Computationally Private Randomizing Polynomials and Their Applications[*]

Benny Applebaum Yuval Ishai Eyal Kushilevitz

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

March 5, 2006

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 uniform-$\oplus\mathrm{L}/poly$ (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 uniform-$\oplus\mathrm{L}/poly$. (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 uniform-$\oplus\mathrm{L}/poly$, 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 uniform-$\oplus\mathrm{L}/poly$ 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 uniform-$\oplus\mathrm{L}/poly$, the task of computing an *arbitrary* (polynomial-time computable) function with computational security can be reduced to the task of securely computing degree-3 polynomials (say, over $\mathrm{GF}(2)$) without further interaction. This gives rise to new, conceptually simpler, constant-round protocols for general functions.

[*]A preliminary version of this paper appeared in the proceedings of CCC 2005. Research 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 [IK00, IK02], and has recently found applications to parallel constructions of cryptographic primitives [AIK04]. 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 [IK00, AIK04]. 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* [IK00]. In other contexts it is natural to view $\hat{f}$ as a function over binary strings and attempt to minimize its parallel time complexity [AIK04]. 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 [IK00, IK02] that every function f in $\oplus\mathrm{L}/poly$[1] can be efficiently represented by degree-3 randomizing polynomials over $\mathrm{GF}(2)$.[2] 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 [AIK04]. 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) [BM84, Yao82], namely one that stretches its seed by just one bit, in $\oplus\mathrm{L}/poly$.[3] 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 4.16.) We note that EPRG is a very mild assumption. In particular, it is

[1]For brevity, all complexity classes are assumed to be polynomial-time uniform by default. In particular, the class $\oplus\mathrm{L}/poly$ is assumed to be polynomial-time uniform. (See Section 2 for a definition of this class.) The class $\oplus\mathrm{L}/poly$ contains $\mathrm{L}/poly$ and NC^1 and is contained in NC^2. In the non-uniform setting nonuniform-$\oplus\mathrm{L}/poly$ also contains nonuniform-$\mathrm{NL}/poly$ [Wig94]. However, such an inclusion is not known to hold in the uniform setting.

[2]This result generalizes to arbitrary finite fields [IK02] or even rings [CFIK03], allowing efficient degree-3 representations of various counting logspace classes.

[3]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.

implied by most concrete intractability assumptions commonly used in cryptography, such as ones related to factoring, discrete logarithm, or lattice problems (see [AIK04, Remark 6.6]). It is also implied by the existence in $\oplus\mathrm{L}/poly$ of a one-way permutation or, using [HILL99], of any *regular* one-way function (OWF); i.e., a OWF $f = \{f_n\}$ that maps the same (polynomial-time computable) number of elements in $\{0,1\}^n$ to every element in $\mathrm{Im}(f_n)$. (This is the case, for instance, for any one-to-one OWF.)[4] 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.

We now give a high-level overview of our construction. Recall that we wish to encode a polynomial-time computable function by an NC^0 function. To do this we rely on a variant of Yao's *garbled circuit* technique [Yao86]. Roughly speaking, Yao's technique allows to efficiently "encrypt" a boolean circuit in a way that enables to compute the output of the circuit but "hides" any other information about the circuit's input. These properties resemble the ones required for randomized encoding.[5] Moreover, the garbled circuit enjoys a certain level of locality (or parallelism) in the sense that gates are encrypted independently of each other. Specifically, each encrypted gate is obtained by applying some cryptographic primitive (typically, a high-stretch PRG or an encryption scheme with special properties), on a constant number of (long) random strings and, possibly, a single input bit. However, the overall circuit might not have constant locality (it might not even be computable in NC) as the cryptographic primitive being used in the gates might be sequential in nature. Thus, the bottleneck of the construction is the parallel time complexity of the primitive being used.

Recently, it was shown in [AIK04] (via "information theoretic" randomized encoding) that under relatively mild assumptions many cryptographic primitives can be computed in NC^0. Hence, we can try to plug one of these primitives into the garbled circuit construction in order to obtain an encoding with constant locality. However, a direct use of this approach would require stronger assumptions[6] than EPRG and result in an NC^0 encoding with inferior parameters. Instead, we use the following variant of this approach.

Our 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. Note that we are only assuming the existence of a minimal PRG, i.e., a PRG that stretches its seed only by one bit. Such a PRG 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[Yao86] to obtain an encoding in NC^0 which uses one-time symmetric encryption as an oracle. By combining these two steps we get an encoding that can be computed by an NC^0 circuit which uses a minimal PRG as an oracle. Finally, using the EPRG assumption and [AIK04], 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 [NR99, AIK04] 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

[4]Using [HILL99] or [HHR05] this regularity requirement can be relaxed. See [AIK04, Footnote 14].

[5]This similarity is not coincidental as both concepts were raised in the context of secure multiparty computation. Indeed, an information theoretic variant of Yao's garbled circuit technique was already used in [IK02] to construct low degree randomized encoding for NC^1 functions.

[6]Previous presentations of Yao's garbled circuit relied on primitives that seem less likely to allow an NC^0 implementation. Specifically, [BMR90, NPS99] require linear stretch PRG and [LP04] requires symmetric encryption that enjoys some additional properties.

known that few primitives, including pseudorandom *functions* [GGM86], cannot even be implemented in AC^0 [LMN93], no similar negative results were known for other primitives.

Very recently, it was shown in [AIK04] 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, (computationally hiding) commitments, and (interactive or non-interactive) zero-knowledge proofs. Thus, given that these primitives at all exist,[7] their existence in NC^0 follows from the EPRG assumption, namely from the existence of a PRG in complexity classes such as $\oplus\mathrm{L}/poly$. Previously (using [AIK04]), the existence of each of these primitives in NC^0 would only follow from the assumption that this particular 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 [AIK04] 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 [AIK04] 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 [GL89] or even to more general types of one-way functions [HILL99, Vio05, HHR05] can be encoded into NC^0-reductions (see [AIK04, Remark 6.7]). 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 [AIK04] 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,[8] 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 [Nao91] requires a linear-stretch PRG (expanding n bits into $n + \Omega(n)$ 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

[7] 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.

[8] 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.

based on the parallel PRF construction of [NR99]. 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.)

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 [AIK04]. (See [RTV04] 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 multi-party computation [Yao86, GMW87, BGW88, CCD88]. The main relevant observations made in [IK00] 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 task of securely computing 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 a protocol for f can be obtained by invoking a protocol for $\hat{f}$ and applying a *local* computation on its outputs (without additional 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" [BGW88] gives a simpler, and in some cases more efficient, alternative to the constant-round protocol of Beaver, Micali and Rogaway [BMR90] (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 constant $c > 0$ and sufficiently large n. For two distribution ensembles $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, we write $\{X_n\}_{n\in\mathbb{N}} \equiv \{Y_n\}_{n\in\mathbb{N}}$ if X_n and Y_n are identically distributed, and say that the two ensembles are statistically indistinguishable if $\mathrm{SD}(X_n,Y_n)$ is negligible in n. A weaker notion of closeness between distributions is that of *computational* indistinguishability: We write $\{X_n\}_{n\in\mathbb{N}} \overset{c}{\equiv} \{Y_n\}_{n\in\mathbb{N}}$ if for every (non-uniform) polynomial-size circuit family $\{A_n\}$, the distinguishing advantage $|\Pr[A_n(X_n) = 1] - \Pr[A_n(Y_n) = 1]|$ is negligible. (We will sometimes simplify notation and write $X_n \overset{c}{\equiv} Y_n$.) By default we adapt this non-uniform notion of indistinguishability. However, our results also apply in a uniform setting in which adversaries are probabilistic polynomial-time algorithms.

We will rely on several standard facts about computational indistinguishability (cf. [Gol01, Chapter 2]).

Fact 2.1 *For every distribution ensembles X, Y and Z, if $X \overset{c}{\equiv} Y$ and $Y \overset{c}{\equiv} Z$ then $X \overset{c}{\equiv} Z$.*

That is, computational indistinguishability is transitive. The following fact asserts that computational indistinguishability is preserved under multiple independent samples:

Fact 2.2 *Let $\{X_n\}, \{X'_n\}, \{Y_n\}$ and $\{Y'_n\}$ be distribution ensembles. Suppose that $X_n \overset{c}{\equiv} Y_n$ and $X'_n \overset{c}{\equiv} Y'_n$. Then $(X_n \times X'_n) \overset{c}{\equiv} (Y_n \times Y'_n)$, where $A \times B$ denotes the product distribution of A, B (i.e., the joint distribution of independent samples from A and B).*

Another basic fact is that computational indistinguishability is preserved under efficient computation:

Fact 2.3 *Suppose that that the distribution ensembles $\{X_n\}$ and $\{Y_n\}$ are computationally indistinguishable. Then for every polynomial-time computable function f we have $f(X_n) \overset{c}{\equiv} f(Y_n)$.*

Consider a case in which two probabilistic (possibly computationally unbounded) algorithms behave "similarly" on every input, in the sense that their output distributions are computationally indistinguishable. The following two facts deal with such a situation. Fact 2.4 asserts that an efficient procedure that gets an oracle access to one of these algorithms cannot tell which algorithm it communicates with. Fact 2.5 asserts that the outputs of these algorithms cannot be distinguished with respect to any (not necessarily efficiently samplable) input distribution. These facts will allow us to argue that, in most applications, computationally-private encodings can be securely used as substitutes for perfectly private encodings.

Fact 2.4 *Let X and Y be probabilistic algorithms such that for every string family $\{z_n\}$ where $z_n \in \{0,1\}^n$, it holds that $X(z_n) \overset{c}{\equiv} Y(z_n)$. Then, for any (non-uniform) polynomial-time oracle machine A, it holds that $A^X(1^n) \overset{c}{\equiv} A^Y(1^n)$ (where A does not have access to the random coins of the given probabilistic oracle).*

Fact 2.5 *Let X and Y be probabilistic algorithms such that for every string family $\{z_n\}$ where $z_n \in \{0,1\}^n$, it holds that $X(z_n) \overset{c}{\equiv} Y(z_n)$. Then, for every distribution ensemble $\{Z_n\}$ where Z_n is distributed over $\{0,1\}^n$, we have $(Z_n, X(Z_n)) \overset{c}{\equiv} (Z_n, Y(Z_n))$.*

Circuits. Boolean circuits are defined in a standard way. That is, 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 class nonuniform-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 use the (somewhat nonstandard) convention that all complexity classes are polynomial-time uniform by default. For instance, NC^0 refers to the class of functions admitting polynomial-time uniform NC^0 circuits, whereas *nonuniform*-NC^0 refers to the class of functions admitting non-uniform NC^0 circuits. We let $\mathrm{NL}/poly$ (resp., $\oplus\mathrm{L}/poly$) denote the class of boolean functions computed by NL (resp., $\oplus$L) Turing machines taking a polynomial-time uniform advice. (The class $\oplus\mathrm{L}/poly$ contains the classes $\mathrm{L}/poly$ and NC^1 and is contained in NC^2.) 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 $\ell(n)$ log-space Turing machines, one for each output bit, taking the same uniform advice. Similarly, we denote by P (resp. BPP) the class of *functions* that can be computed in polynomial time (resp. probabilistic polynomial time). For instance, a function $f : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ is in BPP if there exists a probabilistic polynomial-time machine A such that for every $x \in \{0,1\}^n$ it holds that $\Pr[A(x) \neq f(x)] \leq 2^{-n}$, where the probability is taken over the internal coin tosses of A.

3 Randomized Encodings

We now review the notions of randomized encoding and randomizing polynomials from [IK00, IK02, AIK04], and introduce the new computationally private variant discussed in this paper. The following definition is from [AIK04].

Definition 3.1 (Randomized encoding) *Let $f : \{0,1\}^n \to \{0,1\}^\ell$ 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 *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 $\ell, m, s, \delta, \varepsilon$ 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 [AIK04]. 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 $S(U_\ell) \equiv U_s$. It is said to be *stretch preserving* if $s = \ell + m$. 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 $f = \{f_n : \{0,1\}^n \to \{0,1\}^{\ell(n)}\}_{n\in\mathbb{N}}$ 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 [AIK04] 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 f admitting a computational (resp., statistical, perfect) randomized encoding $\hat{f}$ in NC^0. (As usual, NC^0 is polynomial-time uniform.)*

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 $\mathrm{NL}/poly \subseteq \mathcal{SREN}$ [AIK04]. (We cannot use the fact that nonuniform-$\mathrm{NL}/poly$ $\subseteq$nonuniform-$\oplus\mathrm{L}/poly$ [Wig94] to conclude that $\mathrm{NL}/poly \subseteq \mathcal{PREN}$, since this inclusion is only known to hold in the non-uniform setting.)

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 [AIK04] for the statistical and perfect variants of randomized encodings; we extend it here to the computational variant.

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

Proof: We start with correctness. Let B_g be a $\delta_g(n)$-correct decoder for g and B_h a $\delta_h(n+m_g(n))$-correct decoder for h, where $m_g(n)$ is the randomness complexity of g. Define a decoder B for $\hat{f}$ by $B(\hat{y}) = B_g(B_h(\hat{y}))$. The decoder B errs only if either B_h or B_g err. Thus, by the union bound we have for every $x \in \{0,1\}^n$,

$$\begin{aligned}\Pr_{r_g,r_h}[B(1^n, \hat{f}(x,(r_g,r_h)) \neq f(x)] &\leq \Pr_{r_g,r_h}[B_h(1^n, h((x,r_g),r_h)) \neq g(x,r_g)] + \Pr_{r_g}[B_g(1^n, g(x,r_g)) \neq f(x)] \\ &\leq \delta_h(n+m_g(n)) + \delta_g(n),\end{aligned}$$

and so B is perfectly correct if both the decoders of h and g are perfectly correct. Moreover, since $m_g(n)$ is polynomial in n, if the decoders of h and g are statistically correct then so is B.

To prove computational privacy, we again compose the computationally private simulators of g and h, this time in an opposite order. Specifically, let S_g be a computationally-private simulator for g and S_h be computationally-private simulator for h. We define a simulator S for $\hat{f}$ by $S(y) = S_h(S_g(y))$. Letting $m_g(n)$ and $m_h(n)$ denote the randomness complexity of g and h, respectively, and $\{x_n\}_{n\in\mathbb{N}}$ be a family of strings where $|x_n| = n$, we have,

$$\begin{aligned}S_h(S_g(f(x_n))) &\overset{c}{\equiv} S_h(g(x_n, U_{m_g(n)})) && \text{(since } g \text{ is a comp. private encoding of } f \text{ and by Fact 2.3)} \\ &\overset{c}{\equiv} h((x_n, U_{m_g(n)}), U_{m_h(n)}) && \text{(since } h \text{ is a comp. private encoding of } g \text{ and by Fact 2.5).}\end{aligned}$$

Hence, the transitivity of the relation $\overset{c}{\equiv}$ (Fact 2.1) finishes the proof. ■

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 [AIK04]. The same holds for the class $\mathcal{CREN}$, since we can encode a function $f \in \mathcal{CREN}$ by a computational encoding g in NC^0 and then encode g using a perfect encoding h of degree 3 and locality 4 (promised by the fact that $\mathrm{NC}^0 \subseteq \mathcal{PREN}$). By Lemma 3.5, the function h is a computational encoding for f of degree 3 and locality 4. Moreover, the complexity of h is linearly related to the complexity of g.

4 Computational Encoding in NC^0 for Efficiently Computable Functions

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 [Yao86]. 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", namely, a minimal PRG in $\oplus\mathrm{L}/poly$.

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 [GM84] 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 minimal PRG and one-time symmetric encryption.

Definition 4.1 (Pseudorandom generator) *A pseudorandom generator (PRG) is a deterministic polynomial-time algorithm, G, satisfying the following two conditions:*

- ***Expansion**: There exists a function $\ell(k) : \mathbb{N} \to \mathbb{N}$ satisfying that $\ell(k) > k$ for all $k \in \mathbb{N}$, such that $|G(x)| = \ell(|x|)$ for all $x \in \{0,1\}^*$.*
- ***Pseudorandomness**: The distribution ensembles $\{G(U_k)\}_{k\in\mathbb{N}}$ and $\{U_{\ell(k)}\}_{k\in\mathbb{N}}$ are computationally indistinguishable.*

A PRG that stretches its input by one bit (i.e., $\ell(k) = k + 1$) is referred to as a minimal *PRG. When $\ell(k) = k + \Omega(k)$ we say that G is a* linear-stretch *PRG. We refer to G as a* polynomial-stretch *PRG if $\ell(k) = \Omega(k^c)$ for some constant $c > 1$.*

Definition 4.2 (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 $D_e(E_e(m)) = m$ (where $E_e(m) \overset{def}{=} E(e, m)$ and similarly for D).*
- ***Security:** For every polynomial $\ell(\cdot)$, and every families of plaintexts $\{x_k\}_{k\in\mathbb{N}}$ and $\{x'_k\}_{k\in\mathbb{N}}$ where $x_k, x'_k \in \{0,1\}^{\ell(k)}$, it holds that*

$$\{E_{U_k}(x_k)\}_{k\in\mathbb{N}} \overset{c}{\equiv} \{E_{U_k}(x'_k)\}_{k\in\mathbb{N}}.$$

The integer k serves as the security parameter *of the scheme. The scheme is said to be $\ell(\cdot)$-one-time symmetric encryption scheme if correctness and security hold with respect to plaintexts whose length is bounded by $\ell(k)$.*

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[9] 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. [Gol01, Thm. 3.3.3]). In contrast, this is not known to be the case with

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

respect to parallel reductions. It is not even known whether a linear-stretch PRG is NC-reducible to a minimal PRG (see [Vio05] for some relevant negative results). Thus, a minimal PRG is a more conservative assumption from the point of view of parallel cryptography. 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 [HILL99, Vio05, AIK04].

The above discussion motivates a *direct* parallel construction of one-time symmetric encryption using a minimal PRG, i.e., a construction that does not rely on a "stronger" type of PRG as an intermediate step. We present such an NC^0 construction below.

Construction 4.3 (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 $(k+\ell)$-bit plaintext. Define the probabilistic encryption algorithm $E_e(m,(r_1,\ldots,r_{\ell-1})) \overset{def}{=} (G(e)\oplus r_1, G(r_1)\oplus r_2,\ldots,G(r_{\ell-2})\oplus r_{l-1}, G(r_{\ell-1})\oplus m)$, where $r_i \leftarrow U_{k+i}$ serve as the coin tosses of E. The decryption algorithm $D_e(c_1,\ldots,c_{\ell-1})$ sets $r_0 = e$, $r_i = c_i \oplus G(r_{i-1})$ for $i = 1,\ldots,\ell$, and outputs r_ℓ.*

We prove the security of Construction 4.3 via a standard hybrid argument.

Lemma 4.4 *The scheme (E,D) described in Construction 4.3 is a one-time symmetric encryption scheme.*

Proof: Construction 4.3 can be easily verified to satisfy the correctness requirement. We now prove the security of this scheme. Assume, towards a contradiction, that Construction 4.3 is not secure. It follows that there is a polynomial $\ell(\cdot)$ and two families of strings $x = \{x_k\}$ and $y = \{y_k\}$ where $|x_k| = |y_k| = k+\ell(k)$, such that 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 $k+\ell(k)$ we define for $0 \le i \le \ell(k)$ 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 $1 \le i \le \ell(k)$, 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 $m \in \{0,1\}^{k+\ell(k)}$, all the $\ell(k)$ strings of the hybrid $H_{\ell(k)}(m)$ are distributed uniformly and independently (each of them is the result of XOR with a fresh random string w_i). Therefore, in particular, $H_{\ell(k)}(x_k) \equiv H_{\ell(k)}(y_k)$. 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 $H_{\ell(k)}(x_k)$ and $H_0(x_k)$ with at least $\varepsilon(k)/2$ advantage. Then, since there are $\ell(k)$ hybrids, there must be $1 \le i \le \ell(k)$ such that the neighboring hybrids, $H_{i-1}(x_k), H_i(x_k)$, can be distinguished by A_k with $\frac{\varepsilon(k)}{2\ell(k)}$ 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 $j = 1,\ldots,\ell(k)-1$. We feed A_k with the sample $(r_1,\ldots,r_{i-1}, z\oplus r_i, G(r_i)\oplus r_{i+1},\ldots,G(r_{\ell(k)-1})\oplus x_k)$. 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 $\frac{\varepsilon(k)}{2\ell(k)}$, deriving a contradiction. ■

Since the encryption algorithm described in Construction 4.3 is indeed an NC^0 circuit with oracle access to a minimal PRG, we get the following lemma.

Lemma 4.5 *Let G be a minimal 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. We can parallelize it (without harming the parallelization of the encryption) at the expense of strengthening the assumption we use.

Claim 4.6 *Let PG (resp. LG) be a polynomial-stretch (resp. linear-stretch) PRG. Then, for every polynomial $p(\cdot)$ there exists a $p(\cdot)$-one-time symmetric encryption scheme (E, D) such that $E \in \mathrm{NC}^0[PG]$ and $D \in \mathrm{NC}^0[PG]$ (resp. $E \in \mathrm{NC}^0[LG]$ and $D \in \mathrm{NC}^1[LG]$).*

Proof: Use Construction 4.3 (where $|r_i| = |G(r_{i-1})|$). When the stretch of G is polynomial (resp. linear) the construction requires only $O(1)$ (resp. $O(\log k)$) invocations of G, and therefore, so does the decryption algorithm. ■

4.2 From One-Time Encryption to Computational Encoding

Let $f = \{f_n : \{0,1\}^n \to \{0,1\}^{\ell(n)}\}_{n\in\mathbb{N}}$ 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. The construction uses a variant of Yao's garbled circuit technique [Yao86]. Our notation and terminology for this section borrow from previous presentations of Yao's construction in [Rog91, NPS99, LP04].[10] Before we describe the actual encoding it will be convenient to think of the following "physical" analog that uses locks and boxes.

A physical encoding. To each wire of the circuit we assign a pair of keys: a 0-key that represents the value 0 and a 1-key that represents the value 1. For each of these pairs we randomly color one key black and the other key white. This way, given a key one cannot tell which bit it represents (since the coloring is random). For every gate of the circuit, the encoding consists of four double-locked boxes – a white-white box (which is locked by the white keys of the wires that enter the gate), a white-black box (locked by the white key of the left incoming wire and the black key of the right incoming wire), a black-white box (locked by the black key of the left incoming wire and the white key of the right incoming wire) and a black-black box (locked by the black keys of the incoming wires). Inside each box we put one of the keys of the gate's output wires. Specifically, if a box is locked by the keys that represent the values α, β then for every outgoing wire we put in the box the key that represents the bit $g(\alpha, \beta)$, where g is the function that the gate computes. For example, if the gate is an OR gate then the box which is locked by the incoming keys that represent the bits $(0, 1)$ contains all the 1-keys of the outgoing wires. So if one has a single key for each of the incoming wires, he can open only one box and get a single key for each of the outgoing wires. Moreover, as noted before, holding these keys does not reveal any information about the bits they represent.

Now, fix some input x for f_n. For each wire, exactly one of the keys corresponds to the value of the wire (induced by x); we refer to this key as the *active key* and to the second key as the *inactive* key. We include in the encoding of $f_n(x)$ the active keys of the input wires. (This is the only place in which the encoding depends on the input x.) Using these keys and the locked boxes as described above, one can obtain the active keys of all the wires by opening the corresponding boxes in a bottom-to-top order. To make this information useful (i.e., to enable decoding of $f_n(x)$), we append to the encoding the semantics of the output wires;

[10]Security proofs for variants of this construction were given implicitly in [Rog91, TX03, LP04] in the context of secure computation. However, they cannot be directly used in our context for different reasons. In particular, the analysis of [LP04] relies on a special form of symmetric encryption and does not achieve perfect correctness, while that of [Rog91, TX03] relies on a linear-stretch PRG.

namely, for each output wire we expose whether the 1-key is white or black. Hence, the knowledge of the active key of an output wire reveals the value of the wire.

The actual encoding. The actual encoding is analogous to the above physical encoding. We let random strings play the role of physical keys. Instead of locking a value in a double-locked box, we encrypt it under the XOR of two keys. Before formally defining the construction, we need the following 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 $1 \leq i \leq |C_n|$, 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.

Our encoding $\hat{f}_n(x, (r, W))$ consists of random inputs of two types: $|C_n|$ pairs of strings $W_i^0, W_i^1 \in \{0,1\}^{2k}$, and $|C_n|$ bits (referred to as masks) denoted $r_1, \ldots, r_{|C_n|}$.[11] The strings W_i^0, W_i^1 will serve as the 0-key and the 1-key of the i-th wire, while the bit r_i will determine which of these keys is the black key. We use c_i to denote the value of wire i masked by r_i; namely, $c_i = b_i \oplus r_i$. Thus, c_i is the color of the active key of the i-th wire (with respect to the input x). As before, the encoding $\hat{f}_n(x, (r, W))$ will reveal each active key $W_i^{b_i}$ and its color c_i but will hide the inactive keys $W_i^{1-b_i}$ and the masks r_i of all the wires (except the masks of the output wires). Intuitively, since the active keys and inactive keys are distributed identically, the knowledge of an active key $W_i^{b_i}$ does not reveal the value b_i.

The encoding $\hat{f}_n$ consists of the concatenation of $O(|C_n|)$ functions, which include 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. For every $\beta \in \{0,1\}$ and every i, we view the string W_i^β as if it is partitioned into two equal-size parts denoted $W_i^{\beta,0}, W_i^{\beta,1}$.

Construction 4.7 *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 of $(x, (r, W))$.*

Input wires: *For an input wire i, labeled by a literal ℓ (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 $Q_t^{a_i,a_j}$. This function can be thought of as the box whose color is (a_i, a_j). It is defined as follows:*

$$Q_t^{a_i,a_j}(r, W) \overset{\text{def}}{=} E_{W_i^{a_i - r_i, a_j} \oplus W_j^{a_j - r_j, a_i}} \Big(W_{y_1}^{g(a_i - r_i, a_j - r_j)} \circ (g(a_i - r_i, a_j - r_j) + r_{y_1}) \circ \ldots \\ \circ W_{y_m}^{g(a_i - r_i, a_j - r_j)} \circ (g(a_i - r_i, a_j - r_j) + r_{y_m}) \Big), \quad (4.1)$$

where E is a one-time symmetric encryption algorithm. (For simplicity, the randomness of E is omitted.) That is, the colored keys of all the output wires of this gate are encrypted under a key that depends on the keys 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 colors of the active keys of the input wires as an active label, *and to the other three labels as the* inactive labels.

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

[11] In fact, each application of the encryption scheme will use some additional random bits. To simplify notation, we keep these random inputs implicit.

It is not hard to verify that $\hat{f}_n$ is in $\mathrm{NC}^0[E]$. In particular, a term of the form W_i^ℓ is a 3-local function of W_i^0, W_i^1 and ℓ, since its j-th bit depends on the j-th bit of W_i^0, the j-th bit of W_i^1 and on the literal ℓ. Similarly, the keys that are used in the encryptions are 8-local functions, and the arguments to the encryption are 6-local functions of (r, W).

We will now analyze the complexity of $\hat{f}_n$. The output complexity and randomness complexity of $\hat{f}$ are both dominated by the complexity of the gate labels. Generally, the complexity of these functions is $\mathrm{poly}(|C_n| \cdot k)$ (since the encryption E is computable in polynomial time).[12] However, when the circuit C_n has bounded fan-out (say 2) each invocation of the encryption uses $\mathrm{poly}(k)$ random bits and outputs $\mathrm{poly}(k)$ bits. Hence, the overall complexity is $O(|C_n|) \cdot \mathrm{poly}(k) = O(|C_n| \cdot n^\varepsilon)$ for an arbitrary constant $\varepsilon > 0$. Since any circuit with unbounded fan-out of size $|C_n|$ can be (efficiently) transformed into a bounded-fanout circuit whose size is $O(|C_n|)$ (at the price of a logarithmic factor in the depth), we get an encoding of size $O(|C_n| \cdot n^\varepsilon)$ for every (unbounded fan-out) circuit family $\{C_n\}$.

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 4.8 (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, our decoder computes, for every wire i, the active key $W_i^{b_i}$ and its color c_i. Then, for an output wire i, the decoder retrieves the mask r_i from α and computes the corresponding output bit of $f_n(x)$; i.e., outputs $b_i = c_i - r_i$. (Recall that the masks of the output wires are given explicitly as part of α.) The active keys and their colors 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 colors c_i, c_j of the active keys of the input wires to select the active label $Q_t^{c_i,c_j}$ of the gate t (and ignore the other 3 inactive 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). Hence, we can recover the encrypted plaintext, 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. ■

Remark 4.9 By the description of the decoder it follows that if the circuit C_n is in NC^i, then the decoder is in $\mathrm{NC}^i[D]$, where D is the decryption algorithm. In particular if D is in NC^j then the decoder is in NC^{i+j}. This fact will be useful for some of the applications discussed in Section 5.

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

Lemma 4.10 (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 $S(1^n, f_n(x_n)) \overset{c}{\equiv} \hat{f}_n(x_n, U_{\mu(n)})$.*

[12]Specifically, the encryption is always invoked on messages whose length is bounded by $\ell(n) \overset{\text{def}}{=} O(|C_n| \cdot k)$, hence we can use $\ell(n)$-one-time symmetric encryption.

Remark 4.11 (Information-theoretic variant) Construction 4.7 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 [Sha49]. It follows that the wires' key 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 [IK02]. We will use it in Section 4.3 for obtaining a computational encoding with a parallel decoder.

4.3 Main Results

Combining Lemmas 4.8, 4.10, and 4.5 we get an NC^0 encoding of any efficiently computable function using an oracle to a minimal PRG.

Theorem 4.12 *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 4.16.) Using [AIK04, Thm 6.5], 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 4.13 (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 4.12 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 4.14 *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 4.15 *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 4.14. Since f' is a statistical encoding of f (the simulator and the decoder are simply the identity functions), it follows from Lemma 3.5 that $\hat{f}(x,(z,r)) \stackrel{\text{def}}{=} \hat{f}'((x,z),r)$ 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 4.16 (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 4.14 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 [IK02, AIK04] 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 $\mathrm{NL}/poly$. Another alternative is to assume the existence of a one-time symmetric encryption (E, D) whose encrypting algorithm E is in $\mathcal{SREN}$. According to [AIK04] the last assumption is equivalent to the existence of one-time symmetric encryption (with negligible decryption error) whose encryption algorithm is in NC^0. Hence by plugging this scheme to Construction 4.7, we obtain a computationally private, *statistically* correct randomized encoding in NC^0 for any polynomial-time computable function (and in particular derive Corollary 4.15). In fact, we can even obtain perfect correctness (as in Theorem 4.14) by assuming the existence of one-time symmetric encryption in $\mathrm{NL}/poly$ (which, using [IK02, AIK04], implies such an errorless scheme in NC^0). Note that (PRG $\in \oplus\mathrm{L}/poly$) $\implies$ (PRG $\in \mathrm{NL}/poly$) $\implies$ (one-time encryption $\in \mathrm{NL}/poly$), while the converse implications are not known to hold.

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. Recall that if the circuit computing f is an NC^i circuit and the decryption algorithm (used in the construction) is in NC^j, we obtain a parallel decoder in NC^{i+j} (see Remark 4.9). Unfortunately, we cannot use the parallel symmetric encryption scheme of Construction 4.3 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 [NR04].) In such a case, by Claim 4.6, 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. 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 4.11 or [IK02]). 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 [AIK04] or Lemma 5.2.) Since the symmetric encryption (E', D') employed by the information-theoretic garbled circuit is in NC^0, its decoder can be implemented in $\mathrm{NC}^1[D'] = \mathrm{NC}^1$. Thus, $\hat{D}$ is also in NC^1 (as $\mathrm{NC}^0[\mathrm{decoder}] = \mathrm{NC}^1$). Combining this encryption scheme with Construction 4.7, we get a computational encoding of a function $f \in \mathrm{NC}^i$ with encoding in NC^0 and decoding in NC^{i+1}. Assuming there exists a linear-stretch PRG in NC^1, we can use a similar argument to obtain an NC^0 encoding for f whose decoding in NC^{i+2}. (In this case we use the linear-PRG part of Claim 4.6.) Summarizing, we have the following:

Claim 4.17 *Suppose there exists a PRG with polynomial stretch (resp. linear stretch) in* NC^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} *(resp.* NC^{i+2}*).*

5 Applications

5.1 Relaxed Assumptions for Cryptography in NC^0

In [AIK04] 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) statistical or perfect 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 4.14 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,[13] commitments,[14] signatures, message authentication schemes (MACs), and zero knowledge proofs.[15] (See [Gol01, Gol04] 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 [AIK04] for the case of statistical encoding. For completeness, we provide a full treatment for encryption schemes and sketch the proofs for commitments, signatures, and zero-knowledge proofs. We also show how to use the EPRG assumption to construct an instance-hiding scheme between an NC^0 user and a BPP-oracle.

Definition 5.1 (Public-key encryption) *A* secure public-key encryption scheme *(PKE) is a triple (G, E, D) of probabilistic polynomial-time algorithms satisfying the following conditions:*

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

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

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

- ***Security**: For every polynomial $\ell(\cdot)$, and every families 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*

$$(e \leftarrow G_1(1^n), E(e, x_n)) \overset{c}{\equiv} (e \leftarrow G_1(1^n), E(e, x'_n)), \tag{5.1}$$

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

[13] See Footnote 8.

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

[15] In fact, computational randomized encoding also preserves the security of one-way functions. However, this fact has no application in the current context since OWFs are NC^0-reducible to PRGs.

The definition of a *private-key* encryption scheme is similar, except that the public key is omitted from the ensembles. That is, instead of Equation 5.1 we require that $E(G_1(1^n), x_n) \overset{c}{\equiv} E(G_1(1^n), x'_n)$. An extension to multiple-message security, where the indistinguishability requirement should hold for encryptions of polynomially many messages, follows naturally (see [Gol04, chapter 5] for formal definitions). In the public-key case, multiple-message security is implied by single-message security as defined above, whereas in the private-key case it is a strictly stronger notion. In the following we explicitly address only the (single-message) public-key case, but the treatment easily holds for the case of private-key encryption with multiple-message security.

Lemma 5.2 *Let $\mathcal{E} = (G, E, D)$ be a secure public-key encryption scheme, where $E(e, x, r)$ is viewed as a polynomial-time computable function that encrypts the message x using the key e and randomness r. Let $\hat{E}((e, x), (r, s)) = \hat{E}((e, x, r), s)$ be a computational randomized encoding of E and let $\hat{D}(d, \hat{y}) \overset{\text{def}}{=} D(d, B(\hat{y}))$ be the composition of D with the decoder B of $\hat{E}$. Then, the scheme $\hat{\mathcal{E}} \overset{\text{def}}{=} (G, \hat{E}, \hat{D})$ is also a secure public-key encryption scheme.*

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

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

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

We move on to prove the security of the construction. Let S be the efficient computational simulator of $\hat{E}$. Then, for every polynomial $\ell(\cdot)$, and every families 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

$$\begin{aligned}
(e \leftarrow G_1(1^n), \hat{E}(e, x_n, r_n, s_n)) &\overset{c}{\equiv} (e \leftarrow G_1(1^n), S(E(e, x_n, r_n))) && \text{(by the privacy of } \hat{E} \text{ and Fact 2.5)} \\
&\overset{c}{\equiv} (e \leftarrow G_1(1^n), S(E(e, x'_n, r_n))) && \text{(by the security of } \mathcal{E} \text{ and Fact 2.3)} \\
&\overset{c}{\equiv} (e \leftarrow G_1(1^n), \hat{E}(e, x'_n, r_n, s_n)) && \text{(by the privacy of } \hat{E} \text{ and Fact 2.5),}
\end{aligned}$$

where r_n and s_n are uniformly chosen random strings of an appropriate length. Hence, the security of $\hat{\mathcal{E}}$ follows from the transitivity of the relation $\overset{c}{\equiv}$ (Fact 2.1). ■

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

SIGNATURES. Let $\mathcal{S} = (G, S, V)$ be a signature scheme, where G is a key-generation algorithm that generates the signing and verification keys (s, v), the signing function $S(s, \alpha, r)$ computes a signature β on the document α using the key s and randomness r, and the verification algorithm $V(v, \alpha, \beta)$ verifies that β is a valid signature on α using the verification key v. The 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. Specifically, any polynomial-time adversary that gets the verification key and an oracle access to the signing process $S(s,\cdot)$ fails (except with negligible probability) 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.[16]

Let $\hat{S}$ be a computational randomized encoding of S, and let $\hat{V}(v,\alpha,\hat{\beta}) \stackrel{\text{def}}{=} V(v,\alpha,B(\hat{\beta}))$ be the composition of V with the decoder B of the encoding $\hat{S}$. We claim that the scheme $\hat{\mathcal{S}} \stackrel{\text{def}}{=} (G,\hat{S},\hat{V})$ is also a signature scheme. The efficiency and correctness of $\hat{\mathcal{S}}$ 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 $\hat{\mathcal{S}}$ into an attack on $\mathcal{S}$. Specifically, given an adversary $\hat{A}$ that breaks $\hat{\mathcal{S}}$, we can break $\mathcal{S}$ by invoking $\hat{A}$ and emulating the oracle $\hat{S}$ using the simulator of the encoding and the signature oracle S. By Fact 2.4 the output of $\hat{A}$ when interacting with the emulated oracle is computationally indistinguishable from $\hat{A}$'s output when interacting with the actual signing oracle $\hat{S}(s,\cdot)$. Moreover, if the forged signature $(\alpha,\hat{\beta})$ produced by $\hat{A}$ is valid under $\hat{\mathcal{S}}$, then it is translated into a valid signature (α,β) under $\mathcal{S}$ by using the decoder, i.e., $\beta = B(\hat{\beta})$. Hence, if the scheme $\hat{\mathcal{S}}$ can be broken with non-negligible probability, then so does the scheme $\mathcal{S}$. A similar argument holds also in the private-key setting (i.e., in the case of MACs).

COMMITMENTS. A commitment scheme enables one party (a sender) to commit itself to a value while keeping it secret from another party (the receiver). Later, the sender can reveal the committed value to the receiver, and it is guaranteed that the revealed value is equal to the one determined at the commit stage. We start with the simple case of a non-interactive commitment. Such a scheme can be defined by a polynomial-time computable function $C(b,r)$ 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 $C(b,r)$ is equal to the commitment c. The scheme should be both (computationally) hiding and (perfectly) binding. Hiding requires that $c = C(b,r)$ keeps b computationally secret, that is $C(0,U_n) \stackrel{c}{\equiv} C(1,U_n)$. 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 $C(0,r_0) = C(1,r_1)$.

Let $\hat{C}(b,r,s)$ be a perfectly-correct computationally-private encoding of $C(b,r)$. Then $\hat{C}$ 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 5.2. Namely, it holds that

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

where r and s are uniformly chosen strings of an appropriate length (the first and third transitions follow from the privacy of $\hat{C}$ and Fact 2.5, while the second transition follows from the hiding of C and Fact 2.3).[17] The binding property of $\hat{C}$ follows from the perfect correctness; namely, if there exists an ambiguous pair $(r_0,r'_0),(r_1,r'_1)$ such that $\hat{C}(0,r_0,s_0) = \hat{C}(1,r_1,s_1)$, then by perfect correctness it holds that $C(0,r_0) = C(1,r_1)$ which contradicts the binding of the original scheme. So when the encoding is in NC^0 we get a commitment scheme whose sender is in 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 $\hat{C}(b,r,s)$ and comparing the result to $\hat{c}$. Assuming $\hat{C}$ is in NC^0, the receiver can be implemented by an NC^0 circuit augmented with a single (unbounded fan-in) AND gate. We refer to this special type of AC^0 circuit as an

[16]When the signing algorithm is probabilistic, the attacker does not have an access to the random coin tosses of the signing algorithm.

[17]The resulting scheme is computationally hiding even if the original scheme is unconditionally hiding. This is contrast with the case of statistically-private encodings as discussed in [AIK04].

$\mathrm{AND}_n \circ \mathrm{NC}^0$ circuit.[18]

While the existence of non-interactive commitment schemes is implied by the existence of a 1-1 OWF ([Blu83], [Gol01, Const. 4.4.2]), it is not known how to construct such a scheme based on arbitrary OWF (or equivalently PRG). However, the existence of a OWF allows to construct an *interactive* commitment scheme [Nao91]. In particular, the PRG based commitment scheme of [Nao91] has the following simple form: First the receiver chooses a random string $r \in \{0,1\}^{3n}$ and sends it to the sender, then the sender that wishes to commit to the bit b chooses a random string $s \in \{0,1\}^n$ and sends the value of the function $C(b,r,s)$ to the receiver. (The exact definition of the function $C(b,r,s)$ is not important in our context.) To decommit the sender sends the randomness s and the bit b and the receiver accepts if $C(b,r,s)$ equals to the message he had received in the commit phase. Computational hiding requires that for any choice of r it holds that $(r, C(0,r,s)) \overset{c}{\equiv} (r, C(1,r,s))$. Perfect binding requires that, except with negligible probability (over the randomness of the receiver r), there are no s_0 and s_1 such that $C(0,r,s_0) = C(1,r,s_1)$.

Again, if we replace C by a computationally-private perfectly-correct encoding $\hat{C}$, we get a (constant-round) interactive commitment scheme (this follows by combining the previous arguments with Fact 2.5). Moreover, as in the non-interactive case, when the encoding is in NC^0 the receiver's computation in the decommit phase is in $\mathrm{AND}_n \circ \mathrm{NC}^0$, and so, since the receiver's computation in the commit phase is in NC^0, we get an $\mathrm{AND}_n \circ \mathrm{NC}^0$ receiver. As an immediate application, we obtain a constant-round protocol for coin flipping over the phone [Blu83] between an NC^0 circuit and an $\mathrm{AND}_n \circ \mathrm{NC}^0$ circuit under the EPRG assumption.

ZERO-KNOWLEDGE PROOFS. We move on to the 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 $\Pr[V(x,\sigma,P(x,w,\sigma)) = 1] > \frac{2}{3}$;
- (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 $\Pr[V(x,\sigma,P^*(x,\sigma)) = 1] < \frac{1}{3}$;
- (Zero-knowledge) there exists a probabilistic polynomial-time simulator M such that for every string sequence $\{(x_n,w_n)\}$ where $(x_n,w_n) \in R$ it holds that $\{(x_n,\sigma,P(x_n,w_n,\sigma))\} \overset{c}{\equiv} \{M(x_n)\}$;

(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 2.5 and the privacy of $\hat{P}$ it holds that

[18]There is no commitment scheme in which the receiver's computation is in NC^0. Such a receiver decides whether to accept the opening of the commitment according to a constant number of bits, and hence can guess these bits with constant probability which allows him to open the commitment before the decommit phase. Thus, the locality of the function that the receiver computes has to be super-logarithmic. Hence, an $\mathrm{AND}_n \circ \mathrm{NC}^0$ receiver can be considered as the best one can achieve.

$(x,\sigma,\hat{P}(x,w,\sigma,r)) \overset{c}{\equiv} (x,\sigma,S(P(x,w,\sigma)))$ (where r is the randomness of the encoding $\hat{P}$) while Fact 2.3 ensures that $(x,\sigma,S(P(x,w,\sigma))) \overset{c}{\equiv} \hat{M}(x)$.[19]

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 his 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 2.4 instead of Fact 2.3.) The same construction works for ZK arguments (in which the soundness holds only against computationally bounded cheating prover).

INSTANCE HIDING. 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 [Fei93] for a survey on IHS schemes.)

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 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 2.5, it holds that

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

Hence, under the EPRG assumption, we can construct such a scheme where $\mathsf{WEAK} = \mathrm{NC}^0$ and $\mathsf{STRONG} = \mathrm{BPP}$ (or P if perfect correctness is required).

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

Theorem 5.3 *Suppose that the EPRG assumption holds. Then,*

1. *If there exists a public-key encryption scheme (resp., NIZK with an efficient prover or constant-round ZK proof with an efficient prover for every* NP *relation), then there exists such a scheme in which the encryption (prover) algorithm is in* NC^0_4.

2. *If there exists a non-interactive commitment scheme, then there exists such a scheme in which the sender is in* NC^0_4 *and the receiver is in* $\mathrm{AND}_n \circ \mathrm{NC}^0$.

3. *There exists a stateless symmetric encryption scheme (resp., digital signature, MAC, a constant-round ZK argument for every language in* NP*) in which the encryption (signing, prover) algorithm is in* NC^0_4.

[19]This transformation results in a computational ZK even if the original protocol was a statistical ZK proof system. Again, this is contrasted with the transformations obtained by using *statistically* private randomized encoding described in [AIK04].

4. *There exists a constant-round commitment scheme in which the sender is in* NC^0_4 *and the receiver is in* $\mathrm{AND}_n \circ \mathrm{NC}^0$.

5. *For every polynomial-time computable function we have a (non-adaptive single-oracle) IHS in which the user is in* NC^0_5 *and the oracle is in* BPP.

Note that the existence of (stateless) symmetric encryption, signature, MAC, constant-round commitment scheme and constant-round ZK arguments for NP, 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 [Gol01, Gol04]). For these primitives, we obtain more general (unconditional) results in the next subsection.

Theorem 5.3 reveals an interesting phenomenon. It appears that several cryptographic primitives (e.g., symmetric encryption schemes, digital signatures and MACs) can be implemented in NC^0 despite the fact that their standard constructions rely on pseudorandom functions (PRFs) [GGM86], which cannot be computed even in AC^0 [LMN93]. For such primitives, we actually construct a sequential PRF from the PRG (as in [GGM86]), 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 4.14 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 4.17 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.[20]

Theorem 5.4 *Let* $\mathcal{X} = (G, S, R)$ *be a sender-receiver cryptographic scheme whose security is respected by computational encoding (e.g., encryption, signature, MAC, commitment scheme, NIZK), where* G *is a key-generation algorithm (in case the scheme has one),* $S \in \mathrm{NC}^s$ *is the algorithm of the sender and* $R \in \mathrm{NC}^r$ *is the algorithm of the receiver. Then,*

- *If there exists a polynomial-stretch PRG in* NC^1*, then there exists a similar scheme* $\hat{\mathcal{X}} = (G, \hat{S}, \hat{R})$ *in which* $\hat{S} \in \mathrm{NC}^0$ *and* $\hat{R} \in \mathrm{NC}^{\max\{s+1,r\}}$.
- *If there exists a linear-stretch PRG in* NC^1*, then there exists a a similar scheme* $\hat{\mathcal{X}} = (G, \hat{S}, \hat{R})$*, in which* $\hat{S} \in \mathrm{NC}^0$ *and* $\hat{R} \in \mathrm{NC}^{\max\{s+2,r\}}$.

Proof: If there exists a polynomial-stretch (resp. linear-stretch) PRG in NC^1, then we can use Claim 4.17 and get a computational encoding $\hat{S}$ for S in NC^0 whose decoder B is in NC^{s+1} (resp. NC^{s+2}). 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 $\mathrm{NC}^{\max\{s+1,r\}}$ (resp. $\mathrm{NC}^{\max\{s+2,r\}}$). ■

[20]Similarly, assuming a linear-stretch PRG in NC^1, we can obtain, for every NC function, a (non-adaptive single-oracle) IHS in which the user is in NC^0 and the oracle is in NC.

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 [HILL99], 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 4.12 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 5.5 *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 more general types of one-way functions (see [HILL99, HHR05] and [AIK04, Remark 6.6]), 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 5.6 *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 constant-round commitment scheme (resp., constant-round coin-flipping protocol) in which the sender (first party) is in $\mathrm{NC}^0[G]$ and the receiver (second party) is in $\mathrm{AND}_n \circ \mathrm{NC}^0[G]$.*
- *For every NP language, there exists a constant-round ZK argument in which the prover is in $\mathrm{NC}^0[G]$.*

Note that items 3,4 of Theorem 5.3 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 [Nao91] 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, $\mathrm{AND}_n \circ \mathrm{NC}^0[G]$) where G is a *minimal* PRG. Since it is not known how to reduce a linear-stretch PRG to a minimal PRG even in NC, 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 [GGM86, NR99, Gol04] for these and other applications of PRFs). Since the known construction of a PRF from a PRG is sequential [GGM86], 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 [NR99, NR04]. In particular, Naor and Reingold [NR99] 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 5.6, 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 [NR99] yield NC^1-reductions on both ends.

In contrast to the above, we show that a synthesizer in NC^i can be used to implement encryption in NC^i with decryption in NC.[21] First, we use [NR99] to construct an encryption scheme (E, D) and a polynomial-stretch PRG G such that E and D are in $\mathrm{NC}^1[\mathrm{SYNTH}] = \mathrm{NC}^{i+1}$ and G is in $\mathrm{NC}^0[\mathrm{SYNTH}] = \mathrm{NC}^i$. Next, by Claim 4.6 and Remark 4.9, we obtain an $\mathrm{NC}^0[G] = \mathrm{NC}^i$ computational encoding $\hat{E}$ for E whose decoder B is in NC^{2i}. (We first use Claim 4.6 to construct one-time symmetric encryption (OE, OD) such that OE and OD are in $\mathrm{NC}^0[G] = \mathrm{NC}^i$. Then, we encode E by plugging OE into Construction 4.7 and obtain an $\mathrm{NC}^0[OE] = \mathrm{NC}^i$ computational encoding $\hat{E}$ for E. By Remark 4.9 the decoder B of $\hat{E}$ is in $\mathrm{NC}^i[OD] = \mathrm{NC}^{2i}$.) To decrypt ciphers of $\hat{E}$ we invoke the decoder B, and then apply the original decryption algorithm D to the result. Therefore, the decryption algorithm of our new scheme $\hat{D}$ is in $\mathrm{NC}^{\max(2i,i+1)}$.

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., [Can00, Can01, Gol04], 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 may be 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 [IK00] was that the round complexity of most general

[21]For concreteness, we refer here only to the case of symmetric encryption, the case of other primitives which are NC^0-reducible to PRF (such as identification schemes and MACs) is analogous.

protocols from the literature (e.g., those of [GMW87, BGW88, CCD88]) is related to the algebraic *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 $\hat{f}'(x, r^1, \ldots, r^{t+1}) \stackrel{\text{def}}{=} p(x, r^1 \oplus \ldots \oplus r^{t+1})$ (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 [IK00] 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 [IK00, IK02, CFIK03] 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.

To formulate the above we need the following definitions.

Definition 5.7 (Secure computation) *Let $f(x_1, \ldots, x_n)$ be an m-party functionality, i.e., a (possibly randomized) mapping from m inputs of equal length into m outputs. Let π be an m-party protocol. We formulate the requirement that π securely computes f by comparing the following "real process" and "ideal process".*

The real process. *A t*-bounded *adversary A attacking the* real process *is a probabilistic polynomial-time algorithm, who may corrupt up to t parties and observe all of their internal data. At the end of the interaction, the adversary may output an arbitrary function of its view, which consists of the inputs, the random coin tosses, and the incoming messages of the corrupted parties. We distinguish between* passive *vs.* active *adversaries and between* adaptive *vs.* non-adaptive *adversaries. If the adversary is active, it has full control over the messages sent by the corrupted parties, whereas if it is passive, it follows the protocol's instructions (but may try to deduce information by performing computations on observed data). When the set of corrupted parties has to be chosen in advance, we say that the adversary is non-adaptive, and otherwise say that it is adaptive. Given an m-tuple of inputs $(x_1, \ldots, x_m) \in (\{0,1\}^n)^m$, the* output of the real process *is defined as the random variable containing the* concatenation *of the adversary's output with the outputs and identities of the uncorrupted parties. We denote this output by $\textsc{Real}_{\pi,A}(x_1, \ldots, x_m)$.*

The ideal process. *In the ideal process, an incorruptible trusted party is employed for computing the given functionality. That is, the "protocol" in the ideal process instructs each party to send its input to the trusted party, who computes the functionality f and sends to each party its output. The interaction of a*

t-bounded adversary A' with the ideal process and the output of the ideal process are defined analogously to the above definitions for the real process. The adversary attacking the ideal process will also be referred to as a simulator. *We denote the output of the ideal process on the inputs $(x_1, \ldots, x_m) \in (\{0,1\}^n)^m$ by $\textsc{Ideal}_{f,A'}(x_1, \ldots, x_m)$.*

*The protocol π is said to t-*securely *realize the given functionality f with respect to a specified type of adversary (namely, passive or active, adaptive or non-adaptive) if for any probabilistic polynomial-time t-bounded adversary A attacking the real process, there exists a probabilistic polynomial-time t-bounded simulator A' attacking the ideal process, such that for any sequence of m-tuples $\{\bar{x}_n\}$ such that $\bar{x}_n \in (\{0,1\}^n)^m$, it holds that $\textsc{Real}_{\pi,A}(\bar{x}_n) \overset{c}{\equiv} \textsc{Ideal}_{f,S}(\bar{x}_n)$.*

Secure reductions. To define secure reductions, consider the following *hybrid* model. An m-party protocol augmented with an oracle to the m-party functionality g is a standard protocol in which the parties are allowed to invoke g, i.e., a trusted party to which they can securely send inputs and receive the corresponding outputs. The notion of t-security generalizes to protocols augmented with an oracle in the natural way.

Definition 5.8 *Let f and g be m-party functionalities. A t-*secure *reduction from f to g is an m-party protocol that given an oracle access to the functionality g, t-securely realizes the functionality f (with respect to a specified type of adversary). We say that the reduction is* non-interactive *if it involves a single call to f (and possibly local computations on inputs and outputs), but no further communication.*

Appropriate composition theorems, (e.g. [Gol04, Thms. 7.3.3, 7.4.3]), guarantee that the call to g can be replaced by any secure protocol realizing g, without violating the security of the high-level protocol for f.[22] Using the above terminology, Theorem 4.14 has the following corollary.

Theorem 5.9 *Suppose the EPRG assumption holds. Let $f(x_1, \ldots, x_m)$ 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 $\hat{f}(x_1, \ldots, x_m)$ 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 $\hat{f}'(x'_1, \ldots, x'_m)$ 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.

Proof: The second item follows from the first via standard (non-interactive, degree-preserving) secure reduction from randomized functionalities to deterministic functionalities (see [Gol04, Prop. 7.3.4]). Thus we will only prove the first item. Assume, without loss of generality, that f is a deterministic functionality that returns the same output to all the parties.[23] Let $\hat{f}(x, r)$ be the computational encoding of $f(x)$ promised

[22]Actually, for the composition theorem to go through, Definition 5.7 should be augmented by providing players and adversaries with auxiliary inputs. We ignore this technicality here, and note that the results in this section apply (with essentially the same proofs) to the augmented model as well.

[23]To handle randomized functionalities we use the non-interactive secure reduction mentioned above. Now, we can $(m-1)$-securely reduce f to a single-output functionality by letting each party to mask its output f_i with a private randomness. That is, $f'((x_1, r_1) \ldots, (x_m, r_m)) = ((f_1(x_1) \oplus r_1) \circ \ldots \circ (f_1(x_m) \oplus r_m))$. As both reductions are non-interactive the resulting reduction is also non-interactive. Moreover, the circuit size of f' is linear in the size of the circuit that computes the original function.

by Theorem 4.14. (Recall that $\hat{f}$ is indeed a degree 3 function having $O(s(n) \cdot n^{\epsilon})$ random inputs and outputs.) The following protocol $(m-1)$-securely reduces the computation of $f(x)$ to $\hat{f}(x,r)$ (where $\hat{f}$ is viewed as a randomized functionality whose randomness is r).

- Inputs: Party i gets input $x_i \in \{0,1\}^n$.
- Party i invokes the (randomized) oracle $\hat{f}$ with query x_i, and receives an output $\hat{y}$.
- Outputs: Each party locally applies the decoder B of the encoding to the answer $\hat{y}$ received from the oracle, and outputs the result.

We start by showing that the reduction is $(m-1)$-secure against a passive non-adaptive adversary. Let A be such an adversary that attacks some set $I \subset [m]$ of the players. Then, the output of the real process is $(A(x_I, \hat{f}(x,r)), B(\hat{f}(x,r)), \bar{I})$ where $x_I = (x_i)_{i \in I}$, $\bar{I} \overset{\text{def}}{=} [m] \setminus I$ and r is a uniformly chosen string of an appropriate length. We define a (passive non-adaptive) simulator A' that attacks the ideal process in the natural way: that is, $A'(x_I, y) = A(x_I, S(y))$, where y is the answer received from the trusted party (i.e., $f(x)$) and S is the computationally private simulator of the encoding. Thus, the output of the ideal process is $(A'(x_I, f(x)), f(x), \bar{I})$. By the definition of A', the privacy of the encoding $\hat{f}$ and Fact 2.3, we have,

$$\textsc{Ideal}(x) \equiv (A(x_I, S(f(x))), f(x), \bar{I}) \overset{c}{\equiv} (A(x_I, \hat{f}(x,r)), B(\hat{f}(x,r)), \bar{I}) \equiv \textsc{Real}(x),$$

which finishes the proof.

We now sketch the security proof for the case of an adversary A which is both adaptive and active. (The non-adaptive active case as well as the adaptive passive case are treated similarly.) An attack by A has the following form: (1) Before calling the oracle $\hat{f}$, in each step A may decide (according to his current view) to corrupt some party i and learn its input x_i. (2) When the oracle $\hat{f}$ is invoked A changes the input of each corrupted party i to some value x'_i, which is handed to the $\hat{f}$ oracle. (3) After the parties call the oracle on some (partially corrupted) input $x' = (x'_I, x_{\bar{I}})$, the oracle returns a randomized encoding $\hat{f}(x')$ to the adversary, and now A may adaptively corrupt additional parties. Finally, A outputs some function of its entire view. For every such adversary we construct a simulator A' that attacks the ideal process by invoking A and emulating his interaction with the real process. Namely, (1) Before the call to the trusted party we let A choose (in each step) which party to corrupt and feed it with the input we learn; (2) When the trusted party is invoked we let A pick x'_I according to its current view and send these x'_I to the f oracle; (3) Given the result $y = f(x'_I, x_{\bar{I}})$ returned by the oracle, we invoke the simulator (of the encoding) on y and feed the result to A. Finally, we let A pick new corrupted players as in step (1). We claim that in each step of the protocol the view of A when interacting with the real process is computationally indistinguishable from the view of A when it is invoked by A' in the ideal process. (In fact, before the call to the oracle these views are identically distributed.) Hence, the outputs of the two processes are also computationally indistinguishable. ■

A high-level corollary of Theorem 5.9 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 5.9 gives rise to new, conceptually simpler, constant-round protocols for general functionalities. For instance, a combination of this result with the "BGW protocol" [BGW88] gives a simpler alternative to the constant-round protocol of Beaver, Micali, and Rogaway [BMR90]. 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 [BMR90]. On the downside, Theorem 5.9 relies on a stronger assumption than the protocol from [BMR90] (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 [CFIK03]).

Acknowledgment. We thank Omer Reingold for helpful discussions.

References

[AIK04] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in NC^0. *SIAM J. Comput.* To appear. Preliminary version in FOCS 04.

[Blu83] M. Blum. Coin flipping by telephone: a protocol for solving impossible problems. *SIGACT News*, 15(1):23–27, 1983.

[BM84] 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.

[BMR90] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *Proc. of 22nd STOC*, pages 503–513, 1990.

[BGW88] 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.

[Can00] R. Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd FOCS*, pages 136–145, 2001.

[CCD88] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proc. of 20th STOC*, pages 11–19, 1988.

[CFIK03] R. Cramer, S. Fehr, Y. Ishai, and E. Kushilevitz. Efficient multi-party computation over rings. In *Proc. EUROCRYPT '03*, pages 596–613, 2003.

[Fei93] J. Feigenbaum. Locally random reductions in interactive complexity theory. In *Advances in Computational Complexity Theory*, volume 13 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages pp. 73–98, 1993.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. of the ACM.*, 33:792–807, 1986.

[GL89] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proc. 21st STOC*, pages 25–32, 1989.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, 1984. Preliminary version in Proc. STOC ’82.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game (extended abstract). In *Proc. of 19th STOC*, pages 218–229, 1987.

[Gol01] O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001.

[Gol04] O. Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[HHR05] I. Haitner, D. Harnik, and O. Reingold. On the power of the randomized iterate. *Electronic Colloquium on Computational Complexity (ECCC)*, (135), 2005.

[HILL99] 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.

[IK00] 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.

[IK02] Y. Ishai and E. Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *Proc. 29th ICALP*, pages 244–256, 2002.

[LMN93] 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.

[LP04] Y. Lindell and B. Pinkas. A proof of yao’s protocol for secure two-party computation. *Electronic Colloquium on Computational Complexity*, 11(063), 2004.

[Nao91] M. Naor. Bit commitment using pseudorandomness. *J. of Cryptology*, 4:151–158, 1991.

[NPS99] 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.

[NR99] 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.

[NR04] 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.

[Rog91] P. Rogaway. *The Round Complexity of Secure Protocols.* PhD thesis, MIT, June 1991.

[RTV04] O. Reingold, L. Trevisan, and S. Vadhan. Notions of reducibility between cryptographic primitives. In *TCC ’04*, volume 2951 of *LNCS*, pages 1–20, 2004.

[Sha49] C. E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28-4:656–715, 1949.

[TX03] 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.

[Vio05] E. Viola. On constructing parallel pseudorandom generators from one-way functions. In *Proc. 20th Conference on Computational Complexity (CCC)*, pages 183– 197, 2005.

[Wig94] A. Wigderson. $\mathrm{NL}/poly \subseteq \oplus\mathrm{L}/poly$. In *Proc. 9th Structure in Complexity Theory Conference*, pages 59–62, 1994.

[Yao82] A. C. Yao. Theory and application of trapdoor functions. In *Proc. 23rd FOCS*, pages 80–91, 1982.

[Yao86] A. C. Yao. How to generate and exchange secrets. In *Proc. 27th FOCS*, pages 162–167, 1986.

A Proof of Lemma 4.10

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, an active key and a color; namely, S selects a random string $W_i^{b_i}$ of length $2k(n)$, and a random bit c_i. (Recall that b_i denotes the value of the i-th wire induced by the input x. The simulator, of course, does not know this value.) 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 active 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 inactive 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 $p(\cdot)$, a string family $\{x_n\}$, $|x_n| = n$, such that for infinitely many n's it holds that

$$\Delta(n) \stackrel{\text{def}}{=} |\Pr[A_n(S(1^n, f_n(x_n))) = 1] - \Pr[A_n(\hat{f}_n(x_n, U_{\mu(n)})) = 1]| > \frac{1}{p(n)}.$$

We use a hybrid argument to show that such a distinguisher can be used to break the encryption hence deriving a contradiction.

From now on we fix n and let $k = k(n)$. 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). Hence, if the extreme hybrids can be efficiently distinguished then there must be two neighboring hybrids that can be efficiently distinguished.

The hybrids H_t^n. First, 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 $2k$-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 $r_i = c_i - b_i$.

Claim A.1 *There exist some* $0 \leq t \leq \Gamma(n) - 1$ *such that* A_n *distinguishes between* H_t^n *and* H_{t+1}^n *with advantage* $\frac{\Delta(n)}{\Gamma(n)}$.

Proof: First, note that H_t^n 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 $H_0^n \equiv \hat{f}_n(x_n, U_{\mu(n)})$.

Hence, we can write

$$\Delta(n) = |\Pr[A_n(H_0^n) = 1] - \Pr[A_n(H_{\Gamma(n)}^n) = 1]| \leq \sum_{t=0}^{\Gamma(n)-1} |\Pr[A_n(H_t^n) = 1] - \Pr[A_n(H_{t+1}^n) = 1]|,$$

and so there exists some $0 \leq t \leq \Gamma(n) - 1$ such that A_n distinguishes between H_t^n and H_{t+1}^n with advantage $\frac{\Delta(n)}{\Gamma(n)}$. ■

We now show that distinguishing H_t^n and H_{t+1}^n allows to distinguish whether a single gate is real or fake. To do this, we define two random experiments $P_n(0)$ and $P_n(1)$, that produce a real gate and a fake gate, correspondingly.

The experiments $P_n(0), P_n(1)$**.** 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 $P_n(\sigma)$ as the output distribution of the following random process:

- Uniformly choose the $2k$-bit strings $W_i^{b_i}, W_i^{1-b_i}, W_j^{b_j}, W_j^{1-b_j}, W_{y_1}^{b_{y_1}}, W_{y_1}^{1-b_{y_1}}, \ldots, W_{y_m}^{b_{y_m}}, W_{y_m}^{1-b_{y_m}}$, 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 set $Q_t^{a_i,a_j}$ exactly as in Step 4 of H_t^n; that is, set $r_i = c_i - b_i$, $r_j = c_j - b_j$, and for every choice of $(a_i, a_j) \in \{0,1\}^2$, let

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

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

Claim A.2 *There exist a polynomial size circuit A'_n that distinguishes between $P_n(0)$ and $P_n(1)$ with advantage $\frac{\Delta(n)}{\Gamma(n)}$.*

Proof: The adversary A'_n uses the output of $P_n(\sigma)$ to construct one of the hybrids H_t^n and H_{t+1}^n, and then uses A_n to distinguish between them. Namely, given the output of P_n, the distinguisher A'_n invokes 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. By the definition of P_n, when $P_n(0)$ 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. Hence, by Claim A.1, A'_n has the desired advantage. Finally, since H_t^n runs in polynomial time (when x_n is given), the size of A'_n is indeed polynomial. ■

A delicate point. Note that P_n does not output the inactive keys of the wires i and j (which is crucial for Claim A.3 to hold). However, the hybrid distributions use inactive keys of wires that either enter a real gate or leave a real gate (in the first case the inactive keys are used as the keys of the gate label encryption whereas in the latter case, the inactive keys are being encrypted). Hence, we do not need these inactive keys to construct the rest of the distribution H_t^n (or H_{t+1}^n), as i and j are output wires of gates that precedes t and therefore are "fake" gates. This is the reason for which we had to sort the gates. On the other hand, the process P_n must output the inactive keys of the output wires of the gate $y_1, \ldots, y_m$, since these wires might enter as inputs to another gate $t' > t$ which is a "real" gate in both H_t^n and H_{t+1}^n.

We now define a related experiment P'_n in which some of the randomness used by P_n is fixed. Specifically, we fix the random strings $W_{y_1}^{b_{y_1}}, W_{y_1}^{1-b_{y_1}}, \ldots, W_{y_m}^{b_{y_m}}, W_{y_m}^{1-b_{y_m}}, c_i, c_j, c_{y_1}, \ldots, c_{y_m}$ to some value such that the resulting experiments still can be distinguished by A'_n with advantage $\frac{\Delta(n)}{\Gamma(n)}$. (The existence of such strings is promised by an averaging argument.) For simplicity, we omit the fixed strings from the output of this new experiment. The experiments $P'_n(0)$ and $P'_n(1)$ can still be distinguished by some polynomial size circuit with advantage $\frac{\Delta(n)}{\Gamma(n)}$. (Such distinguisher can be constructed by incorporating the omitted fixed strings into A'_n.) Hence, by the contradiction hypothesis, it follows that this advantage is greater than $\frac{1}{\Gamma(n)p(n)}$ for infinitely many n's. As $\Gamma(n)$ is polynomial in n (since C_n is of polynomial size) we deduce that the distribution ensembles $\{P'_n(0)\}_{n\in\mathbb{N}}$ and $\{P'_n(1)\}_{n\in\mathbb{N}}$ are not computationally indistinguishable, in contradiction with the following claim.

Claim A.3 $\{P'_n(0)\}_{n\in\mathbb{N}} \overset{c}{\equiv} \{P'_n(1)\}_{n\in\mathbb{N}}$.

Proof: Fix some n. For both distributions $P'_n(0)$ and $P'_n(1)$, the first two entries (i.e., $W_i^{b_i}, W_j^{b_j}$) are two uniformly and independently $2k(n)$-length strings, and the active label $Q_t^{b_i,b_j}$ is a function of $W_i^{b_i}, W_j^{b_j}$ and the fixed strings. Hence, the distributions $P'_n(0)$ and $P'_n(1)$ differ only in the inactive 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 $W_i^{a_i-r_i,a_j}, W_j^{a_j-r_j,a_i}$ 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 distributed uniformly and independently of the other entries of $P'_n(1)$'s output. So all the entries of both $P'_n(1)$ and $P'_n(0)$ are independent. Moreover, the security of the encryption scheme implies that the ensemble $\{Q_t^{a_i,a_j}\}$ for $(a_i, a_j) \neq (c_i, c_j)$ produced by $P'_n(1)$ is computationally indistinguishable from the corresponding ensemble produced by $P'_n(0)$, as in both cases some $p(n)$-length message is encrypted under uniformly chosen $k(n)$-length key. (Recall that $k(n)$ is polynomial in n by definition, and $p(n) = O(|C_n|k(n)) = \mathrm{poly}(n)$). Hence, by Fact 2.2, the proof follows. ■