GUARANTEEING THE DIVERSITY OF NUMBER GENERATORS

ADI SHAMIR AND BOAZ TSABAN

ABSTRACT. A major problem in using iterative number generators of the form $x_i = f(x_{i-1})$ is that they can enter unexpectedly short cycles. This is hard to analyze when the generator is designed, hard to detect in real time when the generator is used, and can have devastating cryptanalytic implications. In this paper we define a measure of security, called *sequence diversity*, which generalizes the notion of cycle-length for non-iterative generators. We then introduce the class of counter assisted generators, and show how to turn any iterative generator (even a bad one designed or seeded by an adversary) into a counter assisted generator with a provably high diversity, without reducing the quality of generators which are already cryptographically strong.

1. INTRODUCTION

In this paper we consider the problem of generating long cryptographically secure sequences by iterative number generators which start at some seed value $x_0 = s$, and extend it by computing $x_i = f(x_{i-1})$ where f is some function. The ith output of the generator is a (typically shorter) value $y_i = g(x_i)$ derived from the internal state by some output function g (Figure 1). If f is a secret keyed function, then g may be the identity.

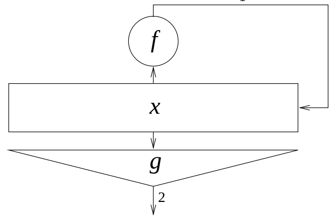

FIGURE 1

Key words and phrases. pseudorandomness, cycle length, cryptography.

A major application of number generators is to encrypt cleartexts by XORing them with the generated outputs. In this case, the seed s is a secret key which is shared by the communicating parties, but is unknown to the eavesdropping adversary.

Since the state space is finite, the sequence of internal states x_i will eventually become periodic with some period p, i.e., $x_i = x_{i+p}$ for all i larger than some i_0. Any cycling of the state sequence causes a cycling of the output sequence with period *at most* p. A particularly worrisome problem is the possibility that i_0 and p may be unexpectedly small, and therefore the cycling point $i_0 + p$ is actually achieved. This can happen even in very complex generators. An interesting example is Knuth's "Super-random" number generator (Algorithm **K**) [9, §3.1], which converges rapidly to a fixed point (that is, i_0 is very small, and $p = 1$).

If the cycling point $i_0 + p$ is achieved, then the XOR of the ith and $i+p$th ciphertexts is equal to the XOR of the ith and $i+p$th cleartexts, for all $i \geq i_0$. If the cleartexts have a sufficiently high redundancy, the cryptanalyst can detect the cycling by noticing the non-uniform statistics of such XOR's, and then recover the actual cleartexts from their known pairwise XOR's. Even if the cleartexts have no redundancy, knowledge of some cleartexts will make it possible to find other cleartexts encrypted with the same repeated values.

1.1. **Partial solutions.**

1.1.1. *Online monitoring.* A possible solution to this problem is to monitor each execution in real time. If a particular seed leads to early cycling, the cryptographic operation is stopped and the seed is replaced. However, this can be very disruptive if the exchange of new seeds is time consuming or difficult to arrange. Note further that real time detection of cycling behavior using hash tables requires a very large memory, whereas other methods such as Floyd's two pointer cycle detection algorithm (see, e.g., [9, p. 7]) are not guaranteed to detect cycles as soon as they are entered.

1.1.2. *Experimental testings.* The designer of the generator can test its behavior by applying f a limited number of times to a limited number of random seeds (see [2]). However, such testing cannot be exhaustive, and thus even if no cycling is ever detected in these tests, the next seed or the next step can lead to a cycling.

1.1.3. *Pseudorandom functions. Pseudorandom functions* $f : X \to X$ are functions which are chosen from the space of all possible functions $g : X \to X$ with a relatively low-entropy distribution, but which are difficult to tell apart from *truly random* functions (which are selected from the space of all possible functions $g : X \to X$ with uniform distribution). For any adversary with unlimited computational power and access to a polynomial (in $\log |X|$) number of values of a pseudorandom function f, the probability that the adversary can tell that these values came from f rather than from a truly random g should be negligible. *Pseudorandom permutations* and *pseudorandom sequences* are defined similarly to be low-entropy but difficult to distinguish from truly random permutations and sequences, respectively. For more precise definitions see, [20], [7], [10], [13, §2.2], and references therein.

It is easy to see (and well known) that sequences generated by iterative number generators with pseudorandom functions f are pseudorandom. Thus, the probability that such a generator enters a small cycle is negligible. However, all known constructions of pseudorandom functions are slow and are based on unproved conjectures (see [16, §17.9]). In fact, all practical functions used in cryptography are ad-hoc constructions which are not proved to be pseudorandom, and nothing is known about the actual structure of the cycles they generate.[1] This is particularly worrisome for the user, since there is no guarantee that the generators that he uses do not contain a trapdoor leading to short cycles.[2]

1.1.4. *Mathematically structured generators.* The need to avoid short cycles is the major motivation behind the development of several families of generators based on mathematical structures. These families include: Linear congruential generators, linear feedback shift registers (LFSR's), clock-controlled LFSR's, additive generators, feedback with carry shift registers, $1/p$ generators (see [16, §§16–17] and references therein), and TSR's [18]. Under certain conditions, these families can be proved to have large cycles.

The drawback of this approach is that their mathematical structure can be often used to cryptanalyze them (see [16, loc. cit.] for references to cryptanalysis of various implementations of the mentioned generators).

1.1.5. *Re-keying.* Chambers [3] suggested a technique to reduce the risk of short cycles by restarting the generator's internal state every fixed number of iterations, with a new key seed taken from a "re-keying"

[1]A notable exception appears in [8] and [5], where the cycle structure of nonlinear feedback shift registers is studied. However, the obtained results cover only degenerate cases. Moreover, in [8] it is proved that the studied generators *must have* short cycles.

[2]Knuth's example could be viewed as such a trapdoor generator.

generator which has a provably large cycle (e.g., one of the generators mentioned in Section 1.1.4).

Given an iterative generator, let p_k, $k = 1, 2, \ldots$, be the probability that the cycling point of the generator occurs after at least k iterations. Assume that we use the generator to get an output sequence of size m. The probability that we do not reach the cycling point in the usual iterative mode is p_m. Now, if we re-key the generator every k iterations, then the probability that we do not reach the cycling point even once is $p_k^{m/k}$. As nothing is known on the cycle structure of the generator, there is no guarantee that $p_k^{m/k}$ is greater than p_m. It may thus be the case that the re-keying mode is worse than the standard iterative mode.

Moreover, if the re-keying generator is cryptographically weak, then it could be cryptanalyzed from the outputs which come immediately after the re-keying phases.

One should note further that, as Schneier points out in [16, §17.11], algorithms that have a long key setup routine are not suitable for this mode.

1.1.6. *Similarity transformations and counter-mode.* Another possible solution is to take some simple permutation u which is guaranteed to have long cycles (e.g., $u(x) = x + 1 \pmod n$), or any of the examples from Section 1.1.4), and then to use fuf^{-1} (instead of f) as the iteration function. This similarity transformation has the same cycle structure as u.

Such a construction is, though, rather degenerate. Let $\langle f, g\rangle$ stand for a generator whose iteration function is f, and whose output function is g. Consider a generator of the form $\langle fuf^{-1}, g\rangle$. Define $\tilde{g} = g \circ f$. Then for all seeds s, setting $\tilde{s} = f^{-1}(s)$ implies that the ith output is $g((fuf^{-1})^i(s)) = g(fu^if^{-1}(s)) = (g \circ f)(u^i(\tilde{s})) = \tilde{g}(u^i(\tilde{s}))$, that is, the generator is equivalent to the generator $\langle u, \tilde{g}\rangle$. This means that the modified generator is equivalent to another generator with a cryptographically weak iteration function.

For $u(x) = x+1 \pmod n$ we conclude that for some $\tilde{g}$, the ith output of the generator equals $\tilde{g}(\tilde{s}+i)$. Generators of the form $y_i = g(s+i)$ are called *counter-mode* generators, and are a standard mode of operation [16, §9.9]. However, such generators have the following unpleasant property: The difference of any two input values $s + i$ and $s + j$ to g is simply $i - j$. If i is close to j, then $i - j$ has a small Hamming weight. This fact could be used in differential or correlation cryptanalysis of g. This is also the case for other choices of u, e.g., if u is an LFSR, then $u^i(s)$ and $u^j(s)$ are equal in all except for $i - j$ bits.

2. The diversity of sequence generators

In this section we propose a new notion of security for sequence generators, which generalizes the cryptographically desirable concept of long cycles.

We first define the notion of diversity for a single infinite sequence.

Definition 2.1. The *diversity of a sequence* $\vec{x} = (x_0, x_1, x_2, \ldots)$ is the function $\mathfrak{D}_{\vec{x}}(k)$ for $k = 1, 2, 3, \ldots$ defined as the minimum number of distinct values occurring in any contiguous subsequence $x_i, x_{i+1}, \ldots, x_{i+k-1}$ of length k in $\vec{x}$.

All of the sequences considered in this paper have a finite sample space of $|X| = n$ possible values. For any sequence $\vec{x}$ in X,

$$1 \leq \mathfrak{D}_{\vec{x}}(k) \leq \mathfrak{D}_{\vec{x}}(k+1) \leq \mathfrak{D}_{\vec{x}}(k) + 1 \leq n.$$

In other words, the diversity grows monotonically and at most linearly with k, and cannot exceed n.

We now generalize the concept from sequences to generators. We first define the types of generators considered in this paper:

Definition 2.2. An *iterative generator* is a structure $\mathcal{G} = \langle X, Y, f : X \to X, g : X \to Y\rangle$, where for all $x \in X$, $f(x)$ and $g(x)$ can be computed in polynomial time from x. X is *the state space*, and Y is *the output space*. We may write $\mathcal{G} = \langle f, g\rangle$ for short, or $\mathcal{G} : x_i = f(x_{i-1})$ if the output function is not relevant. For a generator $\mathcal{G} : x_i = f(x_{i-1})$ and seed $s \in X$, we denote the *state sequence* $(x_0 = s, x_1, \ldots)$ of the generated internal states by $\mathcal{G}(s)$.

We wish to bound from below the diversity of the sequences of internal states generated from possible seeds.

Definition 2.3. The *diversity of an iterative generator* $\mathcal{G} : x_i = f(x_{i-1})$ is the function

$$\mathfrak{D}_{\mathcal{G}}(k) = \min\{\mathfrak{D}_{\mathcal{G}(s)}(k) : s \in X\}$$

defined for $k = 1, 2, 3, \ldots$. The *total diversity* of $\mathcal{G}$ is the limit $\lim_{k\to\infty} \mathfrak{D}_{\mathcal{G}}(k)$.[3]

Iterative generators on finite spaces have simple diversity functions.

Lemma 2.4. *Assume that $\mathcal{G} : x_i = f(x_{i-1})$ is an iterative generator.*

[3]Anderson, et. al., [2] suggested a statistically-oriented notion of diversity for random number generators, based on experimental testings of the generator. These testings give estimations for the *average case* behavior, whereas our notion bounds the *worst case* behavior of the generator. Moreover, the combinatorial nature of our notion will make it possible to use mathematical theory in order to apply it to cases where experimental testings are not suitable (e.g., when the state space is huge). See also Section 1.1.2.

(1) *Let $\vec{x}$ be a sequence (of internal states) created by $\mathcal{G}$. Then $\mathfrak{D}_{\vec{x}}(k) = \min\{k, p\}$ where p is the length of the cycle that $\vec{x}$ enters into.*
(2) $\mathfrak{D}_{\mathcal{G}}(k) = \min\{k, p\}$ *where p is the length of the shortest cycle in f.*

Proof. $\vec{x}$ has distinct values before it enters the cycle and while it completes the first traversal of the cycle. This implies (1), and (2) follows from (1). □

The diversity of an iterative generator is thus directly related to the size of its smallest cycle. It is intended to capture one aspect of the worst case behavior of a generator, in the sense that generators with provably high diversity cannot repeat a small number of internal states a large number of times as a result of an unlucky or adversarial choice of seed.

The diversity measure can be applied to noniterative generators, in which the computation of x_i may depend on its index i as well.

Definition 2.5. A *counter-dependent* generator is a structure $\mathcal{G} = \langle X, Y, F : X \times \mathbb{N} \to X, g : X \to Y\rangle$, where for all $x \in X$ and $i \in \mathbb{N}$, $F(x, i)$ and $g(x)$ can be computed in polynomial time from x. X is *the state space*, and Y is *the output space*. In this type of generators, the next state is calculated by $x_i = F(x_{i-1}, i)$. Here too, we denote the *state sequence* $(x_0 = s, x_1, \dots)$ of generated internal states by $\mathcal{G}(s)$.

Note that iterative as well as counter-mode generators are particular cases of counter-dependent generators. A straightforward generalization of Definition 2.3 for counter-dependent generators is:

Definition 2.6.

(1) The diversity of a counter-dependent generator $\mathcal{G} : x_i = F(x_{i-1}, i)$ is the function $\mathfrak{D}_{\mathcal{G}}(k) = \min\{\mathfrak{D}_{\mathcal{G}(s)}(k) : s \in X\}$ defined for $k = 1, 2, 3, \dots$. The *total diversity* $\mathfrak{D}_{\mathcal{G}}^{\text{total}}$ of $\mathcal{G}$ is the limit $\lim_{k\to\infty} \mathfrak{D}_{\mathcal{G}}(k)$.
(2) A counter-dependent generator $\mathcal{G} : x_i = F(x_{i-1}, i)$ is $\mathfrak{g}(k)$-*diverse* if $D_{\mathcal{G}}(k) \geq \mathfrak{g}(k)$ for all $k = 1, 2, \dots$.

The diversity of a general counter-dependent generator can grow and freeze in an irregular way when k increases, since these generators are not forced into a cycle when they accidentally repeat the same x_i value. The diversity function is thus a natural generalization of the notion of cycle size.

3. Modifying generators

In this section we consider several ways in which we can modify a given iterative generator in order to increase its diversity. The main intuitive conditions we impose on this process are:

Condition 3.1. We do not want to design the new generator from scratch. We usually prefer to use known and well studied primitives such as DES, RC5 or nonlinear feedback shift registers, for which highly optimized code can be easily obtained or reused from other parts of the application. We thus want the modified design to use the same cryptographic ingredients as the original design.

Condition 3.2. The computational complexity of the modified next-state function must not be significantly greater that that of the original one.

Condition 3.3. The modification technique should be uniformly applicable to all iterative generators, treating them as black boxes. We do not want the modification to be based on the mathematical or statistical properties of the given iteration function f. In particular, we can not assume that we know the structure of its cycles.

Condition 3.4. We are more interested in increasing the diversity of the interval values x_i than in increasing the diversity of the output values $y_i = g(x_i)$: If the given generator uses an output function g with a small range (e.g., a single bit) applying diversity measures to the output values is meaningless.

The modification should be a win/win situation: If the given generator has a low diversity, the problem should be rectified, but if the given generator is already strong, we do not want the modification to weaken it. The problem is that we do not have a general quantitative definition of the "goodness" of generators, except when they are "perfect". We thus concentrate in this paper on the following formal interpretation.

Condition 3.5.

(1) For any given iteration function, the modified generator should be $\mathfrak{g}(k)$-diverse for some $\mathfrak{g}(k)$ which is exponential in $\log n$.

(2) If the iteration function f is pseudorandom, then the state sequences generated from random seeds by the modified generator should be pseudorandom.

As in counter-mode (see Section 1.1.6), our black box modification technique is based on turning the iterative generator into a counter-dependent generator, allowing x_i to depend on i in addition to x_{i-1}. To sharpen our intuition, let us consider some *bad* constructions. (In the following examples and throughout the paper, the state space X is identified with the set $\{0, 1, \ldots, n-1\}$, and addition in the state space is carried modulo n.)

Example 3.6. $x_i = i$. This function has maximal diversity, but poor cryptographic quality.

Example 3.7. $x_i = f(i)$. This is the standard counter-mode. Perfect generators remain perfect, but for a constant f the diversity is 1.

Example 3.8. $x_i = f(i) + i$. This is a simple combination of the previous two examples. Perfect generators remain perfect, but for $f(x) = -x$, all the generated x_i are 0, and thus the diversity is 1.

Example 3.9. $x_i = f(x_{i-1} + i)$. This is an attempt to force the next state to depend both on the previous state and on the index. Perfect generators remain perfect, but the generated sequence has diversity 1 when f is a constant function.

Example 3.10. $x_i = f(x_{i-1} + i) + i$. This is the "kitchen sink" approach, trying to combine all the ingredients in all possible ways. However, when the function f is $f(x) = -x$, the sequence generated from any initial seed $x_0 = s$ is $s, -s, s, -s, s, -s, \ldots$ which contains at most two values.

Considering these counterexamples, the reader may suspect that all black box modifications are bad (for some f). In the next section we show that this is not the case.

4. A provably good modification technique

Given an iterative generator $\langle f, g \rangle$, we apply the following black-box modification.

Definition 4.1. A *counter-assisted generator* $\langle f, g \rangle$ is a generator in which $x_0 = s$, and for all $i \geq 1$ $x_i = f(x_{i-1}) + i \pmod{n}$, where n is the size of the state space, and the ith output is $g(x_i)$ (see Figure 2).

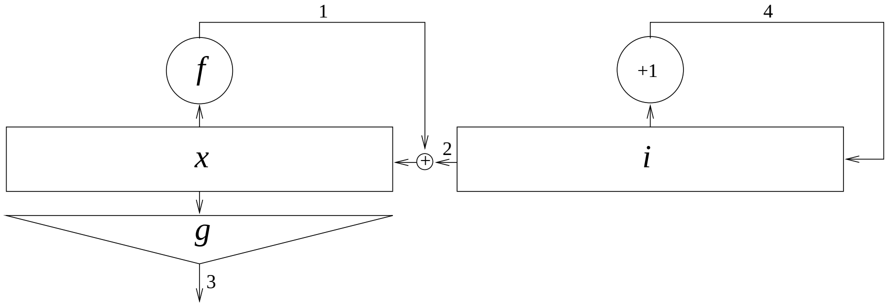

Figure 2

Since it is easy to maintain or obtain a counter for the number of values produced so far (in many applications, one can use either the loop counter or the running block-number as a counter for the counter-assisted mode), and no change is made in the function f or g, the modification technique is completely trivial and can be applied to any iterative generator without increasing its complexity.

Formally, for all generators $\langle X, Y, f, g\rangle$, the counter assisted modified generator is in fact the iterative generator $\langle X \times \{0, \dots, n-1\}, Y, F, G\rangle$, where

$$\begin{aligned} F(x,i) &= (f(x)+i \pmod n, i+1 \pmod n) \\ G(x,i) &= g(x) \end{aligned} \tag{1}$$

However, note that:

(1) The only secret part is located in the x coordinate,
(2) incrementing i has no cryptographical significance, and
(3) the output calculation $G(x,i)$ is independent of the i-coordinate.

Thus applying diversity measures on the whole state space $X\times\{0,\dots,n-1\}$—that is, measuring the diversity of the sequences of pairs (x_i, i), $i = 1, 2, \dots$—is misleading (and, in fact, not informative). This is why the diversity measure is focused on the actual state sequences $\mathcal{G}(s) = (x_0 = s, x_1, \dots)$ rather than on the sequence of pairs (x_i, i).

Lemma 4.2. *Let $\vec{x} = (x_0, x_1, x_2, \dots)$ be a state sequence of a counter assisted generator. Then for all $i \neq j \pmod n$, if $x_i = x_j$ then $x_{i+1} \neq x_{j+1}$ and $x_{i-1} \neq x_{j-1}$.*

Proof. We argue modulo n. By definition, $x_{i+1} = f(x_i) + (i+1)$ and $x_{j+1} = f(x_j)+(j+1)$. If $x_i = x_j$ but $i \neq j$, then necessarily $x_{i+1} \neq x_{j+1}$. Now, for the very same reason, $x_{i-1} = x_{j-1}$ would imply $x_i \neq x_j$, which is not the case. □

In other words, the sequence $\vec{x}$ has the interesting property that equality at any pair of locations implies inequality at the pair of their immediate successors and the pair of their immediate predecessors. We call this *the isolated equality property.* This is the intuitive reason why counter assisted generators cannot enter short cycles: If they accidentally generate the same value at several locations, all the subsequent computations are guaranteed to diverge rather than converge.

Theorem 4.3.

(1) *The black box modification technique modifying $\mathcal{G} : x_i = f(x_{i-1})$ to $\mathcal{G}' : x_i = f(x_{i-1}) + i \pmod n$ is $\max\{\mathfrak{g}(k), \mathfrak{h}(k)\}$-diverse,*

where

$$\mathfrak{g}(k)=\begin{cases}\sqrt{k-1} & k\le n\\ \sqrt{n} & n<k\end{cases},\quad \text{and}\quad \mathfrak{h}(k)=\begin{cases}k/|\operatorname{Im}(f)| & k\le n\\ n/|\operatorname{Im}(f)| & n<k\end{cases}.$$

(2) *If the iteration function f is pseudorandom, then the state sequences generated from random seeds by the modified generator are pseudorandom.*

Proof. (1) We first show that $\mathfrak{g}(k)\le\mathfrak{D}_{\mathcal{G}'}(k)$ for all $k=1,2,\dots$. Consider any sequence of k consecutive values $x_i,x_{i+1},\dots,x_{i+k-1}$ ($k\le n+1$), and assume that it contains exactly ν distinct values. There are ν^2 possible ordered pairs of these values (a,b), and by Lemma 4.2 each one of them can occur at most once in a consecutive pair of locations (x_j,x_{j+1}) along the sequence. Since there are $k-1$ such locations, $\nu^2\ge k-1$, which yields the desired lower bound on ν.

Next, we need to show that $\mathfrak{h}(k)\le\mathfrak{D}_{\mathcal{G}'}(k)$ for all $k=1,2,\dots$. In a sequence of k consecutive values $x_i,x_{i+1},\dots,x_{i+k-1}$ ($k\le n+1$), each x_j is of the form c_j+j, where $c_j\in\operatorname{Im}(f)$. Since we add k distinct values to at most $|\operatorname{Im}(f)|$ values, we get at least $k/|\operatorname{Im}(f)|$ distinct values.

(2) We now sketch the proof of the pseudorandomness part. Consider the following sequence of oracles, which accept a number k (which is polynomial in $\log n$) and output a sequence $x_1,\dots,x_k\in X$. (By *random* we mean statistically independent and uniformly distributed.)

Oracle 1: Returns a random sequence $x_i\in X$ ($i=1,2,\dots,k$).

Oracle 2: Chooses a random seed $x_0=s$, and defines an $f:X\to X$ *on the fly*, as follows:

(1) A flag Birthday is initially set to 0.
(2) For each $i=1,2,\dots,k$:
   – If $f(x_{i-1})$ is undefined, then choose a random $y\in X$ and define $f(x_{i-1})=y$.
   – Otherwise, set Birthday $=1$.
(3) Set $x_i=f(x_{i-1})+i$.

The remaining values of f are chosen randomly.

Oracle 3: Chooses a particular function f with uniform probability from the set of all functions from X to X, chooses a random seed $x_0=s$, and returns the sequence x_i with $x_i=f(x_{i-1})+i$, $i=1,2,\dots,k$.

Oracle 4: Same as Oracle 3, but with f *pseudorandom* instead of truly random.

We say that two oracles are *distinguishable* if there exists a (not necessarily polynomial time) algorithm (called *distinguisher*) which, for some constant $c > 0$, given a sequence of length polynomial in $\log n$, can tell with probability greater than $1/\log(n)^c$ which oracle has generated this sequence. Otherwise, the oracles are *indistinguishable*. It is clear that Oracles 2,3 are indistinguishable. That Oracles 3,4 are indistinguishable follows from the fact that any distinguisher of these oracles can be used to construct a distinguisher of pseudorandom functions from random ones.

It remains to show that Oracles 1,2 are indistinguishable. The only possible constraint on the output of Oracle 2 happens when f is applied twice to the same argument, that is, $\mathsf{Birthday}$ is set to 1. It is well-known that for $k << n$, the probability that no birthday occurs is close to $\frac{k^2}{2n}$ [17], which is negligible if k is polynomial in $\log n$. $\square$

Remark 4.4. The upper bound $\frac{k^2}{2n}$ on the distinguishing probability is tight: In probability close to $\frac{k^2}{2n}$, a birthday $x_i = x_j$ occurs and the distinguisher can check that $x_{i+1} - (i+1) = x_{j+1} - (j+1)$. Provided this, the probability that the output came from Oracle 1 is $1/n$.

5. Asymptotic tightness of the provable diversity

The square root lower bound on the diversity may seem to be an artifact of the proof technique. We first consider the purely combinatorial version of the problem: What is the longest sequence one can construct from ν distinct symbols which has the isolated equality property?

Lemma 5.1. *For any positive integer ν, there exists a sequence of length $\nu^2 + 1$ consisting of ν symbols and having the isolated equality property.*

Proof. Let C be a complete directed graph with ν vertices and ν^2 directed edges (including self loops). As the graph is connected and the indegree and outdegree of each vertex in C is the same ($= \nu$), the graph is Eulerian. Let $v_0 e_0 v_1 e_1 \dots v_{\nu^2-1} e_{\nu^2-1} v_0$ be an Eulerian tour, which includes each directed edge exactly once. Assume that for some distinct i and j, $v_i = v_j$. If $v_{i+1} = v_{j+1}$, then necessarily $e_i = e_j$, which is disallowed in Eulerian tours. Similarly, $v_{i-1} = v_{j-1}$ would imply $e_{i-1} = e_{j-1}$. Consequently, the sequence has the isolated equality property. $\square$

This combinatorial result does not rule out the possibility that sequences created by counter assisted generators must satisfy additional constrains, and as a result the lower bound in Theorem 4.3 can be improved significantly. We will show that this is not the case: We prove the asymptotic tightness of our lower bound by constructing for each

n a specific counter-assisted generator, such that the total diversities of these counter-assisted generators are $O(\sqrt{n})$.

Theorem 5.2. *There exist functions f_n, $n = 1, 2, \dots$ such that the total diversities $\mathfrak{D}^{\text{total}}_{\mathcal{G}_n}$ of the counter assisted generators $\mathcal{G}_n : x_i = f_n(x_{i-1}) + i \pmod n$ are $O(\sqrt{n})$.*

Proof. Fix a natural number n. We will write for short f and $\mathcal{G}$ instead of f_n and $\mathcal{G}_n$, respectively.

The state sequence of $\mathcal{G}$ will be based on two sequences: $a_0, a_1, \dots, a_{\alpha-1}$ and $b_0, b_1, \dots, b_{\beta-1}$ (the values of α and β will be determined later). The sequences are "meshed" as follows:

(1) Locations with even indices contain only the a_i values, and locations with odd indices contain only the b_j values.
(2) The a_i values occur in block order: The first β occurrences are a_0, the next β occurrences are a_1, and so on.
(3) The b_j values occur in cyclic order: The first β occurrences are $b_0, \dots, b_{\beta-1}$ in this order, the next β occurrences are again $b_0, \dots, b_{\beta-1}$ in this order, and so on.

Putting these blocks in consecutive rows, we get a matrix $C = (c_{ij})$ of size $\alpha \times 2\beta$, where $c_{i,2j} = a_i$ and $c_{i,2j+1} = b_j$:

$$C = \begin{pmatrix} a_0 & b_0 & a_0 & b_1 & \cdots & a_0 & b_{\beta-1} \\ a_1 & b_0 & a_1 & b_1 & \cdots & a_1 & b_{\beta-1} \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ a_{\alpha-1} & b_0 & a_{\alpha-1} & b_1 & \cdots & a_{\alpha-1} & b_{\beta-1} \end{pmatrix}$$

We define a function f for which the counter assisted generator $\mathcal{G} : x_i = f(x_{i-1}) + i$, seeded by $x_0 = a_0$, has state sequence equal to our meshed sequence.

We begin with a few simple restrictions on our parameters. For cyclicity the counter must return to 0 after $2\alpha\beta$ steps, that is, $2\alpha\beta = 0 \pmod n$. We will consider α's and β's such that $2\alpha\beta = n$ to make the sequence shorter. The isolated equality property implies that all of the a_i and b_j values are distinct. Thus, the total diversity will be $\alpha + \beta$.

Under these restrictions, we can see via elementary calculus that the choice $\alpha = \beta = \sqrt{n/2}$ yields the minimum possible total diversity of $\alpha + \beta = \sqrt{2n}$ values.

We thus begin with n's for which $n/2$ is a square, and choose $\alpha = \beta = \sqrt{n/2}$.

We now consider the specific values of the elements in our meshed sequence. The conditions are: $c_{i,j+1} = f(c_{ij}) + 2\beta i + (j+1)$, $c_{i+1,0} = f(c_{i,2\beta-1}) + 2\beta(i+1) - 1$, and $c_{00} = f(c_{\alpha-1,2\beta-1}) + 2\alpha\beta$. In terms of the a_i and b_j this is:

$$\begin{aligned} b_j &= f(a_i) + 2\beta i + (2j+1) \\ a_i &= f(b_j) + 2\beta i + (2j+2) \quad (j = 0, \ldots, \beta-2) \\ a_i &= f(b_{\beta-1}) + 2\beta i \end{aligned}$$

Setting $x = f(a_0)$, the first equation yields $b_j = x + (2j+1)$ for $i = 0$. Putting this back in the equation we get that $f(a_i) = x - 2\beta i$ for all i. Similarly, the second equation implies (setting $y = f(b_0)$) $a_i = y + 2\beta i + 2$ and $f(b_j) = y - 2j$ for all $j < \beta - 1$. The third equation with $i = 0$ gives $f(b_{\beta-1}) = a_0 = y + 2$.

We therefore have, for any choice of x, y, the following requirements:

$$\begin{aligned} a_i = y + 2 + 2\beta i &\overset{f}{\mapsto} x - 2\beta i \\ b_j = x + 1 + 2j &\overset{f}{\mapsto} y - 2j \quad (j < \beta - 1) \\ b_{\beta-1} = x - 1 + 2\beta &\overset{f}{\mapsto} y + 2 \end{aligned}$$

It is easy to check that any such definition yields the desired sequence of states, as long as the resultant a_i and b_j's are disjoint. As we assume that n is even, choosing any x and y having the same parity (e.g., $x = y = 0$) will do.

The values of f on $X \setminus \{a_i, b_j\}$ can be arbitrary. It remains to check that the sequence is repeated after every $\alpha \cdot 2\beta$ steps. Indeed, the counter will be $2\alpha\beta = 0 \pmod n$, and thus $x_{2\alpha\beta} = f(x_{2\alpha\beta-1}) + 0 = f(b_{\beta-1}) = a_0$, so we are right where we begun.

We now treat the cases where $n/2$ is not a square. Set $\alpha = \beta = \lfloor \sqrt{n/2} \rfloor$, and define a_i, b_j, and f as above. Now modify $f(x)$ to $f(x \bmod 2\alpha\beta)$. The above argument shows that if we project the state-sequence $\vec{x}$ modulo $2\alpha\beta$, we get diversity at most $\alpha + \beta = O(\sqrt{n})$. Therefore, the actual diversity can be no more than $O(\sqrt{n}) \cdot \lceil n/(2\alpha\beta) \rceil = O(\sqrt{n}) \cdot 2 = O(\sqrt{n})$. □

Remark 5.3. In most practical cases, $n/2$ is not a square and thus we cannot achieve the exact $\sqrt{2n}$ upper bound using our meshing construction. However, in many cases n is an even power of 2 (e.g, 2^{24}, 2^{32}, 2^{64}, 2^{128}, etc.), so we can choose $\alpha = \sqrt{n}$ and $\beta = \sqrt{n}/2$ (note that $2\alpha\beta = n$) to get total diversity $\alpha + \beta = 3\sqrt{n}/2$, which is close to the $\sqrt{2n}$ upper bound achieved in the case where $n/2$ was a square.

Our construction showed that the bound $\sqrt{n}$ for the total diversity is asymptotically tight. However, we do not have a construction where $\mathfrak{D}_{\mathcal{G}}(k)$ is $O(\sqrt{k})$ for all k *simultaneously.*

Open problem 5.4. Does there exist a constant c such that for all sufficiently large n, there exists a counter-assisted generator $\mathcal{G}$ (with state space of size n) such that $\mathfrak{D}_{\mathcal{G}}(k) \leq c\sqrt{k}$ for all k?

6. Cascade counter-assisted generators

In this section we generalize the notion of counter-assisted generators.

A Latin square is a binary function which is uniquely invertible given its output and any one of the inputs. For example, the operations $x + y \pmod{n}$, $x - y \pmod{n}$ and $x \oplus y$ are Latin square operations. Moreover, every group operation is a Latin square operation, and if $x \star y$ is a Latin square operation and P, Q, Z are permutations, then $Z(P(x) \star Q(y))$ is a Latin square operation. Let $\star$ be a Latin square operation.

It is easy to see that the proof of Theorem 4.3 applies when the $+i$ modification is replaced by any Latin square operation $\star i$ (unique invertibility with respect to the i input guarantees the isolated equality property, and unique invertibility with respect to the x_i input guarantees the pseudorandomness of the states). We can thus extend the concept of counter assisted generators to include these cases as well.

Remark 6.1. When n is a power of 2, we can use essentially the same construction as in the proof of Theorem 5.2 to show the optimality of the $\Omega(\sqrt{n})$ lower bound when the $+i \pmod{n}$ modification is replaced by a $\oplus i$ modification.

The next lemma shows that counter-mode generators are a degenerated case of counter-assisted generators.

Lemma 6.2. *Every counter-mode generator is a counter-assisted generator.*

Proof. A counter-mode generator with ith output $g(s \star i)$ is equivalent to the counter-assisted generator $\mathcal{G} = \langle f, g \rangle$, where $f \equiv s$, and the Latin square operation is $\star$, since in this case, $x_i = f(x_{i-1}) \star i = s \star i$. □

We can extend the notion of counter-assisted generators further. Assume that $\mathcal{G} = \langle f, g, X, Y \rangle$ is an iterative generator, and let $\vec{c} = \langle c_0, c_1, \ldots \rangle$ be any sequence of elements in X. Define the *sequence-assisted generator* $\mathcal{G} \star \vec{c}$ to be the generator whose ith state is $x_i = f(x_{i-1}) \star c_i$ (and whose ith output is $g(x_i)$).

Theorem 6.3. *Let $\mathcal{G} = \langle f, g \rangle \star \vec{c}$ be a sequence-assisted generator. Then:*

(1) $\mathfrak{D}_{\mathcal{G}}(k) \geq \sqrt{\mathfrak{D}_{\vec{c}}(k) - 1}$ *for all* $k = 1, 2, \ldots$.

(2) *If the the sequence $\vec{c}$ is pseudorandom, then the state sequence of $\mathcal{G}$ is pseudorandom.*
(3) *If f is pseudorandom, then the state sequence of $\mathcal{G}$ is pseudorandom.*

Proof. (1) As in Lemma 4.2, we can show that $c_i \neq c_j$ implies $(x_{i-1}, x_i) \neq (x_{j-1}, x_j)$. The rest of the proof is similar to the proof of Theorem 4.3(1).

(2) If the state sequence of $\mathcal{G}$ is not pseudorandom, then the sequence $\vec{c}$ can be distinguished from pseudorandom noise by considering $\langle f, g\rangle \star \vec{c}$, and looking at the state sequence of $\mathcal{G}$.

(3) This is proved as in Theorem 4.3(2); the only difference is in the definition of Oracle 3. □

Thus, any sequence $\vec{c}$ with large diversity can be used instead of a counter. In particular, we can use the output of any of the generators mentioned in Section 1.1.4 as the assisting sequence. In general, assume that $\mathcal{C}$ is any generator with output in X. Define $\mathcal{G} \star \mathcal{C} = \mathcal{G} \star \vec{c}$, where $\vec{c} = \langle c_0, c_1, \dots\rangle$ is the output sequence of $\mathcal{C}$ (note that the sequence $\vec{c}$ depends of the initialization of $\mathcal{C}$). The following definition is inductive.

Definition 6.4. $\mathcal{G}$ is a *cascade* counter-assisted generator if:

(1) $\mathcal{G}$ is a (standard) counter-assisted generator, or
(2) $\mathcal{G} = \mathcal{F}\star\mathcal{C}$, where $\mathcal{F}$ is an iterative generator, $\star$ is a Latin square operation, and $\mathcal{C}$ is a cascade counter-assisted generator.

In particular, we have:

Lemma 6.5. *Every iterative generator is a cascade counter-assisted generator.*

Proof. If $\mathcal{G}$ is an iterative generator, and $\mathcal{C}$ is a generator with output function 0, then $\mathcal{G} + \mathcal{C} = \mathcal{G}$ is a cascade counter-assisted generator. □

Thus the notion of cascade counter-assisted generators extends those of iterative, counter-mode and counter-assisted generators.

Ideally, all internal states of the cascaded generators (including the starting position of the counter i) should be initialized by random, independent seeds. If this is not feasible, one can, e.g., initialize the "driving" generator or the counter with a random seed, and then clock the cascade a few times to make all internal states depend on the seed. In this case, however, caution must be taken to make sure that particular choice of output functions does not make the influence of the seed "vanish" while going down the cascade.

Example 6.6. Assume that the generators $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ have state spaces of size $n = 2^{256}$ (256 bits). Assume further that the generator

$\mathcal{C}$ is counter-based with an invertible output function $g_{\mathcal{C}}$, and that the output function $g_{\mathcal{B}}$ of $\mathcal{B}$ is invertible as well. Consider the total diversity of the cascade generator $\mathcal{A}+(\mathcal{B}\oplus\mathcal{C})$ (see Figure 3): As $\mathcal{C}$ is counter-based, we have $\mathfrak{D}_{\mathcal{C}}(n)=n$. Thus by Theorem 6.3 (and discreteness), $\mathfrak{D}_{\mathcal{B}\oplus\mathcal{C}}(n)\geq\lceil\sqrt{n-1}\rceil=2^{128}$, and $\mathfrak{D}_{\mathcal{A}+(\mathcal{B}\oplus\mathcal{C})}(n)\geq\lceil\sqrt{\mathfrak{D}_{\mathcal{B}\oplus\mathcal{C}}(n)-1}\rceil\geq 2^{64}$. Moreover, if the output function of $\mathcal{C}$, or any of the iteration functions of $\mathcal{B}$, $\mathcal{A}$ is pseudorandom, then the state sequence of $\mathcal{A}$ is pseudorandom as well. (We can also use, e.g., a maximal length LFSR instead of the counter-based generator $\mathcal{C}$ to get the same results.)

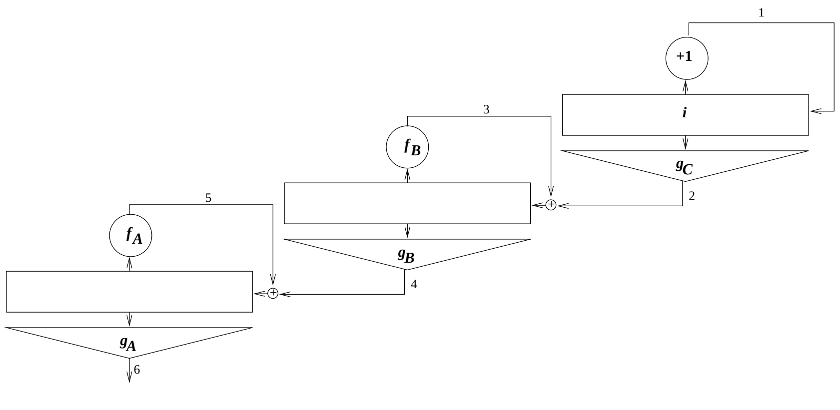

FIGURE 3

Remark 6.7. In this section we have seen that every iterative generator can be viewed as a cascade counter-assisted generator (in a degenerate manner). On the other hand, as mentioned in Section 4, every counter-assisted generator can be viewed as an iterative generator (with a larger state space). The advantage of our approach is that we focus on the cryptographical part of the generator, from which the output is calculated, rather than on the state of the whole system.

7. Generating sequences with maximal diversity

If we allow the design of a new output function g, then we can modify any generator to have the maximal possible diversity $\mathfrak{D}_{\mathcal{G}}(k)=k$ for all $k=1,2,\ldots,n$.

Definition 7.1. Let $\mathcal{G}$ be any iterative generator. Modify its next-state function as follows:

$$\begin{aligned} x_{2i+1} &= f(x_{2i}) \\ x_{2i+2} &= f(x_{2i+1})+i \end{aligned}$$

That is, the counter is incremented and added to the state value only once every two iterations of the generator. The pair of generated values (x_{2i}, x_{2i+1}) is used as the argument of a new output function $g' : X \times X \to Y \times Y$. We call this mode of operation *the two-step counter-assisted mode.* More generally, the *t-step counter-assisted mode* is defined by incrementing and adding the counter once every t iterations, and using each t-tuple as the input of a new output function $\hat{g} : X^t \to Y^t$. Formally, the t-step generator $\mathcal{G} = \langle f, g, X, Y \rangle$ with Latin square operation $\star i$ is the counter-assisted generator $\mathcal{G}^t = \langle \hat{f}, \hat{g}, X^t, Y^t \rangle$ with the (injective) operation $\hat{\star} i$, where

- $\hat{f}(x_0, \ldots, x_{t-1}) = (f(x_{t-1}), f^2(x_{t-1}), \ldots, f^t(x_{t-1}))$,
- $(x_0, \ldots, x_{t-1}) \hat{\star} i = (x_0, \ldots, x_{t-1} \star i)$, and
- i is a cyclic counter in the range $0, 1, \ldots, n-1$.

Note that t-step counter-assisted generators require a state buffer of size t.

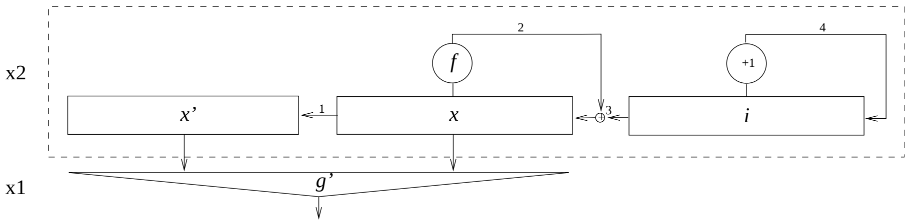

FIGURE 4. A two-step counter-assisted generator

For all $t \geq 2$, any t-step counter-assisted generator has maximal possible diversity:

Theorem 7.2. *For any generator $\mathcal{G} = \langle f, g \rangle$, and for all $t \geq 2$, we have the following:*

(1) *If f is pseudorandom, then the state sequences of $\mathcal{G}^t$ are pseudorandom.*
(2) *$\mathfrak{D}_{\mathcal{G}^t}(k) = k$ for all $k = 1, \ldots, n$.*

Proof. The proof of the pseudorandomness part is similar to that in Theorem 4.3.

To prove the diversity part, assume that for some $i \neq j \pmod{n}$ we have equality between the t-tuples $(x_{it}, \ldots, x_{it+t-1})$ and $(x_{jt}, \ldots, x_{jt+t-1})$. In particular, $x_{it+t-2} = x_{jt+t-2}$. But this implies $x_{it+t-1} = f(x_{it+t-2}) + i \neq f(x_{jt+t-2}) + j = x_{it+t-1} \pmod{n}$, a contradiction. $\square$

7.1. **Black-box modifications of the output function g.** If the computational complexity of evaluating the new output function g' in the two-step mode is at most double that of evaluating g, then on average, the computational complexity of obtaining the next output does not change: We clock the generator twice, but we get two outputs at once. If the output space Y is equal to X then we can get very close to this without designing a new output function.

We will use the terminology of [13]. For a function $g : X \to X$, define the *Feistel permutation* $D_g : X \times X \to X \times X$ by $D_g(L, R) \stackrel{\text{def}}{=} (R, L \oplus g(R))$. (Here too, any Latin square operation $\star$ can be used instead of $\oplus$.)

If the output function g is key-dependent, then we can use a Luby-Rackoff construction. Denote the key space by K, and assume that the size of the key space is exponential in $\log n$.

Theorem 7.3. *Assume that the mapping $\kappa \mapsto g_\kappa$ is pseudorandom, and that κ_1, κ_2, and κ_3 are pseudorandom elements of K. Then for all functions $f : X \to X$ and seeds $x_0 \in X$, the two-step generator $\langle \hat{f}, D_{g_{\kappa_1}} \circ D_{g_{\kappa_2}} \circ D_{g_{\kappa_3}} \rangle$ has pseudorandom output.*

Proof. By Theorem 7.2, for all iteration functions f and seeds $x_0 \in X$, the inputs to $D_{g_{\kappa_1}} \circ D_{g_{\kappa_2}} \circ D_{g_{\kappa_3}}$ are all distinct. By a result of Luby and Rackoff [11], this implies pseudorandomness of the output. □

This construction makes the output calculation slower by a factor of 3:2. The computational complexity of the following alternative is closer to the desired optimum, and is a more straightforward modification.

Theorem 7.4. *Assume that $g : X \to X$ is pseudorandom, and assume that $h : X \to X$ is pseudorandomly chosen from a family H of functions such that for all distinct $x, y \in X$ and for all $z \in X$, the probability that $h(x) \oplus h(y) = z$ ($h \in H$) is negligible. Then for all functions $f : X \to X$ and seeds $x_0 \in X$, the two-step counter-assisted generator $\langle \hat{f}, D_g \circ D_g \circ D_h \rangle$ has pseudorandom output.*

Proof. By a result of Lucks [12] (see also [13]), $D_g \circ D_g \circ D_h$ is pseudorandom. The rest of the proof is like in Theorem 7.3. □

There exist very efficient families H with the property mentioned in Theorem 7.4 (see [13] for examples and references). Thus, the computational overhead of applying h is small, and the resulting generator is almost as efficient as the original one. Note that, unlike the results in earlier sections, we get here a black-box modification of an iterative generator $\langle f, g \rangle$ which has maximal *output* diversity, and if *either one* of the functions f or g is pseudorandom, then the output sequence is pseudorandom.

Example 7.5. Let $f = \mathrm{DES}$ [14], $g = \mathrm{RC5}$ [15], and $h_\kappa : \{0,1\}^{64} \to \{0,1\}^{64}$ be a function from Vazirani's *shift family* (the ith bit of $h_\kappa(x)$ is $\sum_{i=1}^{n} x_i \kappa_{j+i-1} \bmod 2$, see [13] and [19]). The two-step counter-assisted generator $\langle \widehat{\mathrm{DES}}, D_{\mathrm{RC5}} \circ D_{\mathrm{RC5}} \circ D_{h_\kappa} \rangle$ has maximal (state and output) diversity k for all $k = 1, 2, \ldots, 2^{64}$. On average, the calculation of any output 64 bit block requires a single invocation of DES and a single invocation of RC5. The execution time overhead of the rest of the operations is negligible. Furthermore, if *either one of the two* functions DES and RC5 is difficult to distinguish from random, then the output sequence will be difficult to distinguish from random as well.

Open problem 7.6. Assume that both f and g are (truly) random, and consider an output sequence of length m generated from a random seed by the two-step counter-assisted generator $\mathcal{G}^2 = \langle \hat{f}, D_g \circ D_g \rangle$. What is the highest distinguishing probability between such a sequence and a random sequence?

Remark 7.7. Using the results from [13], we get that for all t, the output function of the t-step counter-assisted mode can be modified in a black-box manner with a small computational overhead, to get the same diversity and pseudorandomness results. See [13] for details.

Remark 7.8. In certain cases, when t is large (e.g., $t \geq 4$) it is desirable that the inputs to the t-step output function are distinct in as many entries as possible (for example, this guarantees many active S-boxes in differential cryptanalysis of the output function). We can achieve this goal via letting the next state be the same as when clocking the (standard) counter-assisted generator t times (that is, the counter is incremented and added to the x_i value every clock). By the isolated equality property, this guarantees that any two t-tuples are distinct in at least $\lfloor t/2 \rfloor$ entries. In this mode of operation, the diversity remains maximal as long as $k < n/t$.

7.2. **Safe transition to new generations of cryptographic functions.**

A common practice in the design of new generations of cryptographic functions is to double the input and output length. Nowadays, we experience the evolution from 64 bit functions (such as DES, RC5, etc.) to 128 bit functions (such as the AES candidates [1]). The advantage of old generation functions is that they have gone through years of extensive academic research, and are thus well understood. It will take a long time to gain similar confidence in the new generation functions.

Our two-step counter-assisted mode suggests a natural and straightforward way to combine new and old generation functions in a way that if *either one* of them is pseudorandom, then the resulting generator is pseudorandom: Assume that f is an old generation function and g is a new generation function with double input size. Then we simply use the two-step counter-assisted generator $\langle \hat{f}, g\rangle$.

Example 7.9. In Example 7.5, we can use RC6 instead of $D_{\mathrm{RC5}} \circ D_{\mathrm{RC5}} \circ D_{h_\kappa}$ as the output function. This results in a faster and more elegant generator. Here too, the diversity is maximal for all $k = 1, \dots, 2^{64}$, and the generator is difficult to distinguish from random if either DES or RC6 is.

7.3. **Cascaded multiple-step counter-assisted generators.** If we have enough state-space (this is usually the case with software encryption), we can cascade multiple-step counter-assisted generators without decreasing the diversity. Consider for example generators $\mathcal{G}_0, \mathcal{G}_1, \dots, \mathcal{G}_{m-1}$ having the same state-space and output-space. For any sequence of positive integers $t_0 < t_1 < \dots < t_{m-1}$, and Latin-square operations $\star_{t_0}, \dots, \star_{t_{m-2}}$ (on spaces of size $t_0, t_1, \dots, t_{m-2}$ blocks, respectively), the $(t_0, t_1, \dots, t_{m-1})$-*step cascade* is defined to be

$$\mathcal{G}_{\text{cascade}} = \mathcal{G}_{m-1}^{t_{m-1}} \hat{\star}_{t_{m-2}} \dots \hat{\star}_{t_1} \mathcal{G}_1^{t_1} \hat{\star}_{t_0} \mathcal{G}_0^{t_0}.$$

In the sense of definition 6.4. Here, $(x_0, \dots, x_{t_{j+1}-1}) \hat{\star}_{t_j} (y_0, \dots, y_{t_j-1})$ is defined as the concatenation of $(x_0, \dots, x_{t_{j+1}-t_j-1})$ and $(x_{t_{j+1}-t_j}, \dots, x_{t_{j+1}-1}) \star_{t_j} (y_0, \dots, y_{t_j-1})$.

Using this notation, we have the following:

Theorem 7.10. *For all generators $\mathcal{G}_0, \mathcal{G}_1, \dots, \mathcal{G}_{m-1}$ having the same state-space and output-space, and for any Latin-square operations $\star_{t_0}, \dots, \star_{t_{m-2}}$ (on spaces of size $t_0 < t_1 < \dots < t_{m-2}$ blocks, respectively), the* $(t_0, t_1, \dots, t_{m-1})$-step cascade $\mathcal{G}_{\text{cascade}} = \mathcal{G}_{m-1}^{t_{m-1}} \hat{\star}_{t_{m-2}} \dots \hat{\star}_{t_1} \mathcal{G}_1^{t_1} \hat{\star}_{t_0} \mathcal{G}_0^{t_0}$ *has the following properties:*

(1) $\mathfrak{D}_{\mathcal{G}_{\text{cascade}}}(k) = k$ *for all* $k = 1, 2, \dots n$.
(2) *If either the iteration or the output function of any of the cascaded generators is pseudorandom, then the output of* $\mathcal{G}_{\text{cascade}}$ *is pseudorandom as well.*

Proof. (1) follows from Theorem 7.2, by induction on m. (2) follows readily from Theorem 6.3. □

8. Concluding remarks and further research

We have presented a new mode of operation which makes the diversity of every state sequence provably large with a negligible computational cost. Unlike other solutions, this mode does not introduce new (trivial) risks. The well known threat of "no available theory" on the cycle structure of complicated iterative generators (see, e.g., [4, p. 525], [3, p. 22], [16, §17.6], and [6, p. 347]) is eliminated. It is important to stress, however, that the diversity measures only one aspect of security, and is clearly not sufficient for evaluating the cryptographical strength of the generator.

Our new mode has various possible implementations via multiple-stepping and/or cascading, which allow the user a wide range of choice to fit the implementation to his constraints and needs. All of the suggested modes require a counter, but in most of the applications a counter either already exists or is easy to maintain. The cascaded mode reduces the provable diversity with respect to the simple counter-assisted mode, but it suggests an interesting new way to combine the cryptographic strength of several generators. The multiple-stepping mode requires a larger state buffer (thus may be more suitable in software applications), but assures perfect diversity.

The cryptographical impact of our modification technique when the functions f or g are not pseudorandom remains open. It is easy to find pathological examples of output functions where the modification makes things worse, but we believe that such pathological cases will be easy to inspect. However, if the user wants complete confidence, then he may wish to replace the output function g by one that he trusts. In this case, it may be worthwhile to use the generator in the two-step mode and gain the maximal possible diversity as in Section 7.

As we have proved, in the multiple-stepping modes it is enough that either the iteration *or* the output function is pseudorandom to obtain pseudorandom output. This suggests combining two functions from "orthogonal" sources, such as in Example 7.5, and combining strength of well studied primitives with with new, promising ones, as in Example 7.9.

The counter-assisted mode suggests many open problems. Some of these problems are mentioned in the paper. To these we can add practical problems such as the challenge of finding a seed s for which the counter-assisted generator with DES as the iteration function has $\mathfrak{D}_{DES(s)}(k) \approx \sqrt{k}$ for some large k, and theoretical problems such as statistical analysis of the behavior of the state sequence of counter-assisted generators.

References

[1] NIST's *Advanced Encryption Standard* home page, `http://csrc.nist.gov/encryption/aes/aes_home.htm` .

[2] R. Anderson, R. Gibbens, C. Jagger, F. Kelly, and M. Roe, *Measuring the diversity of random number generators*, preprint.

[3] W.G. Chambers, *On random mappings and random permutations*, Lecture Notes in Computer Science **1008** (1995), 22–28.

[4] D. Gollman and W.G. Chambers, *Clock-controlled shift registers: A review*, IEEE Journal on Selected Areas in Communications **7** (1989), 525–533.

[5] E.M. Coven and G.A. Hedlund, *Periods of some nonlinear shift registers*, Journal of Combinatorial Theory (A) **27** (1979), 186–197.

[6] T.W. Cusick, C. Ding, and A. Renvall, *Stream Ciphers and Number Theory*, North-Holland Mathematical Library **55**, Elsevier, Amsterdam 1998.

[7] O. Goldreich, S. Goldwasser and M. Micali, *How To Construct Random Functions*, J. of the ACM 33 (1986) 792–807.

[8] K. Kjeldsen, *On the cycle structure of a set of nonlinear shift registers with symmetric feedback functions*, Journal of Combinatorial Theory (A) **20** (1976), 154–169.

[9] D.E. Knuth, *The Art of Computer Programming* **2**, Addison-Wesley, Massachusetts: 1981, 5–6.

[10] M. Luby, *Pseudorandomness and its applications*, Princeton University Press, Princeton, NJ: 1996.

[11] M. Luby and C. Rackoff, *How to construct pseudorandom permutations and pseudorandom functions*, SIAM J. Comput. **17** (1988), 373–386.

[12] S. Lucks, *Faster Luby-Rackoff ciphers*, Proc. Fast Software Encryption, Lecture Note in Computer Science **1039** (1996), 189–203.

[13] M. Naor and O. Reingold, *On the construction of pseudorandom permutations: Luby-Rackoff revisited*, J. Cryptology **12** (1999), 29–66.

[14] National Bureau of Standards, *Data encryption standard*, Federal Information Processing Standard, U.S. Department of Commerce, FIPS PUB 46, Washington, DC, 1977.

[15] Ronald L. Rivest, *The RC5 Encryption Algorithm*, Proceedings of the 1994 Leuven Workshop on Fast Software Encryption (Springer 1995), 86–96.

[16] B. Schneier, *Applied Cryptography*, John Wiley and Sons, 1996.

[17] B. Tsaban, *Bernoulli numbers and the probability of a birthday surprise*, Discrete Applied Mathematics, to appear. `http://arxiv.org/abs/math.NA/0304028`

[18] B. Tsaban and U. Vishne, *Efficient linear feedback shift registers with maximal period*, Finite Fields and their Applications **8** (2002), 256–267. `http://arxiv.org/abs/cs.CR/0304010`

[19] U.V. Vazirani, *Randomness, adversaries and computation*, Ph.D. Thesis, U.C. Berkeley: 1986.

[20] A.C. Yao, *Theory and Applications of Trapdoor Functions*, Proc. 23-rd IEEE Symp. Foundations of Computer Science (1982), 80–91.

Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot 76100, Israel
E-mail address: shamir@wisdom.weizmann.ac.il

Department of Mathematics, Bar-Ilan University, Ramat-Gan 52900, Israel
E-mail address: tsaban@macs.biu.ac.il
URL: http://www.cs.biu.ac.il/~tsaban