Fooling Parity Tests with Parity Gates

Dan Gutfreund[1,⋆] and Emanuele Viola[2,⋆⋆]

[1] School of Computer Science and Engineering, The Hebrew University of Jerusalem, Israel, 91904, danig@cs.huji.ac.il

[2] Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138, viola@eecs.harvard.edu

Abstract. We study the complexity of computing k-wise independent and ϵ-biased generators $G : \{0,1\}^n \to \{0,1\}^m$. Specifically, we refer to the complexity of computing G *explicitly*, i.e. given $x \in \{0,1\}^n$ and $i \in \{0,1\}^{\log m}$, computing the i-th output bit of $G(x)$. [MNT90] show that constant depth circuits of size poly(n) (i.e. AC^0) cannot explicitly compute k-wise independent and ϵ-biased generators with seed length $n = 2^{\log^{o(1)} m}$.

In this work we show that DLOGTIME-uniform constant depth circuits of size poly(n) *with parity gates* can explicitly compute k-wise independent and ϵ-biased generators with seed length $n = (\text{poly}) \log m$. In some cases the seed length of our generators is optimal up to constant factors, and in general up to polynomial factors. To obtain our results, we show a new construction of combinatorial designs, and we also show how to compute, in DLOGTIME-uniform AC^0, random walks of length $\log^c n$ over certain expander graphs of size 2^n.

1 Introduction

The notion of *pseudorandom generators* (PRGs) is central to the fields of Computational Complexity and Cryptography. Informally, a PRG is an efficient deterministic procedure that maps a short *seed* to a long output, such that certain tests are *fooled* by the PRG, i.e. they cannot distinguish between the output of the PRG (over a random input) and the uniform distribution. More formally, a PRG $G : \{0,1\}^n \to \{0,1\}^m$ ϵ-fools a test $M : \{0,1\}^m \to \{0,1\}$ if

$$\left| \Pr_{y \in \{0,1\}^m}[M(y) = 1] - \Pr_{x \in \{0,1\}^n}[M(G(x)) = 1] \right| \leq \epsilon.$$

One popular class of tests are *parity* ones (sometimes called linear tests). These are the tests that, given $z \in \{0,1\}^m$, only output the XOR of a fixed subset of the bits of z. A generator is called ϵ*-biased* if it ϵ-fools such tests [NN90]. Another popular type of tests are those whose value, given $z \in \{0,1\}^m$, depends

⋆ Research supported in part by the Leibniz Center, the Israel Foundation of Science, a US-Israel Binational research grant, and an EU Information Technologies grant (IST-FP5).

⋆⋆ Research supported by NSF grant CCR-0133096 and US-Israel BSF grant 2002246.

only on k fixed bits of z, but the dependence on those bits is arbitrary. A generator is called (ϵ, k)*-wise independent* if it ϵ-fools such tests. An important special case of (ϵ, k)-wise independent generators are those where $\epsilon = 0$, i.e. every k fixed output bits of the generator are uniform over $\{0,1\}^k$. Such generators are called k*-wise independent.* ϵ-biased and k-wise independent generators have found several applications in Complexity Theory and Cryptography. For a discussion of these generators we refer the reader to the excellent book by Goldreich [Gol99].

In this paper we study the following general question: *what are the minimal computational resources needed to compute k-wise and ϵ-biased generators?* Throughout this work, when we refer to the complexity of a generator G, we actually refer to the complexity of computing the i-th bit of $G(x)$ given a seed $x \in \{0,1\}^n$ and an index $i \in \{0,1\}^{\log m}$. This is a more refined notion of complexity (compared to the complexity of computing $G(x)$, which is at least linear in the output length m), and it is especially adequate for generators having logarithmic seed length (i.e. $n \approx \log m$), as is the case with k-wise independent and ϵ-biased generators (see below). Generators for which we can efficiently compute the i-th output bit of $G(x)$, given $x \in \{0,1\}^n$ and $i \in \{0,1\}^{\log m}$, are called *explicit*. Explicitness plays a crucial role in many applications where only some portion of $G(x)$ is needed at each time, and the application runs in time polynomial in the *seed length* of the generator (rather than in its output length). These applications include constructions of universal hash functions, lower bounds proofs [Kab03], resource-bounded measure theory [CSS97], and more.

k-wise independent and ϵ-biased generators, $G : \{0,1\}^n \to \{0,1\}^m$, explicitly computable in time $\mathrm{poly}(n)$ are known with seed length logarithmic in the output length, i.e. $n = O(\log m)$ [CG89,ABI86,NN90,AGHP92] (for constant k and ϵ. Later we will be more precise about the dependence on k and ϵ). On the other hand, Mansour, Nisan and Tiwari [MNT90] showed[3] that any k-wise independent or ϵ-biased generator $G : \{0,1\}^n \to \{0,1\}^m$ explicitly computable by constant depth circuits (AC^0) of size $\mathrm{poly}(n)$ must have seed length $n \geq 2^{\log^{\Omega(1)} m} \gg \mathrm{poly}\log m$.

In this paper we ask the following question: Are there k-wise independent and ϵ-biased generators, $G : \{0,1\}^n \to \{0,1\}^m$, with seed length $n = O(\log m)$ (or $n = \mathrm{poly}\log m$), explicitly computable by constant depth circuits *with parity gates* (denoted $AC^0[\oplus]$) of size $\mathrm{poly}(n)$? (Recall that parity is the function $\bigoplus(x) := \sum_i x_i \bmod 2$).

We give an affirmative answer by showing several constructions of such generators explicitly computed by $AC^0[\oplus]$ circuits of size $\mathrm{poly}(n)$ (our results are discussed below). All our circuits are DLOGTIME-uniform. Informally this means that each gate (resp. edge) in the circuit can be specified in time linear *in the name* of the gate (resp. edge). DLOGTIME-uniformity is the strongest notion of uniformity found generally applicable, and gives nice characterizations of $AC^0, AC^0[\oplus]$ [BIS90]. *In this paper whenever we say that a circuit class is uniform we always mean that it is DLOGTIME-uniform.* Note that uniform $AC^0[\oplus]$

[3] In [MNT90] this negative result is stated for hash functions only, but their techniques apply to k-wise independent and ϵ-biased generators as well.

is strictly contained in $L :=$ logarithmic space. Therefore all our generators are in particular explicitly computable in space $O(\log n)$. For background on circuit complexity we refer the reader to the excellent book by Vollmer [Vol99].

Recently there has been some work on constructing k-wise independent and ϵ-biased generators in the very restricted class NC^0 (i.e. each output bit of the generator depends on a constant number of input bits) [CM01,MST03,AIK04]. However, it is not difficult to see that the seed length of these generators must be $n = m^{\Omega(1)}$, which is exponentially bigger than the seed length of the constructions that we present here.

Generators : $\{0,1\}^n \to \{0,1\}^m$ explicitly computable by DLOGTIME-uniform $AC^0[\oplus]$ circuits of size $\mathrm{poly}(n)$			
Seed length	Type	Limitations	Reference
$n = O(\log m)$	k-wise	$k = O(1)$	Theorem 10 (1)
$n = O(k^2 \log(m) \log(k \cdot \log m))$	k-wise	-	Theorem 10 (2)
$n = O(\log m + \log 1/\epsilon)$	ϵ-biased	$\epsilon \geq 1/2^{\mathrm{poly}\log\log(m)}$	Theorem 12 (1)
$n = O(\log m \cdot \log 1/\epsilon)$	ϵ-biased	-	Theorem 12 (2)

Table 1. Our main results.

1.1 Our results

Our main results are summarized in Table 1. We now discuss some of the generators in Table 1.

k-wise independent generators in $AC^0[\oplus]$: k-wise independent generators explicitly computable in time $\mathrm{poly}(n)$ are known with seed length $n = O(k \log m)$ [CG89,ABI86] which is optimal up to constant factors (a lower bound of $(k \log m)/2$ was proven in [CGH+85]).

To understand the difficulty of implementing these generators in $AC^0[\oplus]$ let us discuss a construction from [CG89,ABI86]. For simplicity of exposition, assume that $m = 2^h$ for some integer h. Let $GF(2^h)$ be the field of size 2^h, then for every k, the generator $G : \{0,1\}^n \to \{0,1\}^m$ defined as

$$G(a_0, a_1, \dots, a_{k-1})_i := \sum_{j<k} a_j i^j, \text{ where } a_0, a_1, \dots, a_{k-1}, i \in GF(2^h), \quad (1)$$

is a k-wise independent generator[4] with seed length $n = kh = k \log m$.

For concreteness, in the following discussion let us fix k at most $\mathrm{poly}(h)$, so that $\mathrm{poly}(n) = \mathrm{poly}(h)$. To compute the generator in Equation 1 we must solve two problems: (i) we must find a representation of $GF(2^h)$ (e.g., an irreducible

[4] This generator outputs non-Boolean random variables. To get Boolean random variables we can, say, take the least significant bit of $G(x)_i$.

polynomial of degree h over $GF(2)$), and (ii) we must compute field operations (such as multiplication and exponentiation) over $GF(2^h)$.

We do not know how to solve (i) in *uniform* $AC^0[\oplus]$ circuits of size $\mathrm{poly}(h)$ for *every* given h. However, for h of the special form $h = 2\cdot 3^l$ (for some l), one can use the polynomial $x^h + x^{h/2} + 1$ which is irreducible over $GF(2)$ (see e.g. [vL99], Theorem 1.1.28). (Also, standard algorithms solve (i) under P-uniformity.)

At the time we wrote this paper we did not know how to solve (ii) in uniform $AC^0[\oplus]$. However, subsequent to our work, and actually motivated by it, there has been progress on (ii) (E. Allender, A. Healy and E. Viola, personal communication, August 2004).

In this work we exhibit alternative constructions of k-wise independent generators computable in uniform $AC^0[\oplus]$.

Techniques: Our k-wise independent generators are based on *combinatorial designs*. A combinatorial design is a collection of m subsets $S_1, \ldots, S_m$ of $\{1 \ldots n\}$ with small pairwise intersections. We obtain the i-th output bit of the generator by taking the parity of the bits of the seed indexed by S_i. Roughly speaking, the small intersection size of the sets will guarantee the k-wise independence of the variables. For the case $k = O(1)$, we give a new construction of combinatorial designs with $n = O(\log m)$, which is computable in $AC^0[\oplus]$ (Lemma 7). Following an idea from [HR03] (credited to S. Vadhan), we obtain this construction by "concatenating" Reed-Solomon codes with a design construction from [Vio04].

To compute our design constructions (specifically the Reed-Solomon code) we work over finite fields as well (as does the classic k-wise independent generator in Equation 1). The difference is that in our constructions we need finite fields of size exponentially smaller, i.e. $\mathrm{poly}(n)$ (as opposed to 2^n) . Representations of such fields can be found by brute force (even in uniform AC^0), and results by Agrawal et. al. [AAI+01] show how to perform field operations over fields of size $\mathrm{poly}(n)$ by uniform AC^0 circuits of size $\mathrm{poly}(n)$.

ϵ-biased generators in $AC^0[\oplus]$: Similar to k-wise independent generators, ϵ-biased generators explicitly computable in time $\mathrm{poly}(n)$ are known with seed length $n = O(\log m + \log 1/\epsilon)$ [NN90,AGHP92] which is optimal up to constant factors [AGHP92].

Alon et. al. [AGHP92] give three simple constructions, but none of them seems to be implementable in uniform $AC^0[\oplus]$: they either seem to be inherently sequential, or they need large primes, or they need exponentiation over finite fields of size 2^n with exponent of n bits, all of which do not seem to be computable in uniform $AC^0[\oplus]$ circuits of size $\mathrm{poly}(n)$.

Prior to the work of [AGHP92], Naor and Naor [NN90] gave another construction. We show that their construction (as long as $\epsilon \geq 1/2^{\mathrm{poly}\log\log m}$) can be implemented in uniform $AC^0[\oplus]$. (For $\epsilon \leq 1/2^{\mathrm{poly}\log\log m}$ we obtain seed length $O((\log m)(\log 1/\epsilon))$ with a slight modification of the construction in [NN90]).

Techniques: The construction of [NN90] goes through a few stages. In particular it uses as a component a 7-wise independent generator. By the discussion above, we have such a generator in uniform $AC^0[\oplus]$. [NN90] also needs to compute random walks on an expander graph. We show that, for every fixed c, random walks of length $\log^c n$ on the Margulis expander [Mar73] of size 2^n, can be computed by uniform AC^0 circuits of size $\mathrm{poly}(n)$ (where the depth of the circuit depends on c). We also argue that for this specific expander, our result is tight. Bar-Yossef, Goldreich and Wigderson [BYGW99] show how to efficiently compute walks on the same expander but in an *online* model of computation. Their result is incomparable to ours. Ajtai [Ajt93] shows how to compute in uniform AC^0 of size $\mathrm{poly}(n)$ walks of length $\log n$ on expander graphs of size exponentially smaller, i.e. n.

Almost k-wise independent generators in $AC^0[\oplus]$: Using the approach of [NN90] that combines k-wise independent and ϵ-biased generators to get almost k-wise independent generators, we can use our constructions to obtain an (ϵ, k)-wise independent generator $G : \{0,1\}^n \to \{0,1\}^m$ explicitly computable by uniform $AC^0[\oplus]$ circuits of size $\mathrm{poly}(n, \log m)$. Its seed length is $O(k+\log\log m+\log(1/\epsilon))$ as long as $k = \mathrm{poly}\log\log m$ and $\epsilon = 1/2^{\mathrm{poly}\log\log m}$. In this range of parameters, this seed length matches that of the best known constructions [NN90,AGHP92] up to constant factors. We leave the details of this construction to the interested reader.

Organization: In Section 2 we discuss some preliminaries. In Section 3 we show the connection between combinatorial designs and k-wise independent generators, and we exhibit our construction of combinatorial designs and our constructions of k-wise independent generators in uniform $AC^0[\oplus]$. In Section 4 we give our ϵ-biased generators in uniform $AC^0[\oplus]$. This includes our results about computing walks on expander graphs.

2 Preliminaries

We now define k-wise independent and ϵ-biased generators. Denote the set $\{1, \ldots, m\}$ by $[m]$. For $I \subseteq [m]$ and $G(x) \in \{0,1\}^m$ we denote by $G(x)|_I \in \{0,1\}^{|I|}$ the projection of $G(x)$ on the bits specified by I. Recall that $\bigoplus$ is the *parity* function, i.e. $\bigoplus_{i\in I}(x) := \sum_{i\in I} x_i \pmod 2$.

Definition 1. *Let $G : \{0,1\}^n \to \{0,1\}^m$ be a function.*

- *G is* (ϵ, k)-wise independent *if for every $0 < k' \le k$, $M : \{0,1\}^{k'} \to \{0,1\}$ and $I \subseteq [m]$ such that $|I| = k'$:*

$$\left| \Pr_{y\in\{0,1\}^{k'}}[M(y) = 1] - \Pr_{x\in\{0,1\}^n}[M(G(x)|_I) = 1] \right| \le \epsilon$$

- *G is* k-wise independent *if it is $(0, k)$-wise independent.*
- *G is* ϵ-biased *if for every $\emptyset \neq I \subseteq [m]$: $\left| \Pr_{x\in\{0,1\}^n}[\bigoplus_{i\in I} G(x)_i = 0] - \frac{1}{2} \right| \le \epsilon$.*

We now define the circuit classes of interest in this paper. AC^0 is the class of constant depth circuits with $\neg, \vee$ and $\wedge$ gates, where $\vee$ and $\wedge$ have unbounded fan-in. $AC^0[\oplus]$ is the class of constant depth circuits with $\neg, \vee, \wedge$ and $\bigoplus$ gates, where $\vee, \wedge$ and $\bigoplus$ have unbounded fan-in. A family of circuits $\{C_n\}$ is DLOGTIME-uniform, if there is a Turing machine M running in linear time that decides *the direct connection language of* $\{C_n\}$, which is the language of tuples (t, a, b, n), such that b and a are names of gates in C_n, b is a child of a, and a is of type t [BIS90,Vol99]. *In this paper uniform always means DLOGTIME-uniform.*

Definition 2. *A generator* $G : \{0,1\}^n \to \{0,1\}^m$ *is* explicitly computable by uniform $AC^0[\oplus]$ (resp. AC^0) circuits *of size* g*, if there is a uniform* $AC^0[\oplus]$ *(resp.* AC^0*) circuit* C *of size* g *such that* $C(x,i) = G(x)_i$ *for all* $x \in \{0,1\}^n$ *and* $i \in \{0,1\}^{\log m}$*, where* $G(x)_i$ *is the* i*-th output bit of* $G(x)$*.*

In [MNT90] they essentially prove the following negative result on the ability of AC^0 circuits to explicitly compute (ϵ, k)-wise independent and ϵ-biased generators.

Theorem 3 ([MNT90]). *Fix any constant* $\epsilon < 1/2$*. Let* $G : \{0,1\}^n \to \{0,1\}^m$ *be a generator either* $(\epsilon, 2)$*-wise independent or* ϵ*-biased. Let* C *be a circuit of size* g *and depth* d *such that* $C(x,i) = G(x)_i$ *for every* $x \in \{0,1\}^n, i \in \{0,1\}^{\log m}$*. Then* $\log^{d-1} g \geq \Omega(\log m)$*.*

In some of our constructions we make use of the following result from [AAI$^+$01] about field operations in uniform $AC^0[\oplus]$. We denote by $GF(2^t)$ the field with 2^t elements, and we identify these elements with bit strings of length t.

Lemma 4 ([AAI$^+$01], Theorem 3.2 and proof of Theorem 1.1). *For any functions* $t = O(\log n)$ *and* $k = \mathrm{poly}(n)$ *there is a uniform* $AC^0[\oplus]$ *circuit of size* $\mathrm{poly}(n)$ *such that given a polynomial* $p(x) := \sum_{i=0}^{k} a_i x^i$ *of degree* k *over* $GF(2^t)$*, and* $b \in GF(2^t)$*, computes* $p(b)$*.*

We stress that Lemma 4 do not depend on any specific representation of the field. In fact, even if the representation is not given, a uniform AC^0 circuit of size $\mathrm{poly}(n)$ can find the first (lexicographically) representation by brute force.

3 k-wise independent generators from designs

In this section we present a general approach that gives k-wise independent generators from *combinatorial designs.* First we define combinatorial designs. Then we show how to get k-wise independent generators from combinatorial designs. We then turn to the problem of efficiently constructing designs with good parameters.

Definition 5. *[NW94] A* (l,d)*-design of size* m *over a universe of size* s *is a family* $S = (S_1, \ldots, S_m)$ *of subsets of* $[s]$ *that satisfies: (1) for every* i*,* $|S_i| = l$*, and (2) for every* $i \neq j$*,* $|S_i \cap S_j| \leq d$*.*

We now show how to get k-wise independent generators from combinatorial designs.

Lemma 6. *Let $S = (S_1, \ldots, S_m)$ be a $((k+1)d, d)$-design of size m over a universe of size n. Define the generator $G_S : \{0,1\}^n \to \{0,1\}^m$ as: $G_S(r)_i := \bigoplus_{j \in S_i} r_j$. Then G_S is a k-wise independent generator.*

Proof. Fix k output bits $i_1, \ldots, i_k$. We show that $G_S(x)_{i_k}$ is uniform over $\{0,1\}$ and independent from $G_S(x)_{i_1}, \ldots, G_S(x)_{i_{k-1}}$. By definition, $G_S(x)_{i_k}$ is the parity of the bits in x indexed by S_{i_k}. Since S is a $((k+1)d, d)$-design there is $e \in S_{i_k}$ such that $e \notin \bigcup_{0<j<k} S_{i_j}$. Thus the value of $G_S(x)_{i_k}$ is independent from $G_S(x)_{i_1}, \ldots, G_S(x)_{i_{k-1}}$, because its parity includes a bit that is independent from the bits in the parities of $G_S(x)_{i_1}, \ldots, G_S(x)_{i_{k-1}}$. $\square$

Note that under a certain choice of combinatorial designs, the generator G_S in Lemma 6 is Nisan's generator against constant depth circuits [Nis91]. Indeed it was observed by several researchers that Nisan's generator is $\operatorname{poly}\log m$-wise independent [LN90,CSS97].

We want to point out the connection of our approach to the NC^0 generators of [MST03]. There, the generators are based on certain bipartite graphs that have the *unique-neighbor property* (see [MST03] for the exact definition). Our combinatorial designs, can be seen as bipartite graphs with this property. (By having the elements of the universe on one side and the sets on the other, and placing edges between elements and the sets that contains them). Interestingly, such graphs with the parameters that [MST03] need are unknown explicitly, and this fact makes their constructions nonuniform. On the other hand, as we show below, graphs with the parameters that we need can be constructed very efficiently.

3.1 Combinatorial designs computable in uniform $AC^0[\oplus]$

By Lemma 6, in order to construct a k-wise independent generator, $G : \{0,1\}^n \to \{0,1\}^m$, explicitly computable in uniform $AC^0[\oplus]$, we need to show a construction of a $((k+1)d, d)$-design of size m over a universe of size n, that is *explicitly* computable in uniform $AC^0[\oplus]$. By this we mean that given $1 \leq i \leq m$, we can compute the i'th set in the design (say, as a characteristic vector) in uniform $AC^0[\oplus]$.

We now describe a new construction of combinatorial designs of size m with universe size $O(\log m)$, where the size of the sets is larger than the size of their pairwise intersections by an arbitrarily large constant factor. While there are known constructions that achieve the parameters that we need (e.g. [NW94,KvM99,Vio04]) these constructions are not explicit (not even polynomial-time explicit).

Our construction follows an idea from [HR03] (credited to Salil Vadhan) to combine error-correcting codes with combinatorial designs. While their construction does not achieve the parameters that we need, we can use this idea to obtain designs with the desired parameters combining Reed-Solomon codes with a design construction from [Vio04]. The following lemma states the parameters that we achieve.

Lemma 7. *For every constant $c > 1$ and large enough m there is a family S of $(c^2 \log m, 2c \log m)$-designs of size m over a universe of size $50 \cdot c^3 \log m$. Moreover, there is a uniform $AC^0[\oplus]$ circuit of size $\mathrm{poly}(\log m)$, such that given $i \in \{0,1\}^{\log m}$ computes the characteristic vector of S_i.*

Note that in this construction the ratio (set size)/(pairwise intersection) = $c/2$ can be an arbitrarily large constant.

Lemma 7 uses as a component a family of designs with exponentially smaller parameters computable in AC^0 [Vio04].

Lemma 8 ([Vio04]). *For every constant $c > 1$, and for every large enough n there is a family S of $(c \log n, \log n)$-designs of size n over a universe of size $50 \cdot c^2 \log n$. Moreover, there is a uniform AC^0 circuit of size $\mathrm{poly}(n)$ that, given $i \in \{0,1\}^{\log n}$, computes the characteristic vector of S_i.*

Proof (of Lemma 7). First we describe the construction, then we show it is a design with the claimed parameters and then we study its complexity.

Construction: The idea is 'combining' a Reed-Solomon code with the designs given by Lemma 8. Let n be such that $n \log n = c \log m$. Fix a field $\mathbf{F}$ of size n (without loss of generality we assume that n is a power of 2). Let $h := n/c$. Fix $z \in [n^h] = [2^{n \log n/c}] = [m]$, as a bit string $z = a_0 \dots a_h$ where each a_i has $\log(n)$ bits. Define the polynomial p_z over $\mathbf{F}$ as $p_z := \sum_{i=0}^{h} a_i x^i$. Let $b_1, \dots, b_n$ be an enumeration of all elements of $\mathbf{F}$.

Now consider a family $(D_1, \dots, D_n)$ of $(c \log n, \log n)$ designs of size n over a universe of size $50 \cdot c^2 \log n$ as guaranteed by Lemma 8.

Then $S = (S_1, \dots, S_m)$ is defined as follows: We view $z \in [n^h]$ as an index in $[m]$, and we define the characteristic vector of S_z to be $D_{p_z(b_1)} \cdots D_{p_z(b_n)}$. Namely it is a concatenation of n characteristic vectors of sets from D.

Analysis: S has m sets, each of size $n \cdot c \log n = c^2 \log m$, and the universe size is $n \cdot 50c^2 \log n = 50c^3 \log m$. We now bound the intersection size. Consider $z \neq z'$. Since p_z and $p_{z'}$ are polynomials of degree at most h, there are at most h distinct $b \in \mathbf{F}$ such that $p_z(b) = p_{z'}(b)$. Whenever $p_z(b) \neq p_{z'}(b)$, we have that $D_{p_z(b)}$ and $D_{p_{z'}(b)}$ are distinct sets in the design D, and thus their intersection is at most $\log n$. Therefore

$$|S_z \cap S_{z'}| \leq h \cdot c \log n + (n - h) \cdot \log n \leq 2n \log n = 2c \log m$$

Complexity: By Lemma 4, computing $p_z(b)$ can be done by uniform $AC^0[\oplus]$ circuits of size $\mathrm{poly}(n) = \mathrm{poly}(\log m)$, and D can be computed in uniform AC^0 circuts of size $\mathrm{poly}(n) = \mathrm{poly}(\log m)$ by Lemma 8. □

The construction above gives k-wise independent generators with $k = O(1)$ (see Theorem 10). For $k = \omega(1)$ we use as a component a design construction by Nisan and Wigderson [NW94]. The complexity of computing this construction follows from Lemma 4, we omit the proof.

Lemma 9 ([NW94]). *For every integers ℓ, m such that $\log m \le \ell \le m$, there is a $(\ell, \log m)$-design of size m over a universe of size $O(\ell^2)$. Moreover, there is a uniform $AC^0[\oplus]$ circuit of size $\mathrm{poly}(\ell)$, such that given $i \in \{0,1\}^{\log m}$ computes the characteristic vector of S_i.*

We now state our k-wise independent generators.

Theorem 10. *For every large enough m, there is a k-wise independent generator $G : \{0,1\}^n \to \{0,1\}^m$, that is explicitly computable by uniform $AC^0[\oplus]$ circuits of size $\mathrm{poly}(n)$, where,*

1. *$k \ge 2$ is a constant, and $n = O(\log m)$. Or,*
2. *k is any function (of m), and $n = O(k^2 \log(m) \log(k \cdot \log m))$.*

Proof. For Item (1) we plug the design construction from Lemma 8 (with $c := 2(k+1)$) into Lemma 6. Item (2) is obtained as follows. Consider a $((k+1)\log m, \log m)$-design S of size m over a universe of size $O(k^2 \log^2 m)$ as guaranteed by Lemma 9. By Lemma 6, G_S is a k-wise independent generator, however the seed length of G_S is $n = O(k^2 \log^2 m)$. To reduce the seed length, suppose we have a $(k \cdot (k+1)\log m)$-wise independent generator $G' : \{0,1\}^{n'} \to \{0,1\}^n$. We claim that $G \circ G' : \{0,1\}^{n'} \to \{0,1\}^m$ is still a k-wise independent generator. This is because every k output bits of G depend on at most $k \cdot (k+1)\log m$ output bits of $G'(x)$. Since G' is $(k \cdot (k+1)\log m)$-wise independent, these bits will by uniformly and independently distributed. Now, using for G' the generator of [CG89,ABI86] (Section 1.1, Equation 1) we have $n' = k \cdot (k+1)\log(m) \cdot \log n = O(k^2 \log(m) \log(k \cdot \log m))$. Finally, by Lemma 4, G' is computable by uniform $AC^0[\oplus]$ circuits of size $\mathrm{poly}(n')$. $\square$

A "combinatorial" construction. We note that our approach allows for a construction of k-wise independent generators (for $k = O(1)$) that does not use finite fields of growing size. This is obtained combining, as in Lemma 7, the designs from Lemma 8 with an error correcting code. For the latter, we take an expander code [SS96] based on the Margulis expander discussed in Section 4.1. Since this error-correcting code does not have the minimum distance that we need, we use an expander graph (again, the Margulis construction) to increase its minimum distance, at the price of increasing the alphabet size (this is actually in our favor since we need a large alphabet to concatenate with the designs), as is done by Alon et. al. [ABN+92]. The resulting generators match the parameters of Theorem 10 (1), and they are computable in P-uniform $AC^0[\oplus]$ circuits of size $\mathrm{poly}(n)$. We omit the formal details.

4 ϵ-bias in $AC^0[\oplus]$

In this section we describe our constructions of ϵ-biased generators in uniform $AC^0[\oplus]$. We obtain our constructions by exhibiting uniform $AC^0[\oplus]$ implementations of the ϵ-biased generator due to Naor and Naor [NN90]. We do not describe the ϵ-biased generator in [NN90] here. We only point out that it is built combining 7-wise independent generators, 2-wise independent generators and *random walks on expander graphs.* Using our k-wise independent generators in uniform $AC^0[\oplus]$ (Theorem 10) for the first two, all that is left to do is to compute walks on expander graphs in uniform $AC^0[\oplus]$. We now formally state what random walks we need to get ϵ-biased generators.

A family of graphs $\{G_N\}_N$ is a family of d-regular expander graphs if there is a constant $\lambda < d$ such that for every N the graph G_N has N nodes, is d-regular, and the second largest eigenvalue (in absolute value) of its adjacency matrix is at most λ. To get the ϵ-biased generator of [NN90] with seed length n we need to compute random walks of length $l = O(\log 1/\epsilon)$ on expander graphs of size $2^{O(n)}$. We show in the next section that for every fixed c we can compute walks of length $\log^c n$ on certain expander graphs of size 2^n in uniform AC^0 circuits of size $\mathrm{poly}(n)$.

Theorem 11. *There is a family $\{G_N\}_N$ of 8-regular expander graphs such that for every c there is a uniform AC^0 circuit of size $\mathrm{poly}(n)$ that, given $v \in G_{2^n}$ and a path w of length $\log^c n$ (i.e. $w \in [8]^{log^c n}$), computes the node $v' \in G_{2^n}$ reached starting from v and walking according to w.*

Using the expander walks in Theorem 11 we get, for $\epsilon \geq 1/2^{\log^c(\log m)}$, an ϵ-biased generator with seed length optimal up to constant factors [AGHP92]. For smaller ϵ, we replace the random walk on the expander with random (independent) nodes in the graph, and obtain a generator with larger seed.

Theorem 12. *For every large enough m, there is an ϵ-biased generator G_ϵ : $\{0,1\}^n \rightarrow \{0,1\}^m$, explicitly computable by uniform $AC^0[\oplus]$ circuits of size $\mathrm{poly}(n)$, where*

1. *$n = O(\log m + \log(1/\epsilon))$ and $\epsilon = 1/2^{\log^c(\log m)}$, for an arbitrary constant $c > 0$. Or,*
2. *$n = O((\log m) \cdot \log(1/\epsilon))$ and $\epsilon = \epsilon(m)$ is arbitrary.*

4.1 Expander walks in AC^0

In this section we prove Theorem 11, i.e. we show that there is an expander graph of size $N = 2^n$ where random walks of length $l = O(\log^c n)$ can be computed by uniform AC^0 circuits of size $\mathrm{poly}(n)$ (for every fixed constant c; the depth of the circuit depends on c). We use an expander construction due to Margulis [Mar73] (the needed expansion property was proved later in [GG81,JM87]), which we now recall.

Let $m := \sqrt{N}$ (we assume without loss of generality that m is a power of 2). The vertex set of G_N is $Z_m \times Z_m$, where Z_m is the ring of the integers modulo m. Each vertex v is a pair $v = (x,y)$ where $x,y \in Z_m$. For matrices T_1, T_2 and vectors b_1, b_2 defined below, each vertex $v \in G_N$ is connected to $T_1 v, T_1 v + b_1, T_2 v, T_2 v + b_2$ and the four inverses of these operations.

Theorem 13. *[Mar73,GG81,JM87] The family* $\{G_N\}_N$ *with* $T_1 := \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$, $T_2 := \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, $b_1 := \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ *and* $b_2 := \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ *is a family of* 8*-regular expander graphs (the absolute value of the second eigenvalue of the adjacency matrix is* $5\sqrt{2} < 8$*).*

We now prove Theorem 11, with the family $\{G_N\}_N$ of expander graphs from Theorem 13. That is, we show that walks of length $\log^c n$ on G_{2^n} can be computed in uniform AC^0 circuits of size $\mathrm{poly}(n)$. We note that this result is tight, i.e. there is no $AC^0[\oplus]$ circuit (uniform or not) of size $\mathrm{poly}(n)$ that computes random walks of length $\log^{\omega(1)} n$ on G_{2^n}. This is because computing $\sum_i x_i$ given $x \in \{0,1\}^l$ (which cannot be done in $AC^0[\oplus]$ for $l = \log^{\omega(1)} n$) can be AC^0 reduced to the problem of computing random walks of length $O(l)$ over G_{2^n}. (Proof sketch: Given x, replace '1' in x with a step along the edge associated with T_1, replace '0' in x with a self-loop. Call x' the string thus obtained. Now start at vertex $(0,1)$, and compute a walk according to x'. It is easy to see that the first coordinate of the ending node is $\sum_i x_i$). However this negative result relies on the particular expander graph and on its representation. We do not know if $AC^0[\oplus]$ (or AC^0) circuits of size $\mathrm{poly}(n)$ can compute random walks of length $\log^{\omega(1)} n$ on *some* expander with $\omega(n)$ vertices.

We now turn to the proof of Theorem 11. By the definition of G_N, there are 8 matrices $\tilde{T}_1, \ldots, \tilde{T}_8$ with constant size entries and 8 vectors $\tilde{b}_1, \ldots, \tilde{b}_8$ with constant size entries such that the set of neighbors of a vertex v are $\{\tilde{T}_i v + \tilde{b}_i : i \leq 8\}$. Thus computing the random walk translates to computing

$$v' = A_l(\ldots(A_3(A_2(A_1 v + a_1) + a_2) + a_3)\ldots) + a_l$$

where for every i, $A_i \in \{\tilde{T}_i : i \leq 8\}$ and $a_i \in \{\tilde{b}_i : i \leq 8\}$. We write this as

$$v' = Av + A' \text{ where } A := A_l \ldots A_2 A_1 \text{ and } A' := A_l \ldots A_2 a_1 + A_l \ldots A_3 a_2 + \ldots + a_l.$$

So we are left with the following tasks: computing the matrix A, the vector A', and then computing $Av + A'$. We now show how to solve these problems in uniform AC^0. Recall (from the definition of the expander graph above), that the operations are modulo m. We will show how to do these calculations *without* taking modulo m. This latter operation can be done at the last step, and since we take m to be a power of 2 (i.e. n is even) it amounts to truncating the most significant bits.

First note that since the matrices A_i's have constant size entries, then A and A' have entries of size at most $O(l) = \log^c n$ bits for some c. To compute $Av + A'$ given A, A' and v we use the facts that (1) sum of two n-bit integers is in AC^0 and (2) multiplication of a n-bit integer by a $\log^c n$-bit integer is in AC^0. For (1) see e.g. [Vol99], Theorem 1.20, for (2) see e.g. [Vol99], Theorem 1.21 (this latter theorem shows multiplication of a n-bit integer by a $\log n$-bit integer. The same techniques give (2)). These circuits can be easily shown to be uniform.

We now show how to compute A, the same techniques give A'.

Lemma 14. *For every fixed constant c there is a uniform AC^0 circuit of size* $\mathrm{poly}(n)$ *that, given* $l = \log^c n$*,* 2×2 *matrixes* $A_1, \ldots, A_l$ *with constant size entries, computes* $A = \prod_{i \le l} A_i$.

Proof. Instead of proving the lemma directly, it is convenient to show that the product of n (as opposed to l) given matrices $A_1, \ldots, A_n$ with constant size entries can be computed in space $O(\log n)$, and then appeal to the following lemma (that can be obtained combining results in [Nep70] and in [BIS90], details omitted).

Lemma 15 ([Nep70] + [BIS90]). *Let L be a language computable in logarithmic space. Then for every constant $c \ge 1$ there is a uniform AC^0 circuit of size* $\mathrm{poly}(n)$ *that, given x of size* $\log^c n$*, correctly decides whether x is in L.*

Suppose we are given n matrices $A_1, \ldots, A_n$ with constant size entries. The idea is to first compute $\prod_{i \le n} A_i$ in Chinese Remainder Representation (CRR) and then convert the CRR to binary (using a result by Chiu, Davida and Litow [CDL01]). More formally, note that every entry of $\prod_{i \le k} A_i$, for every $k \le n$, will be at most d^n, for some constant d. By the Chinese Remainder Theorem each number $x \le d^n$ is uniquely determined by its residues modulo $\mathrm{poly}(n)$ primes, each of length $O(\log n)$. The Prime Number Theorem guarantees that there will be more than enough primes of that length. We refer to such a representation of a number x (i.e. as a list of residues modulo primes) as the CRR of x. Note that to find the primes for the CRR we search among the integers of size $O(\log n)$. This clearly can be done in logarithmic space.

We compute the product of the n matrices modulo one prime at a time, reusing the same space for different primes. To compute the product modulo one prime, note that transforming a matrix into CRR is easy because the matrices have constant size entries. Each matrix multiplication is a simple modular sum (modulo a prime of $O(\log n)$ bits) and therefore can be computed in logarithmic space. The machine operates in space $O(\log n)$ because it only needs to store a constant number of residues modulo a prime of length $O(\log n)$.

All that is left to do is to convert the product matrix from CRR to binary, and this can be done in space $O(\log n)$ by a result in [CDL01]. □

Acknowledgments: We thank the following people for helpful discussions and valuable suggestions: Eric Allender, Michael Ben-Or, Yonatan Bilu, Oded Goldreich, Alex Healy, Troy Lee, Eyal Rozenman, Ronen Shaltiel, Salil Vadhan, and Avi Wigderson.

References

[AAI+01] Manindra Agrawal, Eric Allender, Russell Impagliazzo, Toniann Pitassi, and Steven Rudich. Reducing the complexity of reductions. *Comput. Complexity*, 10(2):117–138, 2001.

[ABI86] N. Alon, L. Babai, and A. Itai. A fast and simple randomized algorithm for the maximal independent set problem. *Journal of Algorithms*, 7:567–583, 1986.

[ABN+92] N. Alon, J. Bruck, J. Naor, M. Naor, and R. Roth. Construction of asympototically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Trans. Inform. Theory*, 38:509–516, 1992.

[AGHP92] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures & Algorithms*, 3(3):289–304, 1992.

[AIK04] Benny Appelbaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC_0. In *Proceedings of the 45th Annual Symposium on Foundations of Computer science (to appear)*, 2004.

[Ajt93] Miklós Ajtai. Approximate counting with uniform constant-depth circuits. In *Advances in computational complexity theory (New Brunswick, NJ, 1990)*, pages 1–20. Amer. Math. Soc., Providence, RI, 1993.

[BIS90] David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC_1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990.

[BYGW99] Ziv Bar-Yossef, Oded Goldreich, and Avi Wigderson. Deterministic amplification of space bounded probabilistic algorithms. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity*, pages 188–198, 1999.

[CDL01] A. Chiu, G. Davida, and B. Litow. Division in logspace-uniform NC_1. *RAIRO Theoretical Informatics and Applications*, 35:259–276, 2001.

[CG89] Benny Chor and Oded Goldreich. On the power of two-point based sampling. *Journal of Complexity*, 5(1):96–106, March 1989.

[CGH+85] B. Chor, O. Goldreich, J. Hastad, J. Friedman, S. Rudich, and R. Smolensky. The bit extraction problem and t-resilient functions. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, pages 396–407, Portland, Oregon, 21–23 October 1985. IEEE.

[CM01] Mary Cryan and Peter Bro Miltersen. On pseudorandom generators in NC_0. In *Proceedings of 26th International Symposium on Mathematical Foundations of Computer Science (MFCS 01)*, pages 272–284. Springer-Verlag, 2001.

[CSS97] Jin-Yi Cai, D. Sivakumar, and Martin Strauss. Constant depth circuits and the lutz hypothesis. In *38th Annual Symposium on Foundations of Computer Science*, pages 595–604, Miami Beach, Florida, 20–22 October 1997. IEEE.

[GG81] O. Gabber and Z. Galil. Explicit constructions of linear size superconcentrators. *Journal of Computer and System Sciences*, 22:407–420, 1981.

[Gol99] Oded Goldreich. *Modern cryptography, probabilistic proofs and pseudorandomness*, volume 17 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 1999.

[HR03] T. Hartman and R. Raz. On the distribution of the number of roots of polynomials and explicit weak designs. *Random Structures & Algorithms*, 23(3):235–263, 2003.

[JM87] S. Jimbo and A. Maruoka. Expanders obtained from affine transformations. *Combinatorica*, 7(4):343–355, 1987.

[Kab03] Valentine Kabanets. Almost k-wise independence and hard boolean functions. *Theoretical Computer Science*, 297(1–3):281–295, 2003.

[KvM99] Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (Atlanta, GA, 1999)*, pages 659–667 (electronic). ACM, New York, 1999.

[LN90] Nathan Linial and Noam Nisan. Approximate inclusion-exclusion. *Combinatorica*, 10(4):349–365, 1990.

[Mar73] G. A. Margulis. Explicit construction of concentrator. *Problems Inform. Transmission*, 9:325–332, 1973.

[MNT90] Yishay Mansour, Noam Nisan, and Prasoon Tiwari. The computational complexity of universal hashing. In *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing*, pages 235–243, 1990.

[MST03] E. Mossel, A. Shpilka, and L. Trevisan. On ϵ-biased generators in NC_0. In *Proceedings of the 44th Annual Symposium on Foundations of Computer Science*, Cambridge, Massachusetts, 11–14 October 2003. IEEE.

[Nep70] V.A. Nepomnjaščiĭ. Rudimentary predicates and turing calculations. *Soviet Mathematics-Doklady*, 11(6):1462–1465, 1970.

[Nis91] Noam Nisan. Pseudorandom bits for constant depth circuits. *Combinatorica*, 11(1):63–70, 1991.

[NN90] J. Naor and M. Naor. Small-bias probability spaces: efficient constructions and applications. In *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing*, pages 213–223, 1990.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, October 1994.

[SS96] M. Sipser and D.A. Spielman. Expander codes. *IEEE Trans. Inform. Theory*, 42:1710–1722, 1996.

[Vio04] Emanuele Viola. The complexity of constructing pseudorandom generators from hard functions. Technical Report TR04-020, Electronic Colloquium on Computational Complexity, 2004. `http://www.eccc.uni-trier.de/eccc`. To appear in Journal of Computational Complexity.

[vL99] J. H. van Lint. *Introduction to coding theory*, volume 86 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, third edition, 1999.

[Vol99] Heribert Vollmer. *Introduction to circuit complexity*. Springer-Verlag, Berlin, 1999.