, we can further relax privacy to hold only against efficient algorithms, e.g., to require that for every $x \in \{0,1\}^n$, every polynomial time algorithm A distinguishes between the distributions $S(f(x))$ and $\hat{f}(x, U_m)$ with no more than negligible advantage. Such an encoding is referred to as *computationally* private and it suffices for the purpose of many applications discussed in this paper. (Further details and additional applications appear in [4].) However, while for some of the primitives (such as OWF) computational privacy and statistical correctness will do, others (such as PRGs or one-way permutations) require even stronger properties than perfect correctness and privacy. One such additional property is that the simulator S, when invoked on a uniformly random string from $\{0,1\}^l$ (the output domain of f), will output a uniformly random string from $\{0,1\}^s$ (the output domain of $\hat{f}$). We call this property *balance*. Note that the balance requirement does not impose any uniformity condition on the output of f, which in fact can be concentrated on a strict subset of $\{0,1\}^l$.

Definition 4.4 (Balanced randomized encoding) *A randomized encoding $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$ of a function $f : \{0,1\}^n \to \{0,1\}^l$ is called* balanced *if it has a perfectly private simulator S such that $S(U_l) \equiv U_s$. We refer to S as a* balanced simulator.

A last useful property is a syntactic one: we sometimes want $\hat{f}$ to have the same additive stretch as f. Specifically, we say that $\hat{f}$ is *stretch-preserving* (with respect to f) if $s - (n+m) = l - n$, or equivalently $m = s - l$.

We are now ready to define our two main variants of randomized encoding.

Definition 4.5 (Statistical randomized encoding) *A* statistical randomized encoding *is a randomized encoding that is statistically correct and statistically private.*

Definition 4.6 (Perfect randomized encoding) *A* perfect randomized encoding *is a randomized encoding that is perfectly correct, perfectly private, balanced, and stretch-preserving.*

A combinatorial view of perfect encoding. To gain better understanding of the properties of perfect encoding, we take a closer look at the relation between a function and its encoding. Let $\hat{f} : \{0,1\}^{n+m} \to \{0,1\}^s$ be an encoding of $f : \{0,1\}^n \to \{0,1\}^l$. The following description addresses the simpler case where f is onto. Every $x \in \{0,1\}^n$ is mapped to some $y \in \{0,1\}^l$ by f, and to a 2^m-size multiset $\{\hat{f}(x,r) | r \in \{0,1\}^m\}$ which is contained in $\{0,1\}^s$. Perfect privacy means that this multiset is common to all the x's that share the same image under f; so we have a mapping from $y \in \{0,1\}^l$ to multisets in $\{0,1\}^s$ of size 2^m (such a mapping is defined by the perfect simulator). Perfect correctness means that these multisets are mutually disjoint. However, even perfect privacy and perfect correctness together do not promise that this mapping covers all of $\{0,1\}^s$. The balance property guarantees that the multisets form a perfect tiling of $\{0,1\}^s$; moreover it promises that each element in these multisets has the same multiplicity. If the encoding is also stretch-preserving, then the multiplicity of each element must be 1, so that the multisets are actually sets. Hence, 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 sets of size 2^m.

Remark 4.7 (A padding convention) We will sometimes view $\hat{f}$ as a function of a single input of length $n + m(n)$ (e.g., when using it as a OWF or a PRG). In this case, we require $m(\cdot)$ to be monotone non-decreasing, so that $n + m(n)$ uniquely determines n. We apply a standard padding technique for defining $\hat{f}$ on inputs whose length is not of the form $n + m(n)$. Specifically, if $n + m(n) + t < (n+1) + m(n+1)$ we define $\hat{f}'$ on inputs of length $n + m(n) + t$ by applying $\hat{f}_n$ on the first $n + m(n)$ bits and then appending the t additional input bits to the output of $\hat{f}_n$. This convention respects the security of cryptographic primitives such as OWF, PRG, and collision-resistant hashing, provided that $m(n)$ is efficiently computable and is sufficiently dense (both of which are guaranteed by a uniform encoding). That is, if the unpadded function $\hat{f}$ is secure with respect to its partial domain, then its padded version $\hat{f}'$ is secure in the standard sense, i.e., over the domain of all strings.[9] (See a proof for the case of OWF in [23, Proposition 2.2.3].) Note that the padded function $\hat{f}'$ has the same locality and degree as $\hat{f}$. Moreover, $\hat{f}'$ also preserves syntactic properties of $\hat{f}$; for example it preserves the stretch of $\hat{f}$, and if $\hat{f}$ is a permutation then so is $\hat{f}'$. Thus, it is enough to prove our results for the partially defined unpadded function $\hat{f}$, and keep the above conventions implicit.

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

Definition 4.8 (The classes SREN, PREN) *The class $\mathcal{SREN}$ (resp., $\mathcal{PREN}$) is the class of functions $f : \{0,1\}^* \to \{0,1\}^*$ admitting a statistical (resp., perfect) uniform randomized encoding in* NC^0.

[9] This can be generally explained by viewing each slice of the padded function $\hat{f}'$ (i.e., its restriction to inputs of some fixed length) as a *perfect* randomized encoding of a corresponding slice of $\hat{f}$.

4.2 Basic Properties

We now put forward some useful properties of randomized encodings. 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, and in particular preserves their perfectness.

Lemma 4.9 (Concatenation) *Let $f_i : \{0,1\}^n \to \{0,1\}$, $1 \le i \le l$, be the boolean functions computing the output bits of a function $f : \{0,1\}^n \to \{0,1\}^l$. If $\hat{f}_i : \{0,1\}^n \times \{0,1\}^{m_i} \to \{0,1\}^{s_i}$ is a δ-correct ε-private encoding of f_i, then the function $\hat{f} : \{0,1\}^n \times \{0,1\}^{m_1+\ldots+m_l} \to \{0,1\}^{s_1+\ldots+s_l}$ defined by $\hat{f}(x,(r_1,\ldots,r_l)) \overset{\text{def}}{=} (\hat{f}_1(x,r_1),\ldots,\hat{f}_l(x,r_l))$ is a (δl)-correct, (εl)-private encoding of f. Moreover, if all $\hat{f}_i$ are perfect then so is $\hat{f}$.*

Proof: We start with correctness. Let C_i be a δ-correct decoder for $\hat{f}_i$. Define a decoder C for $\hat{f}$ by $C(\hat{y}_1,\ldots,\hat{y}_l) = (C_1(\hat{y}_1),\ldots,C_l(\hat{y}_l))$. By a union bound argument, C is a (δl)-correct decoder for $\hat{f}$ as required.

We turn to analyze privacy. Let S_i be an ε-private simulator for $\hat{f}_i$. An (εl)-private simulator S for $\hat{f}$ can be naturally defined by $S(y) = (S_1(y_1),\ldots,S_l(y_l))$, where the invocations of the simulators S_i use independent coins. Indeed, for every $x \in \{0,1\}^n$ we have:

$$\begin{aligned}
\|S(f(x)) - \hat{f}(x,(U_{m_1},\ldots,U_{m_l}))\| &= \|(S_1(y_1),\ldots,S_l(y_l)) - (\hat{f}_1(x,U_{m_1}),\ldots,\hat{f}_l(x,U_{m_l}))\| \\
&\le \sum_{i=1}^{l} \|S_i(y_i) - \hat{f}_i(x,U_{m_i})\| \\
&\le \varepsilon l,
\end{aligned}$$

where $y = f(x)$. The first inequality follows from Fact 3.2 and the independence of the randomness used for different i, and the second from the ε-privacy of each S_i.

Note that the simulator S described above is balanced if all S_i are balanced. Moreover, if all $\hat{f}_i$ are stretch preserving, i.e., $s_i - 1 = m_i$, then we have $\sum_{i=1}^{l} s_i - l = \sum_{i=1}^{l} m_i$ and hence $\hat{f}$ is also stretch preserving. It follows that if all $\hat{f}_i$ are perfect then so is $\hat{f}$. ■

We state the following uniform version of Lemma 4.9, whose proof is implicit in the above.

Lemma 4.10 (Concatenation: uniform version) *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function, viewed as a uniform collection of functions $\mathcal{F} = \{f_{n,i}\}_{n\in\mathbb{N},1\le i\le l(n)}$; that is, $f_{n,i}(x)$ outputs the i^{th} bit of $f(x)$ for all $x \in \{0,1\}^n$. Suppose that $\hat{\mathcal{F}} = \{\hat{f}_{n,i}\}_{n\in\mathbb{N},1\le i\le l(n)}$ is a perfect (resp., statistical) uniform randomized encoding of $\mathcal{F}$. Then, the function $\hat{f} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ defined by $\hat{f}(x,(r_1,\ldots,r_{l(|x|)})) \overset{\text{def}}{=} (\hat{f}_{|x|,1}(x,r_1),\ldots,\hat{f}_{|x|,l(|x|)}(x,r_{l(|x|)}))$ is a perfect (resp., statistical) uniform randomized encoding of f.*

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 resulting encoding inherits the perfectness of the encodings from which it is composed.

Lemma 4.11 (Composition) *Let $g(x,r_g)$ be a δ_g-correct, ε_g-private encoding of $f(x)$ and $h((x,r_g),r_h)$ be a δ_h-correct, ε_h-private encoding of $g((x,r_g))$ (viewed as a single-argument function). Then, the function $\hat{f}(x,(r_g,r_h)) \overset{\text{def}}{=} h((x,r_g),r_h)$ is a $(\delta_g+\delta_h)$-correct, $(\varepsilon_g+\varepsilon_h)$-private encoding of f. Moreover, if g, h are perfect (resp., statistical) uniform randomized encodings then so is $\hat{f}$.*

Proof: We start with correctness. Let C_g be a δ_g-correct decoder for g and C_h a δ_h-correct decoder for h. Define a decoder C for $\hat{f}$ by $C(\hat{y}) = C_g(C_h(\hat{y}))$. The decoder C errs only if either C_h or C_g err. Thus, by the union bound we have for every x,

$$\begin{aligned}\Pr_{r_g,r_h}[C(\hat{f}(x,(r_g,r_h))) \neq f(x)] &\leq \Pr_{r_g,r_h}[C_h(h((x,r_g),r_h)) \neq g(x,r_g)] + \Pr_{r_g}[C_g(g(x,r_g)) \neq f(x)] \\ &\leq \delta_h + \delta_g,\end{aligned}$$

as required.

Privacy is argued similarly. Let S_g be an ε_g-private simulator for g and S_h an ε_h-private simulator for h. We define a simulator S for $\hat{f}$ by $S(y) = S_h(S_g(y))$. Letting m_g, m_h denote the randomness complexity of g, h, respectively, we have for every x,

$$\begin{aligned}\|S(f(x)) - \hat{f}(x,(U_{m_g},U_{m_h}))\| &= \|S_h(S_g(f(x))) - h((x,U_{m_g}),U_{m_h})\| \\ &\leq \|S_h(S_g(f(x))) - S_h(g(x,U_{m_g}))\| + \|S_h(g(x,U_{m_h})) - h((x,U_{m_g}),U_{m_h})\| \\ &\leq \varepsilon_g + \varepsilon_h,\end{aligned}$$

where the first inequality follows from the triangle inequality (Fact 3.1), and the second from Facts 3.3 and 3.4.

It is easy to verify that if S_g and S_h are balanced then so is S. Moreover, if g preserves the additive stretch of f and h preserves the additive stretch of g then h (hence also $\hat{f}$) preserves the additive stretch of f. Thus $\hat{f}$ is perfect if both g, h are perfect. All the above naturally carries over to the uniform setting, from which the last part of the lemma follows. ■

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

Lemma 4.12 (Unique randomness) *Suppose $\hat{f}$ is a perfect randomized encoding of f. Then, (a) $\hat{f}$ 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, (b) if f is a permutation then so is $\hat{f}$.*

Proof: Let $f : \{0,1\}^n \to \{0,1\}^l$ and $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$. To prove part (a), assume towards a contradiction that $\hat{f}$ does not satisfy the unique randomness property. Then, by perfect privacy, we have $|\mathrm{Im}(\hat{f})| < |\mathrm{Im}(f)| \cdot 2^m$. On the other hand, letting S be a balanced simulator, we have

$$\begin{aligned}|\mathrm{Im}(\hat{f})| \cdot 2^{-s} &= \Pr_{y \leftarrow U_l}[S(y) \in \mathrm{Im}(\hat{f})] \\ &\geq \Pr_{y \leftarrow U_l}[S(y) \in \mathrm{Im}(\hat{f}) | y \in \mathrm{Im}(f)] \cdot \Pr_{y \leftarrow U_l}[y \in \mathrm{Im}(f)] \\ &= 1 \cdot \frac{|\mathrm{Im}(f)|}{2^l},\end{aligned}$$

where the last equality follows from perfect privacy. Since g is stretch preserving ($s - l = m$), we get from the above that $|\mathrm{Im}(\hat{f})| \geq |\mathrm{Im}(f)| \cdot 2^m$, and derive a contradiction.

If f is a permutation then $n = l$ and since $\hat{f}$ is stretch preserving, we can write $\hat{f} : \{0,1\}^s \to \{0,1\}^s$. Thus, to prove part (b), it is enough to prove that $\hat{f}$ is injective. Suppose that $\hat{f}(x,r) = \hat{f}(x',r')$. Then, since f is injective and $\hat{f}$ is perfectly correct it follows that $x = x'$; hence, by part (a), $r = r'$ and the proof follows. ■

4.3 Constructions

In this section we construct randomized encodings in NC^0. We first review a construction from [38] of degree-3 randomizing polynomials based on mod-2 branching programs and analyze some of its properties. Next, we introduce a general locality reduction technique, allowing to transform a degree-d encoding to a $(d+1)$-local encoding. Finally, we discuss extensions to other types of BPs.

$$\begin{pmatrix} 1 & r_1^{(1)} & r_2^{(1)} & \cdot & \cdot & r_{\ell-2}^{(1)} \\ 0 & 1 & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 1 & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & 1 & \cdot & \cdot \\ 0 & 0 & 0 & 0 & 1 & r^{(1)}_{\binom{\ell-1}{2}} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} * & * & * & * & * & * \\ -1 & * & * & * & * & * \\ 0 & -1 & * & * & * & * \\ 0 & 0 & -1 & * & * & * \\ 0 & 0 & 0 & -1 & * & * \\ 0 & 0 & 0 & 0 & -1 & * \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & r_1^{(2)} \\ 0 & 1 & 0 & 0 & 0 & r_2^{(2)} \\ 0 & 0 & 1 & 0 & 0 & \cdot \\ 0 & 0 & 0 & 1 & 0 & \cdot \\ 0 & 0 & 0 & 0 & 1 & r_{\ell-2}^{(2)} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 4.1: The matrices $R_1(r^{(1)}), L(x), R_2(r^{(2)})$ (from left to right). The symbol $*$ represents a degree-1 polynomial in an input variable.

DEGREE-3 RANDOMIZING POLYNOMIALS FROM MOD-2 BRANCHING PROGRAMS [38]. Let $BP = (G, \phi, s, t)$ be a mod-2 BP of size ℓ, computing a boolean[10] function $f : \{0,1\}^n \to \{0,1\}$; that is, $f(x) = 1$ if and only if the number of paths from s to t in G_x equals 1 modulo 2. 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. Let $A(x)$ be the $\ell \times \ell$ adjacency matrix of G_x viewed as a formal matrix whose entries are degree-1 polynomials in the input variables x. Specifically, the (i,j) entry of $A(x)$ contains the value of $\phi(i,j)$ on x if (i,j) is an edge in G, and 0 otherwise. (Hence, $A(x)$ contains the constant 0 on and below the main diagonal, and degree-1 polynomials in the input variables above the main diagonal.) 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). As before, 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. (See Figure 4.1.)

Fact 4.13 ([38]) *$f(x) = \det(L(x))$, where the determinant is computed over* $\mathrm{GF}(2)$.

Proof sketch: Since G is acyclic, the number of $s-t$ paths in G_x mod 2 can be written as $(I + A(x) + A(x)^2 + \ldots + A(x)^\ell)_{s,t} = (I - A(x))^{-1}_{s,t}$ where I denotes an $\ell \times \ell$ identity matrix and all arithmetic is over $\mathrm{GF}(2)$. Recall that $L(x)$ is the submatrix of $A(x) - I$ obtained by deleting column s and row t. Hence, expressing $(I - A(x))^{-1}_{s,t}$ using the corresponding cofactor of $I - A(x)$, we have:

$$\begin{aligned} (I - A(x))^{-1}_{s,t} &= (-1)^{s+t}\frac{\det(-L(x))}{\det(I - A(x))} \\ &= \det L(x). \end{aligned}$$

■

Let $r^{(1)}$ and $r^{(2)}$ be vectors over $\mathrm{GF}(2)$ of length $\sum_{i=1}^{\ell-2} i = \binom{\ell-1}{2}$ 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. (See Figure 4.1.)

Fact 4.14 ([38]) *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, $\det(M) = \det(M')$ if and only if there exist $r^{(1)}$ and $r^{(2)}$ such that $R_1(r^{(1)})MR_2(r^{(2)}) = M'$.*

[10]The following construction generalizes naturally to a (counting) mod-p BP, computing a function $f : \{0,1\}^n \to Z_p$. In this work, however, we will only be interested in the case $p = 2$.

Proof sketch: Suppose that $R_1(r^{(1)})MR_2(r^{(2)}) = M'$ for some $r^{(1)}$ and $r^{(2)}$. Then, since $\det(R_1(r^{(1)})) = \det(R_2(r^{(2)})) = 1$, it follows that $\det(M) = \det(M')$.

For the second direction assume that $\det(M) = \det(M')$. We show that there there exist $r^{(1)}$ and $r^{(2)}$ such that $R_1(r^{(1)})MR_2(r^{(2)}) = M'$. Multiplying M by a matrix $R_1(r^{(1)})$ on the left is equivalent to adding to each row of M a linear combination of the rows below it. On the other hand, multiplying M by a matrix $R_2(r^{(2)})$ on the right is equivalent to adding to the last column of M a linear combination of the other columns. Observe that a matrix M that contains the constant -1 in each entry of its second diagonal and the constant 0 below this diagonal can be transformed, using such left and right multiplications, to a canonic matrix H_y containing -1's in its second diagonal, an arbitrary value y in its top-right entry, and 0's elsewhere. Since $\det(R_1(r^{(1)})) = \det(R_2(r^{(2)})) = 1$, we have $\det(M) = \det(H_y) = y$. Thus, when $\det(M) = \det(M') = y$ we can write $H_y = R_1(r^{(1)})MR_2(r^{(2)}) = R_1(s^{(1)})M'R_2(s^{(2)})$ for some $r^{(1)}, r^{(2)}, s^{(1)}, s^{(2)}$. Multiplying both sides by $R_1(s^{(1)})^{-1}, R_2(s^{(2)})^{-1}$, and observing that each set of matrices $R_1(\cdot)$ and $R_2(\cdot)$ forms a multiplicative group finishes the proof. ■

Lemma 4.15 (implicit in [38]) *Let BP be a mod-2 branching program computing the boolean function f. 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 showing that the encoding is stretch preserving. 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.

We now describe the decoder and the simulator. 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 $\hat{f}$.) By Facts 4.13 and 4.14, $\det(M) = \det(L(x)) = f(x)$, hence the decoder is perfect.

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.

By Facts 4.13 and 4.14, for every $x \in \{0,1\}^n$ the supports of $\hat{f}(x,U_m)$ and of $S(f(x))$ are equal. Specifically, these supports include all strings in $\{0,1\}^s$ representing matrices with determinant $f(x)$. Since the supports of $S(0)$ and $S(1)$ form a disjoint partition of the entire space $\{0,1\}^s$ (by Fact 4.14) and since S uses $m = s-1$ random bits, it follows that $|\text{support}(S(b))| = 2^m$, for $b \in \{0,1\}$. Since both the simulator and the encoding use m random bits, it follows that both distributions, $\hat{f}(x,U_m)$ and $S(f(x))$, are uniform over their support and therefore are equivalent. Finally, since the supports of $S(0)$ and $S(1)$ halve the range of $\hat{f}$ (that is, $\{0,1\}^s$), the simulator is also 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 4.16 (Locality construction) *Let $f(x) = T_1(x) + \ldots + T_k(x)$, where $f, T_1, \ldots, T_k : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ and summation is over $\mathrm{GF}(2)$. The local encoding $\hat{f} : \mathrm{GF}(2)^{n+(2k-1)} \to \mathrm{GF}(2)^{2k}$ is defined by:*

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

For example, applying the locality construction to the polynomial $x_1x_2 + x_2x_3 + x_4$ results in the encoding $(x_1x_2 - r_1, x_2x_3 - r_2, x_4 - r_3, r_1 - r'_1, r'_1 + r_2 - r'_2, r'_2 + r_3)$.

Lemma 4.17 (Locality lemma) *Let f and $\hat{f}$ be as in Construction 4.16. Then, $\hat{f}$ is a perfect randomized encoding of f. In particular, if f is a degree-d polynomial written as a 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 $s = 2k$, the encoding $\hat{f}$ is stretch preserving. Moreover, given $\hat{y} = \hat{f}(x, r)$ we can decode the value of $f(x)$ by summing up the bits of $\hat{y}$. It is not hard to verify that such a decoder never errs. To prove perfect privacy we define a simulator as follows. Given $y \in \{0, 1\}$, the simulator S uniformly chooses $2k - 1$ random bits $r_1, \ldots, r_{2k-1}$ and outputs $(r_1, \ldots, r_{2k-1}, y - (r_1 + \ldots + r_{2k-1}))$. Obviously, $S(y)$ is uniformly distributed over the $2k$-length strings whose bits sum up to y over $\mathrm{GF}(2)$. It thus suffices to show that the outputs of $\hat{f}(x, U_m)$ 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 $w \in \{0, 1\}^{2k-1}$ to the first $2k - 1$ outputs of $\hat{f}(x, U_m)$, 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 w. Indeed, for $1 \leq i \leq k$, the values of x, w_i uniquely determine r_i. For $1 \leq i \leq k - 1$, the values w_{k+i}, r_i, r'_{i-1} determine r'_i (where $r'_0 \stackrel{\text{def}}{=} 0$). Therefore, $S(f(x)) \equiv \hat{f}(x, U_m)$. Moreover, S is balanced since the supports of $S(0)$ and $S(1)$ halve $\{0, 1\}^s$ and $S(y)$ is uniformly distributed over its support for $y \in \{0, 1\}$. ■

In Appendix B we describe a graph-based generalization of Construction 4.16, which in some cases can give rise to a (slightly) more compact encoding $\hat{f}$.

We now present the main theorem of this section.

Theorem 4.18 $\oplus\mathrm{L}/poly \subseteq \mathcal{PREN}$. *Moreover, any $f \in \mathcal{PREN}$ admits a perfect randomized encoding in* NC^0_4.

Proof: The first part of the theorem is derived by combining the degree-3 construction of Lemma 4.15 together with the Locality Lemma (4.17), using the Composition Lemma (4.11) and the Concatenation Lemma (4.10).

To prove the second part, we first encode f by a perfect encoding $\hat{f}$ in NC^0 (guaranteed by the fact that f is in $\mathcal{PREN}$). Then, since $\hat{f}$ is in $\oplus\mathrm{L}/poly$, we can use our constructions (Lemmas 4.15, 4.17, 4.11, 4.10) to perfectly encode $\hat{f}$ by a function $\hat{f}'$ in NC^0_4. By the Composition Lemma (4.11), $\hat{f}'$ perfectly encodes the function f. ■

Remark 4.19 An alternative construction of perfect randomized encodings in NC^0 can be obtained using a randomizing polynomials construction from [38, Sec. 3], which is based on an information-theoretic variant of Yao's garbled circuit technique [53]. This construction yields an encoding with a (large) constant locality, without requiring an additional "locality reduction" step (of Construction 4.16). This construction is weaker than the current one in that it only efficiently applies to functions in NC^1 rather than $\oplus\mathrm{L}/poly$. For functions in NC^1, the complexity of this alternative (in terms of randomness and output length) is incomparable to the complexity of the current construction.

There are variants of the above construction that can handle non-deterministic branching programs as well, at the expense of losing perfectness [37, 38]. For instance, it is shown in [37] that if f is represented by a non-deterministic BP of size ℓ, then the function $\hat{f}(x, (R_1, R_2)) \stackrel{\text{def}}{=} R_1L(x)R_2$ is a perfectly-private, statistically-correct encoding of f provided that R_1, R_2 are uniformly random $(\ell - 1) \times (\ell - 1)$ matrices over $\mathrm{GF}(p)$, where p is prime and $p > \ell^\ell$. (The matrix $L(x)$ is as defined above, except that here it is interpreted as a matrix over $\mathrm{GF}(p)$.) To obtain an encoding over a binary alphabet, we rely on the facts that one can sample an almost-uniform element of $\mathrm{GF}(p)$ (up to a negligible statistical distance) as well as perform multiplications in $\mathrm{GF}(p)$ using NC^1 boolean circuits. Thus, we get a statistical *binary* encoding in NC^1, which can be converted (using Theorem 4.18 and the composition lemma) to a statistical encoding in NC^0_4. Based on the above, we get the following theorem:

Theorem 4.20 $\mathrm{NL}/poly \subseteq \mathcal{SREN}$. *Moreover, any $f \in \mathcal{SREN}$ admits a statistical randomized encoding in* NC^0_4.

Note that the second part of Theorem 4.20 can be proved similarly to the second part of Theorem 4.18.

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 input $f(x)$, where x is picked from U_n, succeeds only with a negligible probability. Formally,

Definition 5.1 (One-way function) *A function $f : \{0,1\}^* \to \{0,1\}^*$ is called a* one-way function *(OWF) if it satisfies the following two properties:*

- **Easy to compute**: *There exists a deterministic polynomial-time algorithm computing $f(x)$.*
- **Hard to invert**: *For every probabilistic polynomial-time algorithm, B, the probability $\Pr_{x \leftarrow U_n}[B(1^n, f(x)) \in f^{-1}(f(x))]$ is negligible in n (where the probability is taken over a uniform choice of x and the internal coin tosses of B).*

The function f is called weakly one-way *if the second requirement is replaced with the following (weaker) one:*

- **Slightly hard to invert**: *There exists a polynomial $p(\cdot)$, such that for every probabilistic polynomial-time algorithm, B, and all sufficiently large n's $\Pr_{x \leftarrow U_n}[B(1^n, f(x)) \notin f^{-1}(f(x))] > \frac{1}{p(n)}$ (where the probability is taken over a uniform choice of x and the internal coin tosses of B).*

The above definition naturally extends to functions whose domain is restricted to some infinite subset $I \subset \mathbb{N}$ of the possible input lengths, such as ones defined by a randomized encoding $\hat{f}$. As argued in Remark 4.7, such a partially defined OWF can be augmented into a fully defined OWF provided that the set I is polynomially-dense and efficiently recognizable (which is a feature of functions $\hat{f}$ obtained via a uniform encodings).

5.1 Key Lemmas

In the following we show that a perfectly correct and statistically private 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. The case of a statistical randomized encoding that does not enjoy perfect correctness is more involved and will be dealt with later in this section.

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 single-argument function, is also hard to invert.*

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

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

CORRECTNESS: We analyze the success probability of B on input $(1^n, f(x))$ where $x \leftarrow U_n$. Let us assume for a moment that the simulator S is perfect. Observe that, by perfect correctness, if $f(x) \neq f(x')$ then the support sets of $\hat{f}(x, U_m)$ and $\hat{f}(x', U_m)$ are disjoint. Moreover, by perfect privacy the string $\hat{y}$, generated by $\hat{B}$, is always in the support of $\hat{f}(x, U_m)$. Hence, if $\hat{B}$ succeeds (that is, indeed $\hat{y} = \hat{f}(x', r')$) then so does B (namely, $f(x') = y$).

Finally, observe that (by Fact 3.4) the input $\hat{y}$ on which B invokes $\hat{B}$ is distributed identically to $\hat{f}_n(U_n, U_{m(n)})$, and therefore B succeeds with probability $\geq \phi(n+m)$. Formally, we can write,

$$\begin{aligned}
\Pr_{x \leftarrow U_n}[B(1^n, f(x)) \in f^{-1}(f(x))] &\geq \Pr_{x \leftarrow U_n, \hat{y} \leftarrow S(1^n, f(x))}[\hat{B}(1^{n+m}, \hat{y}) \in \hat{f}^{-1}(\hat{y})] \\
&= \Pr_{x \leftarrow U_n, r \leftarrow U_{m(n)}}[\hat{B}(1^{n+m}, \hat{f}_n(x,r)) \in \hat{f}^{-1}(\hat{f}(x,r))] \\
&\geq \phi(n+m).
\end{aligned}$$

When S is only statistically private, we lose negligible success probabilities in the first and second transitions. The first loss is due to the fact that the simulator invoked on $y = f(x)$ might output (with negligible probability) $\hat{y}$ which is not in the support of $\hat{f}(x, U_m)$. The second loss is due to the fact that the input $\hat{y}$ on which B invokes $\hat{B}$ is not distributed identically to $\hat{f}(U_n, U_m)$, on which $\hat{B}$ is guaranteed to succeed with probability $\phi(n+m)$. However, it follows from Fact 3.4 that the second loss is also negligible. Thus, if S is $\varepsilon(n)$-private for a negligible function $\varepsilon(\cdot)$, we have

$$\begin{aligned}
\Pr_{x \leftarrow U_n}[B(1^n, f(x)) \in f^{-1}(f(x))] &\geq \Pr_{x \leftarrow U_n, \hat{y} \leftarrow S(1^n, f(x))}[\hat{B}(1^{n+m}, \hat{y}) \in \hat{f}^{-1}(\hat{y})] - \varepsilon(n) \\
&\geq \Pr_{x \leftarrow U_n, r \leftarrow U_{m(n)}}[\hat{B}(1^{n+m}, \hat{f}_n(x,r)) \in \hat{f}^{-1}(\hat{f}(x,r))] - \varepsilon(n) - \varepsilon(n) \\
&\geq \phi(n+m) - 2\varepsilon(n) > \frac{1}{q(n+m)} - 2\varepsilon(n) > \frac{1}{q'(n)},
\end{aligned}$$

for some polynomial $q'(\cdot)$ and infinitely many n's. It follows that f is not hard to invert, in contradiction to the hypothesis. ■

The efficiency of the simulator S is essential for Lemma 5.2 to hold. Indeed, without this requirement one could encode any one-way permutation f by the identity function $\hat{f}(x) = x$, which is obviously not one-way. (Note that the output of $\hat{f}(x)$ can be simulated inefficiently based on $f(x)$ by inverting f.)

The perfect correctness requirement is also essential for Lemma 5.2 to hold. To see this, consider the following example. Suppose f is a one-way permutation. Consider the encoding $\hat{f}(x,r)$ which equals $f(x)$ except if r is the all-zero string, in which case $\hat{f}(x,r) = x$. This is a statistically-correct and statistically-private encoding, but $\hat{f}$ is easily invertible since on value $\hat{y}$ the inverter can always return $\hat{y}$ itself as a possible pre-image. Still, we show below that such an $\hat{f}$ (which is only statistically correct) is a *distributionally* one-way function. We will later show how to turn a distributionally one-way function in NC^0 into a OWF in NC^0.

Definition 5.3 (Distributionally one-way function [35]) *A polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ is called* distributionally one-way *if there exists a positive polynomial $p(\cdot)$ such that for every probabilistic polynomial-time algorithm, B, and all sufficiently large n's, $\|(B(1^n, f(U_n)), f(U_n)) - (U_n, f(U_n))\| > \frac{1}{p(n)}$.*

Before proving that a statistical randomized encoding of a OWF is distributionally one-way, we need the following lemma.

Lemma 5.4 *Let $f, g : \{0,1\}^* \to \{0,1\}^*$ be two functions that differ on a negligible fraction of their domain; that is, $\Pr_{x \leftarrow U_n}[f(x) \neq g(x)]$ is negligible in n. Suppose that g is slightly hard to invert (but is not necessarily computable in polynomial time) and that f is computable in polynomial time. Then, f is distributionally one-way.*

Proof: Let f_n and g_n be the restrictions of f and g to n-bit inputs, that is $f = \{f_n\}, g = \{g_n\}$, and define $\varepsilon(n) \overset{\text{def}}{=} \Pr_{x \leftarrow U_n}[f(x) \neq g(x)]$. Let $p(n)$ be the polynomial guaranteed by the assumption that g is slightly hard to invert. Assume, towards a contradiction, that f is not distributionally one-way. Then, there exists a polynomial-time algorithm, B, such that for infinitely many n's, $\|(B(1^n, f_n(U_n)), f_n(U_n)) - (U_n, f_n(U_n))\|$

$\leq \frac{1}{2p(n)}$. Since $(U_n, f_n(U_n)) \equiv (x', f_n(U_n))$ where $x' \leftarrow f_n^{-1}(f_n(U_n))$, we get that for infinitely many n's $\|(B(1^n, f_n(U_n)), f_n(U_n)) - (x', f_n(U_n))\| \leq \frac{1}{2p(n)}$. It follows that for infinitely many n's

$$\Pr[B(1^n, f(U_n)) \in g_n^{-1}(f_n(U_n))] \geq \Pr_{x' \leftarrow f_n^{-1}(f_n(U_n))}[x' \in g_n^{-1}(f_n(U_n))] - \frac{1}{2p(n)}. \tag{5.1}$$

We show that B inverts g with probability greater than $1 - \frac{1}{p(n)}$ and derive a contradiction. Specifically, for infinitely many n's we have:

$$\begin{aligned}
\Pr[B(1^n, g_n(U_n)) \in g_n^{-1}(g_n(U_n))] &\geq \Pr[B(1^n, f_n(U_n)) \in g_n^{-1}(f_n(U_n))] - \varepsilon(n) && \text{(since } f, g \text{ are } \varepsilon\text{-close)}\\
&\geq \Pr_{x' \leftarrow f_n^{-1}(f_n(U_n))}[x' \in g_n^{-1}(f(U_n))] - \frac{1}{2p(n)} - \varepsilon(n) && \text{(by Eq. 5.1)}\\
&= \Pr_{x' \leftarrow f_n^{-1}(f_n(U_n))}[g_n(x') = f_n(U_n)] - \frac{1}{2p(n)} - \varepsilon(n)\\
&= \Pr_{x' \leftarrow f_n^{-1}(f_n(U_n))}[g_n(x') = f_n(x')] - \frac{1}{2p(n)} - \varepsilon(n) && \text{(since } f(U_n) = f(x'))\\
&= 1 - \varepsilon(n) - \frac{1}{2p(n)} - \varepsilon(n) && \text{(since } x' \equiv U_n)\\
&\geq 1 - \frac{1}{p(n)} && \text{(since } \varepsilon \text{ is negligible)}.
\end{aligned}$$

■

We now use Lemma 5.4 to prove the distributional one-wayness of a statistically-correct encoding $\hat{f}$ based on the one-wayness of a related, perfectly correct, encoding g.

Lemma 5.5 *Suppose that $f : \{0,1\}^* \to \{0,1\}^*$ is a one-way function and $\hat{f}(x, r)$ is a statistical randomized encoding of f. Then $\hat{f}$, viewed as a single-argument function, is distributionally one-way.*

Proof: Let C and S be the decoder and the simulator of $\hat{f}$. Define the function $\hat{g}(x, r)$ in the following way: if $C(\hat{f}(x, r)) \neq f(x)$ then $\hat{g}(x, r) = \hat{f}(x, r')$ for some r' such that $C(\hat{f}(x, r')) = f(x)$ (such an r' exists by the statistical correctness); otherwise, $\hat{g}(x, r) = \hat{f}(x, r)$. Obviously, $\hat{g}$ is a perfectly correct encoding of f (as C perfectly decodes $f(x)$ from $\hat{g}(x, r)$). Moreover, by the statistical correctness of C, we have that $\hat{f}(x, \cdot)$ and $\hat{g}(x, \cdot)$ differ only on a negligible fraction of the r's. It follows that $\hat{g}$ is also a statistically-private encoding of f (because $\hat{g}(x, U_m) \overset{s}{\approx} \hat{f}(x, U_m) \overset{s}{\approx} S(f(x))$). Since f is hard to invert, it follows from Lemma 5.2 that $\hat{g}$ is also hard to invert. (Note that $\hat{g}$ might not be computable in polynomial time; however the proof of Lemma 5.2 only requires that the simulator's running time and the randomness complexity of $\hat{g}$ be polynomially bounded.) Finally, it follows from Lemma 5.4 that $\hat{f}$ is distributionally one-way as required. ■

5.2 Main Results

Based on the above, we derive the main theorem of this section:

Theorem 5.6 *If there exists a OWF in $\mathcal{SREN}$ then there exists a OWF in NC^0_4.*

Proof: Let f be a OWF in $\mathcal{SREN}$. By Lemma 5.5, we can construct a distributional OWF $\hat{f}$ in NC^0, and then apply a standard transformation (cf. [35, Lemma 1], [23, p. 96], [52]) to convert $\hat{f}$ to a OWF $\hat{f}'$ in NC^1. This transformation consists of two steps: Impagliazzo and Luby's NC^1 construction of weak OWF from distributional

OWF [35], and Yao's NC^0 construction of a (standard) OWF from a weak OWF [52] (see [23, Section 2.3]).[11] Since $\mathrm{NC}^1 \subseteq \mathcal{PREN}$ (Theorem 4.18), we can use Lemma 5.2 to encode $\hat{f}'$ by a OWF in NC^0, in particular, by one with locality 4. ■

Combining Lemmas 5.2, 4.12 and Theorem 4.18, we get a similar result for one-way permutations (OWPs).

Theorem 5.7 *If there exists a one-way permutation in $\mathcal{PREN}$ then there exists a one-way permutation in* NC^0_4.

In particular, using Theorems 4.18 and 4.20, we conclude that a OWF (resp., OWP) in $\mathrm{NL}/poly$ (resp., $\oplus\mathrm{L}/poly$) implies a OWF (resp., OWP) in NC^0_4.

Theorem 5.7 can be extended to trapdoor permutations (TDPs) provided that the perfect encoding satisfies the following *randomness reconstruction* property: given x and $\hat{f}(x,r)$, the randomness r can be efficiently recovered. If this is the case, then the trapdoor of f can be used to invert $\hat{f}(x,r)$ in polynomial time (but not in NC^0). Firstly, we compute $f(x)$ from $\hat{f}(x,r)$ using the decoder; secondly, we use the trapdoor-inverter to compute x from $f(x)$; and finally, we use the randomness reconstruction algorithm to compute r from x and $\hat{f}(x,r)$. The randomness reconstruction property is satisfied by the randomized encodings described in Section 4.3 and is preserved under composition and concatenation. Thus, the existence of trapdoor permutations computable in NC^0_4 follows from their existence in $\oplus\mathrm{L}/poly$.

More formally, a collection of permutations $\mathcal{F} = \{f_z : D_z \to D_z\}_{z \in Z}$ is referred to as a trapdoor permutation if there exist probabilistic polynomial-time algorithms (I, D, F, F^{-1}) with the following properties. Algorithm I is an index selector algorithm that on input 1^n selects an index z from Z and a corresponding trapdoor for f_z; algorithm D is a domain sampler that on input z samples an element from the domain D_z; F is a function evaluator that given an index z and x returns $f_z(x)$; and F^{-1} is a trapdoor-inverter that given an index z, a corresponding trapdoor t and $y \in D_z$ returns $f_z^{-1}(y)$. Additionally, the collection should be hard to invert, similarly to a standard collection of one-way permutations. (For formal definition see [23, Definition 2.4.4].) By the above argument we derive the following theorem.

Theorem 5.8 *If there exists a trapdoor permutation $\mathcal{F}$ whose function evaluator F is in $\oplus\mathrm{L}/poly$ then there exists a trapdoor permutation $\hat{\mathcal{F}}$ whose function evaluator $\hat{F}$ is in* NC^0_4.

Remarks on Theorems 5.6, 5.7 and 5.8.

1. (Constructiveness) In Section 4.3, we give a constructive way of transforming a branching program representation of a function f into an NC^0 circuit computing its encoding $\hat{f}$. It follows that Theorems 5.6, 5.7 can be made constructive in the following sense: there exists a polynomial-time *compiler* transforming a branching program representation of a OWF (resp., OWP) f into an NC^0 representation of a corresponding OWF (resp., OWP) $\hat{f}$. A similar result holds for other cryptographic primitives considered in this paper.

2. (Preservation of security, a finer look) Loosely speaking, the main security loss in the reduction follows from the expansion of the input. (The simulator's running time only has a minor effect on the security, since it is added to the overall running-time of the adversary.) Thus, to achieve a level of security similar 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 construction), we get $m(n) = l(n) \cdot \ell(n)^{O(1)}$, where $l(n)$ is the output length of f. If the degree of all nodes in the BPs is bounded by a constant, the complexity is $m(n) = O(l(n) \cdot \ell(n)^2)$. It is possible to further reduce the overhead of randomized encoding for specific representation models, such as balanced formulas, using constructions of randomizing polynomials from [38, 15].

[11]We will later show a degree preserving transformation from a distributional OWF to a OWF (Lemma 8.2); however, in the current context the standard transformation suffices.

3. (Generalizations) The proofs of the above theorems carry over to OWF whose security holds against efficient *non-uniform* adversaries (inverters). The same is true for all cryptographic primitives considered in this work. The proofs also naturally extend to the case of *collections* of OWF and OWP (see Appendix A for discussion).

4. (Concrete assumptions) The existence of a OWF in $\mathcal{SREN}$ (in fact, even in NC^1) follows from the intractability of factoring and lattice problems [2]. The existence of a OWF *collection* in $\mathcal{SREN}$ follows from the intractability of the discrete logarithm problem. Thus, we get OWFs in NC^0_4 under most standard cryptographic assumptions. In the case of OWP, we can get a collection of OWPs in NC^0_4 based on discrete logarithm [11, 52] (see also Appendix A) or RSA with a small exponent [49].[12] The latter assumption is also sufficient for the construction of TDP in NC^0_4.

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 $l(n) > n$, where we refer to the function $l(n) - n$ as the *stretch* of the generator; and (2) any "computationally restricted procedure" 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 space-bounded computation, have also been considered. The reader is referred to [21, Chapter 3] for a comprehensive and unified treatment of pseudorandomness.

We start by considering cryptographic PRGs. We show that a *perfect* randomized encoding of such a PRG is also a PRG. We then obtain a similar result for other types of PRGs.

6.1 Cryptographic Generators

Definition 6.1 (Pseudorandom generator) *A pseudorandom generator (PRG) is a polynomial-time computable function, $G : \{0,1\}^n \to \{0,1\}^{l(n)}$, satisfying the following two conditions:*

- **Expansion***: $l(n) > n$, for all $n \in \mathbb{N}$.*
- **Pseudorandomness***: For every probabilistic polynomial-time algorithm, D, the distinguishing advantage $|\Pr[D(1^n, G(U_n)) = 1] - \Pr[D(1^n, U_{l(n)}) = 1]|$ is negligible in n.*

Remark 6.2 (PRGs with sublinear stretch) An NC^0 PRG, G, that stretches its input by a single bit can be transformed into another NC^0 PRG, G', with stretch $l'(n) - n = n^c$ for an arbitrary constant $c < 1$. This can be done by applying G on n^c blocks of n^{1-c} bits and concatenating the results. Since the output of any PRG is computationally-indistinguishable from the uniform distribution even by a polynomial number of samples (see [23, Theorem 3.2.6]), the block generator G' is also a PRG. This PRG gains a pseudorandom bit from every block, and therefore stretches $n^c n^{1-c} = n$ input bits to $n + n^c$ output bits. Obviously, G' has the same locality as G.

Remark 6.2 also applies to other types of generators considered in this section, and therefore we only use a crude classification of the stretch as being "sublinear", "linear" or "superlinear".

Lemma 6.3 *Suppose $G : \{0,1\}^n \to \{0,1\}^{l(n)}$ is a PRG and $\hat{G} : \{0,1\}^n \times \{0,1\}^{m(n)} \to \{0,1\}^{s(n)}$ is a uniform perfect randomized encoding of G. Then $\hat{G}$, viewed as a single-argument function, is also a PRG.*

[12] Rabin's factoring-based OWP collection [47] seems insufficient for our purposes, as it cannot be defined over the set of *all* strings of a given length. The standard modification (cf. [24, p. 767]) does not seem to be in $\oplus\mathrm{L}/poly$.

Proof: Since $\hat{G}$ is stretch preserving, it is guaranteed to expand its seed. To prove the pseudorandomness of its output, we again use a reducibility argument. Assume, towards a contradiction, that there exists an efficient distinguisher $\hat{D}$ that distinguishes between U_s and $\hat{G}(U_n, U_m)$ with some non-negligible advantage ϕ; i.e., ϕ such that $\phi(n+m) > \frac{1}{q(n+m)}$ for some polynomial $q(\cdot)$ and infinitely many n's. We use $\hat{D}$ to 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 resulting $\hat{y}$. If y is taken from U_l then the simulator, being balanced, outputs $\hat{y}$ that is distributed as U_s. On the other hand, if y is taken from $G(U_n)$ then, by Fact 3.4, 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 $\hat{G}$. That is, the advantage of D is $\phi'(n) = \phi(n+m)$. Since $m(n)$ is polynomial, this advantage ϕ' is not only non-negligible in $n+m$ but also in n, in contradiction to the hypothesis. ■

Remark 6.4 (The role of balance and stretch preservation) Dropping either the balance or stretch preservation requirements, Lemma 6.3 would no longer hold. To see this consider the following two examples. Let G be a PRG, and let $\hat{G}(x,r) = G(x)$. Then, $\hat{G}$ is a perfectly correct, perfectly private, and balanced randomized encoding of G (the balanced simulator is $S(y) = y$). However, when r is sufficiently long, $\hat{G}$ does not expand its seed. On the other hand, we can define $\hat{G}(x,r) = G(x)0$, where r is a single random bit. Then, $\hat{G}$ is perfectly correct, perfectly private and stretch preserving, but its output is not pseudorandom.

Using Lemma 6.3 and Theorem 4.18, we get:

Theorem 6.5 *If there exists a pseudorandom generator in $\mathcal{PREN}$ (in particular, in $\oplus\mathrm{L}/poly$) then there exists a pseudorandom generator in NC^0_4.*

As in the case of OWF, an adversary that breaks the transformed generator $\hat{G}$ can break, in essentially the same time, the original generator G. Therefore, again, although the new PRG uses extra $m(n)$ random input bits, it is not more secure than the original generator applied to n bits. Moreover, we stress that the 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 typically superlinear in the input length n.

Remark 6.6 (On the existence of a PRG in $\mathcal{PREN}$) The existence of PRGs in $\mathcal{PREN}$ follows from most standard concrete intractability assumptions. In particular, using Theorem 6.5 (applied to PRG collections) one can construct a collection of PRGs in NC^0_4 based on the intractability of factoring [39, 44] and discrete logarithm [11, 52]. The existence of PRGs in $\mathcal{PREN}$ also follows from the existence in $\mathcal{PREN}$ of any *regular* OWF; i.e., a OWF $f = \{f_n\}$ that maps the same (polynomial-time computable) number of elements in $\{0,1\}^n$ to every element in $\mathrm{Im}(f_n)$. (This is the case, for instance, for any one-to-one OWF.) Indeed, the PRG construction from [33] (Theorem 5.4), when applied to a regular OWF f, involves only the computation of universal hash functions and hard-core bits, which can all be implemented in NC^1.[13] Thus a regular OWF in $\mathcal{PREN}$ can be first transformed into a regular OWF in NC^0 and then, using [33], to a PRG in NC^1. Combined with Theorem 6.5, this yields a PRG in NC^0_4 based on any regular OWF in $\mathcal{PREN}$.[14] This way, for example, one can construct a (single) PRG in NC^0_4 based on the intractability of

[13]In the general case (when the OWF f is not regular) the construction of Håstad et al. (see [33, Construction 7.1]) is not in uniform NC^1, as it requires an additional nonuniform advice of logarithmic length. This (slightly) non-uniform NC^1 construction translates into a *polynomial-time* construction by applying the following steps: (1) construct a polynomial number of PRG candidates (each using a different guess for the non-uniform advice); (2) increase the stretch of each of these candidates using the standard transformation of Goldreich and Micali (cf. [23, Theorem 3.3.3]); (3) take the exclusive-or of all PRG candidates to obtain the final PRG. The second step requires polynomially many sequential applications of the PRGs, and therefore this construction is not in NC^1. (If we skip the second step the resulting generator will not stretch its input.)

[14] In fact, the same result can be obtained under a relaxed regularity requirement. Specifically, for each n and $y \in \mathrm{Im}(f_n)$ define the value $D_{f,n}(y) = \log|f_n^{-1}(y)|$ and the random variable $R_n = D_{f,n}(f(U_n))$. The NC^1 construction of [33, Construction 7.1] needs to approximate, in $\mathrm{poly}(n)$ time, the expectations of both R_n and R_n^2. This is trivially possible when f is regular in the strict sense defined above, since in this case R_n is concentrated on a single (efficiently computable) value. Using a recent NC^1 construction from [30], only the expectation of R_n^2 needs to be efficiently approximated. We finally note that in a non-uniform computation model one can rely on [33] (which gives a nonuniform-NC^1 construction of a PRG from any OWF) and get a PRG in *nonuniform*-NC^0_4 from *any* OWF in $\mathcal{SREN}$.

lattice problems [33, 2].

Remark 6.7 (On unconditional NC^0 reductions from PRG to OWF) Our machinery can be used to obtain an NC^0 reduction from a PRG to any regular OWF (in particular, to any one-to-one OWF), regardless of the complexity of f.[15] Moreover, this reduction only makes a *black-box* use of the underlying regular OWF f (given its regularity parameter $|\mathrm{Im}(f_n)|$). The general idea is to encode the NC^1 construction of [33, Construction 7.1] into a corresponding NC^0 construction. Specifically, suppose $G(x) = g(x, f(q_1(x)), \ldots, f(q_m(x)))$ defines a black-box construction of a PRG G from a OWF f, where g is in $\mathcal{PREN}$ and the q_i's are in NC^0. (The functions $g, q_1, ..., q_m$ are fixed by the reduction and do not depend on f.) Then, letting $\hat{g}((x, y_1, \ldots, y_m), r)$ be a perfect NC^0 encoding of g, the function $\hat{G}(x, r) = \hat{g}((x, f(q_1(x)), \ldots, f(q_m(x))), r)$ perfectly encodes G, and hence defines a black-box NC^0 reduction from a PRG to a OWF. The construction of [33, Construction 7.1] is of the form of $G(x)$ above,[16] assuming that f is regular. Thus, $\hat{G}$ defines an NC^0 reduction from a PRG to a regular OWF.

Comparison with lower bounds. The results of [43] rules out the existence of a superlinear-stretch cryptographic PRG in NC^0_4. Thus our NC^0_4 cryptographic PRGs are not far from optimal despite their sublinear stretch. In addition, it is easy to see that there is no PRG with degree 1 or locality 2 (since we can easily decide whether a given string is in the range of such a function). It seems likely that a cryptographic PRG with locality 3 and degree 2 can be constructed (e.g., based on its existence in a higher complexity class), but our positive result is one step far in terms of both locality and degree. (See also Table 6.1.)

6.2 ε-Biased Generators

The proof of Lemma 6.3 uses the balanced simulator to transform a distinguisher for a PRG G into a distinguisher for its encoding $\hat{G}$. Therefore, if this transformation can be made linear, then the security reduction goes through also in the case of ε-biased generators.

Definition 6.8 (ε-biased generator) *An ε-*biased generator *is a polynomial-time computable function, $G : \{0,1\}^n \rightarrow \{0,1\}^{l(n)}$, satisfying the following two conditions:*

- **Expansion***: $l(n) > n$, for all $n \in \mathbb{N}$.*
- **ε-bias***: For every linear function $L : \{0,1\}^{l(n)} \rightarrow \{0,1\}$ and all sufficiently large n's*

$$|\Pr[L(G(U_n)) = 1] - \Pr[L(U_{l(n)}) = 1]| < \varepsilon(n)$$

(where a function L is linear *if its degree over $\mathrm{GF}(2)$ is 1). By default, the function $\varepsilon(n)$ is required to be negligible.*

Lemma 6.9 *Let G be an ε-biased generator and $\hat{G}$ a perfect randomized encoding of G. Assume that the balanced simulator S of $\hat{G}$ is* linear *in the sense that $S(y)$ outputs a randomized linear transformation of y (which is not necessarily a linear function of the simulator's randomness). Then, $\hat{G}$ is also an ε-biased generator.*

Proof: Let $G : \{0,1\}^n \rightarrow \{0,1\}^{l(n)}$ and let $\hat{G} : \{0,1\}^n \times \{0,1\}^{m(n)} \rightarrow \{0,1\}^{s(n)}$. Assume, towards a contradiction, that $\hat{G}$ is not ε-biased; that is, for some linear function $L : \{0,1\}^{s(n)} \rightarrow \{0,1\}$ and infinitely many n's, $|\Pr[L(\hat{G}(U_{n+m})) = 1] - \Pr[L(U_s) = 1]| > \frac{1}{p(n+m)} > \frac{1}{p'(n)}$, where $m = m(n)$, $s = s(n)$, and $p(\cdot), p'(\cdot)$ are polynomials. Using the balance property we get,

$$|\Pr[L(S(G(U_n))) = 1] - \Pr[L(S(U_l)) = 1]| = |\Pr[L(\hat{G}(U_{n+m})) = 1] - \Pr[L(U_s) = 1]| > \frac{1}{p'(n)},$$

[15] Viola, in a concurrent work [50], obtains an AC^0 reduction of this type.

[16] The functions $q_1, ..., q_m$ are simply projections there. Interestingly, the recent NC^1 construction from [30] is not of the above form and thus we cannot encode it into an (unconditional) NC^0 construction.

where S is the balanced simulator of $\hat{G}$ and the probabilities are taken over the inputs as well as the randomness of S. By an averaging argument we can fix the randomness of S to some string ρ, and get $|\Pr[L(S_\rho(G(U_n))) = 1] - \Pr[L(S_\rho(U_{l(n)})) = 1]| > \frac{1}{p'(n)}$, where S_ρ is the deterministic function defined by using the constant string ρ as the simulator's random input. By the linearity of the simulator, the function $S_\rho : \{0,1\}^l \to \{0,1\}^s$ is linear; therefore the composition of L and S_ρ is also linear, and so the last inequality implies that G is not ε-biased in contradiction to the hypothesis. ■

We now argue that the balanced simulators obtained in Section 4.3 are all linear in the above sense. In fact, these simulators satisfy a stronger property: for every fixed random input of the simulator, each bit of the simulator's output is determined by a single bit of its input. This simple structure is due to the fact that we encode non-boolean functions by concatenating the encodings of their output bits. We state here the stronger property as it will be needed in the next subsection.

Observation 6.10 *Let S be a simulator of a randomized encoding (of a function) that is obtained by concatenating simulators (i.e., S is defined as in the proof of Lemma 4.9). Then, fixing the randomness ρ of S, the simulator's computation has the following simple form: $S_\rho(y) = \sigma_1(y_1)\sigma_2(y_2)\cdots\sigma_l(y_l)$, where each σ_i maps y_i (i.e., the i^{th} bit of y) to one of two fixed strings. In particular, S computes a randomized degree-1 function of its input.*

Recall that the balanced simulator of the NC^0_4 encoding for functions in $\oplus\mathrm{L}/poly$ (promised by Theorem 4.18) is obtained by concatenating the simulators of boolean functions in $\oplus\mathrm{L}/poly$. By Observation 6.10, this simulator is linear. Thus, by Lemma 6.9, we can construct a sublinear-stretch ε-biased generator in NC^0_4 from any ε-biased generator in $\oplus\mathrm{L}/poly$. In fact, one can easily obtain a nontrivial ε-biased generator even in NC^0_3 by applying the locality construction to each of the bits of the degree-2 generator defined by $G(x, x') = (x, x', \langle x, x' \rangle)$, where $\langle \cdot, \cdot \rangle$ denotes inner product modulo 2. Again, the resulting encoding is obtained by concatenation and thus, by Observation 6.10 and Lemma 6.9, is also ε-biased. (This generator actually fools a much larger class of statistical tests; see Section 6.3 below.) Thus, we have:

Theorem 6.11 *There is a (sublinear-stretch) ε-biased generator in NC^0_3.*

Building on a construction of Mossel et al., it is in fact possible to achieve linear stretch in NC^0_3. Namely,

Theorem 6.12 *There is a linear-stretch ε-biased generator in NC^0_3.*

Proof: Mossel et al. present an ε-biased generator in NC^0 with degree 2 and linear stretch ([43], Theorem 13).[17] Let G be their ε-biased generator. We can apply the locality construction (4.16) to G (using concatenation) and get, by Lemma 6.9 and Observation 6.10, an ε-biased generator $\hat{G}$ in NC^0_3. We now relate the stretch of $\hat{G}$ to the stretch of G. Let $n, \hat{n}$ be the input complexity of $G, \hat{G}$ (resp.), let $s, \hat{s}$ be the output complexity of $G, \hat{G}$ (resp.), and let $c \cdot n$ be the stretch of G, where c is a constant. The generator $\hat{G}$ is stretch preserving, hence $\hat{s} - \hat{n} = s - n = c \cdot n$. Since G is in NC^0, each of its output bits can be represented as a polynomial that has a constant number of monomials and thus the locality construction adds only a constant number of random bits for each output bit of G. Therefore, the input length of $\hat{G}$ is linear in the input length of G. Hence, $\hat{s} - \hat{n} = s - n = c \cdot n = \hat{c} \cdot \hat{n}$ for some constant $\hat{c}$ and thus $\hat{G}$ has a linear stretch. ■

[17] In fact, the generator of [43, Theorem 13] is in *nonuniform* NC^0_5 (and it has a slightly superlinear stretch). However, a similar construction gives an ε-biased generator in *uniform* NC^0 with degree 2 and linear stretch. (The locality of this generator is large but constant.) This can be done by replacing the probabilistic construction given in [43, Lemma 12] with a uniform construction of constant-degree bipartite expander with some "good" expansion properties – such a construction is given in [13, Theorem 7.1].

Comparison with lower bounds. It is not hard to see that there is no ε-biased generator with degree 1 or locality 2.[18] In [16] it was shown that there is no superlinear-stretch ε-biased generator in NC^0_3. Thus, our linear-stretch NC^0_3 generator (building on the one from [43]) is not only optimal with respect to locality and degree but is also essentially optimal with respect to stretch.

6.3 Generators for Space-Bounded Computation

We turn to the case of PRGs for space-bounded computation. A standard way of modeling a randomized space-bounded Turing machine is by having a random tape on which the machine can access the random bits one by one but cannot "go back" and view previous random bits (i.e., any bit that the machine wishes to remember, it must store in its limited memory). For the purpose of derandomizing such machines, it suffices to construct PRGs that fool any space-bounded distinguisher having a similar one-way access to its input. Following Babai et al. [6], we refer to such distinguishers as *space-bounded distinguishers*.

Definition 6.13 ([6]) (Space-bounded distinguisher) *A* space-$s(n)$ distinguisher *is a deterministic Turing machine M, and an infinite sequence of binary strings $a = (a_1, \ldots, a_n, \ldots)$ called the advice strings, where $|a_n| = 2^{O(s(n))}$. The machine has the following tapes: read-write work tapes, a read-only advice tape, and a read-only input tape on which the tested input string, y, is given. The input tape has a one-way mechanism to access the tested string; namely, at any point it may request the next bit of y. In addition, only $s(n)$ cells of the work tapes can be used. Given an n-bit input, y, the output of the distinguisher, $M^a(y)$, is the (binary) output of M where y is given on the input tape and a_n is given on the advice tape.*

This class of distinguishers is a proper subset of the distinguishers that can be implemented by a space-$s(n)$ Turing machine with a two-way access to the input. Nevertheless, even log-space distinguishers are quite powerful, and many distinguishers fall into this category. In particular, this is true for the class of *linear* distinguishers considered in Section 6.2.

Definition 6.14 (PRG for space-bounded computation) *We say that a polynomial-time computable function $G : \{0,1\}^n \to \{0,1\}^{l(n)}$ is a PRG for space $s(n)$ if $l(n) > n$ and $G(U_n)$ is indistinguishable from $U_{l(n)}$ to any space-$s(n)$ distinguisher. That is, for every space-$s(n)$ distinguisher M^a, the distinguishing advantage $|\Pr[M^a(G(U_n)) = 1] - \Pr[M^a(U_{l(n)}) = 1]|$ is negligible in n.*

Several constructions of high-stretch PRGs for space-bounded computation exist in the literature (e.g., [6, 45]). In particular, a PRG for logspace computation from [6] can be computed using logarithmic space, and thus, by Theorem 4.18, admits an efficient perfect encoding in NC^0_4. It can be shown (see proof of Theorem 6.15) that this NC^0_4 encoding fools logspace distinguishers as well; hence, we can reduce the security of the randomized encoding to the security of the encoded generator, and get an NC^0_4 PRG that fools logspace computation. However, as in the case of ε-biased generators, constructing such PRGs with a low stretch is much easier. In fact, the same "inner product" generator we used in Section 6.2 can do here is well.

Theorem 6.15 *There exists a (sublinear-stretch) PRG for sublinear-space computation in* NC^0_3.

Proof: Consider the inner product generator $G(x, x') = (x, x', \langle x, x' \rangle)$, where $x, x' \in \{0,1\}^n$. It follows from the average-case hardness of the inner product function for two-party communication complexity [14] that G fools all sublinear-space distinguishers. (Indeed, a sublinear-space distinguisher implies a sublinear-communication protocol predicting the inner product of x and x'. Specifically, the party holding x runs the distinguisher until it finishes reading x, and then sends its configuration to the party holding x'.)

[18] A degree 1 generator contains more than n linear functions over n variables, which must be linearly dependent and thus biased. The non-existence of a 2-local generator follows from the fact that every nonlinear function of two input bits is biased.

Applying the locality construction to G, we obtain a perfect encoding $\hat{G}$ in NC^0_3. (In fact, we can apply the locality construction only to the last bit of G and leave the other outputs as they are.) We argue that $\hat{G}$ inherits the pseudorandomness of G. As before, we would like to argue that if $\hat{M}$ is a sublinear-space distinguisher breaking $\hat{G}$ and S is the balanced simulator of the encoding, then $\hat{M}(S(\cdot))$ is a sublinear-space distinguisher breaking G. Similarly to the proof of Lemma 6.9, the fact that $\hat{M}(S(\cdot))$ can be implemented in sublinear space will follow from the simple structure of S. However, in contrast to Lemma 6.9, here it does not suffice to require S to be linear and we need to rely on the stronger property guaranteed by Observation 6.10.[19]

We now formalize the above. As argued in Observation 6.10, fixing the randomness ρ of S, the simulator's computation can be written as $S_\rho(y) = \sigma_1(y_1)\sigma_2(y_2)\cdots\sigma_l(y_l)$, where each σ_i maps a bit of y to one of two fixed strings. We can thus use S to turn a sublinear-space distinguisher $\hat{M}^a$ breaking $\hat{G}$ into a sublinear-space distinguisher $M^{a'}$ breaking G. Specifically, let the advice a' include, in addition to a, the $2l$ strings $\sigma_i(0), \sigma_i(1)$ corresponding to a "good" ρ which maintains the distinguishing advantage. (The existence of such ρ follows from an averaging argument.) The machine $M^{a'}(y)$ can now emulate the computation of $\hat{M}^a(S_\rho(y))$ using sublinear space and a one-way access to y by applying $\hat{M}^a$ in each step to the corresponding string $\sigma_i(y_i)$. ■

6.4 Pseudorandom Generators - Conclusion

We conclude this section with Table 6.1, which summarizes some of the PRGs constructed here as well as previous ones from [43] and highlights the remaining gaps.

Type	Stretch	Locality	Degree
ε-biased	superlinear	5	2 ✓
ε-biased	$n^{\Omega(\sqrt{k})}$	large k	$\Omega(\sqrt{k})$
ε-biased	$\Omega(n^2)$✓	$\Omega(n)$	2 ✓
ε-biased	linear ✓	3 ✓	2 ✓
space	sublinear $\not\checkmark$	3 ✓	2 ✓
cryptographic *	sublinear $\not\checkmark$	4	3

Table 6.1: Summary of known pseudorandom generators. Results of Mossel et al. [43] appear in the top part and results of this paper in the bottom part. A parameter is marked as optimal (✓) if when fixing the other parameters it cannot be improved. A stretch entry is marked with $\not\checkmark$ if the stretch is sublinear and cannot be improved to be superlinear (but might be improved to be linear). The symbol * indicates a conditional result.

7 Other Cryptographic Primitives

In this section, we describe 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; or (2) use known *reductions* between cryptographic primitives, together with NC^0 primitives we already constructed (e.g., OWF or PRG), to obtain new NC^0 primitives. Of course, this approach is useful only when the reduction itself

[19]Indeed, in the current model of (non-uniform) space-bounded computation with *one-way* access to the input (and two-way access to the advice), there exist a boolean function $\hat{M}$ computable in sublinear space and a linear function S such that the composed function $\hat{M}(S(\cdot))$ is not computable in sublinear space. For instance, let $\hat{M}(y_1, \ldots, y_{2n}) = y_1y_2 + y_3y_4 + \ldots + y_{2n-1}y_{2n}$ and $S(x_1, \ldots, x_{2n}) = (x_1, x_{n+1}, x_2, x_{n+2}, \ldots, x_n, x_{2n})$.

is in NC^0.[20] We mainly adopt the first approach, since most of the known reductions between primitives are not in NC^0. (An exception in the case of symmetric encryption will be discussed below.)

7.1 Collision-Resistant Hashing in NC^0

We start with a formal definition of collision-resistant hash-functions (CRHFs).

Definition 7.1 (Collision-resistant hashing) *Let $\ell, \ell' : \mathbb{N} \to \mathbb{N}$ be such that $\ell(n) > \ell'(n)$ and let $Z \subseteq \{0,1\}^*$. A collection of functions $\{h_z\}_{z \in Z}$ is said to be* collision-resistant *if the following holds:*

1. *There exists a probabilistic polynomial-time* key-generation *algorithm, G, that on input 1^n outputs an* index *$z \in Z$ (of a function h_z). The function h_z maps strings of length $\ell(n)$ to strings of length $\ell'(n)$.*
2. *There exists a polynomial-time* evaluation *algorithm that on input $z \in G(1^n), x \in \{0,1\}^{\ell(n)}$ computes $h_z(x)$.*
3. *Collisions are hard to find. Formally, a pair (x, x') is called a* collision *for a function h_z if $x \neq x'$ but $h_z(x) = h_z(x')$. The collision-resistance requirement states that every probabilistic polynomial-time algorithm B, that is given input $(z = G(1^n), 1^n)$, succeeds in finding a collision for h_z with a negligible probability in n (where the probability is taken over the coin tosses of both G and B).*

Lemma 7.2 *Suppose $\mathcal{H} = \{h_z\}_{z \in Z}$ is collision resistant and $\hat{\mathcal{H}} = \{\hat{h}_z\}_{z \in Z}$ is a uniform perfect randomized encoding of $\mathcal{H}$. Then $\hat{\mathcal{H}}$ is also collision resistant.*

Proof: Since $\hat{h}_z$ is stretch preserving, it is guaranteed to shrink its input as h_z. The key generation algorithm G of $\mathcal{H}$ is used as the key generation algorithm of $\hat{\mathcal{H}}$. By the uniformity of the collection $\hat{\mathcal{H}}$, there exists an efficient evaluation algorithm for this collection. Finally, any collision $((x, r), (x', r'))$ under $\hat{h}_z$ (i.e., $(x, r) \neq (x', r')$ and $\hat{h}_z(x, r) = \hat{h}_z(x', r')$), defines a collision (x, x') under h_z. Indeed, perfect correctness ensures that $h_z(x) = h_z(x')$ and unique-randomness (see Lemma 4.12) ensures that $x \neq x'$. Thus, an efficient algorithm that finds collisions for $\hat{\mathcal{H}}$ with non-negligible probability yields a similar algorithm for $\mathcal{H}$. ■

By Lemma 7.2 and Theorem 4.18, we get:

Theorem 7.3 *If there exists a CRHF $\mathcal{H} = \{h_z\}_{z \in Z}$ such that the function $h'(z, x) \overset{def}{=} h_z(x)$ is in $\mathcal{PREN}$ (in particular, in $\oplus\mathrm{L}/poly$), then there exists a CRHF $\hat{\mathcal{H}} = \{\hat{h}_z\}_{z \in Z}$ such that the mapping $(z, y) \mapsto \hat{h}_z(y)$ is in NC^0_4.*

Using Theorem 7.3, we can construct CRHFs in NC^0 based on the intractability of factoring [17], discrete logarithm [46], or lattice problems [25, 48]. All these candidates are computable in NC^1 provided that some pre-computation is done by the key-generation algorithm. Note that the key generation algorithm of the resulting NC^0 CRHF is not in NC^0. For more details on NC^0 computation of collections of cryptographic primitives see Appendix A.

7.2 Encryption in NC^0

We turn to the case of encryption. Suppose that $\mathcal{E} = (G, E, D)$ is a public-key encryption scheme, where G is a key generation algorithm, the encryption function $E(e, x, r)$ encrypts the message x 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 [29]. Let $\hat{E}$

[20] If the reduction is in NC^1 one can combine the two approaches: first apply the NC^1 reduction to an NC^0 primitive of type X that was already constructed (e.g., OWF or PRG) to obtain a new NC^1 primitive of type Y, and then use the first approach to compile the latter primitive into an NC^0 primitive (of type Y). As in the first approach, this construction requires to prove that a randomized encoding of a primitive Y preserves its security.

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 $\hat{E}$. We argue that the scheme $\hat{\mathcal{E}} \stackrel{\text{def}}{=} (G, \hat{E}, \hat{D})$ is also a public-key encryption scheme. The efficiency and correctness of $\hat{\mathcal{E}}$ are guaranteed by the uniformity of the encoding and its correctness. Using the efficient simulator of $\hat{E}$, we can reduce the security of $\hat{\mathcal{E}}$ to that of $\mathcal{E}$. Namely, given an efficient adversary $\hat{A}$ that distinguishes between encryptions of x and x' under $\hat{\mathcal{E}}$, we can break $\mathcal{E}$ by using the simulator to transform original ciphers into "new" ciphers, and then invoke $\hat{A}$. The same argument holds in the private-key setting. We now formalize this argument.

Definition 7.4 (Public-key encryption) *A* secure public-key encryption scheme *(PKE) is a triple* (G, E, D) *of probabilistic polynomial-time algorithms satisfying the following conditions:*

- ***Viability**: On input* 1^n *the key generation algorithm,* G*, outputs a pair of keys* (e, d)*. For every pair* (e, d) *such that* $(e, d) \in G(1^n)$*, and for every plaintext* $x \in \{0,1\}^*$*, the algorithms* E, D *satisfy*

$$\Pr[D(d, E(e,x)) \neq x] \leq \varepsilon(n)$$

  where $\varepsilon(n)$ *is a negligible function and the probability is taken over the internal coin tosses of algorithms* E *and* D*.*

- ***Security**: (Indistinguishability of encryptions of a single message) For every (non-uniform) polynomial-time distinguisher* B*, every polynomial* $p(\cdot)$*, all sufficiently large* n*'s, and pair of plaintexts* x, x' *such that* $|x| = |x'| \leq p(n)$*, the distinguisher cannot distinguish between encryptions of* x *and* x' *with more than* $\frac{1}{p(n)}$ *advantage; namely,*

$$|\Pr_{(e,d)\leftarrow G(1^n)}[B(e, E(e,x)) = 1] - \Pr_{(e,d)\leftarrow G(1^n)}[B(e, E(e,x')) = 1]| \leq \frac{1}{p(n)},$$

  where the probabilities are taken over the coin tosses of G, E*.*

The definition of a *private-key* encryption scheme is similar, except that the distinguisher does not get the the encryption key e as an additional input. An extension to multiple-message security, where the indistinguishability requirement should hold for encryptions of polynomially many messages, follows naturally (see [24, chapter 5] for formal definitions). In the public-key case, multiple-message security is implied by single-message security as defined above, whereas in the private-key case it is a strictly stronger notion. In the following we explicitly address only the (single-message) public-key case, but the treatment easily holds for the case of private-key encryption with multiple-message security.

Lemma 7.5 *Let* $\mathcal{E} = (G, E, D)$ *be a secure public-key encryption scheme, where* $E(e, x, r)$ *is viewed as a polynomial-time computable function that encrypts the message* x *using the key* e *and randomness* r*. Let* $\hat{E}((e,x),(r,s)) = \hat{E}((e,x,r),s)$ *be a uniform statistical 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* $\hat{E}$*. Then, the scheme* $\hat{\mathcal{E}} \stackrel{\text{def}}{=} (G, \hat{E}, \hat{D})$ *is also a secure public-key encryption scheme.*

Proof: The uniformity of the encoding guarantees that the functions $\hat{E}$ and $\hat{D}$ can be efficiently computed. The viability of $\hat{\mathcal{E}}$ follows in a straightforward way from the correctness of the decoder C. Indeed, if (e, d) are in the support of $G(1^n)$, then for any plaintext x we have

$$\begin{aligned}
\Pr_{r,s}[\hat{D}(d, \hat{E}(e,x,r,s)) \neq x] &= \Pr_{r,s}[D(d, C(\hat{E}(e,x,r,s))) \neq x] \\
&\leq \Pr_{r,s}[C(\hat{E}((e,x,r),s)) \neq E(e,x,r)] + \Pr_{r}[D(d, E(e,x,r)) \neq x] \\
&\leq \varepsilon(n),
\end{aligned}$$

where $\varepsilon(\cdot)$ is negligible in n and the probabilities are also taken over the coin tosses of D; the first inequality follows from the union bound and the second from the viability of $\mathcal{E}$ and the statistical correctness of $\hat{E}$.

We move on to prove the security of the construction. Assume, towards a contradiction, that $\hat{\mathcal{E}}$ is not secure. It follows that there exists an efficient (nonuniform) distinguisher $\hat{B}$ and a polynomial $p(\cdot)$, such that for infinitely many n's there exist two plaintexts x, x' such that $|x| = |x'| \leq p(n)$, and

$$|\Pr_{(e,d)\leftarrow G(1^n),r,s}[\hat{B}(e,\hat{E}(e,x,r,s)) = 1] - \Pr_{(e,d)\leftarrow G(1^n),r,s}[\hat{B}(e,\hat{E}(e,x',r,s)) = 1]| > \frac{1}{p(n)},$$

where r, s are uniformly chosen random strings of an appropriate length. We use $\hat{B}$ to construct a distinguisher B that distinguishes between encryptions of x and x' under E and derive a contradiction. Define a (non-uniform) distinguisher B by $B(e,y) \overset{\text{def}}{=} \hat{B}(e, S(y))$, where S is the efficient (statistical) simulator of $\hat{E}$. Then, for some negligible ε,

$$\begin{aligned}
& |\Pr_{(e,d)\leftarrow G(1^n),r}[B(e,E(e,x,r)) = 1] - \Pr_{(e,d)\leftarrow G(1^n),r}[B(e,E(e,x',r)) = 1]| \\
= \; & |\Pr_{(e,d)\leftarrow G(1^n),r}[\hat{B}(e,S(E(e,x,r))) = 1] - \Pr_{(e,d)\leftarrow G(1^n),r}[B(e,S(E(e,x',r))) = 1]| \\
\geq \; & |\Pr_{(e,d)\leftarrow G(1^n),r,s}[\hat{B}(e,\hat{E}(e,x,r,s)) = 1] - \Pr_{(e,d)\leftarrow G(1^n),r,s}[\hat{B}(e,\hat{E}(e,x',r,s)) = 1]| - \varepsilon(n) \\
> \; & \frac{1}{p(n)} - \varepsilon(n) > \frac{1}{q(n)},
\end{aligned}$$

for some polynomial $q(\cdot)$ and infinitely many n's. The first inequality is due to statistical privacy and the second follows from our hypothesis. Hence, we derive a contradiction to the security of $\mathcal{E}$ and the lemma follows. ■

In particular, if the scheme $\mathcal{E} = (G, E, D)$ enables errorless decryption and the encoding $\hat{E}$ is perfectly correct, then the scheme $\hat{\mathcal{E}}$ also enables errorless decryption. Additionally, the above lemma is easily extended to case of private-key encryption with multiple-message security. Thus we get,

Theorem 7.6 *If there exists a secure public-key encryption scheme (respectively, a secure private-key encryption scheme) $\mathcal{E} = (G, E, D)$, such that E is in $\mathcal{SREN}$ (in particular, in $\mathrm{NL}/poly$), then there exists a secure public-key encryption scheme (respectively, a secure private-key encryption scheme) $\hat{\mathcal{E}} = (G, \hat{E}, \hat{D})$, such that $\hat{E}$ is in NC^0_4.*

Specifically, one can construct an NC^0 PKE based on either factoring [47, 28, 10], the Diffie-Hellman Assumption [19, 28] or lattice problems [3, 48]. (These schemes enable an NC^1 encryption algorithm given a suitable representation of the key.)

On decryption in NC^0. Our construction provides an NC^0 encryption algorithm but does not promise anything regarding the parallel complexity of the decryption process. This raises the question whether decryption can also be implemented in NC^0. In Appendix C.1, we argue that, in many settings, decryption in NC^0 is impossible regardless of the complexity of encryption. In contrast, if the scheme is restricted to a *single* message of a bounded length (even larger than the key) we can use our machinery to construct a private-key encryption scheme in which both encryption and decryption can be computed in NC^0. This can be done by using the output of an NC^0 PRG to mask the plaintext. Specifically, let $E(e,x) = G(e) \oplus x$ and $D(e,y) = y \oplus G(e)$, where e is a uniformly random key generated by the key generation algorithm and G is a PRG. Unfortunately, the resulting scheme is severely limited by the low stretch of our PRGs. This approach can be also used to give multiple message security, at the price of requiring the encryption and decryption algorithms to maintain a synchronized *state*. In such a stateful encryption scheme the encryption and decryption algorithms take an additional input and produce an additional output, corresponding to their state before and after the operation. The seed of the generator can be used, in this case, as the state of the scheme. In this setting, we can obtain multiple-message security by refreshing the seed of the generator in each invocation; e.g., when encrypting the current bit the encryption algorithm can randomly choose a new seed for the next session, encrypt it along with current bit, and send this encryption to the receiver (alternatively, see [24, Construction 5.3.3]). In the resulting scheme both encryption and decryption are NC^0 functions whose inputs include the inner state of the algorithm.

Theorem 7.6 can be easily extended to stronger notions of security. In particular, randomized encoding preserves security against chosen plaintext attacks (CPA) as well as a-priory chosen ciphertext attacks (CCA1). However, randomized encoding does not preserve security against a-posteriori chosen ciphertext attack (CCA2). Still, it can be shown that the encoding of a CCA2-secure scheme enjoys a relaxed security property that suffices for most applications of CCA2-security. See Appendix C.2 for further discussion.

7.3 Signatures, Commitments, and Zero-Knowledge Proofs

The construction that was used for encryption can be adapted to other cryptographic primitives including (non-interactive) commitments, signatures, message authentication schemes (MACs), and non-interactive zero-knowledge proofs (for definitions see [23, 24]). In all these cases, we can replace the sender (i.e., the encrypting party, committing party, signer or prover, 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 and decoder. In fact, such a construction can also be generalized to the case of interactive protocols such as zero-knowledge proofs and interactive commitments. As in the case of encryption discussed above, this transformation results in an NC^0 sender but does not promise anything regarding the parallel complexity of the receiver. An interesting feature of the case of commitment is that we can also improve the parallel complexity at the receiver's end (see below). The same holds for applications of commitment such as coin-flipping and ZK proofs. We now briefly sketch these constructions and their security proofs.

SIGNATURES. Let $\mathcal{S} = (G, S, V)$ be a signature scheme, where G is a key-generation algorithm that generates the signing and verification keys (s, v), the signing function $S(s, \alpha, r)$ computes a signature β on the document α using the key s and randomness r, and the verification algorithm $V(v, \alpha, \beta)$ verifies that β is a valid signature on α using the verification key v. The scheme is secure (unforgeable) if it is infeasible to forge a signature in a chosen message attack. Namely, any polynomial-time adversary that gets the verification key and an oracle access to the signing process $S(s, \cdot)$ fails to produce a valid signature β on a document α (with respect to the corresponding verification key v) for which it has not requested a signature from the oracle. Let $\hat{S}$ be a statistical randomized encoding of S, and let $\hat{V}(v, \alpha, \hat{\beta}) \stackrel{\text{def}}{=} V(v, \alpha, C(\hat{\beta}))$ be the composition of V with the decoder C of the encoding $\hat{S}$. We claim that the scheme $\hat{\mathcal{S}} \stackrel{\text{def}}{=} (G, \hat{S}, \hat{V})$ is also a signature scheme. Given an adversary $\hat{A}$ that breaks $\hat{\mathcal{S}}$, we can break $\mathcal{S}$ by invoking $\hat{A}$ and emulating the oracle $\hat{S}$ using the simulator of the encoding and the signature oracle S. If the forged signature $(\alpha, \hat{\beta})$ produced by $\hat{A}$ is valid under $\hat{\mathcal{S}}$, then it is translated into a valid signature (α, β) under $\mathcal{S}$ by using the decoder, i.e., $\beta = C(\hat{\beta})$. A similar argument holds also in the private-key setting (i.e., in the case of MACs).

COMMITMENTS. A commitment scheme enables one party (a sender) to commit itself to a value while keeping it secret from another party (the receiver). Later, the sender can reveal the committed value to the receiver, and it is guaranteed that the revealed value is equal to the one determined at the commit stage. We start with the simple case of a perfectly binding, non-interactive commitment. Such a scheme can be defined by a polynomial-time computable function $\textsc{Send}(b, r)$ that outputs a commitment c to the bit b using the randomness r. We assume, w.l.o.g., that the scheme has a canonical decommit stage in which the sender reveals b by sending b and r to the receiver, who verifies that $\textsc{Send}(b, r)$ is equal to the commitment c. The scheme should be both (computationally) hiding and (perfectly) binding. Hiding requires that $c = \textsc{Send}(b, r)$ keeps b computationally secret (as formalized in Definition 7.4 for the case of encryption). Binding means that it is impossible for the sender to open its commitment in two different ways; that is, there are no r_0 and r_1 such that $\textsc{Send}(0, r_0) = \textsc{Send}(1, r_1)$. Let $\widehat{\textsc{Send}}(b, r, s)$ be some randomized encoding of $\textsc{Send}(b, r)$. It can be shown that if $\widehat{\textsc{Send}}$ is a perfectly correct (and statistically private) encoding of $\textsc{Send}$, then $\widehat{\textsc{Send}}$ defines a computationally hiding perfectly binding, non-interactive commitment: Hiding follows from the privacy of the encoding, as argued for the case of encryption in Section 7.2. The binding property of $\widehat{\textsc{Send}}$ follows from the perfect correctness; namely, given a cheating sender $\hat{S}^*$ for $\widehat{\textsc{Send}}$ that produces ambiguous commitment $(r_0, r'_0), (r_1, r'_1)$ such that $\widehat{\textsc{Send}}(0, r_0, s_0) = \widehat{\textsc{Send}}(1, r_1, s_1)$, we construct a cheating sender S^* for the original scheme that invokes $\hat{S}^*$ and outputs r_0, r_1. By perfect correctness it holds that $\textsc{Send}(0, r_0) = \textsc{Send}(1, r_1)$ and hence the new adversary succeeds with the same probability as the original one.[21]

Using a standard construction ([9], [23, Construction 4.4.2]), it follows that commitments in NC^0 are implied by the existence of a 1-1 OWF in $\mathcal{PREN}$. It is important to note that in contrast to the non-interactive perfectly binding primitives described so far, here we also improve the parallel complexity at the receiver's end. Indeed, on input $\hat{c}, b, r, s$ the receiver's computation consists of computing $\widehat{\textsc{Send}}(b, r, s)$ and comparing the result to $\hat{c}$. Assuming $\widehat{\textsc{Send}}$ is in NC^0, the receiver can be implemented by an NC^0 circuit augmented with a single (unbounded fan-in) AND gate. We refer to this special type of AC^0 circuit as an $\mathrm{NC}^0[\mathrm{AND}]$ circuit. As an immediate application, we get a 3-round protocol for flipping a coin [9] between an NC^0 circuit and an $\mathrm{NC}^0[\mathrm{AND}]$ circuit.

One can apply a similar transformation to other variants of commitment schemes, such as unconditionally hiding (and computationally binding) interactive commitments. Schemes of this type require some initialization phase, which typically involves a random key sent from the receiver to the sender. We can turn such a scheme into a similar scheme between an NC^0 sender and an $\mathrm{NC}^0[\mathrm{AND}]$ receiver, provided that it conforms to the following structure: (1) the receiver initializes the scheme by *locally* computing a random key k (say, a prime modulus and powers of two group elements for schemes based on discrete logarithm) and sending it to the sender; (2) the sender responds with a single message computed by the commitment function $\textsc{Send}(b, k, r)$ which is in $\mathcal{PREN}$ (actually, perfect correctness and statistical privacy suffice); (3) as in the previous case, the scheme has a canonical decommit stage in which the sender reveals b by sending b and r to the receiver, who verifies that $\textsc{Send}(b, k, r)$ is equal to the commitment c. Using the CRHF-based commitment scheme of [18, 31], one can obtain schemes of the above type based on the intractability of factoring, discrete logarithm, and lattice problems. Given such a scheme, we replace the sender's function by its randomized encoding, and get as a result an unconditionally hiding commitment scheme whose sender is in NC^0. The new scheme inherits the round complexity of the original scheme and thus consists of only two rounds of interaction. (The security proof is similar to the previous case of perfectly binding, non-interactive commitment.) If the random key k cannot be computed in $\mathrm{NC}^0[\mathrm{AND}]$ (as in the case of factoring and discrete logarithm based schemes), one can compute k once and for all during the generation of the receiver's circuit and hardwire the key to the receiver's circuit. (See Appendix A.)

ZERO-KNOWLEDGE PROOFS. We end this section by addressing the case of zero-knowledge protocols. Suppose that the prover's computations are in $\mathcal{SREN}$. Then, similarly to the case of encryption, we can compile the prover into its (statistical) randomized encoding, and obtain a prover whose local computations (viewed as a function of its randomness, the common instance of the language, the private witness, and previously received messages) are in NC^0. The new verifier uses the decoder to translate the prover's encoded messages to the corresponding messages of original protocol, and then invokes the original verifier. The completeness and soundness of the new protocol follow from the correctness of the encoding, and its zero-knowledge property from the privacy of the encoding. (The verifier can produce transcripts of the new protocol by composing the simulator of the encoding with the simulator of the original protocol.) A similar transformation applies to zero-knowledge *arguments*.

As before, this general approach does not parallelize the verifier; in fact, the verifier is now required to "work harder" and decode the prover's messages. However, we can improve the verifier's complexity by relying on specific, commitment-based, zero-knowledge protocols from the literature. For instance, in the constant-round protocol for

[21]A modification of this scheme remains secure even if we replace $\textsc{Send}$ with a *statistical* randomized encoding. However, in this modification we cannot use the canonical decommitment stage. Instead, the receiver should verify the decommitment by applying the decoder C to $\hat{c}$ and comparing the result to the computation of the original sender; i.e., the receiver checks whether $C(\hat{c})$ equals to $\textsc{Send}(b, r)$. A disadvantage of this alternative decommitment is that it does not enjoy the enhanced parallelism feature discussed below.

Graph 3-Colorability of [26], the computations of the prover and the verifier consist of invoking two commitments (of both types, perfectly binding as well as statistically hiding), in addition to some AC^0 computations. Hence, we can use the parallel commitment schemes described before to construct a constant-round protocol for 3-Colorability between an AC^0 prover and an AC^0 verifier. Since 3-Colorability is NP complete under AC^0-reductions, we get constant-round zero-knowledge proofs in AC^0 for every language in NP.

7.4 Summary and Discussion

Table 7.1 summarizes the properties of randomized encoding that suffice for encoding different cryptographic primitives. (In the case of trapdoor permutations, efficient randomness recovery is also needed.) We note that in some cases it suffices to use a *computationally-private* randomized encoding, in which the simulator's output should only be computationally indistinguishable from that of the encoding. This relaxation, recently studied in [4], allows to construct (some) primitives in NC^0 under more general assumptions.

Primitive	Encoding	Efficient simulator	Efficient decoder
One-way function	statistical	required	—
One-way permutation	perfect	required	—
Trapdoor permutation	perfect	required	required
Pseudorandom generator	perfect	required	—
Collision-resistant hashing	perfect	—	—
Encryption (pub., priv.)	statistical	required	required
Signatures, MAC	statistical	required	required
Commit + Decommit	perfectly correct	required	—
Zero-knowlege proof	statistical	required	required

Table 7.1: Sufficient properties for preserving the security of different primitives.

THE CASE OF PRFS. It is natural to ask why our machinery cannot be applied to pseudorandom functions (PRFs) (assuming there exists a PRF in $\mathcal{PREN}$), as is implied from the impossibility results of Linial et al. [42]. Suppose that a PRF family $f_k(x) = f(k,x)$ is encoded by the function $\hat{f}(k,x,r)$. There are two natural ways to interpret $\hat{f}$ as a collection: (1) to incorporate the randomness into the key, i.e., $g_{k,r}(x) \overset{\text{def}}{=} \hat{f}(k,x,r)$; (2) to append the randomness to the argument of the collection, i.e., $h_k(x,r) \overset{\text{def}}{=} \hat{f}(k,x,r)$. To rule out the security of approach (1), it suffices to note that the mapping $\hat{f}(\cdot,r)$ is of degree one when r is fixed; thus, to distinguish $g_{k,r}$ from a truly random function, one can check whether the given function is affine (e.g., verify that $g_{k,r}(x) + g_{k,r}(y) = g_{k,r}(x+y) + g_{k,r}(0)$). The same attack applies to the function $h_k(x,r)$ obtained by the second approach, by fixing the randomness r. More generally, the privacy of a randomized encoding is guaranteed only when the randomness is secret and is freshly picked, thus our methodology works well for cryptographic primitives which employ fresh secret randomness in each invocation. PRFs do not fit into this category: while the key contains secret randomness, it is not freshly picked in each invocation.

We finally note that by combining the positive results regarding the existence of various primitives in NC^0 with the negative results of [42] that rule out the possibility of PRFs in AC^0, one can derive a separation between PRFs and other primitives such as PRGs. In particular, we conclude that it is unlikely that a PRF is AC^0-reducible to a PRG.

8 One-Way Functions with Optimal Locality

The results presented so far leave a small gap between the strong positive evidence for cryptography in NC^0_4 and the known impossibility of even OWF in NC^0_2. In this section we attempt to close this gap for the case of OWF, providing positive evidence for the existence of OWF in NC^0_3.

A natural approach for closing the gap would be to reduce the degree of our general construction of randomized encodings from 3 to 2. (Indeed, the locality construction transforms a degree-2 encoding into one in NC^0_3.) However, the results of [37] provide some evidence against the prospects of this general approach, ruling out the existence of degree-2 perfectly private encodings for most nontrivial functions. We thus take the following two alternative approaches: (1) seek *direct* constructions of degree-2 OWF based on specific intractability assumptions; and (2) employ degree-2 randomized encodings with a weak (but nontrivial) privacy property (called *semi-privacy*), which enables the representation of general functions.

In Section 8.1, we use approach (1) to construct a OWF with optimal locality based on the presumed intractability of decoding a random linear code. In Section 8.2 we briefly demonstrate the usefulness of approach (2) by sketching a construction of a OWF with optimal locality based on a OWF that enjoys a certain strong "robustness" property, which is satisfied by a variant of a OWF candidate suggested in [22]. We note that neither of the above approaches yields a general result in the spirit of the results of the previous sections. Thus, we happen to pay for optimal degree and locality with the loss of generality.

8.1 OWF in NC^0_3 from the Intractability of Decoding Random Linear Codes

Several cryptographic schemes are based on hard problems from the theory of error-correcting codes. In particular, the problem of decoding random linear codes, which is a longstanding open question in coding theory, was suggested as a basis for one-way functions [27]. An (n, k, δ) *binary linear code* is a k-dimensional linear subspace of $\mathrm{GF}(2)^n$ in which the Hamming distance between each two distinct vectors (codewords) is at least δn. We refer to the ratio k/n as the *rate* of the code and to δ as its (relative) *distance*. Such a code can be defined by a $k \times n$ *generator matrix* whose rows span the space of codewords. It follows from the Gilbert–Varshamov bound that whenever $k/n < 1 - \mathrm{H}_2(\delta) - \varepsilon$ (where H_2 is the binary entropy function and ε is an arbitrarily small positive constant), almost all $k \times n$ generator matrices form (n, k, δ)-linear codes.

Before defining our intractability assumption imagine the following "decoding game". Let $k/n < 1 - \mathrm{H}_2(\frac{1}{3}) - \varepsilon$ for some constant $\varepsilon > 0$. Pick a random $k \times n$ matrix C representing a linear code (which is with overwhelming probability an $(n, k, \frac{1}{3} + \varepsilon)$ code) and a random information word x. Encode x with C and transmit the resulting codeword $y = xC$ over a binary symmetric channel in which every bit is flipped with probability $\frac{1}{4}$. If more than $\frac{1}{3}$ of the bits were flipped, output the zero word; otherwise, output the noisy codeword $\tilde{y}$ along with the code's description C. In the former event the adversary always wins (however, note that the probability of this event is negligible). In the latter event, the adversary's task is to find some codeword y which is at most $(n/3)$-far from $\tilde{y}$. The fact that the noise is random (rather than adversarial) guarantees, by Shannon's coding theorem, that y will be unique with overwhelming probability.

The intractability assumption on which we rely asserts that every polynomial-time adversary lose in the above game with noticeable probability. That is, roughly speaking, we assume that it is intractable to correct $n/4$ *random* errors in a random linear code of relative distance $\frac{1}{3}$. More precisely:

Intractability Assumption 8.1 (Decoding a random linear code) *There exists a constant $c < 1 - H_2(\frac{1}{3})$ such that the following function f_{code} is a weak OWF:* [22]

$$f_{\mathrm{code}}(C, x, e) \overset{\text{def}}{=} \begin{cases} 0 & \textit{weight}(e_1e_2, \ldots, e_{2n-1}e_{2n}) \geq n/3, \\ (C, xC + (e_1e_2, \ldots, e_{2n-1}e_{2n})) & \textit{otherwise} \end{cases}$$

[22]In fact, it seems likely that the function f_{code} is even strongly one-way.

where C is a $k \times n$ binary generator matrix with $k = \lfloor cn \rfloor$, $x \in \{0,1\}^k$, $e \in \{0,1\}^{2n}$, $weight(\cdot)$ denotes Hamming weight, and arithmetic is over $\mathrm{GF}(2)$.

Namely, inverting f_{code} on a uniformly chosen input corresponds to winning in the above decoding game. (Two random bits, e_i and e_{i+1}, are multiplied to emulate a noise rate of $\frac{1}{4}$.) The plausibility of Assumption 8.1 is supported by the fact that a successful inverter would imply a major breakthrough in coding theory. Similar assumptions were put forward in [27, 8, 23]. It is possible to base our construction on different variants of this assumption (e.g., one in which the number of errors is bounded by half the minimal distance, as in [27]); the above formulation is preferred for simplicity (and seems even weaker than the one in [27]).

We now construct a degree-2 OWF assuming the (weak) one-wayness of f_{code}. Consider the degree-2 function f'_{code} defined by $f'_{\text{code}}(C, x, e) \stackrel{\text{def}}{=} (C, xC + (e_1e_2, \ldots, e_{2n-1}e_{2n}))$. The function f'_{code} by itself is not one-way; indeed, as there is no restriction on the choice of e, an inverter can arbitrarily pick x and then fix e to be consistent with C, x, and $\tilde{y}$. However, f'_{code} is still distributionally one-way. This follows by noting that f'_{code} differs from f_{code} only on a negligible fraction of their domain and by using Lemma 5.4. To conclude the proof we need the following lemma.

Lemma 8.2 *A degree-2 distributional OWF implies a degree-2 OWF in* NC^0_3.

Proof: First observe that a degree-2 weak OWF can be transformed into a degree-2 (standard) OWF (cf. [52],[23, Theorem 2.3.2]). Combined with the locality construction, we get that the existence of a degree-2 weak OWF implies the existence of a degree-2 OWF in NC^0_3. Hence it is enough to show how to transform a degree-2 distributional OWF into a degree-2 weak OWF.

Let f be a degree-2 distributional OWF. Consider the function $F(x, i, h) = (f(x), h_i(x), i, h)$, where $x \in \{0,1\}^n$, $i \in \{1, \ldots, n\}$, $h : \{0,1\}^n \to \{0,1\}^n$ is a pairwise independent hash function, and h_i denotes the i-bit-long prefix of $h(x)$. This function was defined by Impagliazzo and Luby [35], who showed that in this case F is weakly one-way (see also [23, p. 96]). Note that $h(x)$ can be computed as a degree-2 function of x and (the representation of) h by using the hash family $h_{M,v}(x) = xM + v$, where M is an $n \times n$ matrix and v is a vector of length n. However, $h_i(x)$ is not of degree 2 when considered as a function of h, x and i, since "chopping" the last $n - i$ bits of $h(x)$ raises the degree of the function when i is not fixed. We get around this problem by applying n copies of F on independent inputs, where each copy uses a different i. Namely, we define the function $F'((x^{(i)}, h^{(i)})_{i=1}^n) \stackrel{\text{def}}{=} (F(x^{(i)}, i, h^{(i)}))_{i=1}^n$. Since each of the i's is now fixed, the resulting function F' can be computed by degree-2 polynomials over $\mathrm{GF}(2)$. Moreover, it is not hard to verify that F' is weakly one-way if F is weakly one-way. We briefly sketch the argument. Given an efficient inverting algorithm B for F', one can invert $y = F(x, i, h) = (f(x), h_i(x), i, h)$ as follows. For every $j \neq i$, uniformly and independently choose $x^{(j)}, h^{(j)}$, set $z_j = F(x^{(j)}, j, h^{(j)})$ and $z_i = y$, then invoke B on $(z_j)_{j=1}^n$ and output the i^{th} block of the answer. This inversion algorithm for F has the same success probability as B on a polynomially related input. ■

Applying Lemma 8.2 to f'_{code} we get:

Theorem 8.3 *If Assumption 8.1 holds, there is a degree-2 OWF in* NC^0_3.

8.2 OWF in NC^0_3 Using Semi-Private Encoding

In this section we briefly address the possibility of obtaining optimal locality for OWF (i.e., locality 3 rather than 4) by relaxing the privacy requirement of the encoding. Further details appear in [5].

We start by sketching an alternative approach for constructing OWF in NC^0_3 based on Assumption 8.1. The basic idea is the following. Consider the degree-2 function f'_{code} defined above. This function is not one-way. However, it is possible to augment it to a (weakly) one-way function by appending to its output a single bit, $\phi(e)$, indicating whether the error vector e exceeds the weight threshold. That is, $\phi(e) = 1$ iff $weight(e_1e_2, \ldots, e_{2n-1}e_{2n}) \geq n/3$.

(This ensures that, with high probability, the inverter will be forced to pick a low-weight error.) While we cannot encode the predicate $\phi(e)$ using degree-2 polynomials, it turns out that we can achieve this using the following type of *semi-private* encoding. Specifically, we relax the simulation requirement to hold only when $\phi(e) = 0$. Thus, the encoding $\hat{\phi}(e, r)$ keeps e private only when $\phi(e) = 0$, i.e., when e defines a low-weight error vector. It is possible to efficiently construct such a degree-2 semi-private encoding from the branching program representation of ϕ. (This can be done by using a variant of the BP construction described in Section 4.3.) Hence, under Assumption 8.1, the degree-2 encoding $\hat{f}_{\mathrm{code}}((C, x, e), r) \stackrel{\text{def}}{=} (f'_{\mathrm{code}}(C, x, e), \hat{\phi}(e, r))$ is weakly one-way.

Given any OWF f, one could attempt to apply a semi-private encoding as described above to every output bit of f, obtaining a degree-2 function $\hat{f}$. However, $\hat{f}$ will typically not be one-way: every output bit of f that evaluates to 1 might reveal the entire input (through the corresponding block in the output of $\hat{f}$). This motivates the following notion of a *robust* OWF. Loosely speaking, a OWF f is said to be robust if it remains (slightly) hard to invert even if a random subset of its output bits are "exposed", in the sense that all input bits leading to these outputs are revealed. Intuitively, the purpose of the robustness requirement is to guarantee that the information leaked by the semi-private encoding leaves enough uncertainty about the input to make inversion difficult. It can be shown that: (1) every robust OWF with a low locality (say, logarithmic in the number of inputs) can be turned into a OWF in NC^0_3; and (2) a variant of a OWF candidate from [22] satisfies the latter property, assuming that it is indeed one-way. Thus, an intractability assumption of the flavor of the one suggested in [22] implies the existence of OWF in NC^0_3.

9 Conclusions and Open Problems

Our results provide strong 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? We show that a OWF in $\mathrm{NL}/poly$ implies a 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 [43].) We show that there exists a PRG with *sublinear* stretch in NC^0_4, assuming the existence of a PRG in $\oplus\mathrm{L}/poly$.

- Can the existence of a OWF (or PRG) in NC^0_3 be based on more general assumptions? We construct such a OWF under the intractability of decoding a random linear code.

- Is it possible to obtain constant *input* locality, i.e., construct primitives in which each input influences only a constant number of outputs? (A candidate OWF of this type is given in [22].) Note that the results of this work only address the case of a constant *output* locality, which does not imply a constant input locality.

- 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. In [4] we continue this study by considering a relaxed variant of randomized encoding referred to as *computationally-private* encoding. We show that, under relatively mild assumptions, one can encode every polynomial-time computable function by a computationally-private encoding in NC^0. This gives new sufficient conditions for cryptography in NC^0, as well as new NC^0 reductions between different cryptographic primitives.

Acknowledgments We are grateful to Oded Goldreich for many useful suggestions and comments that helped improve this writeup, and in particular for simplifying the proof of Lemma 5.4. We also thank Iftach Haitner and Emanuele Viola for enlightening us about old and new constructions of PRGs from OWFs and for sharing with us the results of [30] and [50]. Finally, we thank Moni Naor and Amir Shpilka for helpful comments.

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. In *Proc. 28th STOC*, pages 99–108, 1996. Full version in Electronic Colloquium on Computational Complexity (ECCC).

[3] M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proc. 29th STOC*, pages 284–293, 1997.

[4] B. Applebaum, Y. Ishai, and E. Kushilevitz. Computationally private randomizing polynomials and their applications. In *Proc. 20th Conference on Computational Complexity (CCC)*, pages 260–274, 2005.

[5] B. Applebaum, Y. Ishai, and E. Kushilevitz. On one-way functions with optimal locality. Unpublished manuscript available at http://www.cs.technion.ac.il/∼abenny, 2005.

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

[7] D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. In *Proc. 18th STOC*, pages 1–5, 1986.

[8] A. Blum, M. Furst, M. Kearns, and R. J. Lipton. Cryptographic primitives based on hard learning problems. In *Advances in Cryptology: Proc. of CRYPTO '93*, volume 773 of *LNCS*, pages 278–291, 1994.

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

[10] M. Blum and S. Goldwasser. An efficient probabilistic public-key encryption scheme which hides all partial information. In *Advances in Cryptology: Proc. of CRYPTO '84*, volume 196 of *LNCS*, pages 289–302, 1985.

[11] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13:850–864, 1984. Preliminary version in FOCS 82.

[12] R. Canetti, H. Krawczyk, and J. Nielsen. Relaxing chosen ciphertext security of encryption schemes. In *Advances in Cryptology: Proc. of CRYPTO '03*, volume 2729 of *LNCS*, pages 565–582, 2003.

[13] M. Capalbo, O. Reingold, S. Vadhan, and A. Wigderson. Randomness conductors and constant-degree lossless expanders. In *Proc. 34th STOC*, pages 659–668, 2002.

[14] B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM J. on Computing*, 17(2):230–261, 1988.

[15] R. Cramer, S. Fehr, Y. Ishai, and E. Kushilevitz. Efficient multi-party computation over rings. In *Proc. EUROCRYPT '03*, pages 596–613, 2003.

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

[17] I. Damgård. Collision free hash functions and public key signature schemes. In *Proc. Eurocrypt'87*, pages 203–216, 1988.

[18] I. Damgård, T. Pedersen, and B. Pfitzmann. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. In *Advances in Cryptology: Proc. of CRYPTO '93*, volume 773 of *LNCS*, pages 250–265, 1994.

[19] T. E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in cryptology: Proc. of CRYPTO '84*, volume 196 of *LNCS*, pages 10–18, 1985. or IEEE Transactions on Information Theory, v. IT-31, n. 4, 1985.

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

[21] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, volume 17 of *Algorithms and Combinatorics*. Springer-Verlag, 1998.

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

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

[24] O. Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[25] O. Goldreich, S. Goldwasser, and S. Halevi. Collision-free hashing from lattice problems. *Electronic Colloquium on Computational Complexity*, 96(042), 1996.

[26] O. Goldreich and A. Kahan. How to construct constant-round zero-knowledge proof systems for NP. *J. of Cryptology*, 9(2):167–189, 1996.

[27] O. Goldreich, H. Krawczyk, and M. Luby. On the existence of pseudorandom generators. *SIAM J. Comput.*, 22(6):1163–1175, 1993. Preliminary version in Proc. 29th FOCS, 1988.

[28] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proc. 21st STOC*, pages 25–32, 1989.

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

[30] I. Haitner, D. Harnik, and O. Reingold. On the power of the randomized iterate. manuscript, 2005.

[31] S. Halevi and S. Micali. Practicle and provably-secure commitment schemes from collision-free hashing. In *Advances in Cryptology: Proc. of CRYPTO '96*, volume 1109 of *LNCS*, pages 201–215, 1996.

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

[33] 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.

[34] C. Y. Hsiao and L. Reyzin. Finding collisions on a public road, or do secure hash functions need secret coins? In *Advances in Cryptology: Proc. of CRYPTO '04*, volume 3152 of *LNCS*, pages 92–105, 2004.

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

[36] R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of Cryptology*, 9:199–216, 1996.

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

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

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

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

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

[42] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, fourier transform, and learnability. *J. ACM*, 40(3):607–620, 1993. Preliminary version in Proc. 30th FOCS, 1989.

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

[44] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, 2004. Preliminary version in Proc. 38th FOCS, 1997.

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

[46] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology: Proc. of CRYPTO '91*, volume 576 of *LNCS*, pages 129–149, 1991.

[47] M. Rabin. Digitalized signatures and public key functions as intractable as factoring. Technical Report 212, LCS, MIT, 1979.

[48] O. Regev. New lattice based cryptographic constructions. In *Proc. 35th STOC*, pages 407–416, 2003.

[49] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. of the ACM*, 21(2):120–126, 1978.

[50] E. Viola. On constructing parallel pseudorandom generators from one-way functions. In *Proc. 20th Conference on Computational Complexity (CCC)*, pages 183– 197, 2005.

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

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

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

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

A On Collections of Cryptographic Primitives

In most cases, we view a cryptographic primitive (e.g., a OWF or a PRG) as a single function $f : \{0,1\}^* \to \{0,1\}^*$. However, it is often useful to consider more general variants of such primitives, defined by a *collection* of functions $\{f_z\}_{z \in Z}$, where $Z \subseteq \{0,1\}^*$ and each f_z is defined over a finite domain D_z. The full specification of such a collection usually consists of a probabilistic polynomial time key-generation algorithm that chooses an index z of a function (given a security parameter 1^n), a domain sampler algorithm that samples a random element from D_z given z, and a function evaluation algorithm that computes $f_z(x)$ given z and $x \in D_z$. The primitive should be secure with respect to the distribution defined by the key-generation and the domain sampler. (See a formal definition for the case of OWF in [23, Definition 2.4.3].)

Collections of primitives arise naturally in the context of parallel cryptography, as they allow to shift "non-parallelizable" operations such as prime number selection and modular exponentiations to the key-generation stage (cf. [44]). They also fit naturally into the setting of P-uniform circuits, since the key-generation algorithm can be embedded in the algorithm generating the circuit. Thus, it will be convenient to assume that z is a description of a circuit computing f_z. When referring to a collection of functions from a given complexity class (e.g., NC^1,NC^0_4, or $\mathcal{PREN}$, cf. Definition 4.8) we assume that the key generation algorithm outputs a description of a circuit from this class. In fact, one can view collections in our context as a natural relaxation of uniformity, allowing the circuit generator to be randomized. (The above discussion also applies to other P-uniform representation models we use, such as branching programs.)

Our usage of collections differs from the standard one in that we insist on D_z being the set of *all* strings of a given length (i.e., the set of all possible inputs for the circuit z) and restrict the domain sampler to be a trivial one which outputs a uniformly random string of the appropriate length. This convention guarantees that the primitive can indeed be invoked with the specified parallel complexity, and does not implicitly rely on a (possibly less parallel) domain sampler.[23] In most cases, it is possible to modify standard collections of primitives to conform to the above convention. We illustrate this by outlining a construction of an NC^1 collection of one-way permutations based on the intractability of discrete logarithm. The key-generator, on input 1^n, samples a random prime p such that $2^{n-1} \leq p < 2^n$ along with a generator g of Z_p^*, and lets z be a description of an NC^1 circuit computing the function $f_{p,g}$ defined as follows. On an n-bit input x (viewed as an integer such that $0 \leq x < 2^n$) define $f_{p,g}(x) = g^x \mod p$ if $1 \leq x < p$ and $f_{p,g}(x) = x$ otherwise. It is easy to verify that $f_{p,g}$ indeed defines a permutation on $\{0,1\}^n$. Moreover, it can be computed by an NC^1 circuit by incorporating $p, g, g^2, g^4, \ldots, g^{2^n}$ into the circuit. Finally, assuming the intractability of discrete logarithm, the above collection is *weakly* one way. It can be augmented into a collection of (strongly) one-way permutations by using the standard reduction of strong OWF to weak OWF (i.e., using $f'_{p,g}(x_1, \ldots, x_n) = (f_{p,g}(x_1), \ldots, f_{p,g}(x_n))$).

When defining the cryptographic security of a collection of primitives, it is assumed that the adversary (e.g., inverter or distinguisher) is given the key z, in addition to its input in the single-function variant of the primitive. Here one should make a distinction between "private-coin collections", where this is all of the information available to the adversary, and "public-coin collections" in which the adversary is additionally given the internal coin-tosses of the key-generator. (A similar distinction has been recently made in the specific context of collision-resistant hash-functions [34]; also, see the discussion of "enhanced TDP" in [24, App. C.1].) The above example for a OWP collection is of the public-coin type. Any public-coin collection is also a private-coin collection, but the converse may not be true.

Summarizing, we consider cryptographic primitives in three different settings:

1. (Single function setting.) The circuit family $\{C_n\}_{n \in \mathbb{N}}$ that computes the primitive is constructed by a deterministic polynomial time circuit generator that, given an input 1^n, outputs the circuit C_n. This is the default setting for most cryptographic primitives.

[23]Note that unlike the key-generation algorithm, which can be applied "once and for all", the domain sampler should be invoked for each application of the primitive.

2. (Public-coin collection.) The circuit generator is a probabilistic polynomial time algorithm that, on input 1^n, samples a circuit from a collection of circuits. The adversary gets as an input the circuit produced by the generator, along with the randomness used to generate it. The experiments defining the success probability of the adversary incorporate the randomness used by the generator, in addition to the other random variables. As in the single function setting, this generation step can be thought of as being done "once and for all", e.g., in a pre-processing stage. Public-coin collections are typically useful for primitives based on discrete logarithm assumptions, where a large prime group should be set up along with its generator and precomputed exponents of the generator.

3. (Private-coin collection.) Same as (2) except that the adversary does not know the randomness that was used by the circuit generator. This relaxation is typically useful for factoring-based constructions, where the adversary should not learn the trapdoor information associated with the public modulus (see [39, 44]).

We note that our general transformations apply to all of the above settings. In particular, given an NC^1 primitive in any of these settings, we obtain a corresponding NC^0 primitive in the same setting.

B A Generalization of the Locality Construction

In the Locality Construction (4.16), we showed how to encode a degree d function by an NC^0_{d+1} encoding. We now describe a graph based construction that generalizes the previous one. The basic idea is to view the encoding $\hat{f}$ as a graph. The nodes of the graph are labeled by terms of f and the edges by random inputs of $\hat{f}$. With each node we associate an output of $\hat{f}$ in which we add to its term the labels of the edges incident to the node. Formally,

Construction B.1 (General locality construction) *Let $f(x) = T_1(x)+\ldots+T_k(x)$, where $f, T_1, \ldots, T_k : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ and summation is over $\mathrm{GF}(2)$. Let $G = (V, E)$ be a directed graph with k nodes $V = \{1, \ldots, k\}$ and m edges. The encoding $\hat{f}_G : \mathrm{GF}(2)^{n+m} \to \mathrm{GF}(2)^k$ is defined by:*

$$\hat{f}_G(x, (r_{i,j})_{(i,j)\in E}) \quad \overset{def}{=} \quad \left(T_i(x) + \sum_{j|(j,i)\in E} r_{j,i} - \sum_{j|(i,j)\in E} r_{i,j} \right)_{i=1}^{k} .$$

From here on, we will identify with the directed graph G its underlying undirected graph. The above construction yields a perfect encoding when G is a tree (see Lemma B.2 below). The locality of an output bit of $\hat{f}_G$ is the locality of the corresponding term plus the degree of the node in the graph. The locality construction described in Construction 4.16 attempts to minimize the maximal locality of a node in the graph; hence it adds k "dummy" 0 terms to f and obtains a tree in which all of the k non-dummy terms of f are leaves, and the degree of each dummy term is at most 3. When the terms of f vary in their locality, a more compact encoding $\hat{f}$ can be obtained by increasing the degree of nodes which represent terms with lower locality.

Lemma B.2 (Generalized locality lemma) *Let f and $\hat{f}_G$ be as in Construction B.1. Then,*

1. *$\hat{f}_G$ is a perfectly correct encoding of f.*
2. *If G is connected, then $\hat{f}_G$ is also a balanced encoding of f (and in particular it is perfectly private).*
3. *If G is a tree, then $\hat{f}_G$ is also stretch preserving; that is, $\hat{f}_G$ perfectly encodes f.*

Proof: (1) Given $\hat{y} = \hat{f}_G(x,r)$ we decode $f(x)$ by summing up the bits of $\hat{y}$. Since each random variable $r_{i,j}$ appears only in the i^{th} and j^{th} output bits, it contributes 0 to the overall sum and therefore the bits of $\hat{y}$ always add up to $f(x)$.

To prove (2) we use the same simulator as in the locality construction (see proof of Lemma 4.17). Namely, given $y \in \{0,1\}$, the simulator S chooses $k-1$ random bits $r_1,\ldots,r_{k-1}$ and outputs $(r_1,\ldots,r_{k-1},y-(r_1+\ldots+r_{k-1}))$. This simulator is balanced since the supports of $S(0)$ and $S(1)$ halve $\{0,1\}^k$ and $S(y)$ is uniformly distributed over its support for $y \in \{0,1\}$. We now prove that $\hat{f}_G(x,U_m) \equiv S(f(x))$. Since the support of $S(f(x))$ contains exactly 2^{k-1} strings (namely, all k-bit strings whose bits sum up to $f(x)$), it suffices to show that for any input x and output $w \in \mathrm{support}(S(f(x)))$ there are $2^m/2^{k-1}$ random inputs r such that $\hat{f}_G(x,r) = w$. (Note that $m \geq k-1$ since G is connected.) Let $T \subseteq E$ be a spanning tree of G. We argue that for any assignment to the $m-(k-1)$ random variables that correspond to edges in $E \setminus T$ there exists an assignment to the other random variables that is consistent with w and x. Fix some assignment to the edges in $E \setminus T$. We now recursively assign values to the remaining edges. In each step we make sure that some leaf is consistent with w by assigning the corresponding value to the edge connecting this leaf to the graph. Then, we prune this leaf and repeat the above procedure. Formally, let i be a leaf which is connected to T by an edge $e \in T$. Assume, without loss of generality, that e is an incoming edge for i. We set r_e to $w_i - (T_i(x) + \sum_{j|(j,i)\in E\setminus T} r_{j,i} - \sum_{j|(i,j)\in E\setminus T} r_{i,j})$, and remove i from T. By this we ensure that the i^{th} bit of $\hat{f}_G(x,r)$ is equal to w_i. (This equality will not be violated by the following steps as i is removed from T.) We continue with the above step until the tree consists of one node. Since the outputs of $\hat{f}_G(x,r)$ always sum up to $f(x)$ it follows that this last bit of $\hat{f}_G(x,r)$ is equal to the corresponding bit of w. Thus, there are at least $2^{|E\setminus T|} = 2^{m-(k-1)}$ values of r that lead to w as required.

Finally, to prove (3) note that when G is a tree we have $m = k-1$, and therefore the encoding is stretch preserving; combined with (1) and (2) $\hat{f}_G$ is also perfect. ■

C More on Encryption Schemes in NC^0

We consider two issues regarding encryption, briefly mentioned in Section 7.2.

C.1 On the Impossibility of NC^0 Decryption

In this section we show that, in many settings, decryption in NC^0 is impossible regardless of the complexity of encryption. Here we consider standard *stateless* encryption schemes in contrast to the discussion at the end of Section 7.2. We begin with the case of multiple-message security (in either the private-key or public-key setting). If a decryption algorithm $D(d,y)$ is in NC^0_k, then an adversary that gets n encrypted messages can correctly guess the first bits of *all* the plaintexts (jointly) with at least 2^{-k} probability. To do so, the adversary simply guesses at random the k (or less) bits of the key d on which the first output bit of D depends, and then computes this first output bit (which is supposed to be the first plaintext bit) on each of the n ciphertexts using the subkey it guessed. Whenever the adversary guesses the k bits correctly, it succeeds to find the first bits of *all* n messages. When $n > k$, this violates the semantic security of the encryption scheme. Indeed, for the encryption scheme to be secure, the adversary's success probability (when the messages are chosen at random) can only be negligibly larger than 2^{-n}. (That is, an adversary cannot do much better than simply guessing these first bits.)

Even in the case of a single-message private-key encryption, it is impossible to implement decryption in NC^0_k with an arbitrary (polynomial) message length. Indeed, when the message length exceeds $(2|d|)^k$ (where $|d|$ is the length of the decryption key), there must be more than 2^k bits of the output of D which depend on the same k bits of the key, in which case we are in the same situation as before. That is, we can guess the value of more than 2^k bits of the message with constant success probability 2^{-k}. Again, if we consider a randomly chosen message, this violates semantic security.

C.2 Security against CPA, CCA1 and CCA2 Attacks

In this section we address the possibility of applying our machinery to encryption schemes that enjoy stronger notions of security. In particular, we consider schemes that are secure against chosen plaintext attacks (CPA), a-priory chosen ciphertext attacks (CCA1), and a-posteriori chosen ciphertext attacks (CCA2). In all three attacks the adversary has to win the standard indistinguishability game (i.e., given a ciphertext $c = E(e, m_b)$ find out which of the two predefined plaintexts m_0, m_1 was encrypted), and so the actual difference lies at the power of the adversary. In a CPA attack the adversary can obtain encryptions of plaintexts of his choice (under the key being attacked), i.e., the adversary gets an oracle access to the encryption function. In CCA1 attack the adversary may also obtain decryptions of his choice (under the key being attacked), but he is allowed to do so only *before* the challenge is presented to him. In both cases, the security is preserved under randomized encoding. We briefly sketch the proof idea.

Let $\hat{B}$ be an adversary that breaks the encoding $\hat{\mathcal{E}}$ via a CPA attack (resp. CCA1 attack). We use $\hat{B}$ to obtain an adversary B that breaks the original scheme $\mathcal{E}$. As in the proof of Lemma 7.5, B uses the simulator to translate the challenge c, an encryption of the message m_b under $\mathcal{E}$, into a challenge $\hat{c}$, which is an encryption of the same message under $\hat{\mathcal{E}}$. Similarly, B answers the encryption queries of $\hat{B}$ (to the oracle $\hat{E}$) by directing these queries to the oracle E and applying the simulator to the result. Also, in the case of CCA1 attack, whenever $\hat{B}$ asks the decryption oracle $\hat{D}$ to decrypt some ciphertext $\hat{c}'$, the adversary B uses the decoder (of the encoding) to translate $\hat{c}'$ into a ciphertext c' of the same message under the scheme $\mathcal{E}$, and then uses the decryption oracle D to decrypt c'. This allows B to emulate the oracles $\hat{D}$ and $\hat{E}$, and thus to translate a successful CPA attack (resp. CCA1 attack) on the new scheme into a similar attack on the original scheme.

The situation is different in the case of a CCA2 attack. As in the case of a CCA1 attack, a CCA2 attacker has an oracle access to the decryption function corresponding to the decryption key in use; however, the adversary can query the oracle *even after* the challenge has been given to him, under the restriction that he cannot ask the oracle to decrypt the challenge c itself.

We start by observing that when applying a randomized encoding to a CCA2-secure encryption scheme, CCA2 security may be lost. Indeed, in the resulting encryption one can easily modify a given ciphertext challenge $\hat{c} = \hat{E}(e, x, r)$ into a ciphertext $\hat{c}' \neq \hat{c}$ which is also an encryption of the same message under the same encryption key. This can be done by applying the decoder (of the randomized encoding $\hat{E}$) and then the simulator on $\hat{c}$, that is $\hat{c}' = S(C(\hat{c}))$. Hence, one can break the encryption by simply asking the decryption oracle to decrypt $\hat{c}'$.

It is instructive to understand why the previous arguments fail to generalize to the case of CCA2 security. In the case of CCA1 attacks we transformed an adversary $\hat{B}$ that breaks the encoding $\hat{\mathcal{E}}$ into an adversary B for the original scheme in the following way: (1) we used the simulator to convert a challenge $c = E(e, m_b)$ into a challenge $\hat{c}$ which is an encryption of the same message under $\hat{\mathcal{E}}$; (2) when $\hat{B}$ asks $\hat{D}$ to decrypt a ciphertext $\hat{c}'$, the adversary B uses the decoder (of the encoding) to translate $\hat{c}'$ into a ciphertext c' of the same message under the scheme $\mathcal{E}$, and then asks the decryption oracle D to decrypt c'. However, recall that in a CCA2 attack the adversaries are not allowed to ask the oracle to decrypt the challenge itself (after the challenge is presented). So if $c' = c$ but $\hat{c}' \neq \hat{c}$, the adversary B cannot answer the (legitimate) query of $\hat{B}$.

To complement the above, we show that when applying a randomized encoding to a CCA2-secure encryption scheme not all is lost. Specifically, the resulting scheme still satisfies *Replayable CCA security (RCCA)*, a relaxed variant of CCA2 security that was suggested in [12]. Loosely speaking, RCCA security captures encryption schemes that are CCA2 secure except that they allow anyone to generate new ciphers that decrypt to the same value as a given ciphertext. More precisely, an RCCA attack is a CCA2 attack in which the adversary cannot ask the oracle to decrypt *any* cipher c' that decrypts to either m_0 or m_1 (cf. [12, Figure 3]). This limitation prevents the problem raised in the CCA2 proof, in which a legitimate query for $\hat{D}$ translates by the decoder into an illegitimate query for D. That is, if $\hat{c}'$ does not decrypt under $\hat{\mathcal{E}}$ to neither m_0 nor m_1, then (by correctness) the ciphertext c' obtained by applying the decoder to $\hat{c}'$ does not decrypt to any of these messages either. Hence, randomized encoding preserves RCCA security. As argued in [12], RCCA security suffices in most applications of CCA2 security.