Pseudorandom Bit Generators that Fool Modular Sums

Extended Abstract

Shachar Lovett* Omer Reingold† Luca Trevisan‡ Salil Vadhan§

April 12, 2009

Abstract

We consider the following problem: for given n, M, produce a sequence $X_1, X_2, \ldots, X_n$ of *bits* that fools every linear test modulo M. We present two constructions of generators for such sequences. For every constant prime power M, the first construction has seed length $O_M(\log(n/\epsilon))$, which is optimal up to the hidden constant. (A similar construction was independently discovered by Meka and Zuckerman [MZ].) The second construction works for every M, n, and has seed length $O(\log n + \log(M/\epsilon)\log(M\log(1/\epsilon)))$.

The problem we study is a generalization of the problem of constructing *small bias* distributions [NN], which are solutions to the $M = 2$ case. We note that even for the case $M = 3$ the best previously known constructions were generators fooling general bounded-space computations, and required $O(\log^2 n)$ seed length.

For our first construction, we show how to employ recently constructed generators for sequences of elements of $\mathbb{Z}_M$ that fool small-degree polynomials (modulo M). The most interesting technical component of our second construction is a variant of the derandomized graph squaring operation of [RV]. Our generalization handles a product of two distinct graphs with distinct bounds on their expansion. This is then used to produce pseudorandom-walks where each step is taken on a different regular directed graph (rather than pseudorandom walks on a single regular directed graph as in [RTV, RV]).

*Department of Computer Science, Weizmann Institute of Science, Rehovot 76100, Israel. shachar.lovett@weizmann.ac.il.

†Department of Computer Science, Weizmann Institute of Science, Rehovot 76100, Israel. omer.reingold@weizmann.ac.il. Research supported by US-Israel BSF grant 2006060.

‡Computer Science Division, University of California, Berkeley, CA, USA. luca@cs.berkeley.edu. This material is based upon work supported by the National Science Foundation under grant No. CCF-0729137 and by the US-Israel BSF grant 2002246.

§School of Engineering and Applied Science, Harvard University, Cambridge, MA 02138. salil@eecs.harvard.edu. Work done in part while visiting U.C. Berkeley, supported by the Miller Institute for Basic Research in Science and a Guggenheim Fellowship. Also supported by US-Israel BSF grant 2006060.

1 Introduction

Pseudorandomness is the theory of generating objects that "look random" despite being constructed using little or no randomness. A primary application of pseudorandomness is to address the question: *Are randomized algorithms more powerful than deterministic ones?* That is, how does randomization trade off with other computational resources? Can every randomized algorithm be converted into a deterministic one with only a polynomial slowdown (*i.e.,* does $\mathbf{BPP} = \mathbf{P}$) or with only a constant-factor increase in space (*i.e.,* does $\mathbf{RL} = \mathbf{L}$)? The study of both these questions has relied on pseudorandom bit generators that fool algorithms of limited computational powers. In particular, generators that fool space-bounded algorithms [AKS, BNS, Nis, INW] were highly instrumental in the study of the **RL** vs. **L** problem (e.g. used in the best known derandomization of **RL** [SZ]).

While the currently available space-bounded generators are extremely powerful tools, their seed length is still suboptimal. For example, if we want to fool a $\log n$-space algorithm then known generators require $\log^2 n$ truly random bits (the seed) in order to generate up to polynomially many pseudorandom bits. On the other hand, for several interesting special cases we do know generators with almost optimal seed length. The special case which serves as a motivation for our work is that of small-biased generators [NN]. These generators produce n bits $X_1, X_2, \ldots, X_n$ that fool all linear tests modulo 2. In other words, for each subset T of the bits, the sum $\Sigma_{i \in T} X_i \mod 2$ is uniformly distributed up to bias ϵ. Explicit constructions of ϵ-biased generators are known with seed-length $O(\log(n/\epsilon))$, which is optimal up to the hidden constant [NN]. Even though linear tests may seem very limited, ϵ-biased generators have turned out to be very versatile and useful derandomization tools [NN, MNN, HPS, Nao, AM1, AR, BSVW, BV, Lov, Vio]

Given the several applications of distributions that fool linear tests modulo 2, it is natural to consider the question of fooling modular sums for larger moduli. It turns out that the notion of small-biased generators can be generalized to larger fields. Such generators produce a sequence $X_1, X_2, \ldots, X_n$ of elements in a field $\mathbb{F}$ that fool every linear test over $\mathbb{F}$. [Kat, AIK$^+$, RSW, EGL$^+$, AM1]. In this work, instead, we consider a different generalization of ϵ-biased generators where we insist on *bit*-generators. Namely we would like to generate a sequence $X_1, X_2, \ldots, X_n$ of bits that fool every linear test modulo a given number M. For every sequence $a_1, a_2, \ldots, a_n$ of integers in $\mathbb{Z}_M = \{0, 1, \ldots, M-1\}$ we want the sum $\Sigma_i a_i \cdot X_i \mod M$ to have almost the same distribution (up to statistical distance at most ϵ) as in the case where the X_i's are uniform and independent random bits. (Note that this distribution may be far from the uniform distribution over $\mathbb{Z}_M$, particularly when only a few a_i's are nonzero.) It turns out that even for $M = 3$ and even if we limit all the a_i's to be either ones or zeros, the best generators that were known prior to this work are generators that fool general space-bounded computations [Nis, INW], and required a seed of length $O(\log^2 n)$. Therefore, obtaining better pseudorandom bit generators that fool modular sums may be considered a necessary step towards improved space-bounded generators. In addition, we consider this notion to be a natural generalization of small-bias generators, which is a central derandomization tool.

Our Results

We give two constructions of pseudorandom bit generators that fool modular sums.

Similarly to [MST], each construction is actually comprised of two generators: one that fools summations $\Sigma_i a_i \cdot X_i$ in which only relatively few coefficients a_i are not zero (the "low-weight" case) and one that fools summations $\Sigma_i a_i \cdot X_i$ such that many coefficients a_i are not zero (the "high

weight" case). The motivation is that fooling low-weight sums and fooling high-weight sums are tasks of a different nature. In the high-weight case, if R_i are truly random bits, then $\Sigma_i a_i \cdot R_i \mod M$ is almost uniformly distributed in $\mathbb{Z}_M$. Thus, in analyzing our generator, we just need to argue that $\Sigma_i a_i \cdot X_i \mod M$ is close to uniform, where $X_1, \ldots, X_n$ is the output of the generator.

On the other hand, in the low-weight case the distribution may be far from uniform and therefore we may need to imitate the behavior of a random sequence of bits more closely.

In each construction, we shall present two generators: one that is pseudorandom against low-weight sums, and one that is pseudorandom against high-weight sums. We shall then combine them by evaluating them on independently chosen seeds and XORing the two resulting sequences.

Construction Based on Pseudorandom Generators for Polynomials

In our first construction, we handle the case of $M = 3$ and any other fixed prime modulus M (in fact, our construction works also for any fixed prime power). For these cases, our seed length is $O(\log(n/\epsilon))$ as in the case of ϵ-biased generators (but the hidden constant depends exponentially on M).

As mentioned above, for every fixed finite field $\mathbb{F}$, there are nearly-optimal known generators that construct a small-bias distribution $X_1, \ldots, X_n$ of *field elements*, while our goal is to generate *bits*. A natural approach to construct a bit generator would be to sample a sequence of field elements $X_1, \ldots, X_n$ from a small bias distribution, to pick a function $g : \mathbb{F} \to \{0, 1\}$ appropriately, and to output the bits sequence $g(X_1), \ldots, g(X_n)$. Unfortunately the small bias property for $g(X_1), \ldots, g(X_n)$ does not seem to follow from the small bias property of $X_1, \ldots, X_n$.

If, however, we start from a sequence of field elements $X_1, \ldots, X_n$ that fools *polynomials* over $\mathbb{F}$, then we can make such an approach work, because g can be chosen to be itself a polynomial (of degree $\Theta(|\mathbb{F}|)$). However, note that when $|\mathbb{F}|$ is odd, g cannot be balanced, and thus $g(X_1), \ldots, g(X_n)$ are only indistinguishable from independent *biased* coins. Thus, this approach only works when the sum has sufficiently high weight so that both biased and unbiased random bits will yield a sum that is almost uniformly distributed over $|\mathbb{F}|$; specifically we need at least k non-zero coefficients a_i, where $k = O(M^2 \log 1/\epsilon)$. For fixed M, there are known constructions [BV, Lov, Vio] of pseudorandom generators that fool polynomials of degree d over $\mathbb{F} = \mathbb{Z}_M$, M prime, and which only require seed length $O_{M,d}(\log n/\epsilon)$.

In order to fool low-weight sums, we observe that a bit generator $X_1, \ldots, X_n$ which is ϵ-almost k-wise independent fools, by definition, every sum $\sum_i a_i X_i \mod M$ of weight at most k, and that such generators are known which require only seed length $O(\log n + k + \log 1/\epsilon)$.

A similar construction was independently discovered by Meka and Zuckerman [MZ].

Construction Based on the INW Generator

In our second construction, we give a pseudorandom bit generator that fools sums modulo *any* given M (not necessarily prime) with seed length $O(\log n + \log(M/\epsilon) \log(M \log(1/\epsilon)))$. In both the low-weight and high-weight cases, this generator relies on versions of the Impagliazzo–Nisan–Wigderson [INW] pseudorandom generator for space-bounded computation. Of course, modular sums are a special case of space-bounded computations, and thus we could directly apply the INW generator. But this would require seed length larger than $\log^2 n$. We obtain better bounds by more indirect use of the INW generator inside our construction.

The most interesting technical contribution underlying this construction is a new analysis of the derandomized graph squaring operation of [RV], which captures the effect of using the INW generator to derandomize random walks on graphs. Here we study the analogue of derandomized

squaring for taking products of two distinct Cayley graphs over an abelian group (namely $\mathbb{Z}_M$). The advantage of the new analysis is that it handles graphs that have distinct bounds on their expansion, and works for bounding each eigenvalue separately. This is then used to produce pseudorandom-walks where each step is taken on a different abelian Cayley graph (rather than pseudorandom walks on a single graph as in [RTV, RV]).

For the purpose of this informal discussion we will assume that M is prime. (The idea for handling composite M's is to analyze each Fourier coefficient of the distribution of the sum separately. We defer further details to Section 2.1.)

Low-Weight Case. Let us first consider the case where the number of non-zero a_i's is at most $M' \cdot \log(1/\epsilon)$, for $M' = \text{poly}(M)$.[1] As before, we could use an almost k-wise almost independent distribution, but then our seed length would depend polynomially on M, while our goal is a polylogarithmic dependency.

First, we use a hash function to split the index set $[n] = \{1, 2, \ldots n\}$ into $B = O(M')$ disjoint subsets T_j such that with high probability (say, $1 - \epsilon/10$) over the splitting, each set T_j contains at most $k = \log(1/\epsilon)$ indices i such that $a_i \neq 0$. We show that the selection of the hash function that determines the splitting can be done using $O(\log n + (\log M/\epsilon) \cdot \log(M \log 1/\epsilon))$ random bits.

Once we have this partition, it is sufficient to independently sample in each block from an ϵ/B-almost k-wise independent distribution, which requires $s = O(\log n + k + \log(B/\epsilon)) = O(\log n + \log(M/\epsilon))$ random bits per block. Then we argue that it is not necessary for the sampling in different blocks to be independent, and instead they can be sampled using a pseudorandom generator for space-bounded computation [Nis, INW]. (This relies on the fact the computation $\sum_i a_i \cdot X_i \bmod M$ can be performed in any order over the i's, in particular the order suggested by $\sum_j \sum_{i \in T_j} a_i \cdot X_i \bmod M$.) Using the INW generator, we can do all the sampling using $O(s + \log B \cdot (\log(B/\epsilon) + \log M)) = O(\log n + \log M \cdot \log(M/\epsilon))$ random bits.

High-Weight Case. We now discuss the generator that fools sums with more than $M' \cdot \log 1/\epsilon$ non-zero coefficients a_i, for $M' = \text{poly}(M)$. Here, we can think of the computation $\Sigma_i a_i \cdot X_i \bmod M$ as an n-step walk over $\mathbb{Z}_M$ that starts at 0. Unlike standard walks, *each step is taken on a different graph* (over the same set of vertices, namely $\mathbb{Z}_M$). Specifically, step i is taken on the (directed) Cayley graph where every node v has two outgoing edges. The first edge is labeled 0 and goes into v itself (*i.e.,* this edge is a self loop). The second edge is labeled 1 and goes into $v + a_i \mod M$. Following the walk along the labels $X_1, X_2, \ldots, X_n$ arrives at the vertex $\Sigma_i a_i \cdot X_i \bmod M$. If the X_i's are uniform (*i.e.,* we are taking a random walk) then the end vertex will be almost uniformly distributed (because the number of steps is larger than $M^2 \cdot \log(1/\epsilon)$). What we are seeking is a pseudorandom walk that is generated using much fewer truly random bits but still converges to the uniform distribution (possibly slower, e.g. using $M' \cdot \log 1/\epsilon$) steps).

Pseudorandom walk generators were constructed in [RTV, RV] for walks on a single regular and connected graph. In our case, we are walking not on a single graph but rather on a sequence of graphs, each of which is indeed regular. It turns out that the pseudorandom generators of [RTV, RV] still work for a sequence of graphs rather than a single graph. The more difficult aspect is that in our walk there is no uniform bound on the expansion of the graphs. In fact, the graphs that correspond to $a_i = 0$ are not connected at all (they consist solely of self loops). In our setting, where the graphs are directed Cayley graphs for the abelian group $\mathbb{Z}_M$, we show how to generate pseudorandom walks on graphs with varying bounds on expansion.

[1] In this preliminary version we did not try to optimize the various constants. In particular, in our analysis $M' = O(M^{24})$. We note that it can be made as small as $O(M^{2+\alpha})$ for any $\alpha > 0$.

We do this by a generalization of the derandomized graph product of [RV]. There, expanders are used to generate two steps on a degree-D graph using less than $2 \log D$ random bits, yet the (spectral) expansion of the resulting graph is almost as good as the square of the original graph. We analyze the analogous derandomization of two steps on two distinct (abelian Cayley) graphs for which we may have distinct bounds on their expansion. Moreover, to handle composite M, we show that the expansion can be analyzed in each eigenspace separately. (For example, for $\mathbb{Z}_6 = \mathbb{Z}_2 \times \mathbb{Z}_3$, a sequence of even coefficients a_i will yield a random walk that does not mix in the $\mathbb{Z}_2$ component, but may mix in the $\mathbb{Z}_3$ component, and our pseudorandom generator needs to preserve this property.)

To obtain our pseudorandom walk generator, we first randomly reorder the index set $[n]$ so that the nonzero coefficients are well-spread out, and then derandomize the walk by a recursive application of our aforementioned derandomized product. As discussed in [RV], the resulting pseudorandom walk generator is the same as the Impagliazzo–Nisan–Wigderson [INW] generator for space-bounded computation, with a different setting of parameters that enables a much smaller seed length than their analysis requires for general space-bounded algorithms.

2 Definitions and Tools

This section contains some of the preliminary definitions and tools. For lack of space we defer some of the definitions and proofs to Appendix A.

We denote by U_n the uniform distribution over $\{0,1\}^n$. We fix an integer $M \geq 2$ for the rest of the paper. We will be interested in constructing pseudorandom generators for bits, that fool sums modulo M. We denote by $\mathbb{Z}_M$ the set $\{0, 1, \ldots, M-1\}$ with arithmetic modulo M.

Definition 1. (pseudorandom distributions against modular sums) A random variable $X = (X_1, ..., X_n)$ taking values in $\{0,1\}^n$ is *ϵ-pseudorandom against sums modulo M* if for any $a_1, ..., a_n \in \mathbb{Z}_M$, the distribution of $a_1 X_1 + ... + a_n X_n$ *modulo M*, is ϵ-close (in statistical distance) to the distribution $a_1 R_1 + ... + a_n R_n$ *modulo M*, where $R_1, ..., R_n$ are uniform and independent random bits.

Definition 2. (pseudorandom bit generators against modular sums) A function $G : \{0,1\}^r \to \{0,1\}^n$ is an ϵ-pseudorandom bit generator against sums modulo M if the distribution $G(U_r)$ is ϵ-pseudorandom against sums modulo M.

Note that ϵ-biased generators is a special case of the definition of pseudorandom bit generators against sums modulo M, for $M = 2$.

Our goal is to build generators that fool sums modulo M, where M can be either prime or composite. Handling prime modulus is somewhat easier, and the following approach allows handling both cases simultaneously. We will show that it is enough to construct pseudorandom generators which fools the bias of a sum modulo M, and under this approach, there is no major difference between primes and composites.

2.1 Small Bias Bit Generators

First we define the *bias* of a linear combination with coefficients $a_1, ..., a_n \in \mathbb{Z}_M$, given some distribution of $X = (X_1, ..., X_n) \in \{0,1\}^n$:

Definition 3. Let $X = (X_1, ..., X_n)$ be a distribution over $\{0,1\}^n$, and $a_1, ..., a_n \in \mathbb{Z}_M$ be a coefficient vector. We define the bias of $a_1, ..., a_n$ according to X to be

$$\text{bias}_X(a_1, .., a_n) = \mathbb{E}\left[\omega^{\sum a_i X_i}\right]$$

where $\omega = e^{2\pi i/M}$ is a primitive M-th root of unity.

Notice that the bias can in general be a complex number, of absolute value at most 1.

Definition 4. We say a distribution $X = (X_1, ..., X_n)$ over n bits is ϵ-bit-biased against sums modulo M if for any coefficient vector $a_1, ..., a_n \in \mathbb{Z}_M$,

$$|\text{bias}_X(a_1, ..., a_n) - \text{bias}_{U_n}(a_1, ..., a_n)| \leq \epsilon$$

Let $G : \{0,1\}^r \to \{0,1\}^n$ be a bit generator. We shorthand $\text{bias}_G(a_1, ..., a_n)$ for $\text{bias}_{G(U^r)}(a_1, ..., a_n)$.

Definition 5. $G : \{0,1\}^r \to \{0,1\}^n$ is an ϵ-bit-biased generator against sums modulo M if the distribution $G(U_r)$ is ϵ-bit-biased against sums modulo M. That is, for any coefficient vector $(a_1, ..., a_n)$,

$$|\text{bias}_G(a_1, ..., a_n) - \text{bias}_{U_n}(a_1, ..., a_n)| \leq \epsilon$$

The name 'bit-biased' in the above definitions is meant to stress the difference from standard ϵ-biased generators modulo M. Here we compare the bias of the generator to the bias of uniformly selected bits (rather than uniformly selected elements in $\mathbb{Z}_M$).

We now reduce the problem of constructing pseudorandom modular generators to that of constructing ϵ-bit-biased modular generators.

Lemma 1. *Let $X = (X_1, ..., X_n)$ be an ϵ-bit-biased distribution against sums modulo M. Then X is $(\epsilon\sqrt{M})$-pseudorandom against sums modulo M.*

From now on, we focus on constructing ϵ-bit-biased generators. We will need to differentiate two types of linear combinations, based on the number on non-zero terms in them.

Definition 6 (Weight of a coefficient vector)**.** The *weight* of a linear combination with coefficients $a_1, \ldots, a_n \in \mathbb{Z}_M$ is the number of non-zero coefficients.

We will construct two generators: one fooling linear combination with small weights, and the other fooling linear combinations with large weight. Our final generator will be the be the bitwise-XOR of the two, where each is chosen independently. Lemma 11 shows this will result in an ϵ-bit-biased generator fooling all linear combinations.

2.2 Hashing

We will use hashing as one of the major ingredients in our construction. A family of functions $\{h_r : [n] \to [k]\}$ is called a family of hash functions, if a randomly chosen function from the family behaves pseudorandomly under some specific meaning. We consider a hash function $H : [n] \to [k]$ to be a random variable depicting a randomly chosen function from the family. We say H can be generated efficiently and explicitly using s random bits, if a random function in the family can be sampled by a randomized polynomial-time algorithm using s random bits, and this function can be evaluated using a deterministic polynomial-time algorithm.

The j-th bucket of H, for $j \in [k]$, is the set of elements mapped to j, i.e.

$$\{s \in [n] : H(s) = j\}$$

Notice that buckets are random variables. We will in particular care about buckets, when we limit our attention to subsets $S \subset [n]$. The j-th bucket of H with respect to S is defined to be the set of elements of S mapped to j, i.e.

$$\{s \in S : H(s) = j\}.$$

We will use three constructions of hash functions, given in Lemmas 14, 15 and 16. The constructions are based on almost t-wise independence. We define those and prove tail bounds as well as the three families of hash functions in Appendix A.3.

2.3 Pseudorandom generators for small space

An ingredient in our construction is the small space pseudorandom generator of Impagliazzo, Nisan, and Wigderson [INW]. See details in Appendix A.4.

3 Construction using PRG for low-degree polynomials

We present in this section a simple construction for prime power M, based on pseudorandom generators for low-degree polynomials. This construction is optimal for constant M, achieving a pseudorandom generator with seed length $O_M(\log(1/\epsilon))$ (the constant depends exponentially on M).

Let $W = \Omega(M^3 \log 1/\epsilon)$. We will construct two generators: one for linear combinations of weight at most W, and one for linear combinations of weight at least W. Lemma 11 shows that the bitwise-XOR of the two generators is a pseudorandom generator for all linear combinations.

For small weights, we will use a distribution which is ϵ-close to W-wise independent variables. Such a distribution trivially fools linear combinations of weight at most W. By Lemma 19, it can be explicitly generated using $O(\log n + W + \log 1/\epsilon) = O_M(\log n/\epsilon)$ random bits.

We now consider large weights. Let $a_1, \ldots, a_n \in \mathbb{Z}_M$ be a coefficient vector of weight at least W. Consider first the distribution of $a_1 R_1 + \ldots a_n R_n$ for independent and uniform bits $R_1, \ldots, R_n$. By Lemma 12, $|\text{bias}_{U_n}(a_1, \ldots, a_n)| < \epsilon/2$.

Consider now $R_i \in \{0, 1\}$, where $\Pr[R_i = 0] = \frac{c}{M}$ for some integer $1 \leq c \leq M-1$. Such a distribution also gives bias of at most $\epsilon/4$, given that $W = \Omega(M^3 \log(1/\epsilon))$ with a large enough constant. By Lemma 13,

$$|\text{bias}_{R_1,\ldots,R_n \sim (\frac{c}{M}, 1-\frac{c}{M})}(a_1, \ldots, a_n)| < \epsilon/4.$$

The benefit of using this skewed distribution, is that it can be simulated by low-degree polynomials modulo M. Since we assume M is a prime power, there is a polynomial $g(z) : \mathbb{Z}_M \to \mathbb{Z}_M$, which maps c elements of $\mathbb{Z}_M$ to 0, and the rest to 1. For example, if $M = p^k$, the polynomial $g(x) = x^{(p-1)p^{k-1}}$ maps elements divisible by p to 0, and the rest to 1. The degree of this g is at most $M - 1$.

Let $Z_1, \ldots, Z_n \in \{0, 1\}^n$ be generated by $g(Y_1), \ldots, g(Y_n)$, where $Y_1, \ldots, Y_n \in \mathbb{Z}_M$ are uniform. We thus have:

$$|\text{bias}_{Z_1,\ldots,Z_n \sim g(U_{Z_M})^n}(a_1, \ldots, a_n)| < \epsilon/4$$

We now use the fact that the bias is in fact the bias of a low-degree polynomial, to derandomize the generation of $Y_1, \ldots, Y_n$. We have:

$$\text{bias}_{Z_1,\ldots,Z_n \sim g(U_{Z_M})^n}(a_1, \ldots, a_n) = \mathbb{E}_{Y_1,\ldots,Y_n \in \mathbb{Z}_M}[\omega^{a_1 g(Y_1) + \ldots + a_n g(Y_n)}]$$

We use a pseudorandom generator for low-degree polynomials, given by Viola [BV, Lov, Vio]. We note the results in these papers are stated for polynomials over prime finite fields, but they hold also for polynomials over $\mathbb{Z}_M$.

Lemma 2. *Let $f(x_1, \ldots, x_n)$ be a degree-d polynomial over $\mathbb{Z}_M$. There is an explicit generator $G : \{0,1\}^r \to \mathbb{Z}_M^n$, such that the distribution of $f(\mathbb{Z}_M^n)$ and $f(G(U_r))$ are ϵ-close in statistical distance. The number of random bits required is $r = O(2^d \log(1/\epsilon) + \log n)$.*

We use the generator of Lemma 2 for error $\epsilon/4$ and degree $d = M - 1$. We thus get an explicit generator for $Y_1, \ldots, Y_n \in \mathbb{Z}_M$, such that:

$$|\mathbb{E}_{Y'_1,\ldots,Y'_n \in G(U_r)}[\omega^{a_1 g(Y'_1)+\ldots+a_n g(Y'_n)}] - \mathbb{E}_{Y_1,\ldots,Y_n \in \mathbb{Z}_M^n}[\omega^{a_1 g(Y_1)+\ldots+a_n g(Y_n)}]| < \epsilon/4$$

Thus, if we set $X_i = g(Y'_i)$ where $Y'_1, \ldots, Y'_n$ are the output of G, we get an explicit generator, such that

$$|\mathrm{bias}_{X_1,\ldots,X_n \in g(G)}(a_1, \ldots, a_n)| < \epsilon/2$$

Thus, we got that the bias of the generator is ϵ close to the bias under the uniform distribution, which is what we wanted, since both of them have absolute value below $\epsilon/2$:

$$|\mathrm{bias}_{X_1,\ldots,X_n \in g(G)}(a_1, \ldots, a_n) - \mathrm{bias}_{R_1,\ldots,R_n \in U_n}(a_1, \ldots, a_n)| < \epsilon$$

The randomness requirement of our generator comes directly from that of G, which is $O(2^{M-1} \log(1/\epsilon) + \log n) = O_M(\log(n/\epsilon))$ for constant M.

4 Construction Based on Pseudorandom Walk Generators

4.1 A generator for small sums

We construct an ϵ-bit-biased generator for weights at most $W = 10^5 M^{24} \log(1/\epsilon)$. Let $a_1, \ldots, a_n \in \mathbb{Z}_M$ be a linear combination of weight at most W.

The construction has three stages:

1. Partitioning the set of indices $[n]$ into W buckets using the hash function H_1. Lemma 14 guarantees that with probability $1 - \epsilon/100$, each bucket contains at most $O(\log(1/\epsilon))$ non-zero coefficients.

2. For each bucket j, generate the X_i's for i's in the j'th bucket using almost $O(\log(1/\epsilon))$-wise independent distribution.

3. Use the INW generator given by Lemma 20 to generate the W seeds for the $O(\log(1/\epsilon))$-wise independent distributions used for the different buckets.

Lemma 3. *The above construction is an ϵ-bit-biased generator against linear combinations of weight at most W, using $O(\log n + \log(M/\epsilon) \log(M \log(1/\epsilon)))$ random bits.*

The proof appears in appendix B.1.

4.2 A generator for large sums

In this section we construct an ϵ-bit-biased distribution for linear combinations of weight at least $W = 10^5 M^{24} \log(1/\epsilon)$,

Recall that by Lemma 12, when the weight is large, the bias under the uniform distribution is small. Thus, to prove that a distribution is ϵ-bit-biased against large weight sums modulo M, it is enough to show that its bias is also small. We construct our ϵ-bit-biased generator in three steps:

- G_1: a generator which has bias at most $1 - \frac{1}{M^2}$ on every linear combination which is not all zeros.
- G_2: a generator which has bias at most $1/2$ on every linear combination of weight at least $100M^{24}$.
- G_3: a generator which has bias at most $\epsilon/2$ on every linear combination of weight at least $10^5 M^{24} \log 1/\epsilon$.

The generator G_3 will be our ϵ-bit-biased generator for large weights. The main ingredient in the construction will be a derandomized expander product, which we now define and analyze.

4.2.1 Derandomized expander products

Definition 7. We say an undirected graph H is a $(2^r, 2^d, \lambda)$-expander if H has 2^r vertices, it is regular of degree 2^d and all eigenvalues but the first have absolute value at most λ. We will identify the vertices of H with $\{0,1\}^r$, and the edges exiting each vertex with $\{0,1\}^d$ in some arbitrary way.

We will need explicit constructions of expanders, which can be obtained from any one of various known constructions [Mar1, GG, JM, AM2, AGM, LPS, Mar2, Mor, RVW]

Lemma 4. *For some constant $Q = 2^q$, there exist an efficient sequence H_k of $(Q^k, Q, 1/100)$-expanders.*

We now define expander products of two generators with common seed length:

Definition 8. Let $G', G'' : \{0,1\}^r \to \{0,1\}^t$ be two bit generators. Let H be a $(2^r, 2^d, \lambda)$-expander. We define $G' \otimes_H G'' : \{0,1\}^{r+d} \to \{0,1\}^{2t}$ to be the concatenation $(G'(x), G''(y))$, where x is a random vertex in H, and y is a random neighbor of x in H.

We relate the bias of $G' \otimes_H G''$ to the biases of G' and G''.

Lemma 5. *Let $G', G'' : \{0,1\}^r \to \{0,1\}^t$ be two bit generators and let H be a $(2^r, 2^d, \lambda)$-expander. Let $(a_1, ..., a_t)$,$(b_1, ..., b_t)$ be two coefficient vectors. Then:*

$$|\mathrm{bias}_{(G' \otimes_H G'')(U_{r+d})}(a_1, ..., a_t, b_1, ..., b_t)| \leq f(|\mathrm{bias}_{G'(U_r)}(a_1, ..., a_t)|, |\mathrm{bias}_{G''(U_r)}(b_1, ..., b_t)|)$$

where $f(x,y) = xy + \lambda\sqrt{1-x^2}\sqrt{1-y^2}$.

The proof appears in Appendix B.2. Some useful properties of the function $f(x,y)$ are given in Appendix B.3.

4.2.2 Construction of G_1

We define now an iterated product, using the expander graphs $H_1, H_2, \ldots$ given by Lemma 4.

$$\begin{aligned}
G'_1 &= id : \{0,1\}^q \to \{0,1\}^q \\
G'_2 &= G'_1 \otimes_{H_1} G'_1 : \{0,1\}^{2q} \to \{0,1\}^{2q} \\
G'_3 &= G'_2 \otimes_{H_2} G'_2 : \{0,1\}^{3q} \to \{0,1\}^{4q} \\
&\cdots \\
G'_\ell &= G'_{\ell-1} \otimes_{H_{\ell-1}} G'_{\ell-1} : \{0,1\}^{\ell q} \to \{0,1\}^{2^{\ell-1} q}
\end{aligned}$$

We will take the first generator $G_1 : \{0,1\}^{s_1} \to \{0,1\}^k$ to be G'_ℓ for the minimal ℓ such that $2^{\ell-1}q \geq k$. Thus, the seed length of this generator is

$$s_1 = (\log(k/q) + 1)q = O(\log k)$$

We will show that the bias of G_1 on a non-zero linear combination is at most $1 - \frac{1}{M^2}$.

Lemma 6. *Let $a_1, \ldots, a_n \in \mathbb{Z}_M$ be a coefficient vector, which is not all zero. Let ℓ be the minimal such that $n \leq 2^{\ell-1}q$. Apply G_ℓ as an ϵ-bit-biased generator to $a_1, \ldots, a_n$, by taking its first n bits. Then:*

$$bias_{G_\ell}(a_1, \ldots, a_n) \leq 1 - \frac{1}{M^2}$$

The proof appears in Appendix B.4.

4.2.3 Construction of G_2

We will construct G_2 based on G_1. We will prove in this subsection the following lemma:

Lemma 7. *There exists an explicit generator $G_2 : \{0,1\}^r \to \{0,1\}^n$, such that for all coefficient vectors $a_1, \ldots, a_n$ of weight at least $100M^{24}$, we have:*

$$\text{bias}_{G_2}(a_1, \ldots, a_n) \leq 0.91.$$

The seed-length of G_2 is $r = O(\log n + \log^2 M)$.

Assume first that we have the following special case where the non-zero coefficients are well-spread out. Let $k = 2^{\ell-1}q$, and $n = k2^s$. Let $a_1, \ldots, a_n$ be a coefficient vector, such that when partitioned into 2^s parts of size k, each part will contain at least one non-zero element. That is, for all $j \in [2^s]$:

$$weight(a_{jk+1}, a_{jk+2}, \ldots, a_{(j+1)k}) > 0$$

We first construct a generator $\tilde{G}_2$ for such "well spread-out" linear combinations. We then show how to use the hash function H_2 to transform each linear combination of weight at least $100M^{24}$ to the well-spread out case.

We define $\tilde{G}_2$ to be $G'_{\ell+s}$, defined in the previous subsection.

Lemma 8. *Let $k = 2^{\ell-1}q$, and $n = k2^s$. Let $a_1, \ldots, a_n$ be a coefficient vector such that for every $j \in [2^s]$, $weight(a_{jk+1}, a_{jk+2}, \ldots, a_{(j+1)k}) > 0$. Then:*

$$\text{bias}_{G'_{\ell+s}}(a_1, \ldots, a_n) \leq \min(1 - (9/8)^s \frac{1}{M^2}, 0.9)$$

In particular, if $s \geq 12 \log M$, then $\text{bias}_{G'_{\ell+s}}(a_1, \ldots, a_k) \leq 0.9$.

The proof appears in Appendix B.5.

We now construct the generator G_2 in three steps:

- Oblivious re-ordering of the coefficients, using the hash function H_2, to guarantee w.h.p the conditions of Lemma 8 (that the non-zero coefficients are well-spread out).
- Using $\tilde{G}_2$ on the re-ordered coefficients.
- Returning the pseudorandom bits back to the original order.

More formally, let $a_1, \ldots, a_n$ be a coefficient vector of weight at least $100M^{24}$. Let 2^s be the minimal power of two above M^{12}. Let $H_2 : [n] \to [2^s]$ be a hash function, as given by Lemma 15. We have that with probability of at least 0.99, all the M^{12} buckets of H_2 are non-empty. Create a re-ordering of the coefficient vector according to the buckets of H_2. The new coefficient vector $a'_1, \ldots, a'_{n2^s}$ is defined as follows: Put the coefficients from the first bucket of H_2 (*i.e.,* a_i's such that i falls in the first bucket) in the first n locations $a'_1, \ldots, a'_n$, and pad them with zero coefficients to have a block of length n. Put the coefficients from the second bucket of H_2 in the next n locations $a'_{n+1}, \ldots, a'_{2n}$, and pad those also with zeros to have a block of length n. Continue in the same way for all the 2^s buckets.

Assuming all the buckets of H_2 indeed are non-empty, the new coefficient vector $a'_1, \ldots, a'_{n2^s}$ fulfills the conditions of Lemma 8. Thus, we can use $\tilde{G_2}$, and since $s \geq 12 \log M$, we get that:

$$\mathrm{bias}_{\tilde{G_2}}(a'_1, \ldots, a'_{n2^s}) \leq 0.9 \quad (\texttt{given that all buckets are non-empty})$$

There is a probability of at most 0.01 that there is some empty bucket. In this case, we can bound the bias by 1. Thus, we have that

$$\mathrm{bias}_{\tilde{G_2}}(a'_1, \ldots, a'_{n2^s}) \leq 0.9 + 0.01 = 0.91$$

We now re-order the pseudorandom bits to the original order. Since the ordering of $a'_1, \ldots, a'_{n2^s}$ was oblivious of their values (and only relied on H_2), we can obliviously reorder the coefficients and their corresponding pseudorandom bits back to the original order, while keeping the same bias. Thus we get that:

$$\mathrm{bias}_{G_2}(a_1, \ldots, a_n) \leq 0.9 + 0.01 = 0.91$$

We now analyze the randomness required for G_2. The randomness required for H_2 is $O(\log n + \log^2 M)$, and the randomness required for $\tilde{G_2}$ is $O(\log n + \log M)$. Thus we get the total randomness of G_2 is $O(\log n + \log^2 M)$. The above discussion implies the proof of Lemma 7.

4.2.4 Construction of G_3

We now use G_2 to build our final ϵ-bit-biased generator G_3. We prove in this section the following lemma:

Lemma 9. *There exists an explicit generator $G_3 : \{0,1\}^r \to \{0,1\}^n$, such that for all coefficient vectors $a_1, \ldots, a_n$ of weight at least $10^5 M^{24} \log(1/\epsilon)$, we have:*

$$\mathrm{bias}_{G_3}(a_1, \ldots, a_n) \leq \epsilon/2.$$

The randomness required by G_3 is $r = O(\log n + \log(M/\epsilon) \log(M \log(1/\epsilon)))$.

The construction of G_3 has three parts, and uses G_2 as an intermediate construction:

- Use H_3 to partition the inputs to $O(\log(1/\epsilon))$ buckets, such that with probability $1 - \epsilon/100$, most buckets contain at least $100M^{24}$ non-zero coefficients.
- Use G_2 on each bucket.
- Combine the generators for the separate buckets using expander products.

We defer additional details and proof to Appendix B.6.

Acknowledgments

We thank Emanuele Viola for drawing our attention to this problem. We thank Andrej Bogdanov for helpful discussions.

References

[AIK+] M. Ajtai, H. Iwaniec, J. Komlós, J. Pintz, and E. Szemerédi. Construction of a thin set with small Fourier coefficients. *Bull. London Math. Soc.*, 22(6):583–590, 1990.

[AKS] M. Ajtai, J. Komlós, and E. Szemerédi. Deterministic Simulation in LOGSPACE. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 132–140, New York City, 25–27 May 1987.

[AGM] N. Alon, Z. Galil, and V. D. Milman. Better expanders and superconcentrators. *J. Algorithms*, 8(3):337–347, 1987.

[AM1] N. Alon and Y. Mansour. ϵ-discrepancy sets and their application for interpolation of sparse polynomials. *Information Processing Letters*, 54(6):337–342, 1995.

[AM2] N. Alon and V. D. Milman. λ_1, isoperimetric inequalities for graphs, and superconcentrators. *J. Combin. Theory Ser. B*, 38(1):73–88, 1985.

[AR] N. Alon and Y. Roichman. Random Cayley graphs and expanders. *Random Structures Algorithms*, 5(2):271–284, 1994.

[BNS] L. Babai, N. Nisan, and M. Szegedy. Multiparty protocols, pseudorandom generators for logspace, and time-space trade-offs. *Journal of Computer and System Sciences*, pages 204–232, 15–17 May 1989.

[BR] M. Bellare and J. Rompel. Randomness-Efficient Oblivious Sampling. In *35th Annual Symposium on Foundations of Computer Science*, pages 276–287, Santa Fe, New Mexico, 20–22 Nov. 1994. IEEE.

[BSVW] E. Ben-Sasson, M. Sudan, S. Vadhan, and A. Wigderson. Randomness-efficient low degree tests and short PCPs via epsilon-biased sets. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, pages 612–621 (electronic), New York, 2003. ACM.

[BV] A. Bogdanov and E. Viola. Pseudorandom Bits for Polynomials. In *FOCS*, pages 41–51. IEEE Computer Society, 2007.

[EGL+] G. Even, O. Goldreich, M. Luby, N. Nisan, and B. Veličković. Efficient approximation of product distributions. *Random Structures Algorithms*, 13(1):1–16, 1998.

[GG] O. Gabber and Z. Galil. Explicit Constructions of Linear-Sized Superconcentrators. *J. Comput. Syst. Sci.*, 22(3):407–420, June 1981.

[HPS] J. Håstad, S. Phillips, and S. Safra. A well-characterized approximation problem. *Information Processing Letters*, 47(6):301–305, 1993.

[INW] R. Impagliazzo, N. Nisan, and A. Wigderson. Pseudorandomness for Network Algorithms. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 356–364, Montréal, Québec, Canada, 23–25 May 1994.

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

[Kat] N. M. Katz. An estimate for character sums. *Journal of the American Mathematical Society*, 2(2):197–200, 1989.

[Lov] S. Lovett. Unconditional pseudorandom generators for low degree polynomials. In R. E. Ladner and C. Dwork, editors, *STOC*, pages 557–562. ACM, 2008.

[LPS] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[Mar1] G. A. Margulis. Explicit constructions of expanders. *Problemy Peredači Informacii*, 9(4):71–80, 1973.

[Mar2] G. A. Margulis. Explicit group-theoretic constructions of combinatorial schemes and their applications in the construction of expanders and concentrators. *Problemy Peredachi Informatsii*, 24(1):51–60, 1988.

[MZ] R. Meka and D. Zuckerman. Personal communication, 2008.

[Mor] M. Morgenstern. Existence and explicit constructions of $q + 1$ regular Ramanujan graphs for every prime power q. *J. Combin. Theory Ser. B*, 62(1):44–62, 1994.

[MST] E. Mossel, A. Shpilka, and L. Trevisan. On ϵ-biased generators in NC^0. *Random Structures Algorithms*, 29(1):56–81, 2006.

[MNN] R. Motwani, J. Naor, and M. Naor. The probabilistic method yields deterministic parallel algorithms. 49(3):478–516, 1994.

[NN] J. Naor and M. Naor. Small-Bias Probability Spaces: Efficient Constructions and Applications. *SIAM J. Comput.*, 22(4):838–856, Aug. 1993.

[Nao] M. Naor. Constructing Ramsey graphs from small probability spaces. Technical Report RJ 8810, IBM Research Report, 1992.

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

[RSW] A. Razborov, E. Szemerédi, and A. Wigderson. Constructing small sets that are uniform in arithmetic progressions. *Combinatorics, Probability and Computing*, 2(4):513–518, 1993.

[RTV] O. Reingold, L. Trevisan, and S. Vadhan. Pseudorandom Walks in Regular Digraphs and the RL vs. L Problem. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC '06)*, 21–23 May 2006. Preliminary version on *ECCC*, February 2005.

[RVW] O. Reingold, S. Vadhan, and A. Wigderson. Entropy Waves, the Zig-Zag Graph Product, and New Constant-Degree Expanders. *Annals of Mathematics*, 155(1), January 2001. Extended abstract in Proc. of *FOCS '00*.

[RV] E. Rozenman and S. Vadhan. Derandomized Squaring of Graphs. In *Proceedings of the 8th International Workshop on Randomization and Computation (RANDOM '05)*, number 3624 in Lecture Notes in Computer Science, pages 436–447, Berkeley, CA, August 2005. Springer.

[SZ] M. Saks and S. Zhou. $\mathrm{BP_HSPACE}(S) \subseteq \mathrm{DSPACE}(S^{3/2})$. *Journal of Computer and System Sciences*, 58(2):376–403, 1999. 36th IEEE Symposium on the Foundations of Computer Science (Milwaukee, WI, 1995).

[Vio] E. Viola. The Sum of d Small-Bias Generators Fools Polynomials of Degree d. In *IEEE Conference on Computational Complexity*, pages 124–127. IEEE Computer Society, 2008.

A Additional Definitions and Tools

Logarithms will always be taken in base 2, and we assume error terms ϵ are always of the form $\epsilon = 2^{-2^e}$, so $\log\log(1/\epsilon)$ will always be an integer. For general error terms, this can be achieved by at most squaring the error term, which will result in an additional constant multiplicative factor in the seed length of our constructions.

Definition 9 (Statistical distance)**.** The statistical distance between two random variables X, Y taking values in $\mathbb{Z}_M$ is

$$dist(X,Y) = \frac{1}{2}\sum_{i=0}^{M-1} |\Pr[X=i] - \Pr[Y=i]|.$$

The variables X and Y are said to be ϵ-close if their statistical distance is at most ϵ.

A.1 Proof of Lemma 1

We will need the following basic facts regarding Fourier coefficients for the proof of Lemma 1:

Definition 10 (Fourier coefficients)**.** Let X be a random variable taking values in $\mathbb{Z}_M$. The k-th Fourier coefficient of X is:

$$\hat{X}_k = \mathbb{E}_{X\in\mathbb{Z}_M}[\omega^{kX}]$$

where $\omega = e^{2\pi i/M}$ is a primitive M-th root of unity

Fact 10.

- *For any distribution X, $\hat{X}_0 = 1$*
- *(Parseval identity) For any two distributions X, Y taking values in $\mathbb{Z}_M$,*

$$\sum_{k=0}^{M-1} (\mathbf{P}[X=k] - \mathbf{P}[Y=k])^2 = \frac{1}{M}\sum_{k=0}^{M-1} (\hat{X}_k - \hat{Y}_k)^2$$

Proof of Lemma 1. Let $a_1, ..., a_n \in \mathbb{Z}_M$ be a coefficient vector. Let $Y = \sum a_i X_i$ *modulo* M and $S = \sum a_i R_i$ *modulo* M, where $R_1, ..., R_n$ are uniform and independent bits. We need to prove that the statistical distance between Y and S is at most $\epsilon\sqrt{M}$. Consider the k-th Fourier coefficients of Y and S, for $k \in \mathbb{Z}_M$:

$$\hat{Y}_k = \mathbb{E}[\omega^{kY}] = \mathbb{E}[\omega^{k\sum a_i X_i}]$$

and similarly $\hat{S}_k = \mathbb{E}[\omega^{k\sum a_i R_i}]$. Applying the assumption that X is an ϵ-bit-biased distribution modulo M for the coefficient vector $ka_1, ..., ka_n$, we have that $|\hat{Y}_k - \hat{S}_k| \leq \epsilon$. Using the Cauchy-Schwartz inequality and the Parseval identity, we bound the statistical distance between Y and S:

$$\begin{aligned} dist(Y,S) =& \frac{1}{2}\sum_{k=0}^{M-1} |\mathbf{P}[Y=k] - \mathbf{P}[S=k]| \\ &\leq \sqrt{M}\sqrt{\sum_{k=0}^{M-1} |\mathbf{P}[Y=k] - \mathbf{P}[S=k]|^2} \\ &= \sqrt{M}\sqrt{\frac{1}{M}\sum_{k=0}^{M-1} |\hat{Y}_k - \hat{S}_k|^2} \leq \epsilon\sqrt{M} \end{aligned}$$

□

Lemma 11. *Fix a weight threshold W. Let $X' = (X'_1, ..., X'_n)$ be a distribution over $\{0,1\}^n$ such that for any vector coefficient $a_1, ..., a_n$ of weight at most W,*

$$|\text{bias}_{X'}(a_1, ..., a_n) - \text{bias}_{U_n}(a_1, ..., a_n)| \leq \epsilon.$$

Let $X'' = (X''_1, ..., X''_n)$ be a distribution over $\{0,1\}^n$ such that for any vector coefficient $a_1, ..., a_n$ of weight at least W,

$$|\text{bias}_{X'}(a_1, ..., a_n) - \text{bias}_{U_n}(a_1, ..., a_n)| \leq \epsilon.$$

Let X be the bitwise-XOR of two independent copies of X' and X'', i.e.

$$X = X' \oplus X'' = (X'_1 \oplus X''_1, ..., X'_n \oplus X''_n).$$

Then X is ϵ-bit-biased against sums modulo M.

Proof. Let $a_1, ..., a_n$ be a coefficient vector of weight w. We need to prove that

$$|\text{bias}_X(a_1, ..., a_n) - \text{bias}_{U_n}(a_1, ..., a_n)| \leq \epsilon$$

Assume that $w \leq W$ (the proof for the other case is analogous). Then:

$$|\text{bias}_{X'}(a_1, ..., a_n) - \text{bias}_{U_n}(a_1, ..., a_n)| \leq \epsilon$$

Let $x'' = (x''_1, ..., x''_n)$ be any value for X'', and let $A = \{i : x''_i = 1\}$. We have:

$$\text{bias}_X(a_1, ..., a_n) = \mathbb{E}\left[\omega^{\sum a_i (X'_i \oplus X''_i)}\right] = \mathbb{E}\left[\omega^{\sum_{i\notin A} a_i X'_i + \sum_{i\in A} a_i (1 - X'_i)}\right]$$

where we used the simple identity $x \oplus 1 = 1 - x$. Let $b_1, ..., b_n$ be a coefficient vector defined by $b_i = a_i$ for $i \notin A$ and $b_i = -a_i$ for $i \in A$. Thus:

$$\text{bias}_{X' \oplus x''}(a_1, ..., a_n) = \omega^{\sum_{i\in A} a_i} \text{bias}_{X'}(b_1, ..., b_n)$$

Similarly, since we can write the uniform distribution as $U_n \oplus x''$, we have

$$\text{bias}_{U_n}(a_1, ..., a_n) = \text{bias}_{U_n \oplus x''}(a_1, ..., a_n) = \omega^{\sum_{i\in A} a_i} \text{bias}_{U_n}(b_1, ..., b_n)$$

Thus:

$$\begin{aligned}
&|\mathrm{bias}_{X'\oplus x''}(a_1,...,a_n) - \mathrm{bias}_{U_n}(a_1,...,a_n)| = \\
&\left|\omega^{\sum_{i\in A} a_i}\left(\mathrm{bias}_{X'}(b_1,...,b_n) - \mathrm{bias}_{U_n}(b_1,...,b_n)\right)\right| = \\
&|\mathrm{bias}_{X'}(b_1,...,b_n) - \mathrm{bias}_{U_n}(b_1,...,b_n)| \leq \epsilon
\end{aligned}$$

where the last inequality follows because the weight of $b_1,...,b_n$ is equal to the weight of $a_1,...,a_n$, and so is also at most W. Taking average over $x'' \in X''$ we get the required inequality. □

A.2 Convergence of the bias for large weights

The bias of a coefficient vector with respect to the uniform distribution can be large if there are only a few non-zero elements in the vector. However, when the weight is large, the bias is guaranteed to be small.

Lemma 12. *Let $a_1,...,a_n$ be a linear combination of weight w. Then*

$$|bias_U(a_1,...,a_n)| \leq \left(1 - \frac{1}{M^2}\right)^w$$

In particular, for $w \geq M^2 \log 1/\epsilon$ the bias is at most $\epsilon/2$.

Proof. By symmetry, we can assume w.l.o.g that $a_1,...,a_t$ are the non-zero elements in the linear combination.

$$|bias_U(a_1,...,a_n)| = |\mathbb{E}_{x_1,...,x_n\in\{0,1\}^n}[\omega^{a_1x_1+...+a_nx_n}]| = \prod_{i=1}^{t} |\mathbb{E}_{x\in\{0,1\}}[\omega^{a_ix}]|$$

Thus it is enough to bound $\mathbb{E}_{x\in\{0,1\}}[\omega^{ax}]$ for a non-zero element a. We have:

$$|\mathbb{E}_{x\in\{0,1\}}[\omega^{ax}]| = |\frac{1+\omega^a}{2}| = |cos(\frac{a}{M}\pi)| \leq cos(\frac{\pi}{M}) \leq 1 - \frac{1}{M^2}.$$

The last inequality follows since $1 - \frac{1}{M^2} \leq e^{-1/M^2}$. □

A similar result holds if we consider the bias of a large weight coefficient vector under a skewed distribution.

Lemma 13. *Let $a_1,...,a_n$ be a linear combination of weight w. Let $Z_1,\ldots,\mathbb{Z}_n \in \{0,1\}$ be independently distributed with $\Pr[Z_i = 0] = \frac{1+\alpha}{2}$, where $|\alpha| < 1$. Then*

$$|bias_{Z_1,...,Z_n}(a_1,...,a_n)| \leq \left(1 - \frac{O(\alpha)}{M^2}\right)^w$$

In particular, for $w \geq cM^2 \log 1/\epsilon/\alpha$ for a sufficiently large constant c, the bias is at most $\epsilon/2$.

A.3 Hashing

We will use the following three constructions of hash functions.

Lemma 14. *Assume k is a power of 2. There exists a hash function $H_1 : [n] \to [k]$ with the following properties. Let $S \subset [n]$ be of size at most $k \log(1/\epsilon)$. The j-th bucket of H_1 with respect to S is said to be bad if it contains more than $100 \log(1/\epsilon)$ elements. H_1 is said to be bad if at least one bucket is bad. Then H_1 is bad with probability at most $\epsilon/100$. Moreover, H_1 can be generated explicitly and efficiently using $O(\log n + \log(k/\epsilon) \log(k \log(1/\epsilon)))$ random bits.*

Lemma 15. *Assume k is a power of 2. There exists a hash function $H_2 : [n] \to [k]$ with the following properties. Let $S \subset [n]$ be of size at least $100k^2$. The j-th bucket of H_2 with respect to S is said to be bad if it is empty. H_2 is said to be bad if at least one bucket is bad. Then H_2 is bad with probability at most $1/100$. Moreover, H_2 can be generated explicitly and efficiently using $O(\log n + \log^2 k)$ random bits.*

Lemma 16. *There exists a hash function $H_3 : [n] \to [16 \log(1/\epsilon)]$ with the following properties. Let $S \subset [n]$ be of size at least $800k \log(1/\epsilon)$. The j-th bucket of H_3 with respect to S is said to be bad if it contains at most k elements. H_3 is said to be bad if there are at least $\log(1/\epsilon)$ bad buckets. Then the probability for H_3 being bad is at most $\epsilon/100$. Moreover, H_3 can be generated explicitly and efficiently using $O(\log n + \log(1/\epsilon) \log(k \log(1/\epsilon)))$ random bits.*

The constructions of the hashes in Lemmas 14, 15 and 16 are based on almost t-wise independence. We define those and prove tail bounds in the following subsection.

A.3.1 Tail bounds on almost t-wise independent variables

We start by defining t-wise independent random variables.

Definition 11 (t-wise independence)**.** A sequence of random variables $X_1, \ldots, X_n \in \{0, 1\}$ is said to be *t-wise independent* if any t random variables in it are independent. That is, for any distinct $i_1, \ldots, i_t \in [n]$,

$$\Pr[X_{i_1} = a_1, \ldots, X_{i_t} = a_t] = \prod_{j=1}^{t} \Pr[X_{i_j} = a_j]$$

Bellare and Rompel [BR] proved tail bounds on t-wise independent distributions:

Lemma 17 (Lemma 2.3 in [BR])**.** *Let $X_1, \ldots, X_n \in \{0, 1\}$ be t-wise independent random variables, for $t \geq 4$ an even integer. Let $X = \sum X_i$ and $\mu = \mathbb{E}[X]$. Then for any $A > 0$:*

$$\Pr[|X - \mu| \geq A] \leq 8 \left(\frac{t\mu + t^2}{A^2} \right)^{t/2}.$$

We will need a version of Lemma 17 for random variables which are *almost t-wise independent*.

Definition 12 (almost t-wise independence)**.** A sequence of random variables $X_1, \ldots, X_n \in \{0, 1\}$ is said to be δ-almost t-wise independent if any t random variables are δ-close in statistical distance to independent. That is, for any distinct $i_1, \ldots, i_t \in [n]$, Let $X'_{i_1}, \ldots, X'_{i_t}$ be independent random variables, such that X'_{i_j} and X_{i_j} are identically distributed. Then

$$dist((X_{i_1}, \ldots, X_{i_t}), (X'_{i_1}, \ldots, X'_{i_t})) \leq \delta$$

Lemma 18. *Let $X_1, \ldots, X_n \in \{0,1\}$ be δ-almost t-wise independent random variables, for $t \geq 4$ an even integer. There exists a global constant c such that the following holds. Let $X = \sum X_i$ and $\mu = \mathbb{E}[X]$. Then for any $A > 0$:*

$$\Pr[|X - \mu| \geq A] \leq 8\left(\frac{t\mu + t^2}{A^2}\right)^{t/2} + (n+\mu)^t\delta$$

Proof sketch. The proof of Lemma 17 in [BR] is done by bounding the t-th moment of $X - \mu$. Let $X'_1, \ldots, X'_n$ be independent random variables, where X'_i is distributed the same as X_i. Bellare and Rompel [BR] prove:

$$\mathbb{E}[(\sum X'_i - \mu)^t] \leq 8\left(\frac{t\mu + t^2}{A^2}\right)^{t/2}.$$

The LHS is a polynomial in $X'_1, \ldots, X'_n$ of degree t, and the L_1 norm of its coefficient is bounded by $(n+\mu)^t$. Since $X_1, \ldots, X_n$ are δ-almost t-wise independent, we have for every monomial of the polynomial that

$$|\mathbb{E}[\prod X_{i_j}] - \mathbb{E}[\prod X'_{i_j}]| \leq \delta$$

Hence we get that

$$\mathbb{E}[(\sum X_i - \mu)^t] \leq \mathbb{E}[(\sum X'_i - \mu)^t] + (n+\mu)^t\delta,$$

which finishes the proof. □

Distributions which are δ-almost t-wise independent, where each bit is uniform in $\{0,1\}$, can be generated efficiently and explicitly.

Lemma 19. *[NN] There is an explicit generator $G : \{0,1\}^s \to \{0,1\}^n$ which is δ-almost t-wise independent, and each bit in the output of G is uniform, with $s = O(\log n + t + \log(1/\delta))$.*

A.3.2 Proofs of hashing lemmas

We now prove Lemma 14, Lemma 15 and Lemma 16.

Proof of Lemma 14. Construct $H_1 : [n] \to [k]$ by taking $n \log k$ bits, which are δ-almost t-wise independent. We will set $t = 10\log(1/\epsilon)$ if $\epsilon < 1/k$, and $t = 10\log(k/\epsilon)$ otherwise, and we set $\delta = 0.1\epsilon^2/((k+2)\log(1/\epsilon))^t$.

We regard the output bits of H_1 as n numbers, each $\log k$-bits long. These define the images of $[n]$ under H_1. This can be achieved by Lemma 19 using $O(\log n + \log(k/\epsilon)\log(k\log(1/\epsilon)))$ random bits.

Consider the j-th bucket. Let Z_i be the indicator for $i \in S$ that $H_1(i) = j$. If the bits were truly independent, we would have $\Pr[Z_i = 1] = 1/k$. Since $t \geq \log k$ and any t bits are δ-close to independent, we have:

$$\Pr[Z_i = 1] \leq \frac{1}{k} + \delta \leq \frac{2}{k}$$

Let $Z = \sum_{i \in S} Z_i$, and let $\mu = \mathbb{E}[Z]$. We have:

$$\mu \leq 2\log(1/\epsilon)$$

We need to bound $\Pr[Z \geq 10\log(1/\epsilon)]$. It is thus enough to bound the probability that $|Z - \mu| \geq 8\log(1/\epsilon)$. Set $A = 8\log(1/\epsilon)$. We have by Lemma 18 that:

$$\Pr[Z \geq 10\log(1/\epsilon)] \leq 8\left(\frac{t\mu + t^2}{A^2}\right)^{t/2} + |S|^t\delta \leq 0.01\min\{\epsilon/k, \epsilon^2\}$$

by our setting of the parameters.

Thus, union bounding over all k buckets, we get that the probability of having some bad bucket is at most $\epsilon/100$. □

Proof of Lemma 15. Construct $H_2 : [n] \to [k]$ by taking $n \log k$ bits, which are δ-almost t-wise independent, for $t = 100 \log k$ and $\delta = 0.001k^t$. We regard those bits as n numbers, each $\log k$-bits long. These define the images of $[n]$ under H_2. This can be achieved by Lemma 19 using $O(\log n + \log^2 k)$ random bits.

Consider the j-th bucket. Let Z_i be the indicator for $i \in S$ that $H_1(i) = j$. If the bits were truly independent, we would have $\Pr[Z_i = 1] = 1/k$. Since $t \geq \log k$ and any t bits are δ-close to independent, we have:

$$\frac{0.99}{k} \leq \frac{1}{k} - \delta \leq \Pr[Z_i = 1] \leq \frac{1}{k} + \delta \leq \frac{1.01}{k}$$

Let $Z = \sum_{i \in S} Z_i$, and let $\mu = \mathbb{E}[Z]$. We have:

$$\mu \geq 99k$$

We need to bound $\Pr[Z = 0]$. It is thus enough to bound the probability that $|Z - \mu| \geq 99k$. Set $A = 99k$. We have by Lemma 18 that:

$$\Pr[Z = 0] \leq 8 \left(\frac{t\mu + t^2}{A^2} \right)^{t/2} + |S|^t \delta \leq 0.01/k$$

by our setting of the parameters.

Thus, union bounding over all k buckets we get that the probability of having a bad bucket is at most $1/100$.

□

Proof of Lemma 16. Let $\ell = 16 \log(1/\epsilon)$. Construct $H_3 : [n] \to [\ell]$ by taking $n \log \ell$ bits, which are δ-almost t-wise independent, for $t = 100log(1/\epsilon)$ and $\delta = 0.001\epsilon^{20}/(800k \log(1/\epsilon))^t$. We regard those bits as n numbers, each $\log \ell$-bits long. These define the images of $[n]$ under H_3. This can be achieved by Lemma 19 using $O(\log n + \log(1/\epsilon) \log(k \log(1/\epsilon)))$ random bits.

Let $S \subset [n]$ be of size at least $800k \log(1/\epsilon)$. We need to bound the probability that there are at least $\ell/16 = \log(1/\epsilon)$ buckets, each containing at most k elements of S.

Let $T \subset [\ell]$ be a subset of size $\ell/16$. We will upper bound the probability that the number of elements mapped to T is at most $k \log(1/\epsilon)$. Union bounding over all possible T will bound the probability for having $\ell/16$ bad buckets.

Let Z_i be the indicator that the i-th element of S is mapped to T, for $1 \leq i \leq 800k \log(1/\epsilon)$. If the bits where independent, we would have $Pr[Z_i = 1] = 1/16$. Since the bits are δ-almost t-wise independent, and $t \geq \log \ell$ we have:

$$|\Pr[Z_i = 1] - \frac{1}{16}| \leq \delta$$

Let $Z = \sum_{i=1}^{800k \log(1/\epsilon)} Z_i$. Let $\mu = \mathbb{E}[Z]$. We have

$$49 \log(1/\epsilon) \leq \mu \leq 51 \log(1/\epsilon)$$

by our setting of δ.

We need to bound the probability of the event that $Z \leq k\log(1/\epsilon)$. We have

$$\Pr[Z \leq k\log(1/\epsilon)] \leq \Pr[|Z-\mu| \geq 40k\log(1/\epsilon)]$$

Setting $A = 40k\log(1/\epsilon)$, we get by lemma 18 that the probability is bounded by

$$\begin{aligned}\Pr[Z \leq k\log(1/\epsilon)] \leq & 8\left(\frac{t\mu+t^2}{A^2}\right)^{t/2} + (800k\log(1/\epsilon))^t\delta \\ \leq & \epsilon^{20}/100\end{aligned}$$

by our setting of the parameters.

The number of different sets $T \subset [\ell]$ of size $\ell/16$ is at most $2^\ell = \epsilon^{16}$. Thus union bounding over all such T, we get that the probability of H_3 having at least $\ell/16$ bad buckets is at most $\epsilon/100$. □

A.4 Pseudorandom generators for small space

An ingredient in our construction is the small space pseudorandom generator of Impagliazzo, Nisan, and Wigderson [INW]. We first define branching program which is a non-uniform model of small-space computations.

Definition 13 (Branching program)**.** A branching program of length n, degree d and width w is a layered graph with $n+1$ layers, where each layer contains at most w vertices. From each vertex in the i-th layer ($1 \leq i \leq n$) there are d outgoint edges, numbered $0, 1, \ldots, d-1$. A vertex in the first layer is designated as the start vertex. Running the branching program on an input $x_1, \ldots, x_n \in [d]$ is done by following the path according to the inputs, starting at the start vertex. The output of the branching program is the vertex reached in the last layer.

Definition 14 (Pseudorandom generator for branching programs)**.** A pseudorandom generator for branching programs of length n, degree d and width w with error ϵ is a function $G : \{0,1\}^r \to [d]^n$, such that for any branching program of length n, degree d and width w, the statistical distance between the output of the branching program when run on uniform element in $[d]^n$, and the output when run on $G(U_r)$, is at most ϵ.

Lemma 20. *[INW] There exists an explicit pseudorandom generators for branching programs of length n, degree d, width w with error ϵ, which uses $r = O(\log d + \log n(\log(n/\epsilon) + \log w))$ truly random bits.*

B Proofs and Details for Section 4

B.1 Proof of Lemma 3

Proof. Let $a_1, \ldots, a_n$ be a linear combination of weight at most W. Let S be the set of non-zero indices, $S = \{i \in [n] : a_i \neq 0\}$. By our assumption $|S| \leq 10^5 M^{24} \log 1/\epsilon$.

Let b_U be the bias of $a_1, \ldots, a_n$ under the uniform distribution, that is,

$$b_U = \mathbb{E}_{(x_i)_{i\in S} \in U_{|S|}}[\omega^{\sum_{i\in S} a_i x_i}]$$

Let $H_1 : [n] \to [M']$ be given by Lemma 14, where $M' = 10^5 M^{24}$. With probability $1 - \epsilon/100$, each bucket contains at most $100 \log 1/\epsilon$ elements of S. H_1 requires $O(\log n + \log(M/\epsilon)\log(M\log(1/\epsilon)))$ random bits.

Let B_j for $j \in [M']$ denote the elements in the j-th bucket. We assume that $|B_j| \leq 100 \log(1/\epsilon)$ for all $j \in [M']$. We can write the bias as:

$$b_U = \prod_{j \in [M']} \mathbb{E}_{(x_i)_{i \in B_j} \in U_{|B_j|}} [\omega^{\sum_{i \in B_j} a_i x_i}]$$

We now use a pseudorandom generator in each bucket. By Lemma 19, there is an explicit generator $G' : \{0,1\}^s \to \{0,1\}^n$ which is δ-almost t-wise independent, for $t = 100 \log(1/\epsilon)$ and $\delta = 0.01\epsilon/M'$, and each bit in the output of G is uniform. This requires $s = O(\log n + \log M + \log 1/\epsilon)$.

We apply an independent copy of G' to each bucket, using the first ℓ bits of G' if the bucket contains ℓ elements.

The bias given by this generator is

$$b_{G'} = \prod_{j \in [M']} \mathbb{E}_{(x_i)_{i \in B_j} \in G'(U_s)} [\omega^{\sum_{i \in B_j} a_i x_i}]$$

We now compare $b_{G'}$ to b_U. In each bucket, since it contains at most t elements from S, the distribution of the bits which correspond to S is δ-close to uniform. Since the bias is a function with absolute value at most 1, we get that:

$$|\mathbb{E}_{(x_i)_{i \in B_j} \in G'(U_s)} [\omega^{\sum_{i \in B_j} a_i x_i}] - \mathbb{E}_{(x_i)_{i \in B_j} \in U_{|B_j|}} [\omega^{\sum_{i \in B_j} a_i x_i}]| \leq \delta$$

Replacing the buckets one by one, we get that

$$|b_{G'} - b_U| \leq \delta M' < \epsilon/2$$

Now, instead of using an independent seed for G' in each bucket, we derandomize using the INW generator given in Lemma 20. First, we can reorder the coefficients a_i and the corresponding bits X_i such that the bits in each bucket are consecutive. This is allowed since summing modulo M is independent of the order of summation. Thus, we can consider an equivalent model using a branching program. This branching program will be of length M' and width M. Each transition in the branching program corresponds to adding the contribution of a bucket to the total sum modulo M. The degree of the branching program thus corresponds to the number of seeds of G', i.e. $d = 2^s$. In order to fool the bias with error $\epsilon/2$, it suffices to fool this branching program with the same error. By Lemma 20, this can be achieved by a generator G using

$$O(s + \log M(\log M + \log(1/\epsilon))).$$

Thus we have that:

$$|b_G - b_{G'}| < \epsilon/2$$

Combining all of the above, we get an ϵ-bit-biased generator for linear combinations of weight at most $10^5 M^{24} \log 1/\epsilon$ requiring a seed of length

$$O(\log n + \log(M/\epsilon) \log(M \log(1/\epsilon))).$$

□

B.2 Proof of Lemma 5

Proof. Let $u \in \mathbb{C}^{2^r}$ be a complex vector, whose elements are indexed by $\{0,1\}^r$. For $s \in \{0,1\}^r$ assume $G'(s) = (x_1, ..., x_t)$. The s-th coordinate of u is defined to be $u_s = \omega^{\sum a_i x_i}$. Notice that $\text{bias}_{G'(U_r)}(a_1, ..., a_t) = 2^{-r} \sum_s u_s$, and all the elements of u have absolute value 1. Similarly define $v \in \mathbb{C}^{2^r}$. For $s \in \{0,1\}^r$, if $G''(s) = (y_1, ..., y_t)$ then $v_s = \omega^{\sum b_i y_i}$, and $\text{bias}_{G''(U_r)}(b_1, ..., b_t) = 2^{-r} \sum_s v_s$. Let $G = G' \otimes_H G''$, and consider the bias of G on the concatenated coefficient vector $(a_1, ..., a_t, b_1, ..., b_t)$.

$$\text{bias}_{G(U_{r+d})}(a_1, ..., a_t, b_1, ..., b_t) = 2^{-(r+d)} \sum_{s' \sim s''} \omega^{\sum a_i (G'(s'))_i + \sum b_i (G''(s''))_i} = 2^{-(r+d)} \sum_{s' \sim s''} u_{s'} v_{s''}$$

where $\sum_{s' \sim s''}$ is the sum over all neighboring vertices s', s'' in H. Let M_H be the adjacency matrix of H, i.e. M_H is a $2^r \times 2^r$ matrix with $(M_H)_{s',s''} = 1$ if s', s'' are neighbors in H, and $(M_H)_{s',s''} = 0$ otherwise. We thus have that

$$\text{bias}_{G(U_{r+d})}(a_1, ..., a_t, b_1, ..., b_t) = 2^{-(r+d)} u^T M_H v$$

We now use the fact that H is an expander. Decompose $u = \alpha \mathbf{1} + u_\perp$, where $\mathbf{1}$ is the all ones vector, and the sum of the elements of $u_\perp$ is zero. Notice that $\alpha = \text{bias}_{G'(U_r)}(a_1, ..., a_t)$, and $\|u_\perp\|_2^2 = 2^r(1 - |\alpha|^2)$.

Similarly decompose $v = \beta \mathbf{1} + v_\perp$. We also have $\beta = \text{bias}_{G''(U_r)}(b_1, ..., b_t)$ and $\|v_\perp\|_2^2 = 2^r(1 - |\beta|^2)$

We now have:

$$\begin{aligned} |u^T M_H v| =& |(\alpha \mathbf{1} + u_\perp) M_H (\beta \mathbf{1} + v_\perp)| = |\alpha\beta \mathbf{1}^T M_H \mathbf{1} + u_\perp M_H v_\perp| \leq \\ & 2^{r+d}|\alpha||\beta| + \lambda \|u_\perp\|_2 \|v_\perp\|_2 = 2^{r+d} \left(|\alpha||\beta| + \lambda \sqrt{1 - |\alpha|^2} \sqrt{1 - |\beta|^2} \right) \end{aligned}$$

□

B.3 Properties of the function $f(x, y)$ from Lemma 5

Lemma 21. *Let $f(x, y) = f_\lambda(x, y) = xy + \lambda\sqrt{1 - x^2}\sqrt{1 - y^2}$, for $0 \leq x, y \leq 1$. Then:*

1. *If $x = 1$ then $f(x, y) = y$ (and vice versa).*
2. *$f(x, y) \leq xy + \lambda$.*
3. *Assume $\lambda < 1/4$ and let $c < 1/4$. If $x, y \leq 1 - c$ then $f(x, y) \leq 1 - \frac{9}{8}c$.*

Proof. (1) and (2) are immediate. To prove (3) let $x = 1 - a$,$y = 1 - b$. Then:

$$\begin{aligned} 1 - f(x, y) =& 1 - f(1 - a, 1 - b) = \\ & a + b - ab - \lambda\sqrt{2a - a^2}\sqrt{2b - b^2} \geq \\ & a + b - ab - 2\lambda\sqrt{ab} = \\ & \lambda(\sqrt{a} - \sqrt{b})^2 + (1 - \lambda)(a + b) - ab \geq \\ & (1 - \lambda)(a + b) - ab \end{aligned}$$

We now need to show that $g(a,b) = (1-\lambda)(a+b)$ is at least $\frac{9}{8}c$ for $c \le a, b \le 1$. The function g is multilinear, thus its minimum values are attained at the vertices of its domain:

$$g(a,b) \ge \min\{g(c,c), g(c,1), g(1,c), g(1,1)\}$$

It is now an easy calculation to show that each of these terms is at least $\frac{9}{8}c$ given that $c \le 1/4$ and $\lambda \le 1/4$. □

B.4 Proof of Lemma 6

Proof. We can assume w.l.o.g that $n = 2^{\ell-1}q$ by appending zeros to the end of the coefficient vector. We prove by induction on ℓ.

For $\ell = 1$, G_1 is the identity mapping. Let $w \ge 1$ be the weight of $a_1, \ldots, a_n$. By Lemma 12, the bias of $a_1, \ldots, a_n$ is at most $(1 - \frac{1}{M^2})^w$.

We assume by induction the claim for $\ell - 1$, and prove for ℓ. Divide the coefficients into the first and second halves, and define:

$$\alpha = \mathrm{bias}_{G_{\ell-1}}(a_1, \ldots, a_{n/2})$$
$$\beta = \mathrm{bias}_{G_{\ell-1}}(a_{n/2+1}, \ldots, a_n)$$

Either the first half or the second half are not all zeros. Assume w.l.o.g it is the first half. Thus by the induction assumption, $\alpha \le 1 - \frac{1}{M^2}$. We now analyze two cases: either the second half is all zeros, or not. In the first case, we have $\beta = 1$. Thus by Lemma 21 item (1) we have

$$\mathrm{bias}_{G_\ell}(a_1, \ldots, a_n) = \mathrm{bias}_{G_{\ell-1} \otimes_{H_{\ell-1}} G_{\ell-1}}(a_1, \ldots, a_n) \le f_{1/100}(\alpha, 1) = \alpha \le 1 - \frac{1}{M^2}$$

In the latter case, we have by induction $\beta \le 1 - \frac{1}{M^2}$. Thus by Lemma 21 item (3), since $\alpha, \beta \le 1 - \frac{1}{M^2}$ we have

$$\mathrm{bias}_{G_\ell}(a_1, \ldots, a_n) = \mathrm{bias}_{G_{\ell-1} \otimes_{H_{\ell-1}} G_{\ell-1}}(a_1, \ldots, a_n) \le f_{1/100}(\alpha, \beta) \le 1 - \frac{9}{8}\frac{1}{M^2} \le 1 - \frac{1}{M^2}$$

□

B.5 Proof of Lemma 8

Proof. Proof by induction on s. For $s = 0$ this follows immediately from Lemma 6. Assume for $s-1$, and we will prove for s. Divide the coefficient vector into the first and second halves, and define

$$\alpha = \mathrm{bias}_{G'_{\ell+s-1}}(a_1, \ldots, a_{n/2}) \text{ and } \beta = \mathrm{bias}_{G'_{\ell+s-1}}(a_{n/2+1}, \ldots, a_n).$$

Also let $\delta = \min(1 - (9/8)^{s-1}\frac{1}{M^2}, 0.9)$, and let $c = 1 - \delta$. We have by induction that $\alpha, \beta \le \delta$. We have:

$$\mathrm{bias}_{G'_{\ell+s}}(a_1, \ldots, a_n) = \mathrm{bias}_{G'_{\ell+s-1} \otimes_{H_{\ell+s-1}} G'_{\ell+s-1}}(a_1, \ldots, a_n) \le f_{1/100}(\alpha, \beta)$$

First assume $\delta = 0.9$. Then by Lemma 21 item (2) we have:

$$\mathrm{bias}_{G'_{\ell+s}}(a_1, \ldots, a_n) \le f_{1/100}(\alpha, \beta) \le \alpha\beta + 1/100 \le (0.9)^2 + 1/100 \le 0.9$$

Otherwise assume $\delta > 0.9$. Thus $\delta = 1 - (9/8)^{s-1}\frac{1}{M^2}$. By Lemma 21 item (3), since $\alpha, \beta \le \delta$, we have:

$$\mathrm{bias}_{G'_{\ell+s}}(a_1, \ldots, a_n) \le f_{1/100}(\alpha, \beta) \le 1 - (9/8)^s \frac{1}{M^2}$$

□

B.6 Proof of Lemma 9

Proof. The construction of G_3 has three parts, and uses G_2 as an intermediate construction:

- Use H_3 to partition the inputs to $O(\log(1/\epsilon))$ buckets, such that with probability $1 - \epsilon/100$, most buckets contain at least $100M^{24}$ non-zero coefficients.
- Use G_2 on each bucket.
- Combine the generators for the separate buckets using expander products.

First, we use H_3 from Lemma 16 to partition the n inputs to $t = 16\log(1/\epsilon)$ buckets. We have that with probability $1 - \epsilon/100$, there are at least $15\log(1/\epsilon)$ buckets, each with at least $100M^{24}$ non-zero coefficients.

Let $a'_1, \ldots, a'_{nt}$ be a new coefficient vector defined as follows: Put the coefficients from the first bucket of H_3 in the first n locations $a'_1, \ldots, a'_n$, and pad them with zero coefficients to have a block of length n. Put the coefficients from the second bucket of H_3 in the next n locations $a'_{n+1}, \ldots, a'_{2n}$, and pad those also with zeros to have a block of length n. Continue in the same way for all the 2^s buckets.

Consider now applying the ϵ-bit-biased generator G_2 to each bucket, or equivalently, to each of the t blocks of n coefficients in $a'_1, \ldots, a'_{nt}$. Let b_i be the bias of G_2 applied to the i-th bucket/block. Assuming that at least $15\log(1/\epsilon)$ of the buckets contain at least $100M^{24}$ non-zero coefficients, we get by Lemma 7 that for these buckets, $b_i \leq 0.91$. Thus, we get that with probability $1 - \epsilon/100$:

$$\prod_{i=1}^{t} b_i \leq 0.91^{15\log(1/\epsilon)} \leq \epsilon^2$$

We will now use derandomized expander products to derandomize the selection of seeds of G_2 for the different buckets. This will give our construction of G_3.

First, we recall basic facts about expanders. Recall the expanders defined in Lemma 4: for some constant $Q = 2^q$, there exist a sequence H_k of $(Q^k, Q, 1/100)$-expanders. The ℓ power of H_k, H_k^ℓ, is a $(Q^k, Q^\ell, 1/100^\ell)$-expander. Thus, for any required second eigenvalue λ, and for every k, we have explicit expanders on Q^k vertices, with second eigenvalue $1/100^\ell < \lambda$, and degree polynomial in λ. We will use such expander in our construction.

Let $\lambda_1, \lambda_2, \ldots, \lambda_{\log t}$ be defined by

$$\lambda_i = 0.91^{2^i} * 0.01$$

We consider the following combination: Let Q^{k_1} be the number of seeds of G_2. Let $H'_1 = H_{k_1}^{\ell_1}$ be such that its second eigenvalue is at most λ_1. Use H'_1 to combine disjoint pairs: the first and the second block, the third and the fourth block, and so on. Let $Q^{k_2} = Q^{k_1+\ell_1}$ be the seed of the combined generator for a pair. Let $H'_2 = H_{k_2}^{\ell_2}$ be such that its second eigenvalue is at most λ_2, and use H'_2 in the derandomized product of the first and second pair, third and fourth pair, and so on. Continue in this fashion, reducing in each step the number of independent blocks by a factor of 2. After $\log t$ steps, we have composed all the blocks. We now analyze the final bias, and the randomness required.

The bias of the pairs, using H'_1 in the expander product, is

$$f_{\lambda_1}(b_1, b_2), f_{\lambda_1}(b_3, b_4), f_{\lambda_1}(b_5, b_6), f_{\lambda_1}(b_7, b_8), \ldots$$

By Lemma 21, the biases are at most

$$b_1b_2 + \lambda_1, b_3b_4 + \lambda_1, b_5b_6 + \lambda_1, b_7b_8 + \lambda_1, \ldots$$

Continuing in this fashion, we get that the final bias is at most

$$(((b_1b_2 + \lambda_1)(b_3b_4 + \lambda_1) + \lambda_2)((b_5b_6 + \lambda_1)(b_7b_8 + \lambda_1) + \lambda_2) + \lambda_3) \ldots$$

We need to bound this expression. Let c_i be defined as follows: if $b_i \leq 0.91$, then set $c_i = 0.91$, and otherwise set $c_i = 1$. Thus $b_i \leq c_i$ and $\prod_{i=1}^{t} c_i \leq \epsilon^2$. Thus, it is enough to bound

$$(((c_1c_2 + \lambda_1)(c_3c_4 + \lambda_1) + \lambda_2)((c_5c_6 + \lambda_1)(c_7c_8 + \lambda_1) + \lambda_2) + \lambda_3) \ldots$$

By our setting of $\lambda_1, \lambda_2, \ldots$, we have that:

$$\lambda_1 = 0.91^2 0.01 \leq 0.01 c_{2i+1}c_{2i+2} \quad \forall i$$
$$\lambda_2 = 0.91^{2^2} 0.01 \leq 0.01 c_{4i+1}c_{4i+2}c_{4i+3}c_{4i+4} \quad \forall i$$

Thus we get that:

$$c_1c_2 + \lambda_1 \leq 1.01 c_1c_2(c_1c_2 + \lambda_1)(c_3c_4 + \lambda_1) + \lambda_2 \leq 1.01^3 c_1c_2c_3c_4$$

and in total, we get that the combined bias is at most

$$(1.01)^{t-1} \prod_{i=1}^{t} b_i \leq (1.01)^{16\log(1/\epsilon)} (0.91)^{15\log(1/\epsilon)} \leq \epsilon^{-1.8} < \epsilon/4$$

Thus, we get that with probability $1 - \epsilon/100$, the bias of G_3 is at most $\epsilon/4$. Thus, the total bias of G_3 is at most $\epsilon/2$.

We now analyze the randomness requirements for G_3. The randomness required for H_3 is $O(\log n + \log(1/\epsilon)\log(M\log(1/\epsilon)))$. The randomness required for G_2 is $O(\log n + \log^2 M)$, and the random bits required for the derandomized product combination is

$$O(\log \lambda_1 + \log \lambda_2 + \ldots + \log \lambda_{\log t}) =$$
$$O(2 + 2^2 + 2^3 + \ldots + 2^{\log t}) =$$
$$O(2^{\log t}) = O(t) = O(\log 1/\epsilon)$$

So in total, the randomness required for G_3 is

$$O(\log n + \log(M/\epsilon)\log(M\log(1/\epsilon)))$$

□