Chapter 4
Cryptography in $\mathbf{NC}^0$

Abstract In this chapter we show that randomized encoding preserves the security of many cryptographic primitives. We also construct an (information-theoretic) encoding in $\mathbf{NC}^0_4$ for any function in $\mathbf{NC}^1$ or even $\oplus\mathbf{L}/poly$. The combination of these results gives a compiler that takes as an input a code of an $\mathbf{NC}^1$ implementation of some cryptographic primitive and generates an $\mathbf{NC}^0_4$ implementation of the same primitive.

4.1 Introduction

As indicated in Chap. 1, the possibility of implementing most cryptographic primitives in $\mathbf{NC}^0$ 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 $\mathbf{P} \neq \mathbf{NP}$, we cannot expect unconditional results and have to rely on some unproven assumptions.[1] 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 $\mathbf{NC}^0$ 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.

4.1.1 Previous Work

Recall that a pseudorandom generator (PRG) $G : \{0,1\}^n \to \{0,1\}^l$ is a deterministic function that expands a short random n-bit string (aka "seed") into a longer "pseudorandom" string of length $l > n$. Pseudorandom *functions* [74] can viewed as PRGs with exponential output length and direct access. Linial et al. [108] show that

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

B. Applebaum, *Cryptography in Constant Parallel Time*,
Information Security and Cryptography,
DOI 10.1007/978-3-642-17367-7_4,

such functions cannot be computed even in $\mathbf{AC}^0$.[2] However, no such impossibility result is known for standard PRGs.

The existence of PRGs in $\mathbf{NC}^0$ has been studied in [50, 112]. Cryan and Miltersen [50] observe that there is no PRG in $\mathbf{NC}^0_2$, and prove that there is no PRG in $\mathbf{NC}^0_3$ achieving a superlinear stretch; namely, one that stretches n bits to $n + \omega(n)$ bits.[3] Mossel et al. [112] extend this impossibility to $\mathbf{NC}^0_4$. Viola [138] shows that a PRG in $\mathbf{AC}^0$ with superlinear stretch cannot be obtained from a one-way function (OWF) via non-adaptive black-box constructions. This result can be extended to rule out such a construction even if we start with a PRG whose stretch is sublinear. Negative results for other restricted computation models appear in [66, 147].

On the positive side, Impagliazzo and Naor [91] construct a (sublinear-stretch) PRG in $\mathbf{AC}^0$, relying on an intractability assumption related to the subset-sum problem. PRG candidates in $\mathbf{NC}^1$ (or even $\mathbf{TC}^0$) are more abundant, and can be based on a variety of standard cryptographic assumptions including ones related to the intractability of factoring [103, 117], discrete logarithms [36, 117, 144] and lattice problems [3, 86].[4]

Unlike the case of pseudorandom generators, the question of one-way functions in $\mathbf{NC}^0$ is relatively unexplored. The impossibility of OWFs in $\mathbf{NC}^0_2$ follows from the easiness of 2-SAT [50, 69]. Håstad [85] constructs a family of permutations in $\mathbf{NC}^0$ whose inverses are P-hard to compute. Cryan and Miltersen [50], improving on [1], present a circuit family in $\mathbf{NC}^0_3$ 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 $\mathbf{NC}^1$ function often serves as the underlying cryptographic *assumption*.) Finally, Goldreich [69] suggests a candidate OWF in $\mathbf{NC}^0$, whose conjectured security does not follow from any well-known assumption.

4.1.2 Our Results

4.1.2.1 A General Compiler

Our main result is that any OWF (resp., PRG) in a relatively high complexity class, containing uniform $\mathbf{NC}^1$ and even $\oplus\mathbf{L}/poly$, can be efficiently "compiled" into a

[2] Indeed, the low noise-sensitivity of $\mathbf{AC}^0$ functions (cf. [137, Lemma 6.6]) allows us to efficiently distinguish them from truly random function, e.g., by querying the function on a pair of random points which are $1/\sqrt{n}$-close to each other in Hamming distance.

[3] 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 $n^{1-\varepsilon}$ bits, for an arbitrary $\varepsilon > 0$. Also, an $\mathbf{NC}^0$ PRG with some linear stretch can be composed with itself a constant number of times to yield an $\mathbf{NC}^0$ PRG with an arbitrary linear stretch.

[4] In some of these constructions it seems necessary to allow a *collection* of $\mathbf{NC}^1$ 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. [70, Sect. 2.4.2]). See Appendix 4.9 for further discussion of this slightly relaxed notion of PRG.

corresponding OWF (resp., sublinear-stretch PRG) in $\mathbf{NC}^0_4$. 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 $\mathbf{NC}^0$ follows from a variety of standard assumptions and is not affected by the potential weakness of a particular algebraic structure. 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 $\mathbf{NC}^0_4$ PRG, as there is no PRG with superlinear stretch in $\mathbf{NC}^0_4$ [112].

The above results extend to other cryptographic primitives including one-way permutation, encryption, signature, commitment, and collision-resistant hashing. Aiming at $\mathbf{NC}^0$ implementations, we can use our machinery in two different ways: (1) compile a primitive in a relatively high complexity class (say $\mathbf{NC}^1$) 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 $\mathbf{NC}^0$ primitives we already constructed (e.g., OWF or PRG), to obtain new $\mathbf{NC}^0$ primitives. Of course, this approach is useful only when the reduction itself is in $\mathbf{NC}^0$. If the reduction is in $\mathbf{NC}^1$ one can combine the two approaches: first apply the $\mathbf{NC}^1$ reduction to an $\mathbf{NC}^0$ primitive of type X that was already constructed (e.g., OWF or PRG) to obtain a new $\mathbf{NC}^1$ primitive of type Y, and then use the first approach to compile the latter primitive into an $\mathbf{NC}^0$ primitive (of type Y). As in the first approach, this construction requires us to prove that a randomized encoding of a primitive Y preserves its security. We mainly adopt the first approach, since most of the known reductions between primitives are not in $\mathbf{NC}^0$. (An exception in the case of symmetric encryption and zero-knowledge proofs will be discussed in Sect. 4.6.)

A Caveat It is important to note that in the case of two-party primitives (such as encryption schemes, signatures, commitments and zero-knowledge proofs) our compiler yields an $\mathbf{NC}^0$ sender (i.e., encrypting party, committing party, signer or prover, according to the case) but does not promise anything regarding the parallel complexity of the receiver (the decrypting party or verifier). In fact, we prove that, in all these cases, it is *impossible* to implement the receiver in $\mathbf{NC}^0$, regardless of the complexity of the sender. An interesting feature of the case of commitment is that we can also improve the parallel complexity at the receiver's end. Specifically, our receiver can be realized by an $\mathbf{NC}^0$ circuit with a single unbounded fan-in AND gate at the top. The same holds for applications of commitment such as coin-flipping and zero-knowledge proofs.

4.1.2.2 Parallel Reductions

In some cases our techniques yield $\mathbf{NC}^0$ reductions between different cryptographic primitives. (Unlike the results discussed above, here we consider *unconditional* reductions that do not rely on unproven assumptions.) Specifically, we get new $\mathbf{NC}^0$ constructions of PRG and non-interactive commitment from OWF and one-to-one

OWF, respectively. These reductions are obtained by taking a standard $\mathbf{NC}^1$ reduction of a primitive $\mathscr{P}$ from a primitive $\mathscr{P}'$, and applying our compiler to it. This works only for reductions that make non-adaptive calls to $\mathscr{P}'$. There are known reductions of PRGs and non-interactive commitments from OWF and one-to-one OWF which admit such a structure, and thus can be compiled into $\mathbf{NC}^0$ reductions. (We remark that if the original reduction uses the underlying primitive $\mathscr{P}'$ as a black-box then so does the new reduction. This should not be confused with the fact that the $\mathbf{NC}^0$ reduction is obtained by using the "code" of the original reduction. Indeed, all the $\mathbf{NC}^0$ reductions obtained in this chapter are black-box reductions.)

4.1.2.3 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 $\mathbf{NC}^0_5$ constructed by Mossel et al. [112], we obtain a linear-stretch ε-biased generator in $\mathbf{NC}^0_3$. This generator has optimal locality, answering an open question posed in [112]. It is also essentially optimal with respect to stretch, since locality 3 does not allow for a superlinear stretch [50]. Our techniques apply also to other types of non-cryptographic PRGs such as generators for space-bounded computation [23, 119], yielding such generators (with sublinear stretch) in $\mathbf{NC}^0_3$.

4.1.3 Overview of Techniques

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

1. (Correctness.) 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. (Privacy.) There exists a simulator algorithm that given $f(x)$ samples from the distribution $\hat{f}(x,r)$ induced by a uniform choice of r. That is, the distribution $\hat{f}(x,r)$ hides all the information about x except for the value $f(x)$. In particular, if $f(x) = f(x')$ then the random variables $\hat{f}(x,r)$ and $\hat{f}(x',r')$ should be identically distributed.

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.

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 us to efficiently encode relatively complex functions f by functions $\hat{f}$ in $\mathbf{NC}^0$. These two issues are addressed in the following subsections.

4.1.3.1 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 the distributions $\hat{f}(x, r)$ and $\hat{f}(x', r')$ be only *statistically close*, or even *computationally indistinguishable*. 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 pseudorandom, or even that its output is longer than its input. However, by imposing suitable "regularity" requirements (e.g., the properties of *perfect* encoding cf. Definition 3.4) 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.

4.1.3.2 Complexity of Randomized Encodings

It remains to address the second issue: can we encode a complex function f by an $\mathbf{NC}^0$ 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[5] based on Barrington's theorem [24], combined with a randomization technique of Kilian [104].

Suppose f is a boolean function in $\mathbf{NC}^1$. (Non-boolean functions are handled by concatenation. See Lemma 3.1.) By Barrington's theorem, evaluating $f(x)$, for such a function f, reduces to computing an iterated product of polynomially many

[5] In fact, a modified version of this approach has been applied for constructing randomizing polynomials in [49].

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 $\mathbf{NC}^0$, since it is impossible to sample a uniform element of S_5 in $\mathbf{NC}^0$ (even up to a negligible statistical distance).[6] 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 $\mathbf{NC}^1$ and achieve a very small constant locality.

Randomizing Polynomials The concept of randomizing polynomials was introduced by Ishai and Kushilevitz [92] 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 $\mathbb{F}$ and the outputs of $\hat{f}$ as multivariate polynomials in the variables x and r. In this work, we will always let $\mathbb{F} = \mathbb{F}_2$.

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 [92, 93] that every boolean function $f : \{0,1\}^n \to \{0,1\}$ admits a representation by *degree-3* randomizing polynomials whose number of inputs and outputs is at most *quadratic* in its (mod-2) branching program size (cf. Sect. 2.3). (Moreover, this degree bound is tight in the sense that most boolean functions do not admit a degree-2 representation.) Recall that a representation of a non-boolean function can be obtained by concatenating representations of its output bits, using independent blocks of random inputs. (See Lemma 3.1.) This concatenation leaves the degree unchanged.

The above positive result implies that functions whose output bits can be computed in the complexity class $\oplus\mathbf{L}/poly$ 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 (i.e., perfect encoding). We note, however, that different constructions from the literature [49, 92, 93] are incomparable

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

in terms of their exact efficiency and the security-preserving features they satisfy. Hence, different constructions may be suitable for different applications.

Degree vs. Locality Combining our general methodology with the above results on randomizing polynomials already brings us close to our goal, as it enables "degree-3 cryptography". Taking on from here, we show that any function $f : \{0,1\}^n \to \{0,1\}^m$ of algebraic degree d admits an efficient randomized encoding $\hat{f}$ of (degree d and) locality $d+1$. That is, each output bit of $\hat{f}$ can be computed by a degree-d polynomial over $\mathbb{F}_2$ 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.

4.1.4 Organization

In Sect. 4.2 we construct a perfect and statistical encoding in $\mathbf{NC}^0_4$ for various log-space classes (i.e., $\oplus\mathbf{L}/poly$ and $\mathbf{NL}/poly$). We then apply randomized encodings to obtain $\mathbf{NC}^0$ implementations of different primitives: OWFs (Sect. 4.3), cryptographic and non-cryptographic PRGs (Sect. 4.4), collision-resistant hashing (Sect. 4.5), public-key and symmetric encryption (Sect. 4.6) and other cryptographic primitives (Sect. 4.7). We conclude in Sect. 4.8 with a summary of the results obtained in this chapter.

4.2 $\mathbf{NC}^0$ Encoding for Functions in $\oplus\mathbf{L}/poly$ and $\mathbf{NL}/poly$

In this section we construct randomized encodings in $\mathbf{NC}^0$. We first review a construction from [93] 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 us to transform a degree-d encoding to a $(d+1)$-local encoding. Finally, we discuss extensions to other types of BPs.

4.2.1 Degree-3 Randomizing Polynomials from mod-2 BPs

Let $BP = (G, \phi, s, t)$ be a mod-2 BP of size ℓ, computing a boolean[7] 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

[7] 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 $p = 2$.

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

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

$\ell \times \ell$ 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 Fig. 4.1.)

Fact 4.1 ([93]) $f(x) = \det(L(x))$, *where the determinant is computed over* $\mathbb{F}_2$.

Proof Sketch Since G is acyclic, the number of $s - t$ paths in G_x mod 2 can be written as $(I + A(x) + A(x)^2 + \cdots + A(x)^\ell)_{s,t} = (I - A(x))^{-1}_{s,t}$ where I denotes an $\ell \times \ell$ identity matrix and all arithmetic is over $\mathbb{F}_2$. 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:

$$\big(I - A(x)\big)^{-1}_{s,t} = (-1)^{s+t} \frac{\det(-L(x))}{\det(I - A(x))} = \det L(x). \qquad \square$$

Let $r^{(1)}$ and $r^{(2)}$ be vectors over $\mathbb{F}_2$ 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 Fig. 4.1.)

Fact 4.2 ([93]) *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* $R_1(r^{(1)}) M R_2(r^{(2)}) = M'$.

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 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 canonical 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 4.1 (**Implicit in** [93]) *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 B 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 4.1 and 4.2, $\det(M) = \det(L(x)) = f(x)$, 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 4.1 and 4.2, 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 4.2) 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. □

4.2.2 Reducing the Locality

It remains to convert the degree-3 encoding into one in $\mathbf{NC}^0$. 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 $\mathbf{NC}^0$ 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 Sect. 4.1.3.2. (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 4.1 (Locality construction) *Let* $f(x) = T_1(x) + \cdots + T_k(x)$, *where* $f, T_1, \ldots, T_k : \mathbb{F}_2^n \to \mathbb{F}_2$ *and summation is over* $\mathbb{F}_2$. *The local encoding* $\hat{f} : \mathbb{F}_2^{n+(2k-1)} \to \mathbb{F}_2^{2k}$ *is defined by*:

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

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 4.2 (Locality lemma) *Let* f *and* $\hat{f}$ *be as in Construction* 4.1. *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 $(r_1, \ldots, r_{2k-1}, y - (r_1 + \cdots + r_{2k-1}))$. Obviously, $S(y)$ is uniformly distributed over the $2k$-length strings whose bits sum up to y over $\mathbb{F}_2$. 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 $1 \le i \le k$, the values of x, w_i uniquely determine r_i. For $1 \le i \le k-1$, 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 Sect. 4.2.3 we describe a graph-based generalization of Construction 4.1, 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 4.1 $\oplus\mathbf{L}/poly \subseteq \mathbf{PREN}$. *Moreover, any* $f \in \oplus\mathbf{L}/poly$ *admits a perfect randomized encoding in* $\mathbf{NC}^0_4$ *whose degree is* 3.

Proof The theorem is derived by combining the degree-3 construction of Lemma 4.1 together with the Locality Lemma 4.2, using the Composition Lemma 3.3 and the Concatenation Lemma 3.2. □

Remark 4.1 An alternative construction of perfect randomized encodings in $\mathbf{NC}^0$ can be obtained using a randomizing polynomials construction from [93, Sect. 3], which is based on an information-theoretic variant of Yao's garbled circuit technique [145]. This construction yields an encoding with a (large) constant locality, without requiring an additional "locality reduction" step (of Construction 4.1). This construction is weaker than the current one in that it only efficiently applies to functions in $\mathbf{NC}^1$ rather than $\oplus\mathbf{L}/poly$. For functions in $\mathbf{NC}^1$, 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 [92, 93]. In particular, Theorem 2 in [93] encodes non-deterministic branching programs by perfectly-correct statistically-private functions of degree 3 (over $\mathbb{F}_2$). Hence, by using Lemmas 4.2, 3.3, we get a perfectly-correct statistically-private encoding in $\mathbf{NC}^0_4$ for functions in $\mathbf{NL}/poly$. (In fact, we can also use [92, 93] to obtain *perfectly-private* statistically-correct encoding in $\mathbf{NC}^0_4$ for non-deterministic branching programs.) Based on the above, we get the following theorem:

Theorem 4.2 $\mathbf{NL}/poly \subseteq \mathbf{SREN}$. *Moreover, any* $f \in \mathbf{NL}/poly$ *admits a perfectly-correct statistically-private randomized encoding in* $\mathbf{NC}^0_4$.

We can rely on Theorem 4.1 to derive the following corollary.

Corollary 4.1 *Any function* f *in* **PREN** (*resp.*, **SREN**, **CREN**) *admits a perfect* (*resp.*, *statistical*, *computational*) *randomized encoding of degree* 3 *and locality* 4 (*i.e.*, *in* $\mathbf{NC}^0_4$).

Proof We first encode f by a perfect (resp., statistical, computational) encoding $\hat{f}$ in $\mathbf{NC}^0$, guaranteed by the fact that f is in **PREN** (resp., **SREN**, **CREN**). Then, since $\hat{f}$ is in $\oplus\mathbf{L}/poly$, we can use Theorem 4.1 to perfectly encode $\hat{f}$ by a function

$\hat{f}'$ in $\mathbf{NC}^0_4$ whose degree is 3. By the Composition Lemmas 3.3 and 3.4, $\hat{f}'$ perfectly (resp. statistically, computationally) encodes the function f. □

4.2.3 A Generalization of the Locality Construction

In the Locality Construction 4.1, we showed how to encode a degree d function by an $\mathbf{NC}^0_{d+1}$ 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 4.2 (General locality construction) *Let $f(x) = T_1(x) + \cdots + T_k(x)$, where $f, T_1, \ldots, T_k : \mathbb{F}_2^n \to \mathbb{F}_2$ and summation is over $\mathbb{F}_2$. Let $G = (V, E)$ be a directed graph with k nodes $V = \{1, \ldots, k\}$ and m edges. The encoding $\hat{f}_G : \mathbb{F}_2^{n+m} \to \mathbb{F}_2^k$ is defined by*:

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

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 4.3 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 4.1 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 4.3 (Generalized locality lemma) *Let f and $\hat{f}_G$ be as in Construction 4.2. 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 4.2). Namely, given $y \in \{0,1\}$, the simulator S chooses $k-1$ random bits $r_1, \ldots, r_{k-1}$ and outputs $(r_1, \ldots, r_{k-1}, y - (r_1 + \cdots + r_{k-1}))$. 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 $w \in \text{support}(S(f(x)))$ 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. □

4.3 One-Way Functions in $\mathbf{NC}^0$

A *one-way function* (OWF) $f : \{0,1\}^* \to \{0,1\}^*$ is a polynomial-time computable function that is hard to invert; namely, every (non-uniform) 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 4.1 (**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 non-uniform polynomial-time algorithm, A, we have

$$\Pr_{x \leftarrow U_n}\left[A\left(1^n, f(x)\right) \in f^{-1}\left(f(x)\right)\right] \leq \text{neg}(n).$$

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 (non-uniform) polynomial-time algorithm, A, and all sufficiently large n's

$$\Pr_{x \leftarrow U_n}\left[A\left(1^n, f(x)\right) \notin f^{-1}\left(f(x)\right)\right] > \frac{1}{p(n)}.$$

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 3.1, 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).

4.3.1 Key Lemmas

In the following we show that a perfectly correct and statistically(or even computationally) private randomized encoding $\hat{f}$ of a OWF f is also a OWF. The idea, as described in Sect. 4.1.3.1, is to argue that the hardness of inverting $\hat{f}$ reduces to the hardness of inverting f. The case of a randomized encoding that does not enjoy perfect correctness is more involved and will be dealt with later in this section.

Lemma 4.4 *Suppose that $f : \{0,1\}^* \to \{0,1\}^*$ is a one-way function (respectively, weak one-way function) and $\hat{f}(x,r)$ is a perfectly correct, computationally private encoding of f. Then $\hat{f}$, viewed as a single-argument function, is also one-way (respectively, weakly one-way).*

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. Let $\hat{A}$ be an adversary that inverts $\hat{f}(x,r)$ with success probability $\phi(n+m)$. We use $\hat{A}$ to construct an efficient adversary A that inverts f with similar success. On input $(1^n, y)$, the adversary A runs S, the simulator of $\hat{f}$, on the input $(1^n, y)$ and gets a string $\hat{y}$ as the output of S. Next, A runs the inverter $\hat{A}$ on the input $(1^{n+m}, \hat{y})$, getting (x', r') as the output of $\hat{A}$ (i.e., $\hat{A}$ "claims" that $\hat{f}(x', r') = \hat{y}$). A terminates with output x'.

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

CORRECTNESS: We analyze the success probability of A 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 $\hat{A}$, is always in the support of $\hat{f}(x, U_m)$. Hence, if $\hat{A}$ succeeds (that is, indeed $\hat{y} = \hat{f}(x', r')$) then so does A (namely, $f(x') = y$). Finally, observe that (by Fact 2.5)

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

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

We now show that when S is computationally private, we lose only negligible success probability in the above; that is, A succeeds with probability $\geq \phi(n+m) - \mathrm{neg}(n)$. To see this, it will be convenient to define a distinguisher D that on input $(1^n, y, \hat{y})$ computes $(x', r') = \hat{A}(1^{n+m}, \hat{y})$, and outputs 1 if $f(x') = y$ and 0 otherwise. Clearly, the success probability of A on $f(U_n)$ can be written as $\Pr_{x \leftarrow U_n}[D(1^n, f(x), S(f(x))) = 1]$. On the other hand, we showed above that $\Pr_{x \leftarrow U_n, r \leftarrow U_{m(n)}}[D(1^n, f(x), \hat{f}(x,r)) = 1] \geq \phi(n+m)$. Also, by the computational privacy of S, and Facts 2.10, 2.8, we have

$$\left(f(U_n), S\left(f(U_n)\right)\right) \stackrel{\mathrm{c}}{\equiv} \left(f(U_n), \hat{f}(U_n, U_{m(n)})\right).$$

Hence, since D is polynomial-time computable, we have

$$\begin{aligned}
&\Pr_{x \leftarrow U_n}\left[D\left(1^n, f(x), S\left(f(x)\right)\right) = 1\right] \\
&\geq \Pr_{x \leftarrow U_n, r \leftarrow U_{m(n)}}\left[D\left(1^n, f(x), \hat{f}(x,r)\right) = 1\right] - \mathrm{neg}(n) \\
&\geq \phi(n+m) - \mathrm{neg}(n).
\end{aligned}$$

Since f is hard to invert (respectively, slightly hard to invert), the quantity $\phi(n+m)$ must be negligible (respectively, bounded by $1 - 1/p(n)$ for some fixed polynomial $p(\cdot)$) and so $\hat{f}$ is also hard to invert (respectively, slightly hard to invert). □

The efficiency of the simulator S is essential for Lemma 4.4 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 4.4 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 $\mathbf{NC}^0$ into a OWF in $\mathbf{NC}^0$.

Definition 4.2 (**Distributionally one-way function** [90]) 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 (non-uniform) polynomial-time algorithm, A, and all sufficiently large n's, $\|(A(1^n, f(U_n)), f(U_n)) - (U_n, f(U_n))\| > \frac{1}{p(n)}$.

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

Lemma 4.5 *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 $\varepsilon(n) \stackrel{\text{def}}{=} \Pr_{x \leftarrow U_n}[f(x) \neq g(x)]$. 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, A, such that for infinitely many n's, $\|(A(1^n, f_n(U_n)), f_n(U_n)) - (U_n, f_n(U_n))\| \leq \frac{1}{2p(n)}$. 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 $\|(A(1^n, f_n(U_n)), f_n(U_n)) - (x', f_n(U_n))\| \leq \frac{1}{2p(n)}$. It follows that for infinitely many n's

$$\begin{aligned}&\Pr\left[A\left(1^n, f(U_n)\right) \in g_n^{-1}\left(f_n(U_n)\right)\right] \\ &\quad \geq \Pr_{x' \leftarrow f_n^{-1}(f_n(U_n))}\left[x' \in g_n^{-1}\left(f_n(U_n)\right)\right] - \frac{1}{2p(n)}. \end{aligned} \tag{4.1}$$

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

$$\begin{aligned}&\Pr\left[A\left(1^n, g_n(U_n)\right) \in g_n^{-1}\left(g_n(U_n)\right)\right] \\ &\geq \Pr\left[A\left(1^n, f_n(U_n)\right) \in g_n^{-1}\left(f_n(U_n)\right)\right] - \varepsilon(n) \\ &\geq \Pr_{x' \leftarrow f_n^{-1}(f_n(U_n))}\left[x' \in g_n^{-1}\left(f(U_n)\right)\right] - \frac{1}{2p(n)} - \varepsilon(n) \\ &= \Pr_{x' \leftarrow f_n^{-1}(f_n(U_n))}\left[g_n\left(x'\right) = f_n(U_n)\right] - \frac{1}{2p(n)} - \varepsilon(n) \\ &= \Pr_{x' \leftarrow f_n^{-1}(f_n(U_n))}\left[g_n\left(x'\right) = f_n\left(x'\right)\right] - \frac{1}{2p(n)} - \varepsilon(n) \\ &= 1 - \varepsilon(n) - \frac{1}{2p(n)} - \varepsilon(n) \\ &\geq 1 - \frac{1}{p(n)}, \end{aligned}$$

where the first inequality is due to the fact that f and g are ε-close, the second inequality uses (4.1), the second equality follows since $f(U_n) = f(x')$, the third equality is due to $x' \equiv U_n$, and the last inequality follows since ε is negligible. □

We now use Lemma 4.5 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 4.6 *Suppose that $f : \{0,1\}^* \to \{0,1\}^*$ is a weak one-way function and $\hat{f}(x,r)$ is a computational randomized encoding of f. Then $\hat{f}$, viewed as a single-argument function, is distributionally one-way.*

Proof Let B and S be the decoder and the simulator of $\hat{f}$. Define the function $\hat{g}(x,r)$ in the following way: if $B(\hat{f}(x,r)) \neq f(x)$ then $\hat{g}(x,r) = \hat{f}(x,r')$ for some r' such that $B(\hat{f}(x,r')) = f(x)$ (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 B perfectly decodes $f(x)$ from $\hat{g}(x,r)$). Moreover, by the statistical correctness of B, 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 computationally-private encoding of f (because $\hat{g}(x,U_m) \stackrel{s}{\equiv} \hat{f}(x,U_m) \stackrel{c}{\equiv} S(f(x))$). Since f is slightly hard to invert, it follows from Lemma 4.4 that $\hat{g}$ is also slightly hard to invert. (Note that $\hat{g}$ might not be computable in polynomial time; however the proof of Lemma 4.4 only requires that the simulator's running time and the randomness complexity of $\hat{g}$ be polynomially bounded.) Finally, it follows from Lemma 4.5 that $\hat{f}$ is distributionally one-way as required. □

4.3.2 Main Results

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

Theorem 4.3 *If there exists a OWF in* **CREN** *then there exists a OWF in* $\mathbf{NC}^0_4$.

Proof Let f be a OWF in **CREN**. By Lemma 4.6, we can construct a distributional OWF $\hat{f}$ in $\mathbf{NC}^0$, and then apply a standard transformation (cf. [90, Lemma 1], [70, p. 96], [144]) to convert $\hat{f}$ to a OWF $\hat{f}'$ in $\mathbf{NC}^1$. This transformation consists of two steps: Impagliazzo and Luby's $\mathbf{NC}^1$ construction of weak OWF from distributional OWF [90], and Yao's $\mathbf{NC}^0$ construction of a (standard) OWF from a weak OWF [144] (see [70, Sect. 2.3]).[8] Since $\mathbf{NC}^1 \subseteq$ **PREN** (Theorem 4.1), we can use Lemma 4.4 to encode $\hat{f}'$ by a OWF in $\mathbf{NC}^0$, in particular, by one with locality 4. □

[8] We will later show a degree-preserving transformation from a distributional OWF to a OWF (Lemma 6.2); however, in the current context the standard transformation suffices.

Combining Lemmas 4.4, 3.5 and Corollary 4.1, we get a similar result for one-way permutations (OWPs).

Theorem 4.4 *If there exists a one-way permutation in* **PREN** *then there exists a one-way permutation in* $\mathbf{NC}^0_4$.

In particular, using Theorems 4.1 and 4.2, we conclude that a OWF (resp., OWP) in $\mathbf{NL}/poly$ or $\oplus\mathbf{L}/poly$ (resp., $\oplus\mathbf{L}/poly$) implies a OWF (resp., OWP) in $\mathbf{NC}^0_4$.

Theorem 4.4 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 $\mathbf{NC}^0$). 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 Sect. 4.2 and is preserved under composition and concatenation. Thus, the existence of trapdoor permutations computable in $\mathbf{NC}^0_4$ follows from their existence in $\oplus\mathbf{L}/poly$.

More formally, a collection of permutations $\mathscr{F} = \{f_z : D_z \to D_z\}_{z\in Z}$ 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 a formal definition see [70, Definition 2.4.4].) By the above argument we derive the following theorem.

Theorem 4.5 *If there exists a trapdoor permutation* $\mathscr{F}$ *whose function evaluator* F *is in* $\oplus\mathbf{L}/poly$ *then there exists a trapdoor permutation* $\hat{\mathscr{F}}$ *whose function evaluator* $\hat{F}$ *is in* $\mathbf{NC}^0_4$.

Remark 4.1 (**On Theorems 4.3, 4.4 and 4.5**)

1. **Constructiveness**. In Sect. 4.2, we give a constructive way of transforming a branching program representation of a function f into an $\mathbf{NC}^0$ circuit computing its encoding $\hat{f}$. It follows that Theorems 4.3, 4.4 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 $\mathbf{NC}^0$ representation of a corresponding OWF (resp., OWP) $\hat{f}$. A similar result holds for other cryptographic primitives considered in this chapter.
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 [49, 93].

3. **Generalizations**. The proofs of the above theorems carry over to OWF whose security holds against efficient *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 4.9 for discussion).
4. **Concrete assumptions**. The existence of a OWF in **SREN** (in fact, even in $\mathbf{NC}^1$) follows from the intractability of factoring and lattice problems [3]. The existence of a OWF *collection* in **SREN** follows from the intractability of the discrete logarithm problem. Thus, we get OWFs in $\mathbf{NC}^0_4$ under most standard cryptographic assumptions. In the case of OWP, we can get a collection of OWPs in $\mathbf{NC}^0_4$ based on discrete logarithm [36, 144] (see also Appendix 4.9) or RSA with a small exponent [127].[9] The latter assumption is also sufficient for the construction of TDP in $\mathbf{NC}^0_4$.

4.4 Pseudorandom Generators in $\mathbf{NC}^0$

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 non-uniform polynomial-time algorithm (alternatively, polynomial-time algorithm). 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 [68, Chap. 3] for a comprehensive and unified treatment of pseudorandomness.

[9] Rabin's factoring-based OWP collection [123] seems insufficient for our purposes, as it cannot be defined over the set of *all* strings of a given length. The standard modification (cf. [72, p. 767]) does not seem to be in $\oplus\mathbf{L}/poly$.

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.

4.4.1 Cryptographic Generators

Definition 4.3 (**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**. The ensembles $\{G(U_n)\}_{n\in\mathbb{N}}$ and $\{U_{l(n)}\}_{n\in\mathbb{N}}$ are computationally indistinguishable.

Remark 4.2 (**PRGs with sublinear stretch**) An $\mathbf{NC}^0$ pseudorandom generator, G, that stretches its input by a single bit can be transformed into another $\mathbf{NC}^0$ 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 [70, 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 4.3 (**PRGs with linear stretch**) We can also transform an $\mathbf{NC}^0_d$ pseudorandom generator, $G : \{0,1\}^n \to \{0,1\}^{cn}$, with some linear stretch factor $c > 1$ into another $\mathbf{NC}^0$ PRG, $G'\{0,1\}^n \to \{0,1\}^{c'n}$, with arbitrary linear stretch factor $c' > 1$ and larger (but constant) output locality d'. This can be done by composing G with itself a constant number of times. That is, we let $G'(x) \stackrel{\text{def}}{=} G^{\lceil \log_c c' \rceil}(x)$ where $G^{i+1}(x) \stackrel{\text{def}}{=} G(G^i(x))$ and $G^0(x) \stackrel{\text{def}}{=} x$. Since the output of a PRG is pseudorandom even if it is invoked on a pseudorandom seed (see [70, p. 176]), the composed generator G' is also a PRG. Clearly, this PRG stretches n input bits to $c'n$ output bits and its locality is $d^{\lceil \log_c c' \rceil} = O(1)$.[10]

Remark 4.2 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 4.7 *Let $\hat{G} : \{0,1\}^n \times \{0,1\}^{m(n)} \to \{0,1\}^{s(n)}$ be a perfect randomized encoding of a PRG $G : \{0,1\}^n \to \{0,1\}^{l(n)}$. Then $\hat{G}$, viewed as a single-argument function, is also a PRG.*

[10] We can also increase the stretch factor by using the standard construction of Goldreich-Micali [70, Sect. 3.3.2]. In this case the locality of G' will be $d^{\lceil (c'-1)/(c-1) \rceil}$.

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 2.5, 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 4.4 (**The role of balance and stretch preservation**) Dropping either the balance or stretch preservation requirements, Lemma 4.7 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 4.7, Theorem 4.1 and Corollary 4.1, we get:

Theorem 4.6 *If there exists a pseudorandom generator in* **PREN** (*in particular, in* $\oplus\mathbf{L}/poly$) *then there exists a pseudorandom generator in* $\mathbf{NC}^0_4$.

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. However, when G is in $\mathbf{NC}^0$, we can transform it into a PRG $\hat{G}$ in $\mathbf{NC}^0_4$ while (partially) preserving its stretch. Formally,

Theorem 4.7 *If there exists a pseudorandom generator with linear stretch in* $\mathbf{NC}^0$ *then there exists a pseudorandom generator with linear stretch in* $\mathbf{NC}^0_4$.

Proof Let G be a PRG with linear stretch in $\mathbf{NC}^0$. We can apply Theorem 4.1 to G and get, by Lemma 4.7, a PRG $\hat{G}$ in $\mathbf{NC}^0_4$. 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 $\mathbf{NC}^0$, each of its output bits is computable by a constant size branching program and thus our 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. □

We can improve Theorem 4.6 by relying on a recent result of [83] which shows that any OWF can be transformed into a PRG via an $\mathbf{NC}^1$ reduction.[11]

Theorem 4.8 *If there exists a one-way function in* **CREN** *then there exists a pseudorandom generator in* $\mathbf{NC}^0_4$.

Proof A OWF in **CREN** can be first transformed into a OWF in $\mathbf{NC}^0$ (Theorem 4.3) and then, using [83], to a PRG in $\mathbf{NC}^1$. Combined with Theorem 4.6, this yields a PRG in $\mathbf{NC}^0_4$. □

It follows that PRGs in $\mathbf{NC}^0_4$ can be constructed assuming the hardness of factoring, lattice problems, and discrete logarithm (in the latter case one gets only a collection of PRGs).

Remark 4.5 (**On unconditional $\mathbf{NC}^0$ reductions from PRG to OWF**) Our machinery can be used to obtain an $\mathbf{NC}^0$ reduction from a PRG to any OWF f, regardless of the complexity of f.[12] Moreover, this reduction only makes a *black-box* use of the underlying OWF f. The general idea is to encode the $\mathbf{NC}^1$ construction of [83] into a corresponding $\mathbf{NC}^0$ 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 **PREN** and the q_i's are in $\mathbf{NC}^0$. (The functions $g, q_1, \ldots, q_m$ 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 $\mathbf{NC}^0$ 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 $\mathbf{NC}^0$ reduction from a PRG to a OWF. The construction of [83] is of the form of $G(x)$ above (the functions $q_1, \ldots, q_m$ are simply projections there).Thus, $\hat{G}$ defines an $\mathbf{NC}^0$ reduction from a PRG to a OWF.

Comparison with Lower Bounds The results of [112] rules out the existence of a superlinear-stretch cryptographic PRG in $\mathbf{NC}^0_4$. Thus our $\mathbf{NC}^0_4$ cryptographic PRGs are not far from optimal despite their sublinear stretch. In addition, it is easy to see

[11] The seminal work of [86] gives a polynomial-time transformation which can be implemented in $\mathbf{NC}^1$ only in special cases, e.g., when the OWF is one-to-one or "regular", or in a nonuniform setting where an additional nonuniform advice of logarithmic length is employed by the construction. (See [82, 86] and [15, Remark 6.6].) Indeed, in older versions of this section [15], which predates [83] the following results (Theorem 4.8, Remark 4.5) were obtained only for the case of regular OWFs.

[12] Viola, in a concurrent work [138], obtains an $\mathbf{AC}^0$ reduction of this type.

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, but Theorem 4.6 leaves us one step short of this goal, in terms of both locality and degree.[13] See also Table 4.1. These gaps will be partially closed later in this work. Specifically, in Chap. 7 we construct a PRG with locality 4 and *linear* stretch whose security follows from a specific intractability assumption proposed by Alekhnovich in [5], while in Chap. 8 we construct a PRG with locality 3 and degree 2 under the assumption that it is hard to decode a random linear code. Moreover, the latter construction also enjoys an optimal *input* locality.

4.4.2 ε-Biased Generators

The proof of Lemma 4.7 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 4.4 (ε-biased generator) An *ε-biased generator* 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 $n \in \mathbb{N}$.
- **ε-bias**. For every linear function $L : \{0,1\}^{l(n)} \to \{0,1\}$ and all sufficiently large n's

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

(where a function L is *linear* if its degree over $\mathbb{F}_2$ is 1). By default, the function $\varepsilon(n)$ is required to be negligible.

Lemma 4.8 *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 $G : \{0,1\}^n \to \{0,1\}^{l(n)}$ and let $\hat{G} : \{0,1\}^n \times \{0,1\}^{m(n)} \to \{0,1\}^{s(n)}$. Assume, towards a contradiction, that $\hat{G}$ is not ε-biased; that is, for some linear function $L : \{0,1\}^{s(n)} \to \{0,1\}$ and infinitely many n's, $|\Pr[L(\hat{G}(U_{n+m})) = 1]$

[13] Note that there exists a PRG with locality 3 if and only if there exists a PRG with degree 2. The "if" direction follows from Lemma 4.2 and Lemma 4.7, while the "only if" direction follows from Claim 3.1 and the fact that each output of an $\mathbf{NC}^0$ PRG must be balanced.

$-\Pr[L(U_s)=1]| > \frac{1}{p(n+m)} > \frac{1}{p'(n)}$, where $m = m(n)$, $s = s(n)$, and $p(\cdot)$, $p'(\cdot)$ are polynomials. Using the balance property we get,

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

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 Sect. 4.2 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 4.1 *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 3.1). 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 $\mathbf{NC}^0_4$ encoding for functions in $\oplus\mathbf{L}/poly$ (promised by Theorem 4.1) is obtained by concatenating the simulators of boolean functions in $\oplus\mathbf{L}/poly$. By Observation 4.1, this simulator is linear. Therefore, by Lemma 4.8, we can construct a sublinear-stretch ε-biased generator in $\mathbf{NC}^0_4$ from any ε-biased generator in $\oplus\mathbf{L}/poly$. In fact, one can easily obtain a nontrivial ε-biased generator even in $\mathbf{NC}^0_3$ 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 4.1 and Lemma 4.8, is also ε-biased. (This generator actually fools a much larger class of statistical tests; see Sect. 4.4.3 below.) Thus, we have:

Theorem 4.9 *There is a (sublinear-stretch) ε-biased generator in $\mathbf{NC}^0_3$.*

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

Theorem 4.10 *There is a linear-stretch ε-biased generator in $\mathbf{NC}^0_3$.*

Proof Mossel et al. present an ε-biased generator in $\mathbf{NC}^0$ with degree 2 and linear stretch ([112], Theorem 13).[14] Let G be their ε-biased generator. We can apply the locality construction (4.1) to G (using concatenation) and get, by Lemma 4.8 and Observation 4.1, an ε-biased generator $\hat{G}$ in $\mathbf{NC}^0_3$. 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 $\mathbf{NC}^0$, 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. □

Comparison with Lower Bounds It is not hard to see that there is no ε-biased generator with degree 1 or locality 2.[15] In [50] it was shown that there is no superlinear-stretch ε-biased generator in $\mathbf{NC}^0_3$. Thus, our linear-stretch $\mathbf{NC}^0_3$ generator (building on the one from [112]) is not only optimal with respect to locality and degree but is also essentially optimal with respect to stretch.

4.4.3 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. [23], we refer to such distinguishers as *space-bounded distinguishers*.

[14] In fact, the generator of [112, Theorem 13] is in nonuniform-$\mathbf{NC}^0_5$ (and it has a slightly superlinear stretch). However, a similar construction gives an ε-biased generator in *uniform* $\mathbf{NC}^0$ 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 [112, Lemma 12] with a uniform construction of constant-degree bipartite expander with some "good" expansion properties – such a construction is given in [44, Theorem 7.1].

[15] 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.

Definition 4.5 (**Space-bounded distinguisher** [23]) 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 Sect. 4.4.2.

Definition 4.6 (**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., [23, 119]). In particular, a PRG for logspace computation from [23] can be computed using logarithmic space, and thus, by Theorem 4.1, admits an efficient perfect encoding in $\mathbf{NC}^0_4$. It can be shown (see proof of Theorem 4.11) that this $\mathbf{NC}^0_4$ 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 $\mathbf{NC}^0_4$ 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 Sect. 4.4.2 can do here as well.

Theorem 4.11 *There exists a* (*sublinear-stretch*) *PRG for sublinear-space computation in* $\mathbf{NC}^0_3$.

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 [46] 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'.)

Applying the locality construction to G, we obtain a perfect encoding $\hat{G}$ in $\mathbf{NC}^0_3$. (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 4.8, 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 4.8, here it does not suffice to require S to be linear and we need to rely on the stronger property guaranteed by Observation 4.1.[16]

We now formalize the above. As argued in Observation 4.1, 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 $\sigma_i(y_i)$. □

4.4.4 Pseudorandom Generators—Conclusion

We conclude this section with Table 4.1, which summarizes some of the PRGs constructed here as well as previous ones from [112] and highlights the remaining gaps. We will partially close these gaps in Chaps. 7 and 8 under specific *intractability* assumptions. (In particular, we will construct a PRG with locality 4 and *linear* stretch, as well as a PRG with output locality 3, input locality 3, and degree 2.)

4.5 Collision-Resistant Hashing in $\mathbf{NC}^0$

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

Definition 4.7 (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:

[16] 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 $\hat{M}(y_1, \ldots, y_{2n}) = y_1y_2 + y_3y_4 + \cdots + y_{2n-1}y_{2n}$ and $S(x_1, \ldots, x_{2n}) = (x_1, x_{n+1}, x_2, x_{n+2}, \ldots, x_n, x_{2n})$.

Table 4.1 Summary of known pseudorandom generators. Results of Mossel et al. [112] appear in the *top part* and results of this chapter 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 ✓ 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

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 ✓	3 ✓	2 ✓
cryptographic*	sublinear ✓	4	3

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 non-uniform polynomial-time algorithm A, 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 A).

Lemma 4.9 *Suppose $\mathscr{H} = \{h_z\}_{z\in Z}$ is collision resistant and $\hat{\mathscr{H}} = \{\hat{h}_z\}_{z\in Z}$ is a perfect randomized encoding of $\mathscr{H}$. Then $\hat{\mathscr{H}}$ 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 $\mathscr{H}$ is used as the key generation algorithm of $\hat{\mathscr{H}}$. By the uniformity of the collection $\hat{\mathscr{H}}$, 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 3.5) ensures that $x \neq x'$. Thus, an efficient algorithm that finds collisions for $\hat{\mathscr{H}}$ with non-negligible probability yields a similar algorithm for $\mathscr{H}$. □

By Lemma 4.9, Theorem 4.1 and Corollary 4.1, we get:

Theorem 4.12 *If there exists a CRHF $\mathscr{H} = \{h_z\}_{z\in Z}$ such that the function $h'(z,x) \stackrel{\text{def}}{=} h_z(x)$ is in* **PREN** *(in particular, in $\oplus\mathbf{L}/poly$), then there exists a CRHF $\hat{\mathscr{H}} = \{\hat{h}_z\}_{z\in Z}$ such that the mapping $(z,y) \mapsto \hat{h}_z(y)$ is in $\mathbf{NC}^0_4$.*

Using Theorem 4.12, we can construct CRHFs in $\mathbf{NC}^0$ based on the intractability of factoring [51], discrete logarithm [122], or lattice problems [73, 125]. All these candidates are computable in $\mathbf{NC}^1$ provided that some pre-computation is done by the key-generation algorithm. Note that the key generation algorithm of the resulting $\mathbf{NC}^0$ CRHF is not in $\mathbf{NC}^0$. For more details on $\mathbf{NC}^0$ computation of collections of cryptographic primitives see Appendix 4.9.

4.6 Encryption in $\mathbf{NC}^0$

We turn to the case of encryption. We first show that computational encoding preserves the security of a semantically secure encryption scheme both in the public-key and in the private-key setting. This result provides an $\mathbf{NC}^0$ 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 $\mathbf{NC}^0$. In Sect. 4.6.2 we show that, except for some special settings (namely, private-key encryption which is secure only for a single message of bounded length or stateful private-key encryption), decryption in $\mathbf{NC}^0$ is impossible regardless of the complexity of encryption. Finally, in Sect. 4.6.3 we explore the effect of randomized encoding on encryption schemes which enjoy stronger notions of security. In particular, we show that randomized encoding preserves security against chosen plaintext attacks (CPA) as well as a priori chosen ciphertext attacks (CCA1). However, randomized encoding does not preserve security against a posteriori chosen ciphertext attack (CCA2). Still, we show that the encoding of a CCA2-secure scheme enjoys a relaxed security property that suffices for most applications of CCA2-security.

4.6.1 Main Results

Suppose that $\mathscr{E} = (G, E, D)$ 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 [80]. Let $\hat{E}$ be a randomized encoding of E, and let $\hat{D}(d, \hat{y}) \stackrel{\text{def}}{=} D(d, C(\hat{y}))$ be the composition of D with the decoder B of $\hat{E}$. We argue that the scheme $\hat{\mathscr{E}} \stackrel{\text{def}}{=} (G, \hat{E}, \hat{D})$ is also a public-key encryption scheme. The efficiency and correctness of $\hat{\mathscr{E}}$ are guaranteed by the uniformity of the encoding and its correctness. Using the efficient simulator of $\hat{E}$, we can reduce the security of $\hat{\mathscr{E}}$ to that of $\mathscr{E}$. Namely, given an efficient adversary $\hat{A}$ that distinguishes between encryptions of x and x' under $\hat{\mathscr{E}}$, we can break $\mathscr{E}$ 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 4.8 (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

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

  where $\varepsilon(n)$ is a negligible function and the probability is taken over the internal coin tosses of algorithms E and D.
- **Security**. For every polynomial $\ell(\cdot)$, and every family of plaintexts $\{x_n\}_{n\in\mathbb{N}}$ and $\{x'_n\}_{n\in\mathbb{N}}$ where $x_n, x'_n \in \{0,1\}^{\ell(n)}$, it holds that

$$\big(e \leftarrow G_1\big(1^n\big), E\big(eG_1\big(1^n\big), x_n\big)\big) \stackrel{c}{\equiv} \big(e \leftarrow G_1\big(1^n\big), E\big(eG_1\big(1^n\big), x'_n\big)\big), \tag{4.2}$$

  where $G_1(1^n)$ denotes the first element in the pair $G(1^n)$.

The definition of a *private-key* encryption scheme is similar, except that the public key is omitted from the ensembles. That is, instead of (4.2) we require that $E(G_1(1^n), x_n) \stackrel{c}{\equiv} E(G_1(1^n), x'_n)$. An extension to multiple-message security, where the indistinguishability requirement should hold for encryptions of polynomially many messages, follows naturally (see [72, Chap. 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 4.10 *Let $\mathscr{E} = (G, E, D)$ 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 computational randomized encoding of E and let $\hat{D}(d, \hat{y}) \stackrel{\text{def}}{=} D(d, B(\hat{y}))$ be the composition of D with the decoder B of $\hat{E}$. Then, the scheme $\hat{\mathscr{E}} \stackrel{\text{def}}{=} (G, \hat{E}, \hat{D})$ 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 $\hat{\mathscr{E}}$ follows in a straightforward way from the correctness of the decoder B. Indeed, if (e, d) are in the support of $G(1^n)$, then for any plaintext x we have

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

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 $\mathscr{E}$ and the statistical correctness of $\hat{E}$.

We move on to prove the security of the construction. Let S be the efficient computational simulator of $\hat{E}$. Then, for every polynomial $\ell(\cdot)$, and every family of plaintexts $\{x_n\}_{n\in\mathbb{N}}$ and $\{x'_n\}_{n\in\mathbb{N}}$ where $x_n, x'_n \in \{0,1\}^{\ell(n)}$, it holds that

$$\begin{aligned}\left(e \leftarrow G_1\left(1^n\right), \hat{E}(e, x_n, r_n, s_n)\right) &\stackrel{c}{\equiv} \left(e \leftarrow G_1\left(1^n\right), S\left(E(e, x_n, r_n)\right)\right) \\ &\stackrel{c}{\equiv} \left(e \leftarrow G_1\left(1^n\right), S\left(E\left(e, x'_n, r_n\right)\right)\right) \\ &\stackrel{c}{\equiv} \left(e \leftarrow G_1\left(1^n\right), \hat{E}\left(e, x'_n, r_n, s_n\right)\right),\end{aligned}$$

where r_n and s_n are uniformly chosen random strings of an appropriate length. In the above the first and third transitions are due to the privacy of $\hat{E}$ and Fact 2.10, and the second transition is due to the security of $\mathscr{E}$ and Fact 2.8. Overall, the security of $\hat{\mathscr{E}}$ follows from the transitivity of the relation $\stackrel{c}{\equiv}$ (Fact 2.6). □

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

Theorem 4.13 *If there exists a secure public-key encryption scheme* (*respectively, a secure private-key encryption scheme*) $\mathscr{E} = (G, E, D)$, *such that* E *is in* **CREN** (*in particular, in* $\mathbf{NL}/poly$ *or* $\oplus\mathbf{L}/poly$), *then there exists a secure public-key encryption scheme* (*respectively, a secure private-key encryption scheme*) $\hat{\mathscr{E}} = (G, \hat{E}, \hat{D})$, *such that* $\hat{E}$ *is in* $\mathbf{NC}^0_4$.

Specifically, one can construct an $\mathbf{NC}^0$ PKE based on either factoring [35, 77, 123], the Diffie-Hellman Assumption [64, 77] or lattice problems [4, 125]. (These schemes enable an $\mathbf{NC}^1$ encryption algorithm given a suitable representation of the key.) We will later (Remark 5.5) relax these assumptions and obtain a PKE in $\mathbf{NC}^0$ based on the combination of an arbitrary (polynomial-time computable) PKE and a one-way function in **CREN**.

4.6.2 On Decryption in $\mathbf{NC}^0$

Our construction provides an $\mathbf{NC}^0$ 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 $\mathbf{NC}^0$.

4.6.2.1 Negative Results

We now show that, in many settings, decryption in $\mathbf{NC}^0$ is impossible regardless of the complexity of encryption. Here we consider standard *stateless* encryption schemes. 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 $\mathbf{NC}^0_k$, 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.)

One may still hope that the decryption function may be implemented in $\mathbf{NC}^0$ for every *fixed* value of the key d; namely, that $D(d, \cdot) \in \mathbf{NC}^0_k$ for every fixed d. It turns out that this is impossible as well. Suppose that n-bit messages are encrypted by $m = m(n)$ bit ciphertexts, i.e., $D(d, \cdot) : \{0,1\}^m \to \{0,1\}^n$. By definition, for every d the boolean function $D_1(d, \cdot)$ which decrypts the first bit of the message depends on at most k bits. Therefore, no matter what d is, $D_1(d, \cdot)$ is taken from a small set of $(2m)^k$ possible functions. An adversary can therefore guess $D_1(d, \cdot)$ and decrypt the first bit of $t > (2m)^k$ ciphertexts with overall success probability of $1/(2m)^k$. Again, it can be shown that this violates semantic security (e.g., by taking the ciphertexts to be encryptions of random n-bit messages).

Finally, let us consider the case of a single-message private-key encryption. In this case, it is impossible to implement the decryption algorithm $D(d, y)$ in $\mathbf{NC}^0_k$ while supporting an arbitrary (polynomial) message length. Indeed, when the message length exceeds $k|d|^k$ (where $|d|$ is the length of the decryption key), there must be at least $k + 1$ 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 in our first attack. That is, we can guess the k bits of the key and learn the value of $k + 1$ bits of the message with success probability 2^{-k}. Again, if we consider a randomly chosen message, this violates semantic security.

4.6.2.2 Positive Results

In contrast to the above, 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 $\mathbf{NC}^0$. This can be done by using the output of an $\mathbf{NC}^0$ 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 [72, Construction 5.3.3]). In the resulting scheme both encryption and decryption are $\mathbf{NC}^0$ functions whose inputs include the inner state of the algorithm.

4.6.3 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 priori 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 $c = E(e, x_b)$ find out which of the two predefined plaintexts x_0, x_1 was encrypted), and so the actual difference lies in 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 $\hat{A}$ be an adversary that breaks the encoding $\hat{\mathscr{E}}$ via a CPA attack (resp. CCA1 attack). We use $\hat{A}$ to obtain an adversary A that breaks the original scheme $\mathscr{E}$. As in the proof of Lemma 4.10, A uses the simulator to translate the challenge c, an encryption of the message x_b under $\mathscr{E}$, into a challenge $\hat{c}$, which is an encryption of the same message under $\hat{\mathscr{E}}$. Similarly, A answers the encryption queries of $\hat{A}$ (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 $\hat{A}$ asks the decryption oracle $\hat{D}$ to decrypt some ciphertext $\hat{c}'$, the adversary A uses the decoder (of the encoding) to translate $\hat{c}'$ into a ciphertext c' of the same message under the scheme $\mathscr{E}$, and then uses the decryption oracle D to decrypt c'. This allows A 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 $\hat{A}$ that breaks the encoding $\hat{\mathscr{E}}$ into an adversary A for the original scheme in the following way: (1) we used the simulator to convert a challenge $c = E(e, x_b)$ into a challenge $\hat{c}$ which is an encryption of the same message under $\hat{\mathscr{E}}$; (2) when $\hat{A}$ asks $\hat{D}$ to decrypt a ciphertext $\hat{c}'$, the adversary A uses the decoder (of the encoding) to translate $\hat{c}'$ into a ciphertext c' of the same message under the scheme $\mathscr{E}$, 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 A cannot answer the (legitimate) query of $\hat{A}$.

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 [43]. 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 x_0 or x_1 (cf. [43, Fig. 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 $\hat{\mathscr{E}}$ to either x_0 or x_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 [43], RCCA security suffices in most applications of CCA2 security.

4.7 Other Cryptographic Primitives

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 [70, 72]). 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 $\mathbf{NC}^0$ sender but does not promise anything regarding the parallel complexity of the receiver. (In all these cases, we show that it is impossible to implement the receiver in $\mathbf{NC}^0$.) 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.

4.7.1 Signatures

Let $\mathscr{S} = (G, S, V)$ 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 three algorithms run in probabilistic polynomial time, and the scheme provides correct verification for legal signatures (ones that were produced by the signing function using the corresponding signing key). The scheme is secure (unforgeable) if it is infeasible to forge a signature in a chosen message attack. Namely, any(non-uniform) 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. (When the signing algorithm is probabilistic, the attacker does not have an access to the random coin tosses of the signing algorithm.)

Let $\hat{S}$ be a computational randomized encoding of S, and $\hat{V}(v, \alpha, \hat{\beta}) \stackrel{\text{def}}{=} V(v, \alpha, B(\hat{\beta}))$ be the composition of V with the decoder B of the encoding $\hat{S}$. We claim that the scheme $\hat{\mathscr{S}} \stackrel{\text{def}}{=} (G, \hat{S}, \hat{V})$ is also a signature scheme. The efficiency and correctness of $\hat{\mathscr{S}}$ follow from the uniformity of the encoding and its correctness. To prove the security of the new scheme we use the simulator to transform an attack on $\hat{\mathscr{S}}$ into an attack on $\mathscr{S}$. Specifically, given an adversary $\hat{A}$ that breaks $\hat{\mathscr{S}}$ with

probability $\varepsilon(n)$, we can break $\mathscr{S}$ with probability $\varepsilon(n) - \mathrm{neg}(n)$ as follows: Invoke $\hat{A}$ and emulate the oracle $\hat{S}$ using the simulator of the encoding and the signature oracle S. Given the output $(\alpha, \hat{\beta})$ of $\hat{A}$ (supposedly a new valid signature under $\hat{\mathscr{S}}$), use the decoder to translate the signature $\hat{\beta}$ into a signature β under $\mathscr{S}$ and output the pair (α, β).

To analyze the success probability of the new attack note that: (1) By Fact 2.9, the output of $\hat{A}$ when interacting with the emulated oracle is computationally indistinguishable from its output when interacting with the actual signing oracle $\hat{S}(s, \cdot)$. (2) The event that the pair $(\alpha, \hat{\beta})$ is a valid "forgery" under $\hat{\mathscr{S}}$ is efficiently detectable, and so, by (1), it happens in our emulation with probability $\varepsilon(n) - \mathrm{neg}(n)$. (3) By definition, the decoder always translates a valid forgery $(\alpha, \hat{\beta})$ under $\hat{\mathscr{S}}$ into a valid forgery (α, β) under $\mathscr{S}$. Hence, if the scheme $\hat{\mathscr{S}}$ can be broken with non-negligible probability, then so can the scheme $\mathscr{S}$.

This argument can be extended to the private-key setting (i.e., in the case of MACs) as follows. First, observe that the indistinguishability in item (1) holds even for a distinguisher who knows the private-key. (As we encoded the function $S(\cdot, \cdot)$ who treats the key s as an additional argument.) In this case, item (2) holds as well, since forgery under $\hat{\mathscr{S}}$ is efficiently detectable given the private-key. Finally, the last item remains as is. We will later (Remark 5.5) show that signatures and MACs whose signing algorithm is in $\mathbf{NC}^0_4$ can be based on the existence of any one-way function in **CREN**, or, more concretely, on the intractability of factoring, the discrete logarithm problem, and lattice problems.[17]

Impossibility of $\mathbf{NC}^0$ Verification It is not hard to see that the verification algorithm V cannot be realized in $\mathbf{NC}^0$. Indeed, if $V \in \mathbf{NC}^0_k$ one can forge a signature on any document α by simply choosing a random string β' of an appropriate length. This attack succeeds with probability 2^{-k} since: (1) the verification algorithm checks the validity of the signature by reading at most k bits of the signature; and (2) the probability that β' agrees with the correct signature β on the bits which are read by V is at least 2^{-k}.

4.7.2 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.

[17] Our $\mathbf{NC}^0$ signing algorithm is probabilistic but this is unavoidable. Indeed, while a signing algorithm may generally be deterministic (see [72, p. 506]), an $\mathbf{NC}^0$ signing algorithm cannot be deterministic as in this case an adversary can efficiently learn it and use it to forge messages.

4.7.2.1 Non-interactive Commitments

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 $\text{SEND}(b, r)$ 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 $\text{SEND}(b, r)$ is equal to the commitment c. The scheme should be both (computationally) hiding and (perfectly) binding. Hiding requires that $c = \text{SEND}(b, r)$ keeps b computationally secret, that is $\text{SEND}(0, U_n) \stackrel{c}{\equiv} \text{SEND}(1, U_n)$. 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 $\text{SEND}(0, r_0) = \text{SEND}(1, r_1)$.

Let $\hat{\text{SEND}}(b, r, s)$ be a perfectly-correct computationally-private encoding of $\text{SEND}(b, r)$. Then $\hat{\text{SEND}}$ 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 Lemma 4.10. Namely, it holds that

$$\hat{\text{SEND}}(0, r, s) \stackrel{c}{\equiv} S\big(\text{SEND}(0, r, s)\big) \stackrel{c}{\equiv} S\big(\text{SEND}(1, r, s)\big) \stackrel{c}{\equiv} \hat{\text{SEND}}(1, r, s)$$

where r and s are uniformly chosen strings of an appropriate length (the first and third transitions follow from the privacy of $\hat{\text{SEND}}$ and Fact 2.10, while the second transition follows from the hiding of SEND and Fact 2.8). The binding property of $\hat{\text{SEND}}$ follows from the perfect correctness; namely, if there exists an ambiguous pair $(r_0, s_0), (r_1, s_1)$ such that $\hat{\text{SEND}}(0, r_0, s_0) = \hat{\text{SEND}}(1, r_1, s_1)$, then by perfect correctness it holds that $\text{SEND}(0, r_0) = \text{SEND}(1, r_1)$ which contradicts the binding of the original scheme.[18] So when the encoding is in $\mathbf{NC}^0$ we get a commitment scheme whose sender is in $\mathbf{NC}^0$.

In fact, in contrast to the 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 $\hat{\text{SEND}}(b, r, s)$ and comparing the result to $\hat{c}$. Assuming $\hat{\text{SEND}}$ is in $\mathbf{NC}^0$, the receiver can be implemented by an $\mathbf{NC}^0$ circuit augmented with a single (unbounded fan-in) AND gate. We refer to this special type of $\mathbf{AC}^0$ circuit as an $\text{AND}_n \circ \mathbf{NC}^0$ circuit. This extension of $\mathbf{NC}^0$ is necessary as the locality of the function f that the receiver computes cannot be constant. (See the end of this subsection.)

Remark 4.6 (**Unconditional $\mathbf{NC}^0$ construction of non-interactive commitment from 1–1 OWF**) We can use our machinery to obtain an unconditional $\mathbf{NC}^0$ re-

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

duction from a non-interactive commitment scheme to any one-to-one OWF. Moreover, this reduction only makes a *black-box* use of the underlying OWF f. As in the case of the $\mathbf{NC}^0$ reduction from PRGs to OWFs (Remark 4.5), the idea is to encode a non-adaptive black-box $\mathbf{NC}^1$ reduction into a corresponding $\mathbf{NC}^0$ construction. Specifically, the reduction of Blum [34] (instantiated with the Goldreich-Levin hard-core predicate [77]) has the following form: $\text{SEND}(b,(x,r)) = (f(x), r, \langle x, r\rangle \oplus b)$ where x, r are two random strings of length n. Whenever f is one-to-one OWF the resulting function SEND is a perfectly binding, non-interactive commitment (see [70, Construction 4.4.2]). Then, letting $\hat{g}((x,r,b),s)$ be a perfect $\mathbf{NC}^0$ encoding of $\langle x, r\rangle \oplus b$, the function $\hat{\text{SEND}}(b,(x,r,s)) = (f(x), r, \hat{g}(x,r,b,s))$ perfectly encodes SEND, and hence defines a black-box $\mathbf{NC}^0$ reduction from a non-interactive commitment scheme to a one-to-one OWF.

It follows that *non-interactive* commitments in $\mathbf{NC}^0$ are implied by the existence of a 1–1 OWF in **PREN** or by the existence of a non-interactive commitment in **PREN** (actually, perfect correctness and computational privacy suffice).(We will later show that such a scheme can be based on the intractability of factoring or discrete logarithm. See Remark 5.5.)

4.7.2.2 Interactive Commitments

While the existence of an arbitrary OWF is not known to imply non-interactive commitment scheme, it is possible to use OWFs to construct an *interactive* commitment scheme [114]. In particular, the PRG-based commitment scheme of [114] has the following simple form: First the receiver chooses a random string $k \in \{0,1\}^{3n}$ and sends it to the sender, then the sender that wishes to commit to the bit b chooses a random string $r \in \{0,1\}^n$ and sends the value of the function $\text{SEND}(b,k,r)$ to the receiver. (The exact definition of the function $\text{SEND}(b,k,r)$ is not important in our context.) To decommit the sender sends the randomness r and the bit b and the receiver accepts if $\text{SEND}(b,k,r)$ equals the message he had received in the commit phase. Computational hiding requires that for any string family $\{k_n\}$ where $k_n \in \{0,1\}^{3n}$, choice of k it holds that $(k_n, \text{SEND}(0,k_n,U_n)) \stackrel{c}{\equiv} (k_n, \text{SEND}(1,k_n,U_n))$. Perfect binding requires that, except with negligible probability (over the randomness of the receiver k), there are no r_0 and r_1 such that $\text{SEND}(0,k,r_0) = \text{SEND}(1,k,r_1)$.

Again, if we replace SEND by a computationally private perfectly correct encoding $\hat{\text{SEND}}$, we get a (two-round) interactive commitment scheme (this follows by combining the previous arguments with Fact 2.10). Moreover, as in the non-interactive case, when the encoding is in $\mathbf{NC}^0$ the receiver's computation in the decommit phase is in $\text{AND}_n \circ \mathbf{NC}^0$. Since the receiver's computation in the commit phase is also in $\mathbf{NC}^0$, we get an $\text{AND}_n \circ \mathbf{NC}^0$ receiver. (In Remark 5.5 we show that such a scheme can be based on the intractability of factoring, discrete logarithm or lattices problems.) As an immediate application, we obtain a constant-round protocol for coin flipping over the phone [34] between an $\mathbf{NC}^0$ circuit and an $\text{AND}_n \circ \mathbf{NC}^0$ circuit.

4.7.2.3 Statistically Hiding Commitments

One can apply a similar transformation to other variants of commitment schemes, such as unconditionally hiding (and computationally binding) interactive commitments. (Of course, to preserve the security of such schemes the privacy of the encoding will have to be *statistical* rather than computational.) Unconditionally hiding commitments 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 $\mathbf{NC}^0$ sender and an $\mathrm{AND}_n \circ \mathbf{NC}^0$ 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 **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. Statistical hiding requires that for any string family $\{k_n\}$ where $k_n \in \{0,1\}^n$, choice of k it holds that $(k_n, \textsc{Send}(0, k_n, U_{m(n)})) \stackrel{s}{\equiv} (k_n, \textsc{Send}(1, k_n, U_{m(n)}))$, where $m(n)$ is the number of random coins the sender uses. Computational binding requires that, except with negligible probability (over the randomness of the receiver k), an efficient adversary cannot find r_0 and r_1 such that $\textsc{Send}(0, k, r_0) = \textsc{Send}(1, k, r_1)$.

Using the CRHF-based commitment scheme of [53, 84], 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 a statistically hiding commitment scheme whose sender is in $\mathbf{NC}^0$. 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 case of perfectly binding, non-interactive commitment, only this time we use Facts 2.5, 2.3 instead of Facts 2.10, 2.8.) If the random key k cannot be computed in $\mathrm{AND}_n \circ \mathbf{NC}^0$ (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 4.9.)

4.7.2.4 Impossibility of an $\mathbf{NC}^0$ Receiver

We show that, in any of the above settings, the receiver cannot be realized in $\mathbf{NC}^0$. Recall that in the decommit stage the sender opens his commitment to the bit b by sending a single message (b, m) to the receiver, which accepts or rejects it according to b, m and his view v of the commitment stage. (This is the case in all the aforementioned variants.) Suppose that the receiver's computation $f(b, m, v)$ is in $\mathbf{NC}^0_k$. Consider an (honest) execution of the protocol up to the decommit stage in which the sender commits to 1, and the view of the receiver is v. There are two cases: (1) there exists an ambiguous opening m_0, m_1 for which $f(0, m_0) = f(1, m_1) = \mathsf{accept}$; and

(2) there is no ambiguous opening, i.e., for all m we have $f(0, m) = \mathsf{reject}$ and $f(1, m_1) = \mathsf{accept}$ for some m_1. We show that we can either break the binding property (in the first case) or the hiding property (in the second case). Indeed, in the first case the sender can choose two random strings m'_0, m'_1 of an appropriate length. The probability that m'_0 (resp., m'_1) agrees with m_0 (resp., m_1) on the bits which are read by the receiver is at least 2^{-k}. Hence, with probability 2^{-2k}, we have $f(0, m'_0) = f(1, m'_1) = \mathsf{accept}$ and the binding property is violated. Now consider case (2). Let r be the substring of m which is read by the receiver. Since $|r| \leq k$ the receiver can efficiently find b (before the decommit stage) by going over all possible r's and checking whether there exists an r such that $f(b, r, v) = \mathsf{accept}$. We conclude that the locality of the receiver's computation f should be superlogarithmic.

4.7.3 Zero-Knowledge Proofs

We move on to the case of non-interactive zero-knowledge proofs (NIZK). For simplicity, we begin with the simpler case of non-interactive zero knowledge proofs (NIZK). Such proof systems are similar to standard zero-knowledge protocols except that interaction is traded for the use of a public random string σ to which both the prover and the verifier have a read-only access. More formally, a NIZK (with an efficient prover) for an **NP** relation $R(x, w)$ is a pair of probabilistic polynomial-time algorithms (P, V) that satisfies the following properties:

- (Completeness) for every $(x, w) \in R$, it holds that $\Pr[V(x, \sigma, P(x, w, \sigma)) = 1] > 1 - \mathrm{neg}(|x|)$;
- (Soundness) for every $x \notin L_R$ (i.e., x such that $\forall w, (x, w) \notin R$) and every prover algorithm P^* we have that $\Pr[V(x, \sigma, P^*(x, \sigma)) = 1] < \mathrm{neg}(|x|)$;
- (Zero-knowledge) there exists a probabilistic polynomial-time simulator M such that for every string sequence $\{(x_n, w_n)\}$ where $(x_n, w_n) \in LR$ it holds that

$$\left\{\left(x_n, \sigma, P(x_n, w_n, \sigma)\right)\right\} \stackrel{\mathrm{c}}{\equiv} \left\{M(x_n)\right\}$$

(where in all the above σ is uniformly distributed over $\{0, 1\}^{\mathrm{poly}(|x|)}$).

Similarly to the previous cases, we can compile the prover into its computational randomized encoding $\hat{P}$, while the new verifier $\hat{V}$ uses the decoder B to translate the prover's encoded message $\hat{y}$ to the corresponding message of the original prover, and then invokes the original verifier (i.e., $\hat{V} = V(x, \sigma, B(\hat{y}))$). The completeness and soundness of the new protocol follow from the correctness of the encoding. The zero-knowledge property follows from the privacy of the encoding. That is, to simulate the new prover we define a simulator $\hat{M}$ that invokes the simulator M of the original scheme and then applies the simulator S of the encoding to the third entry of M's output. By Fact 2.10 and the privacy of $\hat{P}$ it holds that $(x, \sigma, \hat{P}(x, w, \sigma, r)) \stackrel{\mathrm{c}}{\equiv} (x, \sigma, S(P(x, w, \sigma)))$ (where r is the randomness of the encoding $\hat{P}$) while Fact 2.8 ensures that $(x, \sigma, S(P(x, w, \sigma))) \stackrel{\mathrm{c}}{\equiv} \hat{M}(x)$.

The above construction generalizes to *interactive* ZK-proofs with an efficient prover. In this case, we can encode the prover's computation (viewed as a function of its input, the **NP**-witness he holds, his private randomness and all the messages he has received so far), while the new receiver uses the decoder to translate the messages and then invokes the original protocol. The resulting protocol is still computational ZK proof. (The proof is similar to the case of NIZK above, but relies on Fact 2.9 instead of Fact 2.8.) The same construction works for ZK arguments (in which the soundness holds only against a computationally bounded cheating prover). When the encoding is *statistically*-private the above transformation also preserves the statistical ZK property. That is, if the original protocol provides a statistically-close simulation then so does the new protocol.

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 Graph 3-Colorability of [75], 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 $\mathbf{AC}^0$ computations. Hence, we can use the parallel commitment schemes described before to construct a constant-round protocol for 3-Colorability between an $\mathbf{AC}^0$ prover and an $\mathbf{AC}^0$ verifier. Since 3-Colorability is **NP** complete under $\mathbf{AC}^0$-reductions, we get constant-round zero-knowledge proofs in $\mathbf{AC}^0$ for every language in **NP**. (We will later show that the existence of such a protocol is implied by the intractability of factoring, the discrete logarithm problem, and lattice problems. See Remark 5.5.)

Impossibility of an $\mathbf{NC}^0$ Verifier Again, it is impossible to obtain an $\mathbf{NC}^0$ verifier (for non-trivial languages). Suppose that we have a ZK-proof for a language L whose verifier V is in $\mathbf{NC}^0_k$. First, observe that in such a proof system the soundness error is either 0, or at least $2^{-k} = \Omega(1)$. Hence, since we require a negligible error probability, the soundness must be perfect. Thus, we can efficiently decide whether a string x is in L by letting the honest verifier interact with the simulator. More precisely, given x we use the simulator to sample a transcript of the protocol (with respect to an honest verifier), and accept x if the verifier accepts the transcript. If $x \in L$ then, except with negligible probability, the verifier should accept (as otherwise the interaction with the simulator is distinguishable from the interaction with the real prover). On the other hand, if $x \notin L$ then, due to the perfect soundness, the verifier always rejects. Therefore, $L \in \mathbf{BPP}$.

In fact, if the round complexity of the protocol is $t = O(1)$ as in the aforementioned constructions (and the verifier is in $\mathbf{NC}^0_k$), then L must be in $\mathbf{NC}^0_{tk}$. Furthermore, this is true for any interactive proof protocol, not necessarily ZK. To see this, note that the verifier computes its output based on at most tk bits of x. If $L \notin \mathbf{NC}^0_{tk}$ then there exist an input $x \in L$ and an input $y \notin L$ which agree on the tk bits read by V. Let v be a view of V which makes him accept x. Then, the same view makes V accept y, in contradiction to the perfect soundness.

4.7.4 Instance Hiding Schemes

An instance hiding scheme (IHS) allows a powerful machine (an oracle) to help a more limited user compute some function f on the user's input x; the user wishes to keep his input private and so he cannot just send it to the machine. We assume that the user is an algorithm from a low complexity class WEAK whereas the oracle is from a higher complexity class STRONG. In a (non-adaptive, single-oracle) IHS the user first transforms his input x into a (randomized) encrypted instance $y = E(x, r)$ and then asks the oracle to compute $z = g(y)$. The user should be able to recover the value of $f(x)$ from z by applying a decryption algorithm $D(x, r, z)$ (where $D \in \mathsf{WEAK}$) such that $D(x, r, g(E(x, r))) = f(x)$. The hiding of the scheme requires that $E(x, r)$ keeps x secret, i.e., for every string families $\{x_n\}$ and $\{x'_n\}$ (where $|x_n| = |x'_n|$), the ensembles $E(x_n, r)$ and $E(x'_n, r)$ are indistinguishable with respect to functions in STRONG. The default setting of instance hiding considered in the literature refers to a probabilistic polynomial-time user and a computationally unbounded machine. (See [60] for a survey on IHS schemes.) We will scale this down and let the user be an $\mathbf{NC}^0$ function and the oracle be a probabilistic polynomial-time machine.

The notion of randomized encoding naturally gives rise to IHS in the following way: Given f we define a related function $h(x, r) = f(x) \oplus r$ (where $|r| = |f(x)|$). Let $\hat{h}((x, r), s)$ be a computational randomized encoding of h whose decoder is B. Then, we define $E(x, (r, s)) = \hat{h}((x, r), s)$, $g(y) = B(y)$ and $D(x, r, z) = r \oplus z$. The correctness of the scheme follows from the correctness of the encoding. To prove the privacy note that, by Fact 2.10, it holds that

$$\hat{h}(x_n, r, s) \stackrel{c}{\equiv} S\big(f(x_n) \oplus r\big) \equiv S\big(f(y_n) \oplus r\big) \stackrel{c}{\equiv} \hat{h}(y_n, r, s).$$

Hence, we can construct such a scheme where $\mathsf{WEAK} = \mathbf{NC}^0$ and $\mathsf{STRONG} = \mathbf{CREN}$. (Recall that $\mathbf{CREN} \subseteq \mathbf{BPP}$ and thus computational privacy indeed fools a **CREN** oracle.)

4.8 Summary and Discussion

Table 4.2 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.) As mentioned before, in some cases it suffices to use a *computationally-private* randomized encoding. This relaxation allows us to construct (some) primitives in $\mathbf{NC}^0$ under more general assumptions. (See Theorem 5.3.)

Table 4.2 Sufficient properties for preserving the security of different primitives

Primitive	Encoding	Efficient simulator	Efficient decoder
One-way function	computational	required	–
One-way permutation	perfect	required	–
Trapdoor permutation	perfect	required	required
Pseudorandom generator	perfect	required	–
Collision-resistant hashing	perfect	–	–
Encryption (pub., priv.)	computational	required	required
Signatures, MAC	computational	required	required
Perfectly-binding commitment	perfectly correct comp. private	required	–
Statistically-hiding commitment	perfectly correct stat. private	required	–
Zero-knowledge proof	computational	required	required
Stat. ZK proof/arguments	statistical	required	required
Instance hiding	computational	required	required

4.8.1 The Case of PRFs

A PRF family $f_k(x)$ is a collection of efficiently-computable functions keyed by a secret key k, such that it is infeasible to distinguish between a function $f_k(\cdot)$ chosen uniformly at random from the PRF collection to a truly random function by an adversary that has an oracle access to the function. (See [70, Sect. 3.6] for a formal definition.) It is natural to ask why our machinery cannot be applied to pseudorandom functions (PRFs) (assuming there exists a PRF in **PREN**), as is implied from the impossibility results of Linial et al. [108]. 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., $g_{k,r}(x) \stackrel{\text{def}}{=} \hat{f}(k, x, r)$; (2) to append the randomness to the argument of the collection, i.e., $h_k(x, r) \stackrel{\text{def}}{=} \hat{f}(k, x, r)$. 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 $\mathbf{NC}^0$ with the fact that PRFs cannot be implemented in $\mathbf{NC}^0$, or even $\mathbf{AC}^0$ (see footnote 2), one can derive a separation between PRFs and other

primitives such as PRGs. In particular, there is no $\mathbf{AC}^0$ construction of PRFs from PRGs, unless factoring is easy on the average (more generally, unless there is no OWF in **CREN**).

4.8.2 Open Problems

The results described in this chapter provide strong evidence for the possibility of cryptography in $\mathbf{NC}^0$. 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 $\mathbf{NC}^0$? For instance, does the existence of an arbitrary OWF imply the existence of OWF in $\mathbf{NC}^0$? We show that a OWF in $\mathbf{NL}/poly$ implies a OWF in $\mathbf{NC}^0$.
- Can the existence of a OWF (or PRG) in $\mathbf{NC}^0_3$ be based on general assumptions such as the ones that suffice for implementations in $\mathbf{NC}^0_4$? In Chaps. 6 and 8 we construct such a OWF (and even a PRG) under concrete intractability assumptions (e.g., the intractability of decoding a random linear code).
- 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.

4.9 Appendix: 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 [70, Definition 2.4.3].)

Collections of primitives arise naturally in the context of parallel cryptography, as they allow us to shift "non-parallelizable" operations such as prime number selection and modular exponentiations to the key-generation stage (cf. [117]). 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., $\mathbf{NC}^1$, $\mathbf{NC}^0_4$,

or **PREN**, cf. Definition 3.7) 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.[19] 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 $\mathbf{NC}^1$ 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 $\mathbf{NC}^1$ 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 $\mathbf{NC}^1$ 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 [89]; see also the discussion of "enhanced TDP" in [72, Appendix 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.
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

[19] 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.

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 [103, 117]).

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