Computationally Private Randomizing Polynomials and Their Applications

(EXTENDED ABSTRACT)[*]

Benny Applebaum Yuval Ishai Eyal Kushilevitz

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

Abstract

Randomizing polynomials *allow to represent a function $f(x)$ by a low-degree randomized mapping $\hat{f}(x,r)$ whose output distribution on an input x is a* randomized encoding *of $f(x)$. It is known that any function f in $\oplus\mathrm{L}/poly$ (and in particular in NC^1) can be efficiently represented by degree-3 randomizing polynomials. Such a degree-3 representation gives rise to an NC^0_4 representation, in which every bit of the output depends on only 4 bits of the input.*

In this paper, we study the relaxed notion of computationally private *randomizing polynomials, where the output distribution of $\hat{f}(x,r)$ should only be* computationally indistinguishable *from a randomized encoding of $f(x)$. We construct degree-3 randomizing polynomials of this type for every* polynomial-time *computable function, assuming the existence of a cryptographic pseudorandom generator (PRG) in $\oplus\mathrm{L}/poly$. (The latter assumption is implied by most standard intractability assumptions used in cryptography.) This result is obtained by combining a variant of Yao's* garbled circuit *technique with previous "information-theoretic" constructions of randomizing polynomials.*

We then present the following applications:

- **Relaxed assumptions for cryptography in NC^0.** *Assuming a PRG in $\oplus\mathrm{L}/poly$, the existence of an* arbitrary *public-key encryption, commitment, or signature scheme implies the existence of such a scheme in NC^0_4. Previously, one needed to assume the existence of such schemes in $\oplus\mathrm{L}/poly$ or similar classes.*
- **New parallel reductions between cryptographic primitives.** *We show that even some relatively complex cryptographic primitives, including (stateless) symmetric encryption and digital signatures, are NC^0-reducible to a PRG. No parallel reductions of this type were previously known, even in NC. Our reductions make a non-black-box use of the underlying PRG.*
- **Application to secure multi-party computation.** *Assuming a PRG in $\oplus\mathrm{L}/poly$, the task of computing an* arbitrary *(polynomial-time computable) function with computational security efficiently reduces to that of securely computing degree-3 polynomials. This gives rise to new, conceptually simpler, constant-round protocols for general functions.*

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

1. Introduction

To what extent can one simplify the task of computing a function f by settling for computing some (possibly randomized) *encoding* of its output? The study of this question was initiated in the context of secure multi-party computation [17, 18], and has recently found applications to parallel constructions of cryptographic primitives [1]. In this paper we consider a relaxed variant of this question and present some new constructions and cryptographic applications.

The above question can be formally captured by the following notion. We say that a function $\hat{f}(x,r)$ is a *randomized encoding* of a function $f(x)$, if its output distribution depends only on the output of f. More precisely, we require that: (1) given $\hat{f}(x,r)$ one can efficiently recover $f(x)$, and (2) given $f(x)$ one can efficiently sample from the distribution of $\hat{f}(x,r)$ induced by a uniform choice of r.

This notion of randomized encoding defines a nontrivial relaxation of the usual notion of computing, and thus gives rise to the following question: Can we encode "complex" functions f by "simple" functions $\hat{f}$? This question is motivated by the fact that in many cryptographic applications, $\hat{f}$ can be securely used as a substitute for f [17, 1]. For instance, if f is a one-way function then so is $\hat{f}$. It should be noted that different applications motivate different interpretations of the term "simple" above. In the context of multi-party computation, one is typically interested in minimizing the algebraic *degree* of $\hat{f}$, viewing it as a vector of multivariate polynomials over a finite field. In this context, $\hat{f}$ was referred to as a representation of f by *randomizing polynomials* [17]. In other contexts it is natural to view $\hat{f}$ as a function over binary strings and attempt to minimize its parallel time complexity [1]. From here on, we will refer to $\hat{f}$ as a "randomized encoding" of f (or simply "encoding" for short) except when we wish to stress that we are interested in minimizing the degree.

It was shown in [17, 18] that every function f in $\oplus\mathrm{L}/poly$ can be efficiently represented by degree-3 randomizing polynomials over $\mathrm{GF}(2)$.[1] (The class $\oplus\mathrm{L}/poly$ contains $\mathrm{L}/poly$ and NC^1 and is contained in NC^2. In a non-uniform setting it also contains $\mathrm{NL}/poly$ [30].) Moreover, every degree-3 encoding can in turn be converted into an NC^0 encoding with locality 4, namely one in which every bit of the output depends on only 4 bits of the input [1]. A major question left open by the above results is whether every *polynomial-time* computable function admits an encoding in NC^0.

In this work we consider a relaxed notion of *computationally private* randomized encodings, where requirement (2) above is relaxed to allow sampling from a distribution which is *computationally indistinguishable* from $\hat{f}(x, r)$. As it turns out, computationally private encodings are sufficient for most applications. Thus, settling the latter question for the relaxed notion may be viewed as a second-best alternative.

1.1. Overview of Results and Techniques

We construct a computationally private encoding in NC^0 for every *polynomial-time* computable function, assuming the existence of a "minimal" cryptographic pseudorandom generator (PRG) [5, 31], namely one that stretches its seed by just one bit, in $\oplus\mathrm{L}/poly$.[2] We refer to the latter assumption as the "Easy PRG" (EPRG) assumption. (This assumption can be slightly relaxed, e.g., to also be implied by the existence of a PRG in $\mathrm{NL}/poly$; see Remark 4.13.) We note that EPRG is a very mild assumption. In particular, it is implied by most concrete intractability assumptions commonly used in cryptography, such as ones related to factoring, discrete logarithm, or lattice problems. It is also implied by the existence in $\oplus\mathrm{L}/poly$ of a one-way permutation or, using [16], of any one-way function whose "entropy" can be efficiently computed. The NC^0 encoding we obtain under the EPRG assumption has degree 3 and locality 4. Its size is nearly linear in the circuit size of the encoded function.

The construction consists of three steps. The first step is an NC^0 implementation of *one-time symmetric encryption* using a minimal PRG as an oracle. (Such an encryption allows to encrypt a single message whose length may be polynomially larger than the key. Since our PRG extends its seed by just one bit, it cannot be directly used to encrypt long messages.) The second and main step of the construction relies on a variant of Yao's *garbled circuit* technique [32] to obtain an encoding in NC^0 which uses one-time symmetric encryption as an oracle. Finally, using the EPRG assumption and [1], we apply a final step of "information-theoretic" encoding to obtain an encoding in NC^0 with degree 3 and locality 4.

The above result gives rise to several types of cryptographic applications, discussed below.

1.1.1. Relaxed assumptions for cryptography in NC^0.

The question of minimizing the parallel time complexity of cryptographic primitives has been the subject of an extensive body of research (see [23, 1] and references therein). Pushing parallelism to the extreme, it is natural to ask whether one can implement cryptographic primitives in NC^0. While it was known that few primitives, including pseudorandom *functions* [12], cannot even be implemented in AC^0 [20], no similar negative results were known for other primitives.

Very recently, it was shown in [1] that the existence of most cryptographic primitives in NC^0 follows from their existence in higher complexity classes such as $\oplus\mathrm{L}/poly$, which is typically a very mild assumption. This result was obtained by combining the results on (information-theoretic) randomized encodings mentioned above with the fact that the security of most cryptographic primitives is inherited by their randomized encoding.

Using our construction of computationally private encodings, we can further relax the sufficient assumptions for cryptographic primitives in NC^0. The main observation is that the security of most primitives is also inherited by their computationally private encoding. This is the case even for relatively "sophisticated" primitives such as public-key encryption, digital signatures, commitments, and non-interactive zero-knowledge proofs. Thus, given that these primitives at all exist,[3] their existence in NC^0 follows from the EPRG assumption, namely from the existence of a PRG in complexity classes such as $\oplus\mathrm{L}/poly$ or $\mathrm{NL}/poly$. Previously (using [1]), the existence of each of these primitives in NC^0 would only follow from the assumption that this par-

1 This result generalizes to arbitrary finite fields [18] or even rings [9], allowing efficient degree-3 representations of various counting logspace classes.

2 It is not known whether such a minimal PRG implies a PRG in the same class that stretches its seed by a linear or superlinear amount.

3 This condition is redundant in the case of signatures and commitments, whose existence follows from the existence of a PRG. In Section 1.1.2 we will describe a stronger result for such primitives.

ticular primitive can be implemented in the above classes, a seemingly stronger assumption than EPRG.

It should be noted that we cannot obtain a similar result for some other primitives, such as one-way permutations and collision-resistant hash functions. The results for these primitives obtained in [1] rely on certain regularity properties of the encoding that are lost in the transition to computational privacy.

1.1.2. Parallel reductions between cryptographic primitives. The results of [1] also give rise to new NC^0 *reductions* between cryptographic primitives. (Unlike the results discussed in Section 1.1.1 above, here we consider *unconditional* reductions that do not rely on unproven assumptions.) In particular, known NC^1-reductions from PRG to one-way permutations [13] or even to more general types of one-way functions [16, 29] can be encoded into NC^0-reductions. However, these NC^0-reductions crucially rely on the very simple structure of the NC^1-reductions from which they are derived. In particular, it is not possible to use the results of [1] for encoding general NC^1-reductions (let alone polynomial-time reductions) into NC^0-reductions.

As a surprising application of our technique, we get a general "compiler" that converts an arbitrary (polynomial-time) reduction from a primitive $\mathcal{P}$ to a PRG into an NC^0-reduction from $\mathcal{P}$ to a PRG. This applies to all primitives $\mathcal{P}$ that are known to be equivalent to a one-way function, and whose security is inherited by their computationally-private encoding. In particular, we conclude that symmetric encryption,[4] commitment, and digital signatures are all NC^0-reducible to a *minimal* PRG (hence also to a one-way permutation or more general types of one-way functions).

No parallel reductions of this type were previously known, even in NC. The known construction of commitment from a PRG [21] requires a linear-stretch PRG (expanding an n bits $n + \Omega(n)$ bits), which is not known to be reducible *in parallel* to a minimal PRG. Other primitives, such as symmetric encryption and signatures, were not even known to be reducible in parallel to a polynomial-stretch PRG. For instance, the only previous parallel construction of symmetric encryption from a "low-level" primitive is based on the parallel PRF construction of [23]. This yields an NC^1-reduction from symmetric encryption to *synthesizers*, a stronger primitive than a PRG. Thus, we obtain better parallelism and at the same time rely on a weaker primitive. The price we pay is that we cannot generally guarantee parallel *decryption*. (See Section 5.2 for further discussion.)

4 By symmetric encryption we refer to (probabilistic) *stateless* encryption for multiple messages, where the parties do not maintain any state information other than the key. If parties are allowed to maintain synchronized states, symmetric encryption can be easily reduced in NC^0 to a PRG.

An interesting feature of the new reductions is their *non-black-box* use of the underlying PRG. That is, the "code" of the NC^0-reduction we get (implementing $\mathcal{P}$ using an oracle to a PRG) depends on the code of the PRG. This should be contrasted with most known reductions in cryptography, which make a black-box use of the underlying primitive. In particular, this is the case for the abovementioned NC^0-reductions based on [1]. (See [25] for a thorough taxonomy of reductions in cryptography.)

1.1.3. Application to secure computation. The notion of randomizing polynomials was originally motivated by the goal of minimizing the round complexity of secure multiparty computation [32, 14, 3, 8]. The main relevant observations made in [17] were that: (1) the round complexity of most general protocols from the literature is related to the *degree* of the function being computed; and (2) if f is represented by a vector $\hat{f}$ of degree-d randomizing polynomials, then the secure computation of f can be reduced to that of securely computing some *deterministic* degree-d function $\hat{f}'$ which is closely related to $\hat{f}$. This reduction from f to $\hat{f}'$ is fully *non-interactive*, in the sense that it involves only local computation on the outputs received from $\hat{f}'$ and does not require additional rounds of interaction.

A useful corollary of our results is that under the EPRG assumption, the task of securely computing an *arbitrary* polynomial-time computable function f reduces (non-interactively) to that of securely computing a related degree-3 function $\hat{f}'$. This reduction is only *computationally* secure. Thus, even if the underlying protocol for $\hat{f}'$ is secure in an information-theoretic sense, the resulting protocol for f will only be computationally secure. (In contrast, previous constructions of randomizing polynomials maintained *information-theoretic* security, but only efficiently applied to restricted function classes such as $\oplus\mathrm{L}/poly$.) This reduction gives rise to new, conceptually simpler, constant-round protocols for general functions. For instance, a combination of our result with the classical "BGW protocol" [3] gives a simpler, and in some cases more efficient, alternative to the constant-round protocol of [2] (though relies on a stronger assumption).

Organization. Following some preliminaries (Section 2), in Section 3 we review previous notions of randomized encoding and define our new notion of computationally private encoding. In Section 4 we construct a computationally private encoding in NC^0 for every polynomial-time computable function. Finally, applications of this construction are discussed in Section 5.

2. Preliminaries

Probability notation. We let U_n denote a random variable uniformly distributed over $\{0,1\}^n$. If X is a probability distribution, or a random variable, we write $x \leftarrow X$ to indicate that x is a sample taken from X. The *statistical distance* between discrete probability distributions Y and Y', denoted $\mathrm{SD}(Y, Y')$, is defined as the maximum, over all functions A, of the *distinguishing advantage* $|\Pr[A(Y) = 1] - \Pr[A(Y') = 1]|$. A function $\varepsilon(\cdot)$ is said to be *negligible* if $\varepsilon(n) < n^{-c}$ for any $c > 0$ and sufficiently large n. For two distribution ensembles $Y = \{Y_n\}$ and $Y' = \{Y'_n\}$, we write $Y \equiv Y'$ if Y_n and Y'_n are identically distributed, and say that the two ensembles are statistically indistinguishable if $\mathrm{SD}(Y_n, Y'_n)$ is negligible in n. A weaker notion of closeness between distributions is that of *computational* indistinguishability: We write $Y \overset{c}{\equiv} Y'$ if for every polynomial-size circuit family $\{A_n\}$, the distinguishing advantage $|\Pr[A_n(Y_n) = 1] - \Pr[A_n(Y'_n) = 1]|$ is negligible.

Circuits. We define a boolean circuit C as a directed acyclic graph with labeled, ordered vertices of the following types: (1) *input* vertices, each labeled with a literal x_i or $\bar{x}_i$ and having fan-in 0; (2) *gate* vertices, labeled with one of the boolean functions AND,OR and having fan-in 2; (3) *output* vertices, labeled "output" and having fan-in 1 and fan-out 0. The edges of the circuit are referred to as *wires*. A wire that outgoes from an input vertex is called an *input wire*, and a wire that enters an output vertex is called an *output wire*. Any input $x \in \{0,1\}^n$ assigns a unique *value* to each wire in the natural way. The output value of C, denoted $C(x)$, contains the values of the output wires according to the given predefined order. The *size* of a circuit, denoted $|C|$, is the number of wires in C, and its *depth* is the maximum distance from an input to an output (i.e. the length of the longest directed path in the graph).

NC^i*-reductions.* A circuit with an *oracle* access to a function $g : \{0,1\}^* \to \{0,1\}^*$ is a circuit that contains, in addition to the bounded fan-in OR, AND gates, special *oracle gates* with unbounded fan-in that compute the function g. We say that $f : \{0,1\}^* \to \{0,1\}^*$ is NC^i *reducible* to g, and write $f \in \mathrm{NC}^i[g]$, if f can be computed by a uniform family of polynomial size, $O(\log^i n)$ depth circuits with oracle gates to g. (Oracle gates are treated the same as AND/OR gates when defining depth.) Note that if $f \in \mathrm{NC}^i[g]$ and $g \in \mathrm{NC}^j$ then $f \in \mathrm{NC}^{i+j}$.

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

Complexity classes. For brevity, we assume all complexity classes to be polynomial-time uniform by default. For instance, NC^0 refers to the class of functions admitting uniform NC^0 circuits. We let $\mathrm{NL}/poly$ (resp., $\oplus\mathrm{L}/poly$) denote the class of boolean functions computed by NL (resp., $\oplus$L) Turing machines taking a uniform advice. We extend boolean complexity classes, such as $\mathrm{NL}/poly$ and $\oplus\mathrm{L}/poly$, to include non-boolean functions by letting the representation include $l(n)$ log-space Turing machines, one for each output, taking the same uniform advice. Similarly, we denote by P the class of *functions* that can be computed in polynomial time.

3. Randomized Encodings

We now review the notions of randomized encoding and randomizing polynomials from [17, 18, 1], and introduce the new computationally private variant discussed in this paper. The following definition is from [1].

Definition 3.1 (Randomized encoding) *Let $f : \{0,1\}^n \to \{0,1\}^l$ be a function. We say that a function $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$ is a δ-correct, ε-private* randomized encoding *of f, if it satisfies the following:*

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

We refer to the second input of $\hat{f}$ as its *random input*, and to m, s as the *randomness complexity* and the *output complexity* of $\hat{f}$ respectively. The *overall complexity* (or complexity) of $\hat{f}$ is defined to be $m + s$.

We say that $\hat{f}$ is a representation (or encoding) of f by degree-d *randomizing polynomials* if each of its output bits can be computed by a multivariate polynomial over $\mathrm{GF}(2)$ of degree at most d in the inputs.

Definition 3.1 naturally extends to infinite functions $f : \{0,1\}^* \to \{0,1\}^*$. In this case, the parameters $l, m, s, \delta, \varepsilon$ are all viewed as functions of the input length n, and the algorithms B, S receive 1^n as an additional input. By default, we require $\hat{f}$ to be computable in $\mathrm{poly}(n)$ time whenever f is. In particular, both $m(n)$ and $s(n)$ are polynomially bounded. We also require both the decoder and the simulator algorithms to be efficient.

Several variants of randomized encodings were considered in [1]. Correctness (resp., privacy) is said to be *perfect* when $\delta = 0$ (resp. $\varepsilon = 0$) or *statistical* when $\delta(n)$ (resp. $\varepsilon(n)$) is negligible. In order to preserve the security of some primitives (such as pseudorandom generators or one-way permutations) even perfect correctness and privacy might not suffice and additional requirements should be introduced. An encoding is said to be *balanced* if it admits a perfectly private simulator S such that $S(U_l) \equiv U_s$.

It is said to be *stretch preserving* if $s = l+m$. We say that $\hat{f}$ is a *statistical* randomized encoding of f if it is both statistically correct and statistically private, and that it is a *perfect* randomized encoding if it is perfectly correct and private, balanced, and stretch preserving. In this work, we abandon the information theoretic setting and relax the privacy requirement to be computational. That is, we require the ensembles $S(1^n, f_n(x))$ and $\hat{f}_n(x, U_{m(n)})$ to be computationally indistinguishable.

Definition 3.2 (Computational randomized encoding) *Let $f = \{f_n : \{0,1\}^n \to \{0,1\}^{l(n)}\}_{n\in\mathbb{N}}$ be a function family. We say that the function family $\hat{f} = \{\hat{f}_n : \{0,1\}^n \times \{0,1\}^{m(n)} \to \{0,1\}^{s(n)}\}_{n\in\mathbb{N}}$ is a* computational randomized encoding *of f (or computational encoding for short), if it satisfies the following requirements:*

- **Statistical correctness.** *There exists a polynomial-time decoder B, such that for any n and any input $x \in \{0,1\}^n$, $\Pr[B(1^n, \hat{f}_n(x, U_{m(n)})) \neq f_n(x)] \leq \delta(n)$, for some negligible function $\delta(n)$.*
- **Computational privacy.** *There exists a probabilistic polynomial-time simulator S, such that for any family of strings $\{x_n\}_{n\in\mathbb{N}}$ where $|x_n| = n$, we have $S(1^n, f_n(x_n)) \overset{c}{\equiv} \hat{f}_n(x_n, U_{m(n)})$.*

We will also refer to *perfectly correct* computational encodings, where the statistical correctness requirement is strengthened to perfect correctness. In fact, our main construction yields a perfectly correct encoding.

Remark 3.3 The above definition uses n both as an input length parameter and as a cryptographic "security parameter" quantifying computational privacy. When describing our construction, it will be convenient to use a separate parameter k for the latter, where computational privacy will be guaranteed as long as $k \geq n^\epsilon$ for some constant $\epsilon > 0$.

The function classes $\mathcal{SREN}$ and $\mathcal{PREN}$ were introduced in [1] to capture the power of statistical and perfect randomized encodings in NC^0. We define a similar class $\mathcal{CREN}$.

Definition 3.4 (The classes CREN, SREN, PREN) *The class $\mathcal{CREN}$ (resp., $\mathcal{SREN}, \mathcal{PREN}$) is the class of functions admitting a computational (resp., statistical, perfect) randomized encoding in* NC^0.

It follows from the definitions that $\mathcal{PREN} \subseteq \mathcal{SREN} \subseteq \mathcal{CREN}$. Moreover, it is known that $\oplus\mathrm{L}/poly \subseteq \mathcal{PREN}$ and $\mathrm{NL}/poly \subseteq \mathcal{SREN}$ [1].

We end this section by considering the following intuitive composition property: Suppose we encode f by g, and then view g as a single-argument function and encode it again. Then, the resulting function (parsed appropriately) is an encoding of f. The following lemma was stated in [1] for the statistical and perfect variants of randomized encodings; we extend it here to the computational variant.

Lemma 3.5 (Composition) *Let $g(x, r)$ be a computational encoding of $f(x)$ and $h((x, r), r')$ a computational encoding of $g((x, r))$, viewing the latter as a single-argument function. Then, the function $h'(x, (r, r')) \overset{def}{=} h((x, r), r')$ is a computational encoding of $f(x)$ whose random inputs are (r, r'). Moreover, if g, h are perfectly correct then so is h'.*

Proof sketch: A decoder for h' is obtained by composing the decoders of h and g. Specifically, given an output $y_{h'}$ of h' (i.e., $y_{h'} = h'(x, (r, r')) = h((x, r), r')$ for some x, r, r'), it first decodes $y_g = g(x, r)$ by invoking the decoder of h on $y_{h'}$, and then decodes $y_f = f(x)$ by invoking the decoder of g on y_g. This decoder is perfectly (resp., statistically) correct if both the decoders of h and g are perfectly (resp., statistically) correct. To prove computational privacy, we again compose the computationally private simulators of g and h, this time in an opposite order. Specifically, on input $y_f = f(x)$, the simulator of h' first invokes the simulator of g on y_f, obtaining a simulated string y_g, and then invokes the simulator of h on y_g. The computational privacy of this simulator follows from that of the simulators of g, h by a standard hybrid argument. ∎

It follows as a special case that the composition of a computational encoding with a perfect or a statistical encoding is a computational encoding.

Remark 3.6 It is known that any $f \in \mathcal{PREN}$ (resp., $f \in \mathcal{SREN}$) admits a perfect (resp., statistical) encoding of degree 3 and locality 4 [1]. The same holds for the class $\mathcal{CREN}$, since we can encode a function $f \in \mathcal{CREN}$ by a computational encoding in NC^0 and then encode the resulting function using a perfect encoding of degree 3 and locality 4 (promised by the fact that $\mathrm{NC}^0 \subseteq \mathcal{PREN}$). By Lemma 3.5, the result is a computational encoding for f of degree 3 and locality 4.

4. Computational Encoding in NC^0 for Every Efficiently Computable Function

In this section we construct a perfectly correct computational encoding of degree 3 and locality 4 for every efficiently computable function. Our construction consists of three steps. In Section 4.1, we describe an NC^0 implementation of one-time symmetric encryption using a *minimal* PRG as an oracle (i.e., a PRG that stretches its seed by just one bit). In Section 4.2 we describe the main step of the construction, in which we encode an arbitrary circuit using an NC^0 circuit which uses one-time symmetric encryption as an oracle. This step is based on a variant of Yao's garbled circuit technique [32]. Combining the first two steps, we get a computational encoding in NC^0 with an oracle to a minimal PRG. Finally, in Section 4.3, we derive the main result by relying on the existence of an "easy PRG".

4.1. From PRG to One-Time Encryption

An important tool in our construction is a one-time symmetric encryption; that is, a (probabilistic) private-key encryption that is semantically secure [15] for encrypting a single message. We describe an NC^0-reduction from such an encryption to a minimal PRG, stretching its seed by a single bit. We start by defining one-time symmetric encryption.

Definition 4.1 (One-time symmetric encryption) *A* one-time symmetric encryption scheme *is a pair (E, D), of probabilistic polynomial-time algorithms satisfying the following conditions:*

- ***Correctness:*** *For every k-bit key e and for every plaintext $m \in \{0,1\}^*$, the algorithms E, D satisfy $D_e(E_e(m)) = m$.*
- ***Security:*** *For every polynomial-size circuit family $\{A_k\}$, every polynomials $p(\cdot)$ and $\ell(\cdot)$, all sufficiently large k's and every plaintexts $x, y \in \{0,1\}^{\ell(k)}$, it holds that*

$$|\Pr[A_k(E_{U_k}(x)) = 1] - \Pr[A_k(E_{U_k}(y)) = 1]| < \frac{1}{p(k)}$$

  where the probabilities are taken over the random choice of the key and the coin tosses of E.

The integer k serves as the security parameter *of the scheme.*

The above definition enables to securely encrypt polynomially long messages under short keys. This is an important feature that will be used in our garbled circuit construction described in Section 4.2. In fact, it would suffice for our purposes to encrypt messages of some fixed polynomial[5] length, say $\ell(k) = k^2$. This could be easily done in NC^0 if we had oracle access to a PRG with a corresponding stretch. Given such a PRG G, the encryption can be defined by $E_e(m) = G(e) \oplus m$ and the decryption by $D_e(c) = G(e) \oplus c$. However, we would like to base our construction on a PRG with a minimal stretch.

From the traditional "sequential" point of view, such a minimal PRG is equivalent to a PRG with an arbitrary polynomial stretch (cf. [10, Thm. 3.3.3]). In contrast, this is not known to be the case with respect to parallel reductions. It is not even known whether a linear-stretch PRG is NC-reducible to a minimal PRG (see [29] for some relevant negative results). Thus, a minimal PRG is a more conservative assumption from the point of view of parallel cryptography.

5 Applying the construction to circuits with a bounded fan-out, even linear length would suffice.

Moreover, unlike a PRG with linear stretch, a minimal PRG is reducible in parallel to one-way permutations and other types of one-way functions [16, 29, 1].

The above discussion motivates a *direct* parallel construction of one-time symmetric encryption using a minimal PRG. We present such an NC^0 construction below.

Construction 4.2 (From PRG to one-time symmetric encryption) *Let G be a minimal PRG that stretches its input by a single bit, let e be a k-bit key, and let m be a $(k + l)$-bit plaintext. Define the probabilistic encryption algorithm $E_e(m, (r_1, \ldots, r_{l-1})) \stackrel{def}{=} (G(e) \oplus r_1, G(r_1) \oplus r_2, \ldots, G(r_{l-2}) \oplus r_{l-1}, G(r_{l-1}) \oplus m)$, where $r_i \leftarrow U_{k+i}$ serve as the coin tosses of E. The decryption algorithm $D_e(c_1, \ldots, c_{l-1})$ sets $r_0 = e$, $r_i = c_i \oplus G(r_{i-1})$ for $i = 1, \ldots, l$, and outputs r_l.*

The security of Construction 4.2 is proved via a standard hybrid argument.

Lemma 4.3 *The scheme (E, D) described in Construction 4.2 is a one-time symmetric encryption scheme.*

Proof: Construction 4.2 can be easily verified to satisfy the correctness requirement. We now sketch the security proof. Assume, towards a contradiction, that Construction 4.2 is not secure. It follows that there is a polynomial $l(\cdot)$ and two families of strings $x = \{x_k\}$ and $y = \{y_k\}$ where $|x_k| = |y_k| = k + l(k)$, such that for infinitely many k's, the distribution ensembles $E_e(x_k)$ and $E_e(y_k)$ where $e \leftarrow U_k$, can be distinguished by a polynomial size circuit family $\{A_k\}$ with non-negligible advantage $\varepsilon(k)$.

We use a hybrid argument to derive a contradiction. Fix some k. For a string m of length $k + l(k)$ we define for $0 \leq i \leq l(k)$ the distributions $H_i(m)$ in the following way. The distribution $H_0(m)$ is defined to be $E_{r_0}(m, (r_1, \ldots, r_{l-1}))$ where $r_i \leftarrow U_{k+i}$. For $1 \leq i \leq l(k)$, the distribution $H_i(m)$ is defined exactly as $H_{i-1}(m)$ only that the string $G(r_{i-1})$ is replaced with a random string w_{i-1}, which is one bit longer than r_{i-1} (that is, $w_{i-1} \leftarrow U_{k+i}$). Observe that for every $m \in \{0,1\}^{k+l(k)}$, all the $l(k)$ strings of the hybrid $H_{l(k)}(m)$ are distributed uniformly and independently (each of them is the result of XOR with a fresh random string w_i). Therefore, in particular, $H_{l(k)}(x_k) \equiv H_{l(k)}(y_k)$. Since $H_0(x_k) \equiv E_e(x_k)$ as well as $H_0(y_k) \equiv E_e(y_k)$, it follows that our distinguisher A_k distinguishes, w.l.o.g., between $H_{l(k)}(x_k)$ and $H_0(x_k)$ with at least $\varepsilon(k)/2$ advantage. Then, since there are $l(k)$ hybrids, there must be $1 \leq i \leq l(k)$ such that the neighboring hybrids, $H_{i-1}(x_k), H_i(x_k)$, can be distinguished by A_k with $\frac{\varepsilon(k)}{2l(k)}$ advantage.

We now show how to use A_k to distinguish a randomly chosen string from an output of the pseudorandom generator. Given a string z of length $k + i$ (that is either sampled from $G(U_{k+i-1})$ or from U_{k+i}), we uniformly choose the strings $r_j \in \{0,1\}^{k+j}$ for $j = 1,\ldots,l(k)-1$. We feed A_k with the sample $(r_1,\ldots,r_{i-1}, z \oplus r_i, G(r_i) \oplus r_{i+1},\ldots,G(r_{l(k)-1}) \oplus x_k)$. If z is a uniformly chosen string then the above distribution is equivalent to $H_i(x_k)$. On the other hand, if z is drawn from $G(U_i)$ then the result is distributed exactly as $H_{i-1}(x_k)$, since each of the first $i-1$ entries of $H_{i-1}(x_k)$ is distributed uniformly and independently of the remaining entries (each of these entries was XOR-ed with a fresh and unique random w_j). Hence, we constructed an adversary that breaks the PRG with non-negligible advantage $\frac{\varepsilon(k)}{2l(k)}$, deriving a contradiction. ■

Since the encryption algorithm described in Construction 4.2 is indeed an NC^0 circuit with oracle access to a minimal PRG, we get the following lemma.

Lemma 4.4 *Let G be a PRG. Then, there exists one-time symmetric encryption scheme (E,D) in which the encryption function E is in* $\mathrm{NC}^0[G]$.

Note that the decryption algorithm of the above construction is sequential. This issue will be further discussed later.

4.2. From One-Time Encryption to Computational Encoding

Let $f = \{f_n : \{0,1\}^n \to \{0,1\}^{l(n)}\}_{n\in\mathbb{N}}$ be a polynomial-time computable function, computed by the uniform circuit family $\{C_n\}_{n\in\mathbb{N}}$. We use a one-time symmetric encryption scheme (E,D) as a black box to encode f by a perfectly correct computational encoding $\hat{f} = \{\hat{f}_n\}_{n\in\mathbb{N}}$. Each $\hat{f}_n$ will be an NC^0 circuit with an oracle access to the encryption algorithm E, where the latter is viewed as a function of the key, the message, and its random coin tosses. This construction uses a variant of Yao's garbled circuit technique [32]. Our notation and terminology for this section borrow from previous presentations of Yao's construction in [26, 22, 19].[6]

Notation. Denote by $x = (x_1,\ldots,x_n)$ the input for f_n. Let $k = k(n)$ be a security parameter which may be set to n^ε for an arbitrary positive constant ε (see Remark 3.3). Let $\Gamma(n)$ denote the number of gates in C_n. For every $1 \le i \le |C_n|$, denote by $b_i(x)$ the value of the i-th wire induced by the input x; when x is clear from the context we simply use b_i to denote the wire's value.

We would like to garble the circuit C_n evaluated on an input x such that values obtained on all wires other than output wires are never revealed. In order to do this, our encoding $\hat{f}_n(x,(r,W))$ consists of random inputs of two types: $|C_n|$ bits (referred to as masks) denoted $r_1,\ldots,r_{|C_n|}$ corresponding to the $|C_n|$ wires of C_n, and $|C_n|$ pairs of strings (referred to as signals) $W_i^0, W_i^1 \in \{0,1\}^{2k}$ again in correspondence with the $|C_n|$ wires. We use c_i to denote the value of wire i masked by r_i; namely, $c_i = b_i \oplus r_i$. The encoding $\hat{f}_n(x,(r,W))$ will reveal the masked value, c_i, of every wire but will hide the masks r_i's of all the wires except of the output wires. This way, the real values (b_i's) of non-output wires will remain hidden. The signal W_i^0 represents the value 0 on wire i and the signal W_i^1 represents the value 1 on this wire. We refer to the signal $W_i^{b_i}$, that corresponds to the real value of the wire, as an *on-path signal* and to the other signal $W_i^{1-b_i}$ as an *off-path signal*. We view each string W_i^b as if it is broken into two equal-size parts denoted $W_i^{b,0}, W_i^{b,1}$.

The encoding will enable computation of the on-path signal and masked value of each wire, but will hide the off-path signals. Since the on-path signals and off-path signals are distributed identically, the knowledge of an on-path signal $W_i^{b_i}$ does not reveal the value b_i. To compute the on-path signals and masked bits, we add to our encoding a garbled truth table for every gate. The table maps the on-path labels and masked bits of the input wires of a gate into the on-path labels and masked bits of the output wires of this gate, and thus enables a bottom-to-top computation of these values. Namely, for each of the four possible inputs to a gate $\alpha,\beta \in \{0,1\}$ we encrypt the corresponding signals and masked bits of the gate's output wires, where the keys of the encryption are the corresponding on-path signals of the input wires.

We view the encoding $\hat{f}_n$ as the concatenation of $O(|C_n|)$ functions. In particular, we specify several entries for each gate and for each input and output wire. In what follows $\oplus$ denotes bitwise-xor on strings; when we want to emphasize that the operation is applied to single bits we will usually denote it by either $+$ or $-$. We use $\circ$ to denote concatenation.

Construction 4.5 *Let C_n be a circuit that computes f_n. Then, we define $\hat{f}_n(x,(r,W))$ to be the concatenation of the following functions.*

Input wires: *For an input wire i, labeled by a literal ℓ, which is either some variable x_u or its negation, we append the function $W_i^\ell \circ (\ell + r_i)$.*

Gates: *Let $t \in [\Gamma(n)]$ be a gate that computes the function $g \in \{\mathrm{AND},\mathrm{OR}\}$ with input wires i,j and output wires $y_1,\ldots,y_m$. We associate with this gate 4 functions that are referred to as gate labels. Specifically, for each of the 4 choices of $a_i, a_j \in \{0,1\}$, we define a corresponding function. This function can be thought of as the entry that is indexed by (a_i,a_j) in the garbled truth table of the gate. It is*

6 Security proofs for variants of this construction were given implicitly in [26, 28, 19] in the context of secure computation. However, they cannot be directly used in our context for different reasons. In particular, the analysis of [19] relies on a special form of symmetric encryption and does not achieve perfect correctness, while that of [26, 28] relies on a linear-stretch PRG.

defined as follows:

$$Q_t^{a_i,a_j}(r,W) \overset{def}{=} E_{W_i^{a_i-r_i,a_j} \oplus W_j^{a_j-r_j,a_i}}(W_{y_1}^{g(a_i-r_i,a_j-r_j)} \circ (g(a_i-r_i,a_j-r_j)+r_{y_1}) \circ \ldots \circ W_{y_m}^{g(a_i-r_i,a_j-r_j)} \circ (g(a_i-r_i,a_j-r_j)+r_{y_m})),$$

where E is a one-time symmetric encryption algorithm. That is, the signals and the masked bits of all the output wires of this gate are encrypted under a key that depends on the signals of the input wires of the gate. Note that $Q_t^{a_i,a_j}$ depends only on the random inputs. We refer to the label $Q_t^{c_i,c_j}$ that is indexed by the masked bits of the input wires as an on-path label*, and to the other three labels as the* off-path labels.

Output wires: *For each output wire i of the circuit, we add the mask of this wire r_i.*

It is not hard to verify that $\hat{f}_n$ is in $\mathrm{NC}^0[E]$. In particular, observe that a term of the form W_i^ℓ, where ℓ is a literal, is a 3-local function that each of its bits depends on ℓ and the corresponding bits of W_i^1 and W_i^0. Similarly, the keys that are used in the encryptions are 8-local functions, and the encrypted messages are 6-local functions.

We will now analyze the complexity of $\hat{f}_n$. For each wire the construction uses $O(k)$ random bits, in addition each invocation of the encryption uses $\mathrm{poly}(k)$ random bits. Since there are $O(|C_n|)$ invocations, the randomness complexity is $O(|C_n|)\cdot \mathrm{poly}(k) = O(|C_n|\cdot n^\varepsilon)$ for an arbitrary constant $\varepsilon > 0$. The output complexity is dominated by the gate labels, which are strings of length $\mathrm{poly}(k)$, since there are $O(|C_n|)$ such labels the output complexity is also $O(|C_n|)\cdot \mathrm{poly}(k)$, and thus the overall complexity of $\hat{f}$ is $O(|C_n|\cdot n^\varepsilon)$.

Let $\mu(n), s(n)$ be the randomness complexity and the output complexity of $\hat{f}_n$ respectively. We claim that the function family $\hat{f} = \{\hat{f}_n : \{0,1\}^n \times \{0,1\}^{\mu(n)} \to \{0,1\}^{s(n)}\}_{n\in\mathbb{N}}$ defined above is indeed a computationally randomized encoding of the family f. We start with perfect correctness.

Lemma 4.6 (Perfect correctness) *There exists a polynomial time decoder algorithm B such that for every $n \in \mathbb{N}$ and every $x \in \{0,1\}^n$ and $(r,W) \in \{0,1\}^{\mu(n)}$, it holds that $B(1^n, \hat{f}_n(x,(r,W))) = f_n(x)$.*

Proof: Let $\alpha = \hat{f}_n(x,(r,W))$ for some $x \in \{0,1\}^n$ and $(r,W) \in \{0,1\}^{\mu(n)}$. Given α we show how to efficiently compute the on-path signal $W_i^{b_i}$ and masked value c_i of every wire i in the circuit. In particular, by obtaining c_i for an output wire i, we can retrieve the mask r_i from α and compute the corresponding output bit of $f_n(x)$, i.e., output $b_i = c_i - r_i$. (Recall that the masks of the output wires are given explicitly as part of α.) The on-path signals and the masked values of every wire are computed by scanning the circuit from bottom to top.

For an input wire i the desired value, $W_i^{b_i} \circ c_i$, is given as part of α. Next, consider a wire y that goes out of a gate t, and assume that we have already computed the desired values of the input wires i,j of this gate. We use the masked bits c_i, c_j of the input wires, to select the on-path label $Q_t^{c_i,c_j}$ of the gate t (and ignore the other 3 off-path labels of this gate). Consider this label as in Equation (4.1); recall that this cipher was encrypted under the key $W_i^{c_i-r_i,c_j} \oplus W_j^{c_j-r_j,c_i} = W_i^{b_i,c_j} \oplus W_j^{b_j,c_i}$. Since we have already computed the values $c_i, c_j, W_i^{b_i}$ and $W_j^{b_j}$, we can decrypt the label $Q_t^{c_i,c_j}$, (by applying the decryption algorithm D) and recover the encrypted string, that includes, in particular, the value $W_y^{g(b_i,b_j)} \circ (g(b_i,b_j)+r_y)$, where g is the function that gate t computes. Since by definition $b_y = g(b_i,b_j)$, the decrypted string contains the desired value. ■

To argue computational privacy we need to prove the following lemma, whose proof is deferred to Appendix A.

Lemma 4.7 (Computational privacy) *There exists a probabilistic polynomial-time simulator S, such that for any family of strings $\{x_n\}_{n\in\mathbb{N}}$, $|x_n| = n$, it holds that $S(1^n, f_n(x_n)) \overset{c}{\equiv} \hat{f}_n(x_n, U_{\mu(n)})$.*

Remark 4.8 (Information-theoretic variant) Construction 4.5 can be instantiated with a *perfect* (information-theoretic) encryption scheme, yielding a perfectly private randomized encoding. (The privacy proof given in Appendix A can be easily modified to treat this case.) However, in such an encryption the key must be as long as the encrypted message [27]. It follows that the wires' signal length grows exponentially with their distance from the outputs, rendering the construction efficient only for NC^1 circuits. This information-theoretic variant of the garbled circuit construction was previously suggested in [18]. We will use it in Section 4.3 for obtaining a computational encoding with a parallel decoder.

4.3. Main Results

Combining Lemmas 4.6, 4.7, and 4.4 we get an NC^0 encoding of any efficiently computable function using an oracle to a minimal PRG.

Theorem 4.9 *Suppose f is computed by a uniform family $\{C_n\}$ of polynomial-size circuits. Let G be a (minimal) PRG. Then, f admits a perfectly correct computational encoding $\hat{f}$ in $\mathrm{NC}^0[G]$. The complexity of $\hat{f}$ is $O(|C_n|\cdot n^\varepsilon)$ (for an arbitrary constant $\varepsilon > 0$).*

We turn to the question of eliminating the PRG oracles. We follow the natural approach of replacing each oracle with an NC^0 implementation. (A more general but less direct approach will be described in Remark 4.13.) Using [1, Theorem 6.2], a minimal PRG in NC^0 is implied by a PRG in $\mathcal{PREN}$, and in particular by a PRG in NC^1 or even $\oplus\mathrm{L}/poly$. Thus, we can base our main theorem on the following "easy PRG" assumption.

Assumption 4.10 (Easy PRG (EPRG)) *There exists a PRG in* $\oplus\mathrm{L}/poly$.

As discussed in Section 1.1, EPRG is a very mild assumption. In particular, it is implied by most standard cryptographic intractability assumptions, and is also implied by the existence in $\oplus\mathrm{L}/poly$ of one-way permutations and other types of one-way functions.

Combining Theorem 4.9 with the EPRG assumption, we get a computational encoding in NC^0 for every efficiently computable function. To optimize its parameters we apply a final step of perfect encoding, yielding a computational encoding with degree 3 and locality 4 (see Remark 3.6). Thus, we get the following main theorem.

Theorem 4.11 *Suppose f is computed by a uniform family $\{C_n\}$ of polynomial-size circuits. Then, under the EPRG assumption, f admits a perfectly correct computational encoding $\hat{f}$ of degree* 3*, locality* 4 *and complexity $O(|C_n| \cdot n^{\varepsilon})$ (for an arbitrary constant $\varepsilon > 0$).*

Corollary 4.12 *Under the EPRG assumption,* $\mathcal{CREN} = \mathrm{BPP}$.

Proof: Let $f(x)$ be a function in BPP. It follows that there exists a function $f'(x,z) \in \mathrm{P}$ such that for every $x \in \{0,1\}^n$ it holds that $\Pr_z[f'(x,z) \neq f(x)] \leq 2^{-n}$. Let $\hat{f}'((x,z),r)$ be the NC^0 computational encoding of f' promised by Theorem 4.11. It follows that $\hat{f}(x,(z,r)) \overset{\text{def}}{=} \hat{f}'((x,z),r)$ is a computational encoding of f in NC^0.

Conversely, suppose $f \in \mathcal{CREN}$ and let $\hat{f}$ be an NC^0 computational encoding of f. A BPP algorithm for f can be obtained by first computing $\hat{y} = \hat{f}(x,r)$ on a random r and then invoking the decoder on $\hat{y}$ to obtain the output $y = f(x)$ with high probability. ■

Remark 4.13 (Relaxing the EPRG assumption) The EPRG assumption is equivalent to the existence of a PRG in NC^0 or in $\mathcal{PREN}$. It is possible to base Theorem 4.11 on a seemingly more liberal assumption by taking an alternative approach that does not rely on a *perfect* encoding. The idea is to first replace each PRG oracle with an implementation G from some class $\mathcal{C}$, and only then apply a (perfectly correct) *statistical* encoding to the resulting $\mathrm{NC}^0[G]$ circuit. Thus, we need G to be taken from a class $\mathcal{C}$ such that $\mathrm{NC}^0[\mathcal{C}] \subseteq \mathcal{SREN}$. It follows from [18, 1] that the class $\mathrm{NL}/poly$ satisfies this property (and furthermore, functions in $\mathrm{NL}/poly$ admit a statistical NC^0 encoding with perfect correctness). Thus, we can replace $\oplus\mathrm{L}/poly$ in the EPRG assumption with $\mathrm{NL}/poly$.

On the parallel complexity of the decoder. As we shall see in Section 5, it is sometimes useful to obtain a computational encoding whose decoder is also parallelized. By the description of the decoder of Construction 4.5, it follows that if the circuit computing f is an NC^i circuit then the decoder is in $\mathrm{NC}^i[D]$, where D is the decryption algorithm. Thus, if D is in NC^j we obtain a parallel decoder in NC^{i+j}. Unfortunately, we cannot use the parallel symmetric encryption scheme of Construction 4.2 for this purpose because of its sequential decryption.

We can get around this problem by strengthening the EPRG assumption. Suppose we have a *polynomial-stretch* PRG in NC^1. (This is implied by some standard cryptographic assumptions, see [24].) In such a case, we can obtain a one-time symmetric encryption scheme (E, D) (for messages of a fixed polynomial length) in which both E and D are in NC^1. To encrypt a message we simply apply the generator to the key and mask the result with the plaintext message; decryption is implemented analogously. Our goal is to turn this into a scheme $(\hat{E}, \hat{D})$ in which the encryption $\hat{E}$ is in NC^0 and the decryption is still in NC^1. We achieve this by applying to (E, D) the encoding given by the information-theoretic variant of the garbled circuit construction (see Remark 4.8 or [18]). That is, $\hat{E}$ is a (perfectly correct and private) NC^0 encoding of E, and $\hat{D}$ is obtained by composing D with the decoder of the information-theoretic garbled circuit. (The resulting scheme $(\hat{E}, \hat{D})$ is still a secure encryption scheme, see [1] or Example 5.1.) Since the symmetric encryption employed by the information-theoretic garbled circuit is in NC^0, its decoder can be implemented in $\mathrm{NC}^1[\mathrm{NC}^0] = \mathrm{NC}^1$. Thus, $\hat{D}$ is also in NC^1. Combining this encryption scheme with Construction 4.5, we get a computational encoding of a function $f \in \mathrm{NC}^i$ with encoding in NC^0 and decoding in NC^{i+1}. Summarizing, we have the following:

Claim 4.14 *Suppose there exists a PRG with polynomial stretch in NC^1 (i.e., a PRG that stretches k bits into k^c bits for some $c > 1$). Then, every function $f \in \mathrm{NC}^i$ admits a perfectly-correct computational encoding in NC^0 whose decoder is in NC^{i+1}.*

5. Applications

5.1. Relaxed Assumptions for Cryptography in NC^0

In [1] it was shown that, under relatively mild assumptions, many cryptographic tasks can be implemented in NC^0. This was proved by arguing that: (1) the security of most primitives is inherited by their statistical or perfect randomized encoding; and (2) perfect or statistical encodings can be obtained for functions in relatively high complexity classes such as NC^1, $\oplus\mathrm{L}/poly$ or $\mathrm{NL}/poly$. Thus, if a primitive $\mathcal{P}$ can be computed in these classes, then it can also be computed in NC^0.

In this work, we consider primitives whose security is also inherited by their computational encoding. (In some cases we will need to rely on perfect correctness, which we get "for free" in our main construction.) It follows from Theorem 4.11 that, under the EPRG assumption, any such primitive $\mathcal{P}$ can be computed in NC^0 *if it exists at all* (i.e., can be computed in polynomial time).

Some primitives, such as collision resistant hash functions and one-way permutations, do not respect computational encoding. However, many others do. These include public-key encryption, symmetric encryption,[7] commitments,[8] signatures, message authentication schemes (MACs), and non-interactive zero knowledge proofs (NIZK). (See [10, 11] for detailed definitions of these cryptographic primitives.) 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 computational 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. These reductions are analogous to those given in [1] for the case of statistical encoding. For completeness, we sketch below the construction and security proof for the case of public-key encryption.

Example 5.1 (Public-key encryption) Suppose that $\mathcal{E} = (G, E, D)$ is a (probabilistic) public-key encryption scheme, where G is a key-generation algorithm generating a pair (e, d) of encryption and decryption keys, $E(e, m, r)$ is the encryption function that encrypts the message m using the key e and randomness r, and $D(d, y)$ is the decryption function that decrypts the cipher y using the decryption key d. The encryption function E should be semantically secure as in Definition 4.1, except that here the distinguisher is also given the public key e. Let $\hat{E}$ be a 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 the encoding $\hat{E}$. We argue that the scheme $\hat{\mathcal{E}} \stackrel{\text{def}}{=} (G, \hat{E}, \hat{D})$ is also a public-key encryption scheme. The efficiency and correctness of $\hat{\mathcal{E}}$ are guaranteed by the efficiency of the encoding and its correctness. The security of $\hat{\mathcal{E}}$ reduces to that of $\mathcal{E}$ using the efficient simulator S of $\hat{E}$. This is argued as follows. Given a distinguisher $\hat{A}$ which efficiently distinguishes between encryptions under $\hat{E}$ of messages x, x', we obtain a similar distinguisher A for E by letting $A(e, y) = \hat{A}(e, S(y))$. By computational privacy, if y is a random encryption of m under E with a key e, then $S(y)$ is computationally indistinguishable from a random encryption of m under $\hat{E}$ with the same key e. It follows that if $\hat{A}$ has a non-negligible advantage distinguishing between encryptions of x and x', then so does A.

7 See Footnote 4.

8 In this work we refer to computationally hiding commitments. Computational encoding does not respect the security of statistically hiding commitments.

A transformation as above can be used to construct an NC^0 sender, but does not promise anything concerning the parallel-time complexity of the receiver. In the following, we argue that 1) for some primitives the parallel complexity at the receiver's end can also be improved, and 2) for other primitives, we can implement the sender in NC^0 without substantial increase in the parallel complexity of the receiver.

Consider first the case of commitment. As argued in [1], in this case we can also improve the complexity at the receiver's end to AC^0. Indeed, the sender can decommit by sending its random coins, and the receiver needs only to emulate the computation of the sender and compare it with the message it received in the commit stage. Thus, the receiver can be implemented as an NC^0 circuit with a single unbounded fan-in AND gate. Such a commitment scheme can then be used to implement coin flipping over the phone [4] between an NC^0 circuit and an AC^0 circuit.

We summarize some consequences of the EPRG assumption obtained so far.

Theorem 5.2 *Suppose that the EPRG assumption holds. Then,*

- *If there exists a public-key encryption (resp., NIZK) scheme, then there exists such a scheme in which the encryption (prover) algorithm is in* NC^0_4.
- *There exists a stateless symmetric encryption scheme (resp., digital signature or MAC) in which the encryption (signing) algorithm is in* NC^0_4.
- *There exists a commitment scheme (resp., coin-flipping protocol) in which the sender (first party) is in* NC^0_4 *and the receiver (second party) is in* AC^0.

Note that the existence of all the above primitives, except of public-key encryption and NIZK, does not require any additional assumption other than EPRG. This is a consequence of the fact that they all can be constructed (in polynomial time) from a PRG (see [10, 11]). For these primitives, we obtain more general (unconditional) results in the next subsection.

Theorem 5.2 reveals an interesting phenomenon. It appears that several cryptographic primitives can be implemented in NC^0 despite the fact that their standard constructions rely on pseudorandom functions (PRFs) [12], which cannot be computed even in AC^0 [20]. For such primitives, we actually construct a sequential PRF from the PRG (as in [12]), use it as a building block to obtain a sequential construction of the desired primitive (e.g., symmetric encryption), and finally reduce the parallel-time complexity of the resulting function using our machinery. Of course, the security of the PRF primitive itself is not inherited by its computational (or even perfect) encoding.

Parallelizing the receiver. As mentioned above, the computational encoding promised by Theorem 4.11 does not support parallel decoding. Thus, we get primitives in which the sender is in NC^0 but the receiver is not known to be in NC, even if we started with a primitive that has an NC receiver. The following theorem tries to partially remedy this state of affairs. Assuming the existence of a PRG with a good stretch in NC^1, we can rely on Claim 4.14 to convert sender-receiver schemes in which both the receiver and the sender are in NC to ones in which the sender is in NC^0 and the receiver is still in NC.

Theorem 5.3 *Let $\mathcal{X} = (G, S, R)$ be a sender-receiver cryptographic scheme, where G is a key-generation algorithm, $S \in \mathrm{NC}^s$ is the algorithm of the sender and $R \in \mathrm{NC}^r$ is the algorithm of the receiver. Suppose $\mathcal{X}$ implements a primitive $\mathcal{P}$ whose security is respected by computational encoding. Suppose further that there exists a polynomial-stretch PRG in NC^1. Then there exists a scheme $\hat{\mathcal{X}} = (G, \hat{S}, \hat{R})$ that securely implements the same cryptographic task as $\mathcal{X}$, and in which $\hat{S} \in \mathrm{NC}^0$ and $\hat{R} \in \mathrm{NC}^{\max\{s+1,r\}}$.*

Proof: If there exists a polynomial-stretch PRG in NC^1, then we can use Claim 4.14 and get a computational encoding $\hat{S}$ for S in NC^0 whose decoder B is in NC^{s+1}. As usual, the new receiver $\hat{R}$ uses B to decode the encoding, and then applies the original receiver R to the result. Thus, $\hat{R}$ is in $\mathrm{NC}^{\max\{s+1,r\}}$. ■

Alternatively, it is possible to relax the above assumption to the existence of a *linear-stretch* PRG in NC^1, at the cost of increasing the complexity of the receiver to $\mathrm{NC}^{\max\{s+2,r\}}$.

5.2. Parallel Reductions between Cryptographic Primitives

In the previous section we showed that many cryptographic tasks can be performed in NC^0 if they can be performed at all, relying on the assumption that an easy PRG exists. Although EPRG is a very reasonable assumption, it is natural to ask what types of parallel reductions between primitives can be guaranteed *unconditionally*. In particular, such reductions would have consequences even if there exists a PRG in, say, NC^4.

In this section, we consider the types of unconditional reductions that can be obtained using the machinery of Section 4. We focus on primitives that can be reduced to a PRG (equivalently, using [16], to a one-way function). We argue that for any such primitive $\mathcal{F}$, its polynomial-time reduction to a PRG can be collapsed into an NC^0-reduction to a PRG. More specifically, we present an efficient "compiler" that takes the code of an arbitrary PRG G and outputs a description of an NC^0 circuit C, having oracle access to a function G', such that for any (minimal) PRG G' the circuit $C[G']$ implements $\mathcal{F}$.

A compiler as above proceeds as follows. Given the code of G, it first constructs a code for an efficient implementation f of $\mathcal{F}$. (In case we are given an efficient *black-box* reduction from $\mathcal{F}$ to a PRG, this code is obtained by plugging the code of G into this reduction.) Then, applying a constructive form of Theorem 4.9 to the code of f, the compiler obtains a code $\hat{f}$ of an NC^0 circuit which implements $\mathcal{F}$ by making an oracle access to a PRG. This code of $\hat{f}$ defines the required NC^0 reduction from $\mathcal{F}$ to a PRG, whose specification depends on the code of the given PRG G. Thus, the reduction makes a *non-black-box* use of the PRG primitive, even if the polynomial-time reduction it is based on is fully black-box.

Based on the above we can obtain the following informal "meta-theorem":

Meta-Theorem 5.4 *Let $\mathcal{F}$ be a cryptographic primitive whose security is respected by computational encoding. Suppose that $\mathcal{F}$ is polynomial-time reducible to a PRG. Then, $\mathcal{F}$ is NC^0-reducible to a (minimal) PRG.*

Since a minimal PRG can be reduced in NC^0 to one-way permutations or one-way functions with efficiently computable "entropy" (see [16, 1, 29]), the minimal PRG in the conclusion of the above theorem can be replaced by these primitives.

Instantiating $\mathcal{F}$ by concrete primitives, we get the following corollary:

Corollary 5.5 *Let G be a PRG. Then,*

- *There exists a stateless symmetric encryption scheme (resp., digital signature or MAC) in which the encryption (signing) algorithm is in $\mathrm{NC}^0[G]$.*
- *There exists a commitment scheme (resp., coin-flipping protocol) in which the sender (first party) is in $\mathrm{NC}^0[G]$ and the receiver (second party) is in $\mathrm{AC}^0[G]$.*

Note that the last two items of Theorem 5.2 can be derived from the above corollary, up to the exact locality.

The above results can be used to improve the parallel complexity of some known reductions. For example, Naor [21] shows a commitment scheme in which the sender is in $\mathrm{NC}^0[LG]$, where LG is a *linear-stretch* PRG. By using his construction, we derive a commitment scheme in which the sender (respectively, the receiver) is in $\mathrm{NC}^0[G]$ (respectively, $\mathrm{AC}^0[G]$) where G is a *minimal* PRG. Since it is not known how to NC-reduce a linear-stretch PRG to a minimal PRG, we get a nontrivial parallel reduction.

Other interesting examples arise in the case of primitives that are based on PRFs, such as MACs, symmetric encryption, and identification (see [12, 23, 11] for these and other applications of PRFs). Since the known construction of a PRF from a PRG is sequential [12], it was not known how to reduce these primitives in parallel to (even a polynomial-stretch) PRG. This fact motivated the study of parallel constructions of PRFs in [23, 24]. In particular, Naor and Reingold [23] introduce a new cryptographic primitive called a synthesizer (SYNTH), and show that PRFs can be implemented in $\mathrm{NC}^1[\mathrm{SYNTH}]$. This gives an NC^1-reduction from cryptographic primitives such as symmetric encryption to synthesizers. By Corollary 5.5, we get that these primitives are in fact NC^0-reducible to a PRG. Since (even a polynomial-stretch) PRG can be implemented in $\mathrm{NC}^0[\mathrm{SYNTH}]$ while synthesizers are not known to be even in $\mathrm{NC}[\mathrm{PRG}]$, our results improve both the complexity of the reduction and the underlying assumption. It should be noted, however, that our reduction only improves the parallel-time complexity of the encrypting party, while the constructions of [23] yield NC^1-reductions on both ends. (A partial and conditional workaround is given by Theorem 5.3.)

5.3. Secure Multi-Party Computation

Secure multi-party computation (MPC) allows several parties to evaluate a function of their inputs in a distributed way, so that both the privacy of their inputs and the correctness of the outputs are maintained. These properties should hold, to the extent possible, even in the presence of an adversary who may corrupt at most t parties. This is typically formalized by comparing the adversary's interaction with the *real process*, in which the uncorrupted parties run the specified protocol on their inputs, with an ideal function evaluation process in which a trusted party is employed. The protocol is said to be *secure* if whatever the adversary "achieves" in the real process it could have also achieved by corrupting the ideal process. A bit more precisely, it is required that for every adversary A interacting with the real process there is an adversary A' interacting with the ideal process, such that outputs of these two interactions are *indistinguishable* from the point of view of an external environment. See, e.g., [6, 7, 11], for more detailed and concrete definitions.

There is a variety of different models for secure computation. These models differ in the power of the adversary, the network structure, and the type of "environment" that tries to distinguish between the real process and the ideal process. In the *information-theoretic* setting, both the adversary and the distinguishing environment are computationally unbounded, whereas in the *computational* setting they are both bounded to probabilistic polynomial time.

The notion of randomizing polynomials was originally motivated by the goal of minimizing the round complexity of MPC. The motivating observation of [17] was that the round complexity of most general protocols from the literature (e.g., those of [14, 3, 8]) is related to the *degree* of the function being computed. Thus, by reducing the task of securely computing f to that of securely computing some related low-degree function, one can obtain round-efficient protocols for f.

Randomizing polynomials (or low-degree randomized encodings) provide precisely this type of reduction. More specifically, suppose that the input x to f is distributed between the parties, who wish to all learn the output $f(x)$. If f is represented by a vector $\hat{f}(x, r)$ of degree-d randomizing polynomials, then the secure computation of f can be *non-interactively* reduced to that of $\hat{f}$, where the latter is viewed as a *randomized* function of x. This reduction only requires each party to invoke the decoder of $\hat{f}$ on its local output, obtaining the corresponding output of f. The secure computation of $\hat{f}$, in turn, can be non-interactively reduced to that of a related *deterministic* function $\hat{f}'$ of the same degree d. The idea is to let $\hat{f}'(x, r^1, \ldots, r^{t+1}) \stackrel{\text{def}}{=} p(x, r^1 + \ldots + r^{t+1})$ (where t is a bound on the number of corrupted parties), assign each input vector r^j to a distinct player, and instruct it to pick it at random. (See [17] for more details.) This second reduction step is also non-interactive. Thus, any secure protocol for $\hat{f}'$ or $\hat{f}$ gives rise to a secure protocol for f with the same number of rounds. The non-interactive nature of the reduction makes it insensitive to almost all aspects of the security model.

Previous constructions of (perfect or statistical) randomizing polynomials [17, 18, 9] provided *information-theoretic* reductions of the type discussed above. In particular, if the protocol used for evaluating $\hat{f}'$ is information-theoretically secure, then so is the resulting protocol for f. The main limitation of these previous reductions is that they efficiently apply only to restricted classes of functions, typically related to different log-space classes. This situation is remedied in the current work, where we obtain (under the EPRG assumption) a *general* secure reduction from a function f to a related degree-3 function $\hat{f}'$. The main price we pay is that the security of the reduction is no longer information-theoretic. Thus, even if the underlying protocol for $\hat{f}'$ is secure in the information-theoretic sense, the resulting protocol for f will only be computationally secure.

Formulating the above using the terminology of reductions between secure functionalities (cf. [11]), Theorem 4.11 has the following corollary.

Theorem 5.6 *Suppose the EPRG assumption holds. Let $f(x)$ be an m-party functionality computed by a (uniform) circuit family of size $s(n)$. Then, for any $\epsilon > 0$, there is a non-interactive, computationally $(m-1)$-secure reduction from f to either of the following two efficient functionalities:*

- *A randomized functionality $\hat{f}(x, r)$ of degree 3 (over $\mathrm{GF}(2)$) with a random input and output of length $O(s(n) \cdot n^{\epsilon})$ each;*
- *A deterministic functionality $\hat{f}'(x')$ of degree 3 (over $\mathrm{GF}(2)$) with input length $O(m \cdot s(n) \cdot n^{\epsilon})$ and output length $O(s(n) \cdot n^{\epsilon})$.*

Both reductions are non-interactive in the sense that they involve a single call to $\hat{f}$ or $\hat{f}'$ and no further interaction. They both apply regardless of whether the adversary is passive or active, adaptive or non-adaptive.

A high-level corollary of Theorem 5.6 is that computing arbitrary polynomial-time computable functionalities is as easy as computing degree-3 functionalities. Thus, when designing new MPC protocols, it suffices to consider degree-3 functionalities which are often easier to handle.

More concretely, Theorem 5.6 gives rise to new, conceptually simpler, constant-round protocols for general functionalities. For instance, a combination of this result with the "BGW protocol" [3] gives a simpler alternative to the constant-round protocol of Beaver, Micali, and Rogaway [2]. The resulting protocol will be more round-efficient, and in some cases (depending on the number of parties and the "easiness" of the PRG) even more communication-efficient than the protocol of [2]. On the downside, Theorem 5.6 relies on a stronger assumption than the protocol from [2] (an easy PRG vs. an arbitrary PRG).

An interesting open question, which is motivated mainly from the point of view of the MPC application, is to come up with an "arithmetic" variant of the construction. That is, given an arithmetic circuit C, say with addition and multiplication gates, construct a vector of computationally private randomizing polynomials of size $\mathrm{poly}(|C|)$ which makes a *black-box* use of the underlying field. The latter requirement means that the same polynomials should represent C over any field, ruling out the option of simulating arithmetic field operations by boolean operations. Such a result is known for weaker arithmetic models such as formulas and branching programs (see [9]).

Acknowledgment. We thank Omer Reingold for helpful discussions.

References

[1] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in NC^0. In *Proc. 45st FOCS*, pages 166–175, 2004.

[2] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *Proc. of 22nd STOC*, pages 503–513, 1990.

[3] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. of 20th STOC*, pages 1–10, 1988.

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

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

[6] R. Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[7] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42st FOCS*, pages 136–145, 2001.

[8] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proc. of 20th STOC*, pages 11–19, 1988.

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

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

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

[12] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. of the ACM.*, 33:792–807, 1986.

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

[14] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game (extended abstract). In *Proc. of 19th STOC*, pages 218–229, 1987.

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

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

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

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

[19] Y. Lindell and B. Pinkas. A proof of Yao's protocol for secure two-party computation. *Electronic Colloquium on Computational Complexity*, 11(063), 2004.

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

[21] M. Naor. Bit commitment using pseudorandomness. *J. of Cryptology*, 4:151–158, 1991.

[22] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proc. 1st ACM Conference on Electronic Commerce*, pages 129–139, 1999.

[23] M. Naor and O. Reingold. Synthesizers and their application to the parallel construction of pseudo-random functions. *J. of Computer and Systems Sciences*, 58(2):336–375, 1999. Preliminary version in Proc. 36th FOCS, 1995.

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

[25] O. Reingold, L. Trevisan, and S. Vadhan. Notions of reducibility between cryptographic primitives. In *TCC '04*, volume 2951 of *LNCS*, pages 1–20, 2004.

[26] P. Rogaway. *The Round Complexity of Secure Protocols.* PhD thesis, MIT, June 1991.

[27] C. E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28-4:656–715, 1949.

[28] S. R. Tate and K. Xu. On garbled circuits and constant round secure function evaluation. CoPS Lab Technical Report 2003-02, University of North Texas, 2003.

[29] E. Viola. On parallel pseudorandom generators. *Electronic Colloquium on Computational Complexity*, 11(074), 2004. To appear in 20th IEEE Conference on Computational Complexity.

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

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

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

A. Proof of Lemma 4.7

The simulator. We start with the description of the simulator S. Given 1^n and $f_n(x)$ for some $x \in \{0,1\}^n$, the simulator chooses for every wire i of the circuit C_n a random string $W_i^{b_i}$, and a random bit c_i. For an input wire i, the simulator outputs $W_i^{b_i} \circ c_i$. For a gate t with input wires i, j and output wires $y_1, \ldots y_m$ the simulator computes the on-path label $Q_t^{c_i,c_j} = E_{W_i^{b_i,c_j} \oplus W_j^{b_j,c_i}}(W_{y_1}^{b_{y_1}} \circ c_{y_1} \circ \ldots \circ W_{y_m}^{b_{y_m}} \circ c_{y_m})$ and sets the other three off-path labels of this gate to be encryptions of all-zeros strings of appropriate length under random keys; that is, for every two bits $(a_i, a_j) \neq (c_i, c_j)$, the simulator chooses uniformly a $k(n)$-bit string R_{a_i,a_j} and outputs $Q_l^{a_i,a_j} = E_{R_{a_i,a_j}}(0^{|W_{y_1}^{b_{y_1}} \circ c_{y_1} \circ \ldots \circ W_{y_m}^{b_{y_m}} \circ c_{y_m}|})$. Finally, for an output wire i, the simulator outputs $r_i = c_i - b_i$ (recall that b_i is known since $f_n(x)$ is given).

Since C_n can be constructed in polynomial-time and since the encryption algorithm runs in polynomial time the simulator is also a polynomial-time algorithm. We refer to the gate labels constructed by the simulator as "fake" gate labels and to gate labels of $\hat{f}_n$ as "real" gate labels.

Assume, towards a contradiction, that there exists a (non-uniform) polynomial-size circuit family $\{A_n\}$, a polynomial $p(\cdot)$, and a string family $\{x_n\}$, $|x_n| = n$, such that A_n distinguishes between the distributions $S(1^n, f_n(x_n))$ and $\hat{f}_n(x_n, U_{\mu(n)})$ with non-negligible advantage $\varepsilon(n)$. We construct a sequence of hybrid distributions that depend on x_n, and mix "real" gates labels and "fake" ones, such that one endpoint corresponds to the simulated output (in which all the gates have "fake" labels) and the other endpoint corresponds to $\hat{f}_n(x_n, U_{\mu(n)})$ (in which all the gates have real labels). If the extreme hybrids can be efficiently distinguished then there must be two neighboring hybrids that can be efficiently distinguished. We show that such a distinguisher can be used to break the encryption hence deriving a contradiction.

The hybrids H_t^n. First, fix n and let $k = k(n)$. Next, we order the gates of C_n in topological order. That is, if the gate t uses the output of gate t', then $t' < t$. Now, for every $t = 0, \ldots, \Gamma(n)$, we define the hybrid algorithm H_t^n that constructs "fake" labels for the first t gates and "real" labels for the rest of the gates:

1. For every wire i uniformly choose two k-bit strings $W_i^{b_i}, W_i^{1-b_i}$ and a random bit c_i.
2. For every input wire i output $W_i^{b_i} \circ c_i$.
3. For every gate $t' \leq t$ with input wires i, j and output wires $y_1, \ldots, y_m$ output
$$Q_{t'}^{c_i,c_j} = E_{W_i^{b_i,c_j} \oplus W_j^{b_j,c_i}}(W_{y_1}^{b_{y_1}} \circ c_{y_1} \circ \ldots \circ W_{y_m}^{b_{y_m}} \circ c_{y_m}),$$
and for every choice of $(a_i, a_j) \in \{0,1\}^2$ that is different from (c_i, c_j), uniformly choose a k-bit string R_{a_i,a_j} and output $Q_{t'}^{a_i,a_j} = E_{R_{a_i,a_j}}(0^{|W_{y_1}^{b_{y_1}} \circ c_{y_1} \circ \ldots \circ W_{y_m}^{b_{y_m}} \circ c_{y_m}|})$.
4. For every gate $t' > t$, let g be the function that t' computes (AND or OR), let i, j be the input wires of t' and let $y_1, \ldots y_m$ be its output wires. Use x_n to compute the value of $b_i(x_n), b_j(x_n)$, and set $r_i = c_i - b_i$ and $r_j = c_j - b_j$. For every choice of $(a_i, a_j) \in \{0,1\}^2$, compute $Q_{t'}^{a_i,a_j}$ exactly as in Equation 4.1, and output it.
5. For every output wire i compute b_i and output $r_i = c_i - b_i$.

First, observe that H_t^n runs in polynomial time (when x_n is given). Second, note that this hybrid algorithm uses the string x_n only when constructing real labels, that is in Step 4. Steps 1–3 can be performed without any knowledge on x_n, and Step 5 requires only the knowledge of $f_n(x_n)$. Obviously, the algorithm $H_{\Gamma(n)}^n$ is just a different description of the simulator S, and therefore $S(1^n, f_n(x_n)) \equiv H_{\Gamma(n)}^n$. We also claim that the second extreme hybrid, H_0^n coincides with the distribution of the "real" encoding, $\hat{f}_n(x_n, U_{\mu(n)})$. To see this note that (1) the strings W_i^0, W_i^1 are chosen uniformly and independently by H_0^n, as they are in $\hat{f}_n(x_n, U_{\mu(n)})$; and

(2) since H_0^n chooses the c_i's uniformly and independently and sets $r_i = c_i - b_i$ then the r_i's themselves are also distributed uniformly and independently exactly as they are in $\hat{f}_n(x_n, U_{\mu(n)})$. Since for every gate t the value of $Q_t^{a_i,a_j}$ is a function of the random variables, and since it is computed by H_0^n in the same way as in $\hat{f}_n(x_n, U_{\mu(n)})$, we get that $H_0^n \equiv \hat{f}_n(x_n, U_{\mu(n)})$.

Since A_n can distinguish the extreme hybrids with non-negligible advantage $\varepsilon(n)$ and since the number of hybrids (i.e., the size of C_n) is polynomial in n, it follows that A_n can also distinguish some neighboring hybrids with non-negligible advantage; namely, there exists some $0 \leq t \leq \Gamma(n) - 1$ such that C_n distinguishes between H_t^n and H_{t+1}^n with advantage $\varepsilon'(n) \geq \varepsilon(n)/\Gamma(n)$. We use A_n, x_n, t to construct an adversary that breaks the encryption E.

Let i, j be the input wires of the gate t, let $y_1, \ldots, y_m$ be the output wires of t, let g be the function that gate t computes, and let $b_i, b_j, b_{y_1}, \ldots, b_{y_m}$ be the values of the corresponding wires induced by the input x_n. For $\sigma \in \{0,1\}$, define the distribution ensemble $P_n(\sigma)$ as the output distribution of the following random process:

- Uniformly choose the k-bit strings $W_i^{b_i}, W_j^{b_j}, W_{y_1}^{b_{y_1}}, W_{y_1}^{1-b_{y_1}}, \ldots, W_{y_m}^{b_{y_m}}, W_{y_m}^{1-b_{y_m}}$, and the random bits $c_i, c_j, c_{y_1}, \ldots, c_{y_m}$.
- If $\sigma = 0$ then set $Q_t^{c_i,c_j}$ and the other three $Q_t^{a_i,a_j}$ exactly as in Step 3 of H_t^n.
- If $\sigma = 1$ then uniformly choose $W_i^{1-b_i}, W_j^{1-b_j}$, set $r_i = c_i - b_i$, $r_j = c_j - b_j$, and for every choice of $(a_i, a_j) \in \{0,1\}^2$, set $Q_t^{a_i,a_j}$ exactly as in Step 4 of H_t^n, that is:

$$Q_t^{a_i,a_j} = E_{W_i^{a_i-r_i,a_j} \oplus W_j^{a_j-r_j,a_i}}(W_{y_1}^{g(a_i-r_i,a_j-r_j)} \circ g(a_i - r_i, a_j - r_j) + r_{y_1} \circ \ldots \circ W_{y_m}^{g(a_i-r_i,a_j-r_j)} \circ g(a_i - r_i, a_j - r_j) + r_{y_m})$$

- Output $(W_i^{b_i}, W_j^{b_j}, W_{y_1}^{b_{y_1}}, W_{y_1}^{1-b_{y_1}}, \ldots, W_{y_m}^{b_{y_m}}, W_{y_m}^{1-b_{y_m}}, c_i, c_j, c_{y_1}, \ldots, c_{y_m}, Q_t^{0,0}, Q_t^{0,1}, Q_t^{1,0}, Q_t^{1,1})$.

The following claim reduces the indistinguishability of the distribution ensembles $P_n(0), P_n(1)$ to the indistinguishability of the one-time symmetric encryption scheme.

Claim A.1 *If E is a one-time symmetric encryption then the distribution ensembles $P_n(0)$ and $P_n(1)$ are computationally indistinguishable.*

Proof sketch: Clearly, the ensembles $P_n(0)$ and $P_n(1)$ differ only in the off-path labels $Q_t^{a_i,a_j}$ for $(a_i, a_j) \neq (c_i, c_j)$. In $P_n(0)$ each of these entries is an all-zeros string that was encrypted under uniformly and independently chosen key R_{a_i,a_j}. In the second distribution $P_n(1)$, the entry $Q_t^{a_i,a_j}$ is an encryption of a "meaningful" message that was encrypted under the key $W_i^{a_i-r_i,a_j} \oplus W_j^{a_j-r_j,a_i}$, since $(a_i, a_j) \neq (c_i, c_j)$ at least one of the strings $W_i^{a_i-r_i,a_i}, W_j^{a_j-r_j,a_j}$ is not given in the output of $P_n(1)$ as part of $W_i^{b_i}, W_j^{b_j}$. Also, each of the strings $W_i^{a_i-r_i,a_j}, W_j^{a_j-r_j,a_i}$ was chosen uniformly and it appears only in $Q_t^{a_i,a_j}$ and not in any of the other gate labels, therefore the key $W_i^{a_i-r_i,a_j} \oplus W_j^{a_j-r_j,a_i}$ is a uniformly chosen key that is not correlated with other entries of $P_n(1)$'s output. So the difference between these two distributions amounts to the difference between three encryptions of zeros (in $P_n(0)$) and three encryptions of some other strings in the same length as the zeros strings (in $P_n(1)$), where in both cases the three encryptions are encrypted under three uniformly and independently chosen keys. The indistinguishability of such encryptions easily follows from the security of the encryption scheme. ■

We now show how to construct a non-uniform distinguisher A'_n for $P_n(0)$ and $P_n(1)$ from the distinguisher A_n, and thus derive a contradiction to the previous claim. The adversary A'_n uses the output of $P_n(\sigma)$ to construct the hybrids H_t^n and H_{t+1}^n, and then uses A_n to distinguish between them. Given the output of P_n, we invoke the algorithm H_t^n where the values of $(W_i^{b_i}, W_j^{b_j}, W_{y_1}^{b_{y_1}}, W_{y_1}^{1-b_{y_1}}, \ldots, W_{y_m}^{b_{y_m}}, W_{y_m}^{1-b_{y_m}}, c_i, c_j, c_{y_1}, \ldots, c_{y_m}, Q_t^{0,0}, Q_t^{0,1}, Q_t^{1,0}, Q_t^{1,1})$ are set to the values given by P_n. If $P_n(0)$ is invoked we get the distribution of H_t^n that is the gate t is "fake"; on the other hand, if $P_n(1)$ is invoked then the gate t is "real" and we get the distribution of H_{t+1}^n. Note that P_n does not output the off-path signals of the wires i and j (which is crucial for Claim A.1 to hold). However, we do not need these off-path signals to construct the rest of the distribution, since i and j are output wires of gates that precedes t and therefore are "fake" gates in which off-path signals are not encrypted. This is the reason for which we had to sort the gates. On the other hand, the process P_n must output the off-path signals of the output wires of the gate $y_1, \ldots, y_m$; since these wires enter as inputs to another gate $t' > t$ which is a "real" gate in both H_t^n and H_{t+1}^n, and therefore uses the off-path signals of its input wires. Overall, we derive a contradiction to Claim A.1, which completes the privacy proof of Lemma 4.7.