Generating k-independent variables in constant time

Tobias Christiani
IT University of Copenhagen
tobc@itu.dk

Rasmus Pagh
IT University of Copenhagen
pagh@itu.dk

Abstract—The generation of pseudorandom elements over finite fields is fundamental to the time, space and randomness complexity of randomized algorithms and data structures. We consider the problem of generating k-independent random values over a finite field $\mathbb{F}$ in a word RAM model equipped with constant time addition and multiplication in $\mathbb{F}$, and present the first nontrivial construction of a generator that outputs each value in *constant time*, not dependent on k. Our generator has period length $|\mathbb{F}|\operatorname{poly}\log k$ and uses $k\operatorname{poly}(\log k)\log|\mathbb{F}|$ bits of space, which is optimal up to a $\operatorname{poly}\log k$ factor. We are able to bypass Siegel's lower bound on the time-space tradeoff for k-independent functions by a restriction to sequential evaluation.

I. Introduction

Pseudorandom generators transform a short random seed into a longer output sequence. The output sequence has the property that it is indistinguishable from a truly random sequence by algorithms with limited computational resources. Pseudorandom generators can be classified according to the algorithms (distinguishers) that they are able to fool. An algorithm from a class of algorithms that is fooled by a generator can have its randomness replaced by the output of the generator, while maintaining the performance guarantees from the analysis based on the assumption of full randomness. When truly random bits are costly to generate or supplying them in advance requires too much space, a pseudorandom generator can reduce the time, space and randomness complexity of an algorithm.

This paper presents an explicit construction of a pseudo-random generator that outputs a k-independent sequence of values in *constant time* per value, not dependent on k, on a word RAM [1]. The generator works over an arbitrary finite field that allows constant time addition and multiplication over $\mathbb{F}$ on the word RAM.

Previously, the most efficient methods for generating k-independent sequences were either based on multipoint evaluation of degree $k-1$ polynomials, or on direct evaluation of constant time hash functions. Multipoint evaluation has a time complexity of $O(\log^2 k \log\log k)$ field operations per value while hash functions with constant evaluation time use excessive space for non-constant k by Siegel's lower

bound [2]. We are able to get the best of both worlds: constant time generation and near-optimal seed length and space usage.

Significance: In the analysis of randomized algorithms and in the hashing literature in particular, k-independence has been the dominant framework for limited randomness. Sums of k-independent variables have their jth moment identical to fully random variables for $j \leq k$ which preserves many properties of full randomness. For output length n, $\Theta(\log n)$-independence yields Chernoff-Hoeffding bounds [3] and random graph properties [4], while $\Theta(\operatorname{poly}\log n)$-independence suffices to fool AC^0 circuits [5].

Our generator is particularly well suited for randomized algorithms with time complexity $O(n)$ that use a sequence of k-independent variables of length n, for non-constant k. For such algorithms, the generation of k-independent variables in constant time by evaluating a hash function over its domain requires space $O(n^\epsilon)$ for some constant $\epsilon > 0$. In contrast, our generator uses space $O(k\operatorname{poly}\log k)$ to support constant time generation. Algorithms for randomized load balancing such as the simple process of randomly throwing n balls into n bins fit the above description and presents an application of our generator. Using the bounds by Schmidt et al. [3, Theorem 2] it is easy to show that $\Theta(\log n/\log\log n)$-independence suffices to obtain a maximal load of any bin of $O(\log n/\log\log n)$ with high probability. This guarantee on the maximal load is asymptotically the same as under full randomness. Using our generator, we can allocate each ball in constant time using space $O(\log n \operatorname{poly}\log\log n)$ compared to the lower bound of $O(n^\epsilon)$ of hashing-based approaches to generating k-independence. In Section VI we show how our generator improves upon existing solutions to a dynamic load balancing problem.

The generation of pseudorandomness for Monte Carlo experiments presents another application. Limited independence between Monte Carlo experiments can be shown to yield Chernoff-like bounds on the deviation of an estimator from its expected value. Consider a randomized algorithm $\mathcal{A}(Y)$ that takes m random elements from $\mathbb{F}$ encoded as a string Y and returns a value in the interval $[0, 1]$. Let $\mu_{\mathcal{A}} > 0$ denote the expectation of the value returned by $\mathcal{A}(Y)$ under the assumption that Y encodes a truly

random input and define the estimator $\hat{\mu}_{\mathcal{A}} = \frac{1}{t} \sum_{i=1}^{t} \mathcal{A}(Y_i)$. Due to a result by Schmidt et al. [3, Theorem 5], for every choice of constants $\epsilon, \alpha > 0$, it suffices that $Y_1, \ldots, Y_t$ encodes a sequence of $\Theta(m \log t)$-independent variables over $\mathbb{F}$ to yield the following high probability bound on the deviation of $\hat{\mu}_{\mathcal{A}}$ from $\mu_{\mathcal{A}}$.

$$Pr[|\hat{\mu}_{\mathcal{A}} - \mu_{\mathcal{A}}| \geq \epsilon\mu_{\mathcal{A}}] \leq O(t^{-\alpha}). \tag{1}$$

We hope that our generator can be a useful tool to replace heuristic methods for generating pseudorandomness in applications where theoretical guarantees are important. In order to demonstrate the practicality of our techniques, we present experimental results on a variant of our generator in Section VII. Our experiments show that k-independent values can be generated nearly as fast as output from heuristic pseudorandom generators, even for large k.

Methods: Our construction is a surprisingly simple combination of bipartite unique neigbor expanders with multipoint polynomial evaluation. The basic, probabilistic construction of our generator proceeds in two steps: First we use multipoint evaluation to fill a table with $\Theta(k)$-independent values from a finite field, using an average of $\operatorname{poly} \log k$ operations per table entry. Next we apply a bipartite unique neighbor expander with constant outdegree and with right side nodes corresponding to entries in the table and a left side that is $\operatorname{poly} \log k$ times larger than the right side. For each node in the left side of the expander we generate a k-independent value by returning the sum of its neighboring table entries. Our main result stated in Theorem 1 uses the same idea, but instead of relying on a single randomly constructed expander graph, we employ a cascade of explicit constant degree expanders and show that this is sufficient for constant time generation.

Relation to the literature: Though the necessary ingredients have been known for around 10 years, we believe that a constant time generator has evaded discovery by residing in a blind spot between the fields of hashing and pseudorandom generators. The construction of constant time k-independent *hash functions* has proven to be a difficult task, and a fundamental result by Siegel [2] showed a time-space tradeoff that require hashing-based generators with sequence length n to use $O(n^\epsilon)$ space for some constant $\epsilon > 0$. On the other hand, from the point of view of pseudorandom generators, a generator of k-independent variables, for non-constant k, can not be used as an efficient method of derandomization: A lower bound by Chor et al. [6] shows that the sample space of such generators must be superpolynomial in their output length. Consequently, research shifted towards generators that produce other types of outputs such as biased sequences or almost k-independent variables [7], [8], [9].

It is relevant to ask whether there already exist constructions of constant time pseudorandom generators on the word RAM that can be used instead of generators that output k-independent variables. For example, Nisan's

pseudorandom generator [10] uses constant time to generate a pseudorandom word and has remarkably strong properties: Every algorithm running in $\text{SPACE}(s)$ that uses n random words can have its random input replaced by the output of a constant time generator with seed length $O(s \log n)$. The probability that the outcome of the algorithm differs when using pseudorandomness as opposed to statistical randomness is decreasing exponentially in the seed length.

In spite of this strong result, there are many natural applications where the restrictions on Nisan's model means that we cannot use his generator directly to replace the use of a k-generator. An example is the analysis that uses a union bound over all subsets of k words of a randomly generated structure described by n words. Algorithms shown to be derandomized by Nisan's generator are restricted to one-way access to the output of the generator. Therefore the output of Nisan's generator can not be used to derandomize an algorithm that tests for the events of the union bound without using excessive space. In this case, k-independence can directly replace the use of full randomness without changing the analysis.

A. Our contribution

We present three improved constructions of *k-generators*, formally defined in Section II, that are able to generate a sequence of k-independent values over a finite field $\mathbb{F}$. Our results are stated in a word RAM model equipped with constant time addition and multiplication in $\mathbb{F}$. Our main result is a fully explicit generator:

Theorem 1. *For every finite field $\mathbb{F}$ with constant time arithmetic there exists a data structure that for every choice of $k \leq |\mathbb{F}|/\operatorname{poly}\log|\mathbb{F}|$ is an explicit constant time k-generator. The generator has range $\mathbb{F}$, period $|\mathbb{F}| \operatorname{poly}\log k$, and seed length, space usage and initialization time $k \operatorname{poly}\log k$.*

We further investigate how the space usage and seed length may be reduced by employing a probabilistic construction that has a certain probability of error:

Theorem 2. *For every finite field $\mathbb{F}$ with constant time arithmetic and every choice of positive constants ε, δ there exists a data structure that for every choice of $k = O(|\mathbb{F}|)$ is a constant time k-generator with failure probability δ, range $\mathbb{F}$, period $|\mathbb{F}|$, seed length $O(k)$, space usage $O(k \log^{2+\varepsilon} k)$, and initialization time $O(k \operatorname{poly}\log k)$.*

Finally, we improve existing k-generators with optimal space complexity:

Theorem 3. *For every finite field $\mathbb{F}$ that supports computing the discrete Fourier transform of length k in $O(k \log k)$ operations, there exists a data structure that, for every choice of $k \leq |\mathbb{F}|$ and given a primitive element ω, is an explicit $O(\log k)$ time k-generator with range $\mathbb{F}$, period $|\mathbb{F}|$, seed length k, space usage $O(k)$, and initialization time $O(k \log k)$.*

Construction	Time	Space	Seed length	Comment
Polynomials [11], [12]	$O(k)$	$O(k)$	k	Assumes $O(1)$ mult.
Multipoint [13, ch. 10.1]	$O(\log^2 k \log \log k)$	$O(k \log k)$	k	
Multipoint [14]	$O(\log k \log \log k)$	$O(k)$	k	Requires ω.
Siegel [2]	$O(1)$	$O(|\mathbb{F}|^\varepsilon)$	$O(k)$	Probabilistic.
Theorem 1	$O(1)$	$O(k \operatorname{poly} \log k)$	$O(k \operatorname{poly} \log k)$	Explicit.
Theorem 2	$O(1)$	$O(k \log^{2+\varepsilon} k)$	$O(k)$	Probabilistic.
Theorem 3	$O(\log k)$	$O(k)$	k	Requires ω_k and FFT.

Table notes: We use ε to denote an arbitrary positive constant and ω and ω_k to denote, respectively, a primitive element and a kth root of unity of $\mathbb{F}$. The unit for space and seed length is the number of elements of $\mathbb{F}$ that need to be stored, i.e., a factor $\log_2 |\mathbb{F}|$ from the number of bits. Probabilistic constructions rely on random generation of objects for which no explicit construction is known, and may fail with some probability.

Table I summarizes our results along with previous methods of generating sequences of k-independent values over $\mathbb{F}$. All the methods output sequences that have a length of at least $|\mathbb{F}|$.

Overview of paper: In Section II we define k-generators and related concepts and review results that lead up to our main results. Section III presents the details of our explicit construction of constant time generators. In Section IV we apply the same techniques with a probabilistic expander construction to obtain generators with improved space and randomness complexity. Section V presents an algorithm for evaluating a polynomial over all elements of $\mathbb{F}$ that improves existing generators with optimal space. Section VI applies our generator to improve the time-space tradeoff of previous solutions to a load balancing problem. Section VII presents experimental results on the generation time of different k-generators for a range of values of k.

II. PRELIMINARIES

We begin by defining two fundamental concepts:

Definition 1. A sequence $(X_1, X_2, \ldots, X_n)$ of n random variables with finite range R is an (n, k)-*sequence* if the variables at every set of k positions in the sequence are independent and uniformly distributed over R.

Definition 2. A family of functions $\mathcal{F} \subseteq \{f \mid f : U \to R\}$ is k-*independent* if for every set of k distinct inputs $x_1, x_2, \ldots, x_k$ it holds that $f(x_1), f(x_2), \ldots, f(x_k)$ are independent and uniformly distributed over R when f is selected uniformly at random from $\mathcal{F}$. We say that a function f selected uniformly at random from $\mathcal{F}$ is a k-*independent function*.

We now give a formal definition of the generator data structure.

Definition 3. A k-*generator* with range R, period n and failure probability δ is a data structure with the following properties:

- It supports an initialization operation that takes a random seed s as input.
- After initialization it supports an `emit()` operation that returns a value from R.
- There exists a set B such that $\Pr[s \in B] \leq \delta$ and conditioned on $s \notin B$ the sequence $(X_1, X_2, \ldots, X_n)$ of values returned by `emit()` is an (n, k)-sequence.

A k-generator is *explicit* if the initialization and emit operation has time complexity $\operatorname{poly} k$ and the probability of failure is zero. We refer to a k-generator as a constant time k-generator if the `emit()` operation has time complexity $O(1)$, not dependent on k.

A k-generator differs from a data structure for representing a k-independent hash function by only allowing sequential access to the underlying (n, k)-sequence. It is this restriction on generators that allows us to obtain a better time-space tradeoff for the problem of generating k-independent variables than is possible by using a k-independent hash function directly as a generator. We are interested in the following parameters of k-generators: seed length, period, probability of failure, space needed by the data structure, the time complexity of the initialization operation and the time complexity of a single `emit()` operation.

Model of computation: Our results are stated in the word RAM model of computation with word length $w = \Theta(\log |\mathbb{F}|)$ bits. In addition to the standard bit manipulation and integer arithmetic instructions, we also assume the ability to perform arithmetic operations $(+, -, \times)$ over $\mathbb{F}$ in constant time. In the context of our results that use abelian groups $(A, +)$ we assume that an element of A can be stored in a constant number of words and that addition can be performed in constant time.

Let $\mathbb{F}_q$ denote a field of cardinality $q = p^z$ for p prime and z a positive integer. Constant time arithmetic in $\mathbb{F}_p$ is supported on a standard word RAM with integer multiplication [15]. The full version of this paper presents

additional details on algorithms for arithmetic over $\mathbb{F}_p$ and $\mathbb{F}_{2^w}$ and how they relate to a standard word RAM with integer multiplication.

A. k-independent functions from the literature

We now review the literature on k-independent functions and how they can be used to construct k-generators. We distinguish between a k-independent function $f : U \to R$ and a k-independent hash function by letting the latter refer to a data structure that after initialization supports random access to the (n, k)-sequence defined by evaluating f over U. There exists an extensive literature that focuses on how to construct k-independent hash functions that offer a favorable tradeoff between representation space and evaluation time [16]. We note that a family of k-independent hash functions can be used to construct a k-generator by setting the seed to a random function in the family.

Constant time k-independent hash functions: A fundamental cell probe lower bound by Siegel [2] shows that a data structure to support constant time evaluation of f on every input in U cannot use less than $\Omega(|U|^\epsilon)$ space for some constant $\epsilon > 0$. This bound holds even for amortized constant evaluation time over functions in the family and elements in the domain. From Siegel's lower bound, it is clear that we cannot use k-independent hash functions directly to obtain a constant time k-generator that uses only $O(k \operatorname{poly} \log k)$ words of space.

Known constructions of k-independent hash functions with constant evaluation time are based on expander graphs. Siegel [2] gave a probabilistic construction of a family of k-independent hash functions in the word RAM model based on an iterated product of bipartite expander graphs. Thorup [17] showed that a simple tabulation hash function with high probability yields the type of expander graphs required by Siegel's construction. Unfortunately only randomized constructions of the expanders required by these hash functions is known, introducing a positive probability of error in k-generators based on them.

Polynomials: Here we briefly review the classic construction of k-independent functions based on polynomials over finite fields.

Lemma 1 (Joffe [11], Carter and Wegman [12])**.** *For every choice of finite field $\mathbb{F}$ and every $k \leq |\mathbb{F}|$, let $\mathcal{H}_k \subset \mathbb{F}[X]$ be the family of polynomials of degree at most $k - 1$ over $\mathbb{F}$. $\mathcal{H}_k \subset \{f \mid f : \mathbb{F} \to \mathbb{F}\}$ is a family of k-independent functions.*

An advantage of using families of polynomials as hash functions is that they use near optimal randomness, allow any choice of $k \leq |\mathbb{F}|$, and have no probability of failure. It can also be noted that in the case where $k = O(\log |\mathbb{F}|)$ and we are restricted to linear space $O(k)$, polynomial hash functions evaluated using Horner's scheme are optimal k-independent hash functions [18], [2].

Using slightly more space and for sufficiently large k, a data structure by Kedlaya and Umans [19] supports evaluation of a polynomial of degree k over $\mathbb{F}$. The space usage and preprocessing time of their data structure is $k^{1+\epsilon} \log^{1+o(1)} |\mathbb{F}|$ for constant $\epsilon > 0$. After preprocessing a polynomial f, the data structure can evaluate f in an arbitrary point of $\mathbb{F}$ using time $\operatorname{poly}(\log k) \log^{1+o(1)} |\mathbb{F}|$.

Multipoint evaluation: Using algorithms for multipoint evaluation of polynomials we are able to obtain a k-generator with $\operatorname{poly} \log k$ generation time and space usage that is linear in k. Multipoint evaluation of a polynomial $f \in \mathbb{F}[X]$ of degree at most $k - 1$ in k arbitrary points of $\mathbb{F}$ has a time complexity of $O(k \log^2 k \log \log k)$ in the word RAM model that supports field operations [13, Corollary 10.8]. Bostan and Schost [14] mention an algorithm for multipoint evaluation of f over a geometric progression of k elements with running time $O(k \log k \log \log k)$. In order to use this method to construct a k-generator with period $|\mathbb{F}|$ it is necessary to know a primitive element ω of $\mathbb{F}_q$ so we can perform multipoint evaluation over $\mathbb{F}^* = \{\omega^0, \omega^1, \ldots, \omega^{q-2}\}$. Given the prime factorization of $q-1$ there exists a Las Vegas algorithm for finding ω with expected running time $O(\log^4 q)$ [20, Chapter 11]. In the following lemma we summarize the properties of k-generators based on multipoint evaluation of polynomials over finite fields.

Lemma 2 (Gathen and Gerhard [13, Corollary 10.8], Bostan and Schost [14])**.** *For every finite field $\mathbb{F}$ there exists for every $k \leq |\mathbb{F}|$ and bijection $\pi : [|\mathbb{F}|] \to \mathbb{F}$ an explicit k-generator with period $|\mathbb{F}|$ and seed length k. The space required by the generator and the initialization and generation time depends on the choice of π and multipoint evaluation algorithm.*

– *For arbitrary choice of π there exists a k-generator with generation time $O(\log^2 k \log \log k)$, intialization time $O(k \log^2 k \log \log k)$ and space usage $O(k \log k)$.*

– *Given a primitive element ω of $\mathbb{F}$ and a bijection $\pi(i) = \omega^i$ there exists a generator with generation time $O(\log k \log \log k)$, initialization time $O(k \log k \log \log k)$ and space usage $O(k)$.*

Space lower bounds: Since randomness can be viewed as a resource like time and space, we are naturally interested in generators that can output long k-independent sequences using as few random bits as possible. Families of k-independent functions $f : U \to R$ with $U = R$ and $k \leq |U|$ will trivially have to use at least $k \log |U|$ random bits — a bound matched by polynomial hash functions. We are often interested in generators with $|U| \gg |R|$, for example if we wish to use a generator for randomized load balancing in the heavily loaded case. A lower bound by Chor et al. [6] shows that even in this case the minimal seed length required for k-independence is $\Omega(k \log |U|)$ for every $|R| \leq |U|$.

B. Expander graphs

All graphs in this paper are bipartite with cm vertices on the left side, m vertices on the right side and left outdegree d. Graphs are specified by their edge function $\Gamma : [cm] \times [d] \to [m]$ where the notation $[n]$ is used to denote the set $\{0, 1, \ldots, n-1\}$. Let S be a subset of left side vertices. For convenience we use $\Gamma(S)$ to denote the neighbors of S.

Definition 4. The bipartite graph $\Gamma : [cm] \times [d] \to [m]$ is (c, m, d, k)-*unique* (k-unique) if for every $S \subseteq [cm]$ with $|S| \leq k$ there exists $y \in \Gamma(S)$ such that y has a unique neighbor in S. An expander graph is *explicit* if it has a deterministic description and Γ is computable in time polynomial in $\log cm + \log d$.

The performance of our generator constructions are directly tied to the parameters of such expanders. In particular, we would like explicit expanders that simultanously have a low outdegree d, are highly unbalanced and are k-unique for k as close to m as possible. A direct application of a result by Capalbo et al. [21, Theorem 7.1] together with an equivalence relation between different types of expander graphs from Ta-Shma et al. [22, Theorem 8.1] yields explicit constructions of unbalanced unique neighbor expanders.[1]

Lemma 3 (Capalbo et al. [21, Theorem 7.1]). *For every choice of c and m there exists a (c, m, d, k)-unique expander with $d = \operatorname{poly} \log c$ and $k = \Omega(m/d)$. For constant c the expander is explicit.*

We note the following simple technique for constructing a larger k-unique expander from a smaller k-unique expander.

Lemma 4. *Let Γ be a (c, m, d, k)-unique expander with $cm \times m$ adjacency matrix $\mathbf{M}$. For any positive integer b define $\Gamma^{(b)}$ as the bipartite graph with block diagonal adjacency matrix $\mathbf{M}^{(b)} = \operatorname{diag}(\mathbf{M}, \ldots, \mathbf{M})$ with b blocks in the diagonal. Then $\Gamma^{(b)}$ is a (c, bm, d, k)-unique expander.*

From expanders to independence: By associating each right vertex in a (c, m, d, k)-unique expander with a position in a (m, dk)-sequence over an abelian group $(A, +)$, we can generate a (cm, k)-sequence over A. This approach was pioneered by Siegel and has been used in different constructions of families of k-independent hash functions [2], [17].

Lemma 5 (Siegel [2, Lemma 2.6, Corollary 2.11]). *Let $\Gamma : [cm] \times [d] \to [m]$ be a k-unique expander and let $h : [m] \to A$ be a dk-independent function with range an abelian group. Let $g : [cm] \to A$ be defined as*

$$g(x) = \sum_{y \in \Gamma(\{x\})} h(y). \qquad (2)$$

Then g is a k-independent function.

[1]We state the results here without the restriction from [21] that c and m are powers of two. We do this to simplify notation and it only affects constant factors in our results.

III. EXPLICIT CONSTANT TIME GENERATORS

In this section we show how to obtain a constant time k-generator by combining an explicit $\operatorname{poly} k$-generator with a cascading composition of unbalanced unique neighbor expanders. Our technique works by generating a small number of highly independent elements in an abelian group and then successively applying constant degree expanders to produce a greater number of less independent elements. We continue this process up until the point where the final number of elements is large enough to match the cost of generating the smaller batch of highly independent elements.

The generator has two components. The first component is an explicit m-generator $g_0 : [n] \to A$ with period n and range an abelian group A. The second component is an explicit sequence $(\Gamma_i)_{i=1}^{t}$ of unbalanced unique neighbor expanders. The expanders are constructed such that the left side of the ith expander matches the right side of the $(i+1)$th expander. By Lemma 3, for every choice of imbalance c, target independence k and length of the expander sequence t there exists a sequence of expanders with the property that

$$\Gamma_i \text{ is } (c, c^{i-1}m, d, d^{t-i}k)\text{-unique}, \qquad (3)$$

for $m = O(d^t k)$ and $d = \operatorname{poly} \log c$. For constant c each expander in the sequence is explicit.

We now combine the explicit m-generator g_0 and the sequence of expanders $(\Gamma_i)_{i=1}^{t}$ to define the k-independent function g_t. Let $b = m/n$ and assume for simplicity that m divides n. For each Γ_i we use the technique from Lemma 4 to construct a $(c, c^{i-1}n, d, d^{t-i}k)$-unique expander $\Gamma_i^{(b)}$. Let x_i denote a number in $[c^i n]$ corresponding to a vertex in the right side of $\Gamma_i^{(b)}$. We are now ready to give a recursive definition of $g_i : [c^i n] \to A$.

$$g_i(x_i) = \sum_{x_{i-1} \in \Gamma_i^{(b)}(\{x_i\})} g_{i-1}(x_{i-1}), \quad 1 \leq i \leq t. \qquad (4)$$

Lemma 6. *g_i is $d^{t-i}k$-independent.*

Proof: We proceed by induction on i. By definition, $g_0 : [n] \to A$ is $d^t k$-independent. Assume by induction that $g_i : [c^i n] \to A$ is $d^{t-i}k$-independent. By definition $\Gamma_{i+1}^{(b)}$ is a $(c, c^i n, d, d^{t-(i+1)}k)$-unique expander. Applying Lemma 5 we have that $g_{i+1} : [c^{i+1}n] \to A$ is $d^{t-(i+1)}k$-independent. ∎

We will now show that g_t supports fast sequential evaluation and prove that we can use g_t to construct an explicit constant time k-generator from any explicit m-generator, for an appropriate choice of m. Divide the domain of each g_i evenly into $b = n/m$ batches of size $c^i m$ corresponding to each block of the adjacency matrix of Γ_i used to construct $\Gamma_i^{(b)}$ and index the batches by $j \in [b]$. In order to evaluate g_{i+1} over batch number j it suffices to know Γ_{i+1} and the values of g_i over batch number j. Fast sequential evaluation of g_t is achieved in the following steps. First we tabulate

the sequence of expanders $(\Gamma_i)_{i=1}^t$ such that $\Gamma_i(\{x_i\})$ can be read in d operations. Secondly, to evaluate g_t over batch j, we begin by tabulating the output of g_0 over batch j and then successively apply our tabulated expanders to produce tables for the output of $g_1, g_2, \ldots, g_t$ over batch j.

Given tables for the sequence of expanders and assuming that the generator underlying g_0 has been initialized, we now consider the average number of operations used per output when performing batch-evaluation of g_t. The number of values output is $c^t m$. The cost of emitting m values from g_0 is by definition at most $\text{poly}(m)$. The cost of producing tables for the output of $g_1, g_2, \ldots, g_t$ for the current batch is given by $\sum_{i=1}^t dc^i m = O(dc^t m)$ for $c > 1$. The average number of operations used per output when performing batch-evaluation of g_t is therefore bounded from above by

$$\frac{O(dc^t m) + \text{poly}\, m}{c^t m} = O(d) + \frac{\text{poly}\, m}{c^t}. \quad (5)$$

The following lemma states that we can obtain a constant time k-generator from every explicit m-generator by setting $t = O(\log k)$ and choosing c to be an appropriately large constant.

Lemma 7. *Let A be an abelian group with constant time addition. Suppose there exists an explicit m-generator with range A, period n and space usage $\text{poly}\, m$. Then there exists a positive constant ϵ such that for every $k \leq m^\epsilon$ there exists an explicit constant time k-generator with range A, period n, and seed length, space usage and initialization time $\text{poly}\, k$.*

Proof: The sequence of expanders $(\Gamma_i)_{i=1}^t$ with the properties given in (3) exists for $m = O(d^t k)$ and $d = \text{poly} \log c$ and is explicit for c constant. By inserting $m = O(d^t k)$ into equation (5) it can be seen that the average number of operations is constant for $c = O(1)$ and $t = O(\log k)$ with constants that depend on the parameters of the m-generator. The k-generator is initialized by initializing the m-generator, finding and tabulating the sequence of expanders and producing the first batch of values, all of which can be done in $\text{poly}\, k$ time and space. After initialization, each call to `emit()` will return a value from the current batch and use a constant number of operations for the task of preparing the next batch of outputs. ∎

We now show our main theorem about explicit constant time k-generators over finite fields. The construction uses an explicit m-generator based on multipoint evaluation. Combined with the approach of Lemma 7 this yields a near-optimal time-space tradeoff for k-generation.

Theorem 1 (Repeated). *For every finite field $\mathbb{F}$ with constant time arithmetic there exists a data structure that for every choice of $k \leq |\mathbb{F}|/\text{poly} \log |\mathbb{F}|$ is an explicit constant time k-generator. The generator has range $\mathbb{F}$, period $|\mathbb{F}|\, \text{poly} \log k$, and seed length, space usage and initialization time $k\, \text{poly} \log k$.*

Proof: Fix the choice of finite field $\mathbb{F}$. By Lemma 2 there exists an explicit m-generator in $\mathbb{F}$ for $m \leq |\mathbb{F}|$ with period $|\mathbb{F}|$ that uses time $O(m \log^3 m)$ to emit m values. Fix some constant $c > 1$ and let $(\Gamma_i)_{i=1}^t$ denote an explicit sequence of constant degree expanders with the properties given by (3). The average number of operations per k-independent value output by g_t when performing batch evaluation is given by

$$\frac{O(dc^t m) + O(m \log^3 m)}{c^t m} = O(d) + \frac{O(\log^3 d^t k)}{c^t}. \quad (6)$$

Setting $t = O(\log \log k)$ and following the approach of Lemma 7 we obtain a k-generator with the stated properties. ∎

Based on the discussion in a paper by Capalbo [23] that introduces unbalanced unique neighbor expanders for concrete values of c and d, it appears likely that the constants hidden in Theorem 1 for the current best explicit constructions make our explicit generators unsuited for practical use since c is close to 1 when d is reasonably small. The next section explores how randomly generated unique neighbor expanders can be used to show stronger existence results and yield k-generators with tractable constants.

IV. Constant Time Generators with Optimal Seed Length

Randomly constructed expanders of the type used in this paper have stronger properties than known explicit constructions, and can be generated with an overwhelming probability of success. There is no known efficient algorithm for verifying whether a given graph is a unique neighbor expander. Therefore randomly generated expanders cannot be used to replace explicit constructions without some probability of failure.

In this section we apply the probabilistic method to show the existence of k-generators with better performance characteristics than those based on known explicit constructions of expanders. We are able to show the existence of constant time generators with optimal seed length that use $O(k \log^{2+\varepsilon} k)$ words of space for any constant $\varepsilon > 0$. Furthermore, such generators can be constructed for any choice of constant failure probability $\delta > 0$. The generators we consider in this section use only a single expander graph but are otherwise identical to the generators described in Section III. Using a single expander graph suffices for constant time generation because the probabilistic constructions are powerful enough to support an imbalance of $c = \text{poly} \log k$ while maintaining constant degree. This imbalance is enough to amortize the cost of multipoint evaluation in a single expansion step as opposed to the sequence of explicit expanders employed in Theorem 1. Our arguments are a straightforward application of the probabilistic method, but we include them for completeness and because we are interested in somewhat nonstandard parameters.

We consider the following randomized construction of a (c, m, d, k)-unique expander Γ. For each vertex x in $[cm]$, we add an edge between x and each distinct node of d nodes selected uniformly at random from $[m]$. By a standard argument, the graph can only fail to be unique neighbor expander if there exists a subset S of left hand side vertices with $|S| \leq k$ such that $|\Gamma(S)| \leq \lfloor d|S|/2 \rfloor$ [2, Lemma 2.8]. In the following we assume that $kd \leq m$.

$$\Pr[\Gamma \text{ is not a unique neighbor expander}]$$
$$\leq \Pr[\exists S \subseteq [cm], |S| \leq k : |\Gamma(S)| \leq \lfloor d|S|/2 \rfloor]$$
$$\leq \sum_{\substack{S \subseteq [cm] \\ |S| \leq k}} \Pr[|\Gamma(S)| \leq \lfloor d|S|/2 \rfloor]$$
$$\leq \sum_{i=1}^{k} \binom{cm}{i} \binom{m}{\lfloor id/2 \rfloor} \left(\frac{\lfloor id/2 \rfloor}{m} \right)^{id}$$
$$\leq \sum_{i=1}^{k} \left(\frac{cme}{i} \right)^{i} \left(\frac{me}{id/2} \right)^{id/2} \left(\frac{id/2}{m} \right)^{id}$$
$$= \sum_{i=1}^{k} \left(cme^{1+d/2} \left(\frac{(d/2)i^{1-1/(d/2)}}{m} \right)^{d/2} \right)^{i} \tag{7}$$

If the expression in the outer parentheses in (7) can be bounded from above by $1/2$ for $i = 1, 2, \ldots, k$, then the expander exists. We also note that the randomized expander construction can be performed using dk-independent variables without changing the result in (7). Let $\gamma > 1$ be a number that may depend on k and let δ denote an upper bound on the probability that the randomized construction fails. By setting $m = O(dk\gamma)$ we are able to obtain the following expression for the relation between δ, the imbalance c and the left outdegree bound d.

$$\delta = e \frac{cd}{\gamma^{d/2-1}} \tag{8}$$

Equation (8) reveals tradeoffs for the parameters of the randomly constructed k-unique expander graphs. For example, increasing γ makes it possible to make the graph more unbalanced while maintaining the same upper bound on the probability of failure δ. The increased imbalance comes at the cost of an increase in m, the size of the right side of the graph. Similarly it can be seen how increasing d can be used to reduce the probability of error. Setting the parameters to minimize the space occupied by the expander while maintaining constant outdegree and by extension constant generation time, we obtain Theorem 2.

Theorem 2 (Repeated). *For every finite field $\mathbb{F}$ with constant time arithmetic and every choice of positive constants ε, δ there exists a data structure that for every choice of $k = O(|\mathbb{F}|)$ is a constant time k-generator with failure probability δ, range $\mathbb{F}$, period $|\mathbb{F}|$, seed length $O(k)$, space usage $O(k \log^{2+\varepsilon} k)$, and initialization time $O(k \operatorname{poly} \log k)$.*

Proof: Let $\tilde{\varepsilon} < \varepsilon$ be a constant and set $\gamma = \log^{\tilde{\varepsilon}} k$. Choosing d to be a sufficiently large constant (dependent on $\tilde{\varepsilon}$), equation (8) shows that for every $\delta > 0$ there exists a (c, m, d, k)-unique expander Γ with $c = \Omega(\log^{2+\varepsilon} k)$ and $m = O(k\gamma)$. Using multipoint evaluation, the right side vertices of Γ can be associated with $\Theta(k)$-independent variables over $\mathbb{F}$ using $O(k \log^{2+\varepsilon} k)$ operations. By the properties of Γ and applying Lemma 5 we are able to generate batches of k-independent variables of size $\Omega(k \log^{2+\varepsilon} k)$ using $O(k \log^{2+\varepsilon})$ operations. The seed length of $O(k)$ holds by the observation that randomized construction of the expander only requires $O(k)$-independence. The $O(k \operatorname{poly} \log k)$ initialization time is obtained by using multipoint evaluation to construct a table for Γ. ∎

V. FASTER MULTIPOINT EVALUATION FOR k-GENERATORS

This section presents an improved generator based directly on multipoint evaluation of a polynomial hash function $h \in \mathcal{H}_k$ over a finite field. For our purpose of generating an (n, k)-sequence from h, we are free to choose the order of elements of $\mathbb{F}$ in which to evaluate h. We present an algorithm for the systematic evaluation of h over disjoint size k subsets of $\mathbb{F}$ using Fast Fourier Transform (FFT) algorithms. Our technique yields a k-generator over $\mathbb{F}$ with generation time $O(\log k)$, and space usage and seed length that is optimal up to constant factors. The algorithm depends upon the structure of $\mathbb{F}$, similarly to other FFT algorithms over finite fields [24].

The nonzero elements of $\mathbb{F}$ form a multiplicative cyclic group $\mathbb{F}^*$ of order $q - 1$. The multiplicative group has a primitive element ω which generates $\mathbb{F}^*$.

$$\mathbb{F}^* = \{\omega^0, \omega^1, \omega^2, \ldots, \omega^{q-2}\}. \tag{9}$$

For k that divides $q - 1$, we can construct a multiplicative subgroup $S_{k,0}^*$ of order k with $\omega_k = \omega^{(q-1)/k}$ as the generating element. $S_{k,0}^*$ contains k distinct elements of $\mathbb{F}$. Define for $j = 0, 1, \ldots, (q-1)/k - 1$,

$$S_{k,j}^* = \omega^j S_{k,0} = \{\omega^j \omega_k^0, \omega^j \omega_k^1, \ldots, \omega^j \omega_k^{k-1}\}. \tag{10}$$

Viewed as subsets of $\mathbb{F}^*$ the sets $S_{k,j}^*$ form an exact cover of $\mathbb{F}^*$. We now consider how to evaluate a degree $k - 1$ polynomial $h(x) \in \mathbb{F}[X]$ in the points of $S_{k,j}^*$. The polynomial takes the form

$$h(x) = a_0 x^0 + a_1 x^1 + \cdots + a_{k-1} x^{k-1}. \tag{11}$$

Rewriting the polynomial evaluation over $S_{k,j}^*$ in matrix notation:

$$\begin{bmatrix} \omega_k^{0 \cdot 0} & \omega_k^{0 \cdot 1} & \ldots & \omega_k^{0 \cdot (k-1)} \\ \omega_k^{1 \cdot 0} & \omega_k^{1 \cdot 1} & \ldots & \omega_k^{1 \cdot (k-1)} \\ \omega_k^{2 \cdot 0} & \omega_k^{2 \cdot 1} & \ldots & \omega_k^{2 \cdot (k-1)} \\ \vdots & \vdots & & \vdots \\ \omega_k^{(k-1) \cdot 0} & \omega_k^{(k-1) \cdot 1} & \ldots & \omega_k^{(k-1)^2} \end{bmatrix} \begin{bmatrix} \omega^{j \cdot 0} a_0 \\ \omega^{j \cdot 1} a_1 \\ \omega^{j \cdot 2} a_2 \\ \vdots \\ \omega^{j \cdot (k-1)} a_{k-1} \end{bmatrix} \tag{12}$$

We assume that the coefficients of h and ω^j are given and consider algorithms for efficient evaluation of the matrix-vector product. The coefficients $\tilde{a}_{j,i} = \omega^{j \cdot i} a_i$ for $i = 0, 1, \ldots, k-1$ can be found in $O(k)$ operations and define a polynomial $\tilde{h}_j(x) = \sum_{i=0}^{k-1} \tilde{a}_{i,j} x^i$. Evaluating $\tilde{h}_0(x)$ over $S^*_{k,0}$ corresponds to computing the Discrete Fourier Transform over a finite field.

Theorem 3 (Repeated). *For every finite field $\mathbb{F}$ that supports computing the discrete Fourier transform of length k in $O(k \log k)$ operations, there exists a data structure that, for every choice of $k \leq |\mathbb{F}|$ and given a primitive element ω, is an explicit $O(\log k)$ time k-generator with range $\mathbb{F}$, period $|\mathbb{F}|$, seed length k, space usage $O(k)$, and initialization time $O(k \log k)$.*

Proof: Evaluation of $\tilde{h}_j(x)$ over $S^*_{k,j}$ takes $O(k \log k)$ operations by assumption. For every batch j starting at $j = 0$, the value of ω^j is stored and used to compute the coefficients of $\tilde{h}_{j+1}(x)$ ing $O(k)$ operations. ∎

We now discuss the validity of the assumption that we are able to compute the DFT over a finite field in $O(k \log k)$ operations. Assume that $k \mid (q-1)$ and that ω_k is known. If k is highly composite there exist Fast Fourier Transforms for computing (12) in $O(k \log k)$ field operations [25]. If k is not highly composite there exists an algorithm for computing the DFT in equation (12) in $O(kz \log kz)$ operations for fields of cardinality $q = p^z$ in our model of computation [26]. For $q = p^{O(1)}$ this reduces to the desired $O(k \log k)$ operations.

VI. A load balancing application

We next consider how our new generator yields stronger guarantees for load balancing. Our setting is motivated by applications such as splitting a set of tasks of unknown duration among a set of m machines, in order to keep the load as balanced as possible. Once a task is assigned to a machine, it cannot be reassigned, i.e., we do not allow *migration*. For simplicity we consider the *unweighted* case where we strive to keep the *number* of tasks on each machine low, and we assume that m divides $|\mathbb{F}|$ for some field $\mathbb{F}$ with constant time operations on a word RAM. Suppose that each machine has capacity (e.g. memory enough) to handle b tasks at once, and that we are given a sequence of t tasks $T_1, \ldots, T_t$, where we identify each task with its duration (an interval in $\mathbb{R}$). Now let $k = mb$ and suppose that we use our constant time k-generator to determine for each $i = 1, \ldots, t$ which machine should handle T_i. (We emphasize that this is done without knowledge of T_i, and without coordination with the machines.) Compared to using a fully random choice this has the advantage of requiring only $k \operatorname{poly} \log k$ words of random bits, which in turn may make the algorithm faster if random number generation is a bottleneck. Yet, we are able to get essentially the same guarantee on load balancing as in the fully random case. To see this let $L(x) = \{i \mid x \in T_i\}$ be the set of tasks active

at time x, and let $L_q(x)$ be the subset of $L(x)$ assigned to machine q using our generator. We have:

Lemma 8. *For $\varepsilon > 0$, if $|L(x)|(1 + \varepsilon) < mb$ then $\Pr[\max_q |L_q(x)| > b] < m \exp(-\varepsilon^2 b / 3)$.*

Proof: Since $|L(x)| < mb = k$ we have that the assignment of tasks in $L(x)$ to machines is uniformly random and independent. This means that the number of tasks assigned to each machine follows a binomial distribution with mean $b/(1+\varepsilon)$, and we can apply a Chernoff bound of $\exp(-\varepsilon^2 b/3)$ on the probability that more than b tasks are assigned to a particular machine. A union bound over all m machines yields the result. ∎

Lemma 8 allows us to give a strong guarantee on the probability of exceeding the capacity b of a machine at any time, assuming that the average load is bounded by $b/(1+\varepsilon)$. In particular, let $S \subseteq \mathbb{R}$ be a set of size at most $2t$ such that every workload $L(y)$ is equal to $L(x)$ for some $x \in S$. The existence of S is guaranteed since the t tasks are intervals, and they have at most $2t$ end points. This means that

$$\sup_{x \in \mathbb{R}} \max_q |L_q(x)| = \max_{x \in S} \max_q |L_q(x)|,$$

so a union bound over $x \in S$ gives

$$\Pr[\sup_{x \in \mathbb{R}} \max_q |L_q(x)| > b] < 2tm \exp(-\varepsilon^2 b/3) \ .$$

For constant ε and whenever $b = \omega(\log k)$ and $tm = 2^{o(b)}$ we get an error probability that is exponentially small in b. Such a strong error guarantee can not be achieved with known constant time hashing methods [2], [27], [28], [17] in reasonable space, since they all have an error probability that decreases polynomially with space usage. Even if explicit constructions for the expanders needed in Siegel's hash functions were found, the resulting space usage would be polynomially higher than with our k-generator.

VII. Experiments

This section contains experimental results of an implementation of a k-generator over $\mathbb{F}_{2^{64}}$. There are two main components to the generator: an algorithm for filling a table of size m with dk-independent variables and a bipartite unbalanced expander graph.

For the first component, we use an implementation of Gao-Mateer's additive FFT [29, Algorithm 2.]. Utilizing the Gao-Mateer algorithm we can generate a batch of k elements of an $(|\mathbb{F}|, k)$-sequence using space $O(k)$ and $O(k \log^2 k)$ operations on a word RAM that supports arithmetic over $\mathbb{F}$. The additive complexity of the FFT algorithm is $O(k \log^2 k)$ while the multiplicative complexity is $O(k \log k)$. Addition in $\mathbb{F}_{2^{64}}$ is implemented as an XOR-operation on 64-bit words. Multiplication is implemented using the PCLMUL instruction along with the techniques for modular reduction specified by Gueron et al. [30].

The second component of the generator is the expander graph. Let $\mathbb{F}_{2^w}$ be a field of characteristic two and let $\mathbf{M}$ be a $cm \times m$ adjacency matrix of a graph Γ where each entry of $\mathbf{M}$ is viewed as an element of $\mathbb{F}_{2^w}$. By a similar argument to the one used in Lemma 5 the linear system $\mathbf{M}x$ defines a (cm, k)-sequence if x is a vector of dk-independent variables over $\mathbb{F}_{2^w}$ and $\mathbf{M}$ has row rank at least k. We consider randomized constructions of $\mathbf{M}$ over $\mathbb{F}_2$ with at most d 1s in each row and row rank at least k. It is easy to see that a matrix $\mathbf{M}$ over $\mathbb{F}_2$ with these properties also defines a matrix with the same properties over $\mathbb{F}_{2^w}$. Since k-uniqueness of Γ implies that $\mathbf{M}$ has row rank k, but not the other way around, we are able to obtain better performance characteristics of generators over $\mathbb{F}_{2^w}$ by focusing on randomized constructions of $\mathbf{M}$.

The matrix $\mathbf{M}$ is constructed in the following way. Independently, for each $i \in [cm]$ sample d integers uniformly with replacement from $[m]$ and define the ith row of $\mathbf{M}$ as the vector constructed by taking the zero vector and adding 1s in the d positions sampled for row i. Observe that if $\mathbf{M}$ does not have row rank at least k then some non-empty subset of at most k rows of $\mathbf{M}$ sum to the zero vector. In order for a non-empty set of vectors over $\mathbb{F}_2^m$ to sum to the zero vector, the bit-parity must be even in each of the m positions of the sum. The sum of any i rows of $\mathbf{M}$ corresponds to a balls and bins process that distributes id balls into m bins, independently and uniformly at random. Let id be an even number. Then there are $(id - 1)!!$ ways of ordering the balls into pairs and the probability that the outcome is equal to any particular pairing is $(1/m)^{id/2}$. This yields the following upper bound on the probability that a subset of i rows sums to zero:

$$\beta_{pair}(i, d, m) = (id - 1)!! \left(\frac{1}{m}\right)^{id/2}. \tag{13}$$

A comparison between this bound and the bound for k-uniqueness from equation (7) shows that, for each term in the sum, the multiplicative factor applied to the binomial coefficient $\binom{cm}{i}$ is exponentially smaller in id for the bound in (13).

The pair-based approach which yields the bound β_{pair} overestimates the probability of failure on subsets of size i, increasingly as id grows large compared to m. We therefore introduce a different bound based on the Poisson approximation to the binomial distribution: the number of balls in each of the m positions can approximately be modelled as independent Poisson distributed variables [31, Ch. 5.4]. The probability that that the parity is even in each of the m positions in a sum of i rows is bounded by

$$\beta_{poisson}(i, d, m) = e\sqrt{id} \left(\frac{1 + e^{-2\frac{id}{m}}}{2}\right)^m, \tag{14}$$

where we use the same approach as Mitzenmacher et al. [32]. For any given subset of rows of $\mathbf{M}$, we are free to choose between the two bounds. The probability that a randomly constructed matrix $\mathbf{M}$ fails to be have rank at least k can be bounded from above using a union bound over subsets of rows of $\mathbf{M}$.

$$\delta \leq \sum_{i=1}^{k} \binom{cm}{i} \min(\beta_{pair}(i, d, m), \beta_{poisson}(i, d, m)). \tag{15}$$

Table II shows the generation time in nanoseconds per 64-bit value using Horner's scheme, Gao-Mateer's FFT and the implementation of our generator (FFT+Γ). The choice of parameters for the expander graphs were based on a search for the fastest generation time over every combination of imbalance $c \in \{16, 32, 64\}$ and outdegree $d \in \{4, 8, 16\}$. Given choices of d, c and independence k, the size of the right side of the expander m was increased until existence could be guaranteed by the bound in (15). The experiments were run on a machine with an Intel Core i5-4570 processor with 6MB cache and 8GB of RAM.

The generation time for Horner's scheme is approximately linear in k and logarithmic in k for the FFT, as predicted by theory. The FFT is faster than using Horner's scheme already at $k = 64$ and orders of magnitude faster for large k. For small values of k, our generator is an order of magnitude faster than the FFT and comes close to the 4 ns per output performance of the 64-bit C++11 implementation of the popular Mersenne Twister.

In practice, the memory hierarchy appears to be the primary obstacle to maintaining a constant generation time as k increases. Our generator reads the expander graphs sequentially and performs random lookups into the table of

Table II
GENERATION TIME IN NANOSECONDS PER 64-BIT VALUE

k	Horner	FFT	FFT+Γ
2^5	177	243	15
2^6	361	294	16
2^7	730	338	19
2^8	1470	375	23
2^9	2950	412	24
2^{10}	5902	449	25
2^{11}	11808	487	35
2^{12}	23627	523	43
2^{13}	47183	561	54
2^{14}	94429	599	68
2^{15}	188258	638	69
2^{16}	376143	678	77
2^{17}	751781	719	85
2^{18}	1505016	765	93
2^{19}	3015969	808	110
2^{20}	6082313	864	175

dk-independent values. As k grows large, the table can no longer fit into cache and for large imbalance c, the expander can no longer be stored in main memory.

REFERENCES

[1] T. Hagerup, "Sorting and searching on the word RAM," in *Proc. STACS'98*, 1998, pp. 366–398.

[2] A. Siegel, "On universal classes of extremely random constant-time hash functions," *SIAM J. Comput.*, vol. 33, no. 3, pp. 505–543, 2004.

[3] J. P. Schmidt, A. Siegel, and A. Srinivasan, "Chernoff-Hoeffding bounds for applications with limited independence," *SIAM J. Discrete Math.*, vol. 8, no. 2, pp. 223–250, 1995.

[4] N. Alon and N. Asaf, "k-wise independent random graphs," in *Proc. FOCS'08*, 2008, pp. 813–822.

[5] M. Braverman, "Polylogarithmic independence fools AC^0 circuits," *J. ACM*, vol. 57, no. 5, pp. 28:1–28:10, 2010.

[6] B. Chor, O. Goldreich, J. Hastad, J. Freidmann, S. Rudich, and R. Smolensky, "The bit extration problem or t-resilient functions," in *Proc. FOCS'85*, 1985, pp. 396–407.

[7] N. Alon, O. Goldreich, J. Håstad, and R. Peralta, "Simple constructions of almost k-wise independent random variables," *Random Structures & Algorithms*, vol. 3, no. 3, pp. 289–304, 1992.

[8] J. Naor and M. Naor, "Small-bias probability spaces: efficient constructions and applications," *SIAM J. Comput.*, vol. 22, no. 4, pp. 838–856, 1993.

[9] O. Goldreich, *A primer on pseudorandom generators*. Providence, RI: American Math. Soc., 2010, vol. 55.

[10] N. Nisan, "Pseudorandom generators for space-bounded computation," *Combinatorica*, vol. 12, no. 4, pp. 449–461, 1992.

[11] A. Joffe, "On a set of almost deterministic k-independent random variables," *Ann. Prob.*, vol. 2, no. 1, pp. 161–162, 1974.

[12] J. L. Carter and M. N. Wegman, "New hash functions and their use in authentication and set equality," *J. Comput. System Sci.*, vol. 22, no. 3, pp. 265–279, 1981.

[13] J. von zur Gathen and J. Gerhard, *Modern computer algebra*, 3rd ed. Cambridge, UK: Cambridge University Press, 2013.

[14] A. Bostan and É. Schost, "Polynomial evaluation and interpolation on special sets of points," *J. Complexity*, vol. 21, no. 4, pp. 420–446, 2005.

[15] T. Grandlund and P. L. Montgomery, "Division by invariant integers using multiplication," *SIGPLAN Not.*, vol. 29, no. 6, pp. 61–72, 1994.

[16] M. Dietzfelbinger, "On randomness in hash functions (invited talk)," in *Proc. STACS'12*, 2012, pp. 25–28.

[17] M. Thorup, "Simple tabulation, fast expanders, double tabulation, and high independence," in *Proc. FOCS'13*, 2013, pp. 90–99.

[18] K. G. Larsen, "Higher cell probe lower bounds for evaluating polynomials," in *Proc. FOCS'12*, 2012, pp. 293–301.

[19] K. S. Kedlaya and C. Umans, "Fast modular composition in any characteristic," in *Proc. FOCS'08*, 2008, pp. 146–155.

[20] V. Shoup, *A computational introduction to number theory and algebra*, 2nd ed. Cambridge, UK: Cambridge University Press, 2009.

[21] M. Capalbo, O. Reingold, S. Vadhan, and A. Wigderson, "Randomness conductors and constant-degree lossless expanders," in *Proc. STOC'02*, 2002, pp. 659–668.

[22] A. Ta-Shma, C. Umans, and D. Zuckerman, "Lossless condensers, unbalanced expanders, and extractors," *Combinatorica*, vol. 27, no. 2, pp. 213–240, 2007.

[23] M. Capalbo, "Explicit bounded-degree unique-neighbor concentrators," *Combinatorica*, vol. 25, no. 4, pp. 379–391, 2005.

[24] M. Bhattacharya, R. Creutzburg, and J. Astola, "Some historical notes on the number theoretic transform," in *Proc. 2004 Int. TICS Workshop on Spectral Methods and Multirate Signal Processing*, 2004.

[25] P. Duhamel and M. Vetterli, "Fast Fourier transforms: a tutorial review and state of the art," *Signal Processing*, vol. 19, no. 4, pp. 259–299, 1990.

[26] F. P. Preparata and D. V. Sarwate, "Computational complexity of Fourier transforms over finite fields," *Mathematics of Computation*, vol. 31, no. 131, pp. 740–751, 1977.

[27] A. Pagh and R. Pagh, "Uniform hashing in constant time and optimal space," *SIAM J. Comput.*, vol. 38, no. 1, pp. 85–96, 2008.

[28] M. Dietzfelbinger and P. Woelfel, "Almost random graphs with simple hash functions," in *Proc. STOC'03*, 2003, pp. 629–638.

[29] S. Gao and T. Mateer, "Additive fast Fourier transforms over finite fields," *IEEE Trans. Inf. Theory*, vol. 56, no. 12, pp. 6265–6272, 2010.

[30] S. Gueron and M. E. Kounavis, "Intel carry-less multiplication instruction and its usage for computing the CGM mode," Intel Corporation, Tech. Rep., 2014.

[31] M. Mitzenmacher and E. Upfal, *Probability and computing*. New York, NY: Cambridge University Press, 2005.

[32] M. Mitzenmacher, R. Pagh, and N. Pham, "Efficient estimation for high similarities using odd sketches," in *Proc. WWW'14*, 2014, pp. 109–118.