DYNAMICAL PSEUDORANDOM NUMBER GENERATORS

CLARISE HAN

Abstract. Pseudorandom number generators (PRNGs) are essential to cryptography, simulations, and randomized algorithms. This paper explores the mathematical foundations and properties of dynamical PRNGs, which, in the context of arithmetic dynamics, are generators based on iterated arithmetic dynamical maps over finite fields. We first examine the classical linear congruential generators (LCGs) and demonstrate their vulnerability to partial-output attacks. In contrast, we then investigate a family of nonlinear multivariate polynomial systems with controlled degree growth, which offer strong uniformity guarantees. By comparing linear and nonlinear constructions, we highlight the trade-offs between efficiency, security, and pseudorandomness and the potential of nonlinear dynamical systems in building robust PRNGs.

Contents

1. Introduction

Random number generators (RNGs) lie at the heart of many computational applications, including cryptography, simulations, and randomized algorithms. There are two main types: true random number generators (TRNGs) and pseudorandom number generators (PRNGs). TRNGs generate truly random numbers based on

Date: May 14, 2025.

unpredictable physical processes, such as thermal noise. While this provides high-quality randomness, it also causes the outputs to be expensive to generate and non-reproducible.

Due to these drawbacks, most practical applications turn to PRNGs. A PRNG, broadly speaking, is an algorithm that produces a sequence of numbers that appears random despite being deterministically generated. A PRNG can be viewed as a dynamical system, generating sequences by iterating a map $f : S \to S$ on a state space S, starting from an initial value u_0 called the seed:

$$u_0, \quad u_{n+1} = f(u_n), \quad n = 0, 1, \ldots.$$

A fundamental challenge in designing PRNGs is achieving a suitable balance between pseudorandomness, security against prediction, and computational efficiency.

Historically, many widely-adopted PRNGs have relied on linear maps. A canonical example is the linear congruential generator (LCG) over a finite field $\mathbb{F}_p$, defined by the recurrence relation

$$u_{n+1} \equiv au_n + b \pmod{p},$$

where $a, b, u_0 \in \mathbb{F}_p$ and $a \neq 0$. While LCGs are efficient and easy to implement, their linear structure renders them vulnerable in cryptographic settings. Even partial knowledge of an LCG's output (e.g., masking certain coordinates of each output) can be exploited to recover the LCG's seed and predict future values, as demonstrated in [GIGPS13].

These limitations have led researchers to explore nonlinear PRNGs, which are typically more resistant to structural attacks due to the generators' added complexity. Of particular interest are polynomial systems with controlled degree growth, as in the system defined in [OS10], where iterated polynomials exhibit polynomial rather than exponential degree growth, enabling more efficient computation and stronger uniformity guarantees.

This paper explores the mathematical foundations, constructions, and properties of PRNGs. In Section 2, we formalize the notion of a dynamical PRNG and outline criteria for PRNG effectiveness. Section 3 examines linear generators over finite fields and their insecurity against partial-output attacks. Section 4 analyzes nonlinear polynomial systems with controlled degree growth, focusing on their implications on pseudorandom guarantees.

2. Background

2.1. Dynamical PRNGs.

Definition 2.1. A *dynamical PRNG* is defined by a pair $\langle S, f \rangle$, where S is the state space and $f : S \to S$ is a deterministic map. The generator produces sequences via iteration:

$$u_0, \quad u_{n+1} = f(u_n), \quad n = 0, 1, \ldots.$$

In the context of arithmetic dynamics, f is often a polynomial or rational function, and S is typically a finite field or residue ring.

2.2. Examples of Dynamical PRNGs.

Key examples of dynamical PRNGs include the following:

2.2.1. *Linear Congruential Generators (LCGs).*

$$u_{n+1} \equiv au_n + b \pmod{m},$$

where $a, b \in \mathbb{Z}/m\mathbb{Z}$ and $a \neq 0$. LCGs are efficient but suffer from predictability due to their linear structure.

2.2.2. *Inversive Congruential Generators (ICGs).*

$$u_{n+1} \equiv au_n^{-1} + b \pmod{m},$$

where $a, b \in \mathbb{Z}/m\mathbb{Z}$, $a \neq 0$, and u_n^{-1} denotes the modular inverse of u_n modulo m. ICGs are nonlinear, which improves their resistance to prediction due to their increased complexity. However, computing modular inverses is often more computationally intensive than linear operations.

2.2.3. *Power Generators.*

$$u_{n+1} \equiv u_n^d \pmod{m},$$

where $d \in \mathbb{Z}_{>1}$ and $\gcd(u_n, m) = 1$ to ensure the exponentiation is well-defined modulo m.

2.2.4. *Polynomial Systems.* More generally, one can consider polynomial systems over finite fields. Given a system of r polynomials $\mathcal{F} = \{f_0, \ldots, f_{r-1}\}$ in r variables over a finite field $\mathbb{F}_q$, the sequence is defined by

$$\mathbf{u}_{n+1} = \mathcal{F}(\mathbf{u}_n), \quad \mathbf{u}_0 \in \mathbb{F}_q^r.$$

2.3. **Criteria for "Good" Dynamical PRNGs.** The effectiveness of dynamical PRNGs can be evaluated based on three primary criteria: pseudorandomness, security, and efficiency [AK09, Section 9].

2.3.1. *Pseudorandomness.* Pseudorandomness refers to how closely the output sequence of a PRNG resembles a truly random sequence, measured using a chosen set of statistical tests.

The most fundamental requirement is *uniformity*, meaning that the outputs should be evenly distributed. Another important property is a *long period*, which ensures that the output sequence does not repeat prematurely. In practice, pseudorandomness is typically assessed using standard test suites like NIST and DIEHARD.

In this paper, we focus primarily on uniformity, which can be quantified using a metric called *discrepancy*. Intuitively, the discrepancy of a sequence measures the worst-case deviation between the sequence's distribution and the uniform distribution: the more uniform a sequence, the lower its discrepancy. We define discrepancy formally as follows:

Definition 2.2. Consider a sequence Γ of N points in the m-dimensional unit cube $[0,1)^m$, represented as

$$\Gamma = (\gamma_{n,0}, \ldots, \gamma_{n,m-1})_{n=0}^{N-1},$$

where each point $(\gamma_{n,0}, \ldots, \gamma_{n,m-1}) \in [0,1)^m$.

Let B be a subinterval or "box" of this m-dimensional unit cube of the form

$$B = [\alpha_1, \beta_1) \times \ldots \times [\alpha_m, \beta_m) \subseteq [0,1)^m,$$

where $0 \leq \alpha_j \leq \beta_j \leq 1$ for each $j = 1, \ldots, m$. The volume of such a box is denoted by $|B| = \prod_{j=1}^{m} (\beta_j - \alpha_j)$.

Define $T_\Gamma(B)$ as the number of points in Γ that fall inside the box B. Since the points are distributed in $[0,1)^m$, the expected number of points in B under uniform distribution is $N \cdot |B|$.

Then, the *discrepancy* $\Delta(\Gamma)$ of the sequence Γ is given by the worst-case deviation between the actual and expected proportions of points in any box B. Formally,

$$\Delta(\Gamma) = \sup_{B \subseteq [0,1)^m} \left| \frac{T_\Gamma(B)}{N} - |B| \right|,$$

where the supremum is taken over all boxes $B \subseteq [0,1)^m$ as defined above.

To analyze the uniformity of sequences generated by PRNGs, we can bound the discrepancy. For a sequence Γ of N points as defined in Definition 2.2, the trivial discrepancy bound is $\Delta(\Gamma) \leq 1$. This is because if all the points in the sequence are the same and we consider the box that contains all the points, which has zero volume, we get $T_\Gamma(B) = N$ and $|B| = 0$, which gives the maximum discrepancy of $\left|\frac{N}{N} - 0\right| = 1$. The general discrepancy bound is given by the Koksma-Szüsz inequality from [DT97, Theorem 1.21]:

LEMMA 2.3. *For a positive integer parameter $L > 1$ and a sequence Γ of N points as defined in Definition 2.2, we can bound the discrepancy as follows:*

$$\Delta(\Gamma) < O\left(\frac{1}{L} + \frac{1}{N} \sum_{\substack{|a_0|,\ldots,|a_{m-1}| \leq L \\ a_0^2 + \cdots + a_{m-1}^2 > 0}} \prod_{j=0}^{m-1} \frac{1}{|a_j| + 1} \left| \sum_{n=0}^{N-1} \exp\left(2\pi i \sum_{j=0}^{m-1} a_j \gamma_{n,j} \right) \right| \right),$$

where $\mathbf{a} \in \mathbb{Z}^m$ is an m-dimensional integer vector.

More details about the Koksma-Szüsz inequality and proof can be found in [DT97, Theorem 1.21]. In later sections, we will use this inequality as a tool to bound discrepancy.

The inner absolute value of the Koksma-Szüsz inequality corresponds to the *exponential sum*, defined formally as follows:

DEFINITION 2.4. For a sequence $(\mathbf{u}_n)$ of length N and an integer vector $\mathbf{a} \in \mathbb{Z}^m$, the *exponential sum* is

$$S_\mathbf{a}(N) = \sum_{n=0}^{N-1} \mathbf{e}\left(\sum_{i=0}^{m-1} a_i u_{n,i} \right),$$

where $\mathbf{e}(z) = \exp(2\pi i z/p)$ maps z to a point on the unit circle.

Thus, bounding the exponential sum helps bound the discrepancy. The trivial bound for the exponential sum is $|S_a(N)| \leq N$ since the exponential sum adds N terms $\mathbf{e}(z)$ with magnitude 1, so the maximum magnitude of the exponential sum is N. The general exponential bound is given by the Weil Bound from [LN97, Chapter 5]:

LEMMA 2.5. *For a non-constant polynomial $F \in \mathbb{F}_p[X_0, \ldots, X_m]$ of total degree D, we can bound the exponential sum as follows:*

$$\left| \sum_{x_0,\ldots,x_m=1} \mathbf{e}(F(x_0, \ldots, x_m)) \right| < D p^{\frac{m+1}{2}}.$$

More details about the Weil Bound and proof can be found in [LN97, Chapter 5]. In later sections, we will see how stronger exponential sum bounds, enabled by controlled degree growth of the polynomial, can be used to improve the discrepancy bound, thus giving more robust uniformity guarantees.

2.3.2. *Security.* Security refers to the computational difficulty of predicting the PRNG based on the outputs. This property is essential in cryptographic applications, where predictability of outputs can lead to a complete compromise of security. A standard security test is the *next-bit test*: given the first i outputs of a PRNG, no polynomial-time adversary should be able to predict the $(i+1)$-th output. Another important security test is resistance to *state compromise extensions*: if an attacker gains access to part of an output of a PRNG, they should not be able to recover past or future outputs.

Security tests assume that the attacker knows the algorithm being used, which is formalized by *Kerckhoffs's principle*, a foundational concept in cryptography stating that a system should remain secure even if the system's underlying algorithm, except the initial seed, is public knowledge.

Example 2.6. For example, for LCGs, under Kerckhoffs's principle, an attacker would know that the PRNG has the form $u_{n+1} \equiv au_n + b \pmod{p}$, but does not know the value of u_0.

2.3.3. *Efficiency.* Efficiency refers to the computational cost of producing each output of the PRNG. This cost is typically measured by the number and complexity of basic operations required per iteration, including *arithmetic operations* (addition, multiplication, subtraction, division) and *bitwise logical operations* (XOR, OR, AND, NOT).

In theoretical analysis, the cost of these basic operations depends on the bit-length k of the operands, as outlined in Table 1. Under the standard model of computation, we assume operations are carried out via grade-school methods:

Example 2.7. For example, addition requires $O(k)$ time: each bit of the two inputs is processed once, with a carry that may propagate. Subtraction also processes inputs bit-by-bit, requiring $O(k)$ time. Similarly, bitwise logical operations operate bit-by-bit and require $O(k)$ time.

In contrast, multiplication requires $O(k^2)$ time: each bit of the first operand must be multiplied by each bit of the second. More efficient algorithms like Karatsuba or FFT-based multiplication can reduce this to sub-quadratic time, but the $O(k^2)$ model remains standard for baseline analysis.

Division and modular reduction also require $O(k^2)$ time in the standard model. Division is typically implemented via long division, which involves $O(k)$ steps, each performing $O(k)$-bit comparisons and subtractions. Modular reduction is usually performed by computing the quotient and remainder through division: given $x = qm + r$, we recover $x \mod m = r$. Therefore, modular reduction has the same complexity as division. When the modulus m has a special form (e.g., a power of 2), modular reduction can be much faster since it can be implemented using bitwise masking or shifts.

In practice, when k is small enough to fit within a machine word (e.g., 32 or 64 bits), arithmetic and bitwise logical operations are implemented directly in hardware and execute in constant time. However, for cryptographic applications that

operate on large integers (e.g., 256-bit or 2048-bit fields), the theoretical complexities become more relevant.

Operation	**Theoretical Complexity**	**Practical Complexity**
Addition and Subtraction	$O(k)$	$O(1)$
Multiplication	$O(k^2)$	$O(1)$
Division and Modular Reduction	$O(k^2)$	$O(1)$ (optimized for constants or powers of 2)
Bitwise Logical	$O(k)$	$O(1)$

TABLE 1. Operation Complexity.

We will now analyze the efficiency of some example PRNGs.

EXAMPLE 2.8. LCGs use the recurrence $u_{n+1} \equiv au_n+b \pmod{m}$, which involves one multiplication, an addition, and a modular reduction per iteration. Thus, for k-bit inputs, the theoretical cost is $O(k^2)$ per iteration.

EXAMPLE 2.9. Inversive congruential generators (ICGs) use the recurrence $u_{n+1} \equiv au_n^{-1}+b \mod m$. Each iteration involves computing a modular inverse, followed by a multiplication, an addition, and a modular reduction. The inversion is computed using the extended Euclidean algorithm (EEA), which finds integers x, y such that $u_n x + my = \gcd(u_n, m)$. If $\gcd(u_n, m) = 1$, then $x \equiv u_n^{-1} \mod m$. The EEA takes $O(\log m)$ iterations, each involving $O(k^2)$ arithmetic on k-bit integers, so the modular inversion step requires $O(k^3)$. Thus, for k-bit inputs, the theoretical cost is $O(k^3)$ per iteration, which is much less efficient than the LCG.

Using these three criteria, we can compare PRNGs, as summarized in Table 2.

3. Linear Dynamical PRNG

Although computationally efficient, linear generators over finite fields have severe security vulnerabilities. From just a few outputs, the generator can be predicted. Even hiding certain coordinates of each output does not protect the generator, which has been shown using an algorithm developed by [GIGPS13].

In this section, we will first formalize linear generators over finite fields and give an example. Then, we will discuss how such generators can be attacked using raw outputs. After that, we will present the algorithm that predicts the generator using masked outputs.

3.1. **Construction.**

DEFINITION 3.1 (Linear Generator Over a Finite Field). Let p be a prime number and s be a positive integer. Let $q = p^s$ where $\mathbb{F}_q$ is the finite field over q elements. The *linear generator over* $\mathbb{F}_q$ produces a sequence (u_n) defined by the recurrence relation:

$$u_{n+1} = au_n + b, \quad a, b \in \mathbb{F}_q, \quad a \neq 0, \quad n = 0, 1, 2, \ldots \tag{3.1}$$

where $u_0 \in \mathbb{F}_q$ is the initial seed.

PRNG Type	Pseudorandomness	Efficiency	Security
Linear Congruential Generator ($f(x) = (ax+b) \bmod p$)	Max uniformity when $b \not\equiv 0 \pmod p$, $a \equiv 1 \pmod p$ when p odd, $a \equiv 1 \pmod 4$ when $p = 2$ [AK09]	Fast because only linear operations needed to generated each output; assuming k-bit integers, the theoretical cost is $O(k^2)$ per iteration (see Example 2.8)	Very poor since there are many attacks (e.g., [GIGPS13]) (see Section 3)
Inversive Congruential Generator ($f(x) = (ax^{-1}+b) \bmod p$)	Stronger discrepancy than general bounds: $O(N^{-1/2}p^{1/4}\log p)$, where N is the sequence length and $N \geq p^{1/2}\log p$ [GNS99]	Slow because modular inverse is computationally expensive; assuming k-bit integers, the theoretical cost is $O(k^3)$ per iteration (see Example 2.9)	Better than linear since the attacks are weaker (e.g., [BGPGS03])
Power Generator ($f(x) = x^d \bmod p$)	Stronger discrepancy than general bounds: $O(N^{-5/9}p^{1/4})$, where N is the sequence length [EM08]	Slow due to exponentiation; the degree growth after k iterations is $O(d^k)$ (exponential degree growth)	Better than linear since the attacks are weaker (e.g., [KKK20]); secure for large $d \approx p$
Nonlinear Polynomial System Generator [OS10] (system with $m+1$ polynomials over the finite field $\mathbb{F}_p$)	Stronger discrepancy than general bounds: $O(N^{-\frac{1}{4(m+1)^2}+o(1)}(\log N)^m)$, where N is the sequence length and $N \approx p^{m+1}$ (see Section 4)	Not that slow because the degree growth after k iterations for a system with $m+1$ polynomials is $O(k^m)$ (polynomial degree growth) (see Section 4)	Not fully tested

TABLE 2. PRNG Effectiveness Summary.

Each element in $\mathbb{F}_q$ can be represented in $\mathbb{F}_p$ using a unique coordinate vector. Given a basis $(\gamma_1, \ldots, \gamma_s)$ of $\mathbb{F}_q$ over $\mathbb{F}_p$, each $\alpha \in \mathbb{F}_q$ can be expressed uniquely using the basis expansion as

$$\alpha = c_1\gamma_1 + \cdots + c_s\gamma_s, \quad c_1, \ldots, c_s \in \mathbb{F}_p$$

where $(c_1, \ldots, c_s)$ is the coordinate vector of α, whose elements can be called the *coefficients of α in the basis* $(\gamma_1, \ldots, \gamma_s)$.

We use $w(\alpha)$ to denote the *weight* of α, which refers to the number of nonzero coefficients in the basis expansion of α.

3.2. **Example.**

EXAMPLE 3.2. Let $\mathbb{F}_4$ be the finite field with $4 = 2^2$ elements. We can construct $\mathbb{F}_4$ as $\mathbb{F}_2[\theta]/(\theta^2+\theta+1)$, where θ is a root of the irreducible polynomial x^2+x+1 over $\mathbb{F}_2$. Given the basis $(1, \theta)$ of $\mathbb{F}_4$ over $\mathbb{F}_2$, every element of $\mathbb{F}_4$ can be written uniquely as $c_1 + c_2\theta$ for $c_1, c_2 \in \mathbb{F}_2$.

To construct the linear generator from Definition 3.1, define the recurrence:

$$u_{n+1} = u_n + (\theta + 1), \quad u_0 = 0,$$

where the multiplier $a = 1$ and the increment $b = \theta + 1$. Then the sequence (u_n) is:

$$\begin{aligned} u_0 &= 0, \\ u_1 &= u_0 + (\theta + 1) = \theta + 1, \\ u_2 &= u_1 + (\theta + 1) = (\theta + 1) + (\theta + 1) = \theta, \\ u_3 &= u_2 + (\theta + 1) = \theta + (\theta + 1) = 1, \\ u_4 &= u_3 + (\theta + 1) = 1 + \theta + 1 = 0, \end{aligned}$$

with the first four elements repeating. Note that this generator cycles through all four elements of $\mathbb{F}_4$ and thus has the maximal period of 4. Thus, strong uniformity of LCGs can be made possible with careful choices of a and b, as further described in [AK09, Chapter 9] and summarized in Table 2.

3.3. **Attack.** The linear structure of linear generators makes them susceptible to security attacks. Given a sequence of consecutive outputs, we will show how we can recover the multiplier a and the increment b, as well as the initial seed u_0. Suppose we observe $t \geq 3$ consecutive outputs $w_0, \ldots, w_{t-1}$. Then, we know that $w_1 = aw_0 + b$ and $w_2 = aw_1 + b$. Subtracting these equations, we have $w_2 - w_1 = a(w_1 - w_0)$, so

$$a = \frac{w_2 - w_1}{w_1 - w_0}.$$

Once a is known, we can solve for b:

$$b = w_1 - aw_0.$$

Now that we know a and b, with any single output u_n, we can compute

$$u_{n-1} = (u_n - b)a^{-1}$$

to recursively recover all past outputs, including u_0.

3.4. **Masked Attack.** Given the predictability of linear generators, a proposed strategy to overcome this is to discard or "mask" certain coordinates of the outputs. We formalize masking through the concept of I-approximations:

Definition 3.3 ([GIGPS13, Definition 1]). Given a basis $(\gamma_1, \ldots, \gamma_s)$ of $\mathbb{F}_q$ over $\mathbb{F}_p$, let two elements $\alpha, \beta \in \mathbb{F}_q$ have coordinate vectors $(c_1, \ldots, c_s)$ and $(d_1, \ldots, d_s)$, respectively. Let $I \subseteq \{1, 2, \ldots, s\}$ be a set of coordinate positions. Then α and β are called *I-approximations* of each other if $c_i = d_i$ for all $i \notin I$.

In other words, α and β differ only in the coordinates indexed by I. Equivalently, $\alpha - \beta \in L(I)$, where

$$L(I) = \left\{ \sum_{i \in I} c_i \gamma_i \mid c_i \in \mathbb{F}_p, \quad \forall i \in I \right\}.$$

A masked output is defined as follows:

Definition 3.4. Let $u_n \in \mathbb{F}_q$ be a true output of the generator and $w_n \in \mathbb{F}_q$ be an I-approximation of u_n. Then w_n is called an *I-masked output* of u_n.

Let us now see how we can attack the linear generator using masked outputs rather than true outputs. Given a and b from Eq. 3.1 and a sequence of I-masked outputs w_n for each true output w_n, we will show how we can recover which coordinates are masked (i.e., the set I) and the initial seed u_0 using an algorithm by [GIGPS13] that runs in polynomial time of complexity $(\log q)^{O(1)}$.

The algorithm operates in two stages:

- recovering the set of hidden positions I.
- recovering the initial seed u_0.

3.4.1. *Recovering the Set I.* First, we will describe how the algorithm finds I.

THEOREM 3.5 ([GIGPS13, Theorem 2]). *Given $s+1$ consecutive I-masked outputs $w_0, w_1, \ldots, w_s$ of the true outputs $u_0, u_1, \ldots, u_s$, and $|I| = k$, we can recover I in polynomial time, as long as the error terms $(\epsilon_0, \ldots, \epsilon_s)$, where $\epsilon_i = u_i - w_i \in L(I)$ for $i = 0, \ldots, s$, are not in a certain exceptional set $E(a, b, I) \subset L(I)^{s+1}$, where*

$$E(a,b,I) = \left\{ (x_0, \ldots, x_s) \in L(I)^{s+1} \mid \left(\sum_{i=0}^{s} a_i x_i \right)_j = 0 \right\}$$

with cardinality $|E(a, b, I)| = p^{k(s+1)-1}$.

Proof. Since $\mathbb{F}_q$ is an s-dimensional extension of $\mathbb{F}_p$, the multiplier a in Eq. 3.1 satisfies a minimal polynomial

$$P(a) = a_0 + a_1 a + \ldots + a_s a^s = 0 \quad \text{for} \quad a_i \in \mathbb{F}_p$$

where at least one of $a_0, \ldots, a_s \in \mathbb{F}_p$ is not zero.

We consider two cases for b in Eq. 3.1: $b = 0$ and $b \neq 0$. If $b = 0$, then $u_n = a^n u_0$. Then

$$a_0 u_0 + a_1 u_1 + \ldots + a_s u_s = 0.$$

With $\epsilon_i = u_i - w_i$, we get

$$a_0 \epsilon_0 + \ldots + a_s \epsilon_s = w$$

where

$$w = -a_0 w_0 - \ldots - a_s w_s.$$

Expanding w in the basis $(\gamma_0, \ldots, \gamma_s)$, we get

$$w = \sum_{i=1}^{s} d_i \gamma_i.$$

If $d_j \neq 0$, then the algorithm returns $j \in I$. If $d_j = 0$, the algorithm returns $j \notin I$, which is correct unless $(\epsilon_0, \ldots, \epsilon_s) \in E(a, b, I)$, where $E(a, b, I)$ is the hyperplane

$$E(a,b,I) = \left\{ (x_0, \ldots, x_s) \in L(I)^{s+1} \mid \left(\sum_{i=0}^{s} a_i x_i \right)_j = 0 \right\}.$$

Since $|E(a, b, I)| = p^{k(s+1)-1}$, the failure probability is

$$\frac{|E(a,b,I)|}{|L(I)^{s+1}|} = \frac{p^{k(s+1)-1}}{p^{k(s+1)}} = \frac{1}{p}.$$

Thus, the algorithm is correct with probability $1 - 1/p \geq 1/2$.

If we repeat the process with multiple tuples of consecutive approximations and make a majority decision on whether $j \in I$, the probability of an incorrect result becomes exponentially smaller.

For $b \neq 0$, define $v_n = u_n - b(1 + a + \ldots + a^{n-1})$. Since $v_n = a^n u_0$, the argument for the $b = 0$ case applies to v_n, concluding the proof.

□

3.4.2. *Recovering the Seed u_0.* Next, we will describe how the algorithm uses the recovered I to find u_0.

THEOREM 3.6 ([GIGPS13, Theorem 3]). *Given a set $I \subseteq \{1, \ldots, s\}$ with $|I| = k$ and t consecutive I-masked outputs $w_0, w_1, \ldots, w_{t-1}$ of the true outputs $u_0, u_1, \ldots, u_{t-1}$ where $k + 1 \geq t > 2$, we can recover u_0 in polynomial time, as long as the multiplier a in Eq. 3.1 is not in a certain exceptional set $F(I) \subset \mathbb{F}_q$ with cardinality $|F(I)| < 2p^{\delta_t} + \binom{k}{t-2} p^{2k-t+2}$, where δ_t is the largest divisor of s less than t. We define $F(I)$ as $F(I) = F_1 \cup F_2$, where*

$$F_1 = \{a \in \mathbb{F}_q \mid a\alpha = \beta, 0 < w(\alpha) \leq k - (t-2), w(\beta) \leq k\},$$

$$F_2 = \{a \in \mathbb{F}_q \mid F(a) = 0, F(X) \in \mathbb{F}_p[X]^*, \deg F \leq t - 1\},$$

where $w(\alpha)$ is the weight of α as defined in Section 3.1.

Proof Sketch. From Eq. 3.1 and the error terms $\epsilon_i = u_i - w_i$ for $i = 0, \ldots, t-1$, we get

$$\epsilon_{i+1} + w_{i+1} = a(\epsilon_i + w_i) + b \quad \text{for} \quad i = 0, \ldots, t-2.$$

Isolating the error terms, we derive $a\epsilon_i - \epsilon_{i+1} = w_{i+1} - (b + aw_i) \quad \text{for} \quad i = 0, \ldots, t-2$. Letting $e_i = w_{i+1} - (b + aw_i)$ for $i = 0, \ldots, t-2$, we can formulate the system

$$a\eta_i - \eta_{i+1} = e_i \quad \text{for} \quad i = 0, \ldots, t-2 \tag{3.2}$$

where we are trying to solve for the variables η_i. By construction, this system has at least one solution $(\eta_0, \ldots, \eta_{t-1}) = (\epsilon_0, \ldots, \epsilon_{t-1})$, which can be found using Gaussian elimination.

If this solution is unique, we can recover the initial seed

$$u_0 = w_0 + \epsilon_0.$$

However, this fails if the solution is not unique. We will show that this happens when a is in a certain set $F(I) = F_1 \cup F_2$, where

$$F_1 = \{a \in \mathbb{F}_q \mid a\alpha = \beta, 0 < w(\alpha) \leq k - (t-2), w(\beta) \leq k\},$$

$$F_2 = \{a \in \mathbb{F}_q \mid F(a) = 0, F(X) \in \mathbb{F}_p[X]^*, \deg F \leq t - 1\},$$

where $w(\alpha)$ is the weight of α as defined in Section 3.1.

Let the system given by Eq. 3.2 have another solution $(\tilde{\epsilon_0}, \ldots, \tilde{\epsilon_{t-1}})$. Since $(\epsilon_0, \ldots, \epsilon_{t-1})$ and $(\tilde{\epsilon_0}, \ldots, \tilde{\epsilon_{t-1}})$ are solutions, we have that $a\epsilon_i - \epsilon_{i+1} = e_i$ and $a\tilde{\epsilon_i} - \tilde{\epsilon_{i+1}} = e_i$, so $a(\epsilon_i - \tilde{\epsilon_i}) - (\epsilon_{i+1} - \tilde{\epsilon_{i+1}}) = 0$ for $i = 0, \ldots, t-2$. If we define the difference

$$d_i = \epsilon_i - \tilde{\epsilon_i} \quad \text{for} \quad i = 0, \ldots, t-1,$$

then $(d_0, \ldots, d_{t-1})$ is a nontrivial solution to

$$ay_j - y_{j+1} = 0 \quad \text{for} \quad j = 0, \ldots, t-2.$$

This implies that $d_j = a^j d_0$.

We will now use these differences d_i to characterize F_1 and F_2. Let's fix a basis of $\mathbb{F}_q$ over $\mathbb{F}_p$ and consider the linear subspace formed by $d_0, \ldots, d_{t-1}$ over $\mathbb{F}_p$. Then we have the following two cases that correspond to sets F_1 and F_2:

The first case is when $d_0, \ldots, d_{t-1}$ are linearly independent over $\mathbb{F}_p$. Then, Gaussian elimination shows that there exists a nonzero element α that is a linear combination of $d_0, \ldots, d_{t-2}$ with weight $w(\alpha) \leq k - (t-2)$, such that $\beta = a\alpha$ has all nonzero components within the set I. This corresponds to $a \in F_1$.

The second case is when $d_0, \ldots, d_{t-1}$ are linearly dependent over $\mathbb{F}_p$. Then, for some $j \in \{1, \ldots, t-1\}$, we have $d_j = c_0 d_0 + c_1 d_1 + \ldots + c_{j-1} d_{j-1}$. Since $d_j = a^j d_0$, we get $a^j = c_0 + c_1 a + \ldots + c_{j-1} a^{j-1}$. Thus, a satisfies a polynomial equation of degree $\leq t-1$ over $\mathbb{F}_p$, which corresponds to $a \in F_2$.

Analyzing the sizes of F_1 and F_2, we can bound the size of the set $F(I) = F_1 \cup F_2$. For F_1, we count the possible choices for α with $w(\alpha) \leq k - (t-2)$ and β with $w(\beta) \leq k$ to get the bound $|F_1| \leq \binom{k}{t-2} * p^{2k-t+2}$. For F_2, we know that, by definition, any $a \in F_2$ is a root of a polynomial with degree $\leq t-1$ over $\mathbb{F}_p$. Since $a \in \mathbb{F}_q = \mathbb{F}_{p^s}$, a has a minimal polynomial that is irreducible over $\mathbb{F}_p$, so $\prod_{a \in F_2}(X - a) \mid \prod_{j|s, j<t}(X^{p^j} - a)$. Thus, $|F_2| \leq \sum_{j|s, j<t} p^j \leq \sum_{j=1}^{\delta_t} p^j \leq 2p^{\delta_t}$. Therefore, the combined set $F(I) = F_1 \cup F_2$ has cardinality $|F(I)| < 2p^{\delta_t} + \binom{k}{t-2} p^{2k-t+2}$.

Thus, the failure probability is bounded by

$$\frac{|F(I)|}{|\mathbb{F}_q|} < \frac{2p^{\delta_t} + \binom{k}{t-2} p^{2k-t+2}}{p^s} = 2p^{\delta_t - s} + \binom{k}{t-2} p^{2k-t+2-s}.$$

If we consider a special optimal case, we can further simplify the bound for $|F(I)|$. Using the trivial estimate $\delta_t \leq t-1$, we get $|F(I)| < 2p^{t-1} + \binom{k}{t-2} p^{2k-t+2}$. Then if we use the optimal number of masked outputs $t = k+1$, we get $|F(I)| < 2p^{k+1} + \binom{k}{k-1} p^{k+1} = (k+2)p^{k+1}$. Thus, for this special case, the failure probability is bounded by $\frac{|F(I)|}{|\mathbb{F}_q|} < \frac{(k+2)p^{k+1}}{p^s} = (k+2)p^{k-s+1}$. □

4. Nonlinear Dynamical PRNG

The security limitations of linear generators exposed by [GIGPS13] motivate the study of nonlinear alternatives.

One such nonlinear generator is the *inversive generator*, which has attracted attention due to its potential for producing pseudorandom sequences. Although it has been attacked, as in [BGPGS03], the attacks are much weaker than for linear generators. Rather strong estimates on the randomness properties of sequences generated by inversive generators are available, as demonstrated in [GNS99]. However, the inversive generator involves a modular inversion at each step, which is a computationally expensive operation, as seen in Example 2.9.

Another promising nonlinear alternative is the *power generator*. These generators have also been attacked, as in [KKK20], but the attacks are much weaker than for linear generators. Additionally, the power generator can produce pseudorandom sequences with strong statistical properties, which has been demonstrated by [EM08]. However, while the power generator offers improved uniformity over linear generators, it is still not immune to the computational challenges inherent in more complex nonlinear systems.

Motivated by these limitations, the authors of [OS10] construct a family of multivariate polynomial systems where iterations exhibit controlled degree growth, enabling both efficient computation and robust uniformity guarantees. Although the security of this system has not yet been fully tested, it is more complex than the linear case, making it harder to predict.

In this section, we will first define the nonlinear multivariate polynomial system by [OS10] and give an example. Then we will provide results for the degree growth and the uniformity bounds in terms of exponential sum and discrepancy.

4.1. **Construction.**

Definition 4.1 (Nonlinear Polynomial System [OS10]). Let $\mathbb{F}_p$ be a finite field, and let $\mathcal{F} = \{f_0, \dots, f_m\}$ be a system of $m+1$ polynomials in $\mathbb{F}_p[X_0, ..., X_m]$ defined as follows:

$$\begin{aligned} f_0(X_0, \dots, X_m) &= X_0 g_0(X_1, \dots, X_m) + h_0(X_1, \dots, X_m) \\ f_1(X_0, \dots, X_m) &= X_1 g_1(X_2, \dots, X_m) + h_1(X_2, \dots, X_m) \\ &\vdots \\ f_{m-1}(X_0, \dots, X_m) &= X_{m-1} g_{m-1}(X_m) + h_{m-1}(X_m) \\ f_m(X_0, \dots, X_m) &= aX_m + b \end{aligned} \tag{4.1}$$

where

$$a, b \in \mathbb{F}_p, \quad a \neq 0, \quad g_i, h_i \in \mathbb{F}_p[X_{i+1}, \dots, X_m], \quad i = 0, 1, \dots, m-1.$$

We impose certain conditions on the polynomials g_i and h_i:

- Each g_i must have a unique leading monomial $X_{i+1}^{s_{i,i+1}} \dots X_m^{s_{i,m}}$:
$$g_i(X_{i+1}, \dots, X_m) = X_{i+1}^{s_{i,i+1}} \dots X_m^{s_{i,m}} + \tilde{g}_i(X_{i+1}, \dots, X_m)$$
where $\deg \tilde{g}_i < \deg g_i = s_{i,i+1} + \dots + s_{i,m}$.
- The degree of h_i is bounded by the degree of g_i:
$$\deg h_i \leq \deg g_i.$$

We define the iterations of the system from Definition 4.1 formally as follows:

Definition 4.2. For the system $\mathcal{F} = \{f_0, \dots, f_m\}$ from Definition 4.1, the *k-th iteration* of the polynomials f_i for each $i = 0, \dots, m$ is given by the recurrence relation

$$f_i^{(0)} = f_i, \quad f_i^{(k)} = f^{(k-1)}(f_0, \dots, f_m), \quad k = 0, 1, \dots. \tag{4.2}$$

Definition 4.3. Iterating the system from Definition 4.1, we can construct a dynamical PRNG. The output sequence is generated by the recurrences

$$u_{n+1,i} \equiv f_i(u_{n,0}, \dots, u_{n,m}) \pmod{p}, \quad n = 0, 1, \dots,$$

for $i = 0, \dots, m$ with some initial values $u_{0,0}, \dots, u_{0,m} \in \mathbb{F}_p$. In terms of iterations, we have

$$u_{n+k,i} \equiv f_i^{(k)}(u_{n,0}, \dots, u_{n,m}) \pmod{p}, \quad n, k \geq 0, \quad i = 0, \dots, m.$$

Using vector notation $\mathbf{w}_n = (u_{n,0}, \dots, u_{n,m})$ and $\mathcal{F} = (f_0(X_0, \dots, X_m), \dots, f_m(X_0, \dots, X_m))$, we can write

$$\mathbf{w}_{n+1} = \mathcal{F}(\mathbf{w}_n).$$

In terms of iterations, we have

$$\mathbf{w}_{n+k} = \mathcal{F}^{(k)}(\mathbf{w}_n).$$

The sequence $(\mathbf{w}_n)$ is eventually periodic with period $T \leq p^{m+1}$ because each vector $\mathbf{w}_n$ has $m+1$ components with p possible values, so there are p^{m+1} possible vectors which must eventually repeat by the pigeonhole principle.

Since the last component in each vector $\mathbf{w}_n$ is generated by the linear polynomial, making this last component vulnerable to security attacks, we discard it and use

$$\mathbf{u}_n = (u_{n,0}, \ldots, u_{n,m-1})$$

as each output in the output sequence of the PRNG.

4.2. **Example.**

EXAMPLE 4.4. Let $\mathbb{F}_{101}$ be the finite field. To construct the PRNG from Definition 4.3, we first define the system $\mathcal{F} = \{f_0, f_1\}$ of 2 polynomials in $\mathbb{F}_{101}[X_0, X_1]$ from Definition 4.1:

$$f_0(X_0, X_1) = X_0X_1 + 3$$
$$f_1(X_0, X_1) = 2X_1 + 5.$$

Then the output sequence of the PRNG is generated by the recurrences

$$u_{n+1,0} \equiv X_0X_1 + 3 \pmod{101}$$
$$u_{n+1,1} \equiv 2X_1 + 5 \pmod{101},$$

where the initial values are $u_{0,0} = 1$ and $u_{0,1} = 1$, so $\mathbf{w}_0 = (u_{0,0}, u_{0,1}) = (1, 1)$. Then the sequence $(\mathbf{w}_n)$ is given by the following calculations:

n	$u_{n,0}$	$u_{n,1}$
0	1	1
1	$f_0(1,1) = 1*1+3 = 4$	$f_1(1,1) = 2*1+5 = 7$
2	$f_0(4,7) = 4*7+3 = 31$	$f_1(4,7) = 2*7+5 = 19$
3	87	43
4	7	91

Discarding the last component of each $\mathbf{w}_n$, we get our PRNG output sequence $(\mathbf{u}_n) = (1, 4, 31, 87, 7)$, which does indeed look random.

4.3. **Degree Growth.** One of the fundamental challenges in polynomial systems is managing degree growth of iterations, which impacts the PRNG's computational efficiency and pseudorandomness guarantees. In typical polynomial systems, repeated iteration causes degrees to grow exponentially.

EXAMPLE 4.5. For example, when iterating the polynomial $f(x) = x^d$, the degree after k iterations is d^k.

However, for the polynomial system from Definition 4.1, the degree grows polynomially rather than exponentially:

LEMMA 4.6 ([OS10, Lemma 1]). *Let $d_{k,i}$ be the degree of the polynomial f_i from the system defined by Definition 4.1 after k iterations. Then*

$$d_{k,i} = \frac{1}{(m-i)!}k^{m-i}s_{i,i+1}\ldots s_{m-1,m} + \psi_i(k), \quad i = 0, \ldots, m-1$$

where $\psi_i(T) \in \mathbb{Q}[T]$ is a polynomial of degree less than $m - i$. For the last component, $d_{k,m} = 1$ for all k.

Proof Sketch. Intuitively, this polynomial degree growth stems from the triangular structure of the system from Definition 4.1. Since each polynomial f_i depends only on variables X_j with indices $j \geq i$, the system avoids the large combinations of terms that would normally occur in general multivariate polynomial systems and cause exponential degree growth. □

Example 4.7. For example, in the system from Example 4.4 where $m = 1$, we can see that for $i = 0$, we have degree $d_{k,0} = \frac{1}{(1-0)!}k^{1-0}s_{0,1} + \psi_i(k) = O(k)$, which is linear, and for $i = 1$, we have degree $d_{k,1} = 1$.

This controlled degree growth offers two main benefits. First, the computational efficiency of this system is improved compared to that of general polynomial systems. Second, the polynomial degree growth enables tighter bounds on uniformity guarantees.

4.4. **Discrepancy.** To analyze uniformity, we can bound the discrepancy for the PRNG from Definition 4.3 as follows:

Theorem 4.8 ([OS10, Theorem 6]). *Let the sequence $(\mathbf{u}_n)$ generated by the PRNG defined by Definition 4.3 have length N and period T where $N \leq T$. For the scaled sequence*

$$\left(\frac{u_{n,0}}{p}, \ldots, \frac{u_{n,m-1}}{p}\right), \quad n = 0, \ldots, N-1,$$

and any fixed integer $\nu \geq 1$, the discrepancy D_N satisfies the bound

$$D_N = O(p^{\alpha_{m,\nu}} N^{-\beta_{m,\nu}} (\log p)^m)$$

where

$$\alpha_{m,\nu} = \frac{2m^2 + 2m\nu + 2m + \nu}{4\nu(m+\nu)} \quad \text{and} \quad \beta_{m,\nu} = \frac{1}{2\nu}.$$

To prove this theorem, we will use the exponential sum bound for the same PRNG from Definition 4.3:

Theorem 4.9 ([OS10, Theorem 4]). *For the sequence $(\mathbf{u}_n)$ generated by the PRNG from Definition 4.3 with length N and period T where $N \leq T$, and any fixed integer $\nu \geq 1$, the exponential sum satisfies the bound*

$$\max_{\gcd(a_0,\ldots,a_{m-1},p)=1} |S_{\mathbf{a}}(N)| = O(p^{\alpha_{m,\nu}} N^{1-\beta_{m,\nu}})$$

where

$$\alpha_{m,\nu} = \frac{2m^2 + 2m\nu + 2m + \nu}{4\nu(m+\nu)} \quad \text{and} \quad \beta_{m,\nu} = \frac{1}{2\nu}.$$

Proof Sketch. This can be proven using the Weil Bound from [LN97, Chapter 5]:

$$\left| \sum_{x_0,\ldots,x_m=1} \mathbf{e}(F(x_0, \ldots, x_m)) \right| < Dp^{\frac{m+1}{2}},$$

and the crucial fact that the system has polynomial degree growth with iterations, as given by Theorem 4.6. This keeps the total degree D small, tightening the exponential bound and giving us the desired bound. □

We are now ready to sketch the proof of Theorem 4.8:

Proof Sketch. We will use the general discrepancy bound given by the Koksma-Szüsz inequality from [DT97, Theorem 1.21]:

$$\Delta(\Gamma) < O\left(\frac{1}{L} + \frac{1}{N} \sum_{\substack{|a_0|,\ldots,|a_{m-1}| \leq L \\ a_0^2+\cdots+a_{m-1}^2 > 0}} \prod_{j=0}^{m-1} \frac{1}{|a_j|+1} \left| \sum_{n=0}^{N-1} \exp\left(2\pi i \sum_{j=0}^{m-1} a_j u_{n,j}/p\right)\right|\right).$$

Taking $L = p-1$ and applying the exponential sum bound from Theorem 4.9, we get the desired discrepancy bound. □

This discrepancy bound from Theorem 4.8 exponentially improves upon the general bound. Consider T, N close to the maximum p^{m+1}, i.e., $T \geq N \geq p^{m+1+o(1)}$, where $o(1)$ is an approximation error term. Then $p \approx N^{\frac{1}{m+1}}$. For $\nu = 1$, we have $\alpha_{m,1} = \frac{2m^2+4m+1}{4(m+1)}$ and $\beta_{m,1} = \frac{1}{2}$. Then we can rewrite the bound as

$$\begin{aligned} D_N &= O((N^{\frac{1}{m+1}})^{\alpha_{m,1}} N^{-\beta_{m,1}} (\log p)^m) \\ &= O\left(N^{-\frac{1}{4(m+1)^2}+o(1)} (\log p)^m\right). \end{aligned}$$

Compared to the general bound, this bound $O\left(N^{-\frac{1}{4(m+1)^2}+o(1)} (\log p)^m\right)$ achieves a power-law improvement.

References

[AK09] Vladimir Anashin and Andrei Khrennikov. *Applied Algebraic Dynamics*. De Gruyter, Berlin, New York, 2009.

[BGPGS03] Simon R. Blackburn, Domingo Gomez-Perez, Jaime Gutierrez, and Igor E. Shparlinski. Predicting the inversive generator. In *Cryptography and coding*, volume 2898 of *Lecture Notes in Comput. Sci.*, pages 264–275. Springer, Berlin, 2003.

[DT97] Michael Drmota and Robert F. Tichy. *Sequences, discrepancies and applications*, volume 1651 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1997.

[EM08] Edwin D. El-Mahassni. On the distribution of the power generator over a residue ring for parts of the period. *Revista Matematica Complutense*, 21:319–325, 2008.

[GIGPS13] Jaime Gutierrez, Alvar Ibeas, Domingo Gomez-Perez, and Igor E. Shparlinski. Predicting masked linear pseudorandom number generators over finite fields. *Design, Codes and Cryptography*, (67):395–402, 2013.

[GNS99] Frances Griffin, Harald Niederreiter, and Igor E. Shparlinski. On the distribution of nonlinear recursive congruential pseudorandom numbers of higher orders. In *Applied algebra, algebraic algorithms and error-correcting codes (Honolulu, HI, 1999)*, volume 1719 of *Lecture Notes in Comput. Sci.*, pages 87–93. Springer, Berlin, 1999.

[KKK20] Novak Kaluđerović, Thorsten Kleinjung, and Dusan Kostic. Improved key recovery on the legendre PRF. Cryptology ePrint Archive, Paper 2020/098, 2020.

[LN97] Rudolf Lidl and Harald Niederreiter. *Finite fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, second edition, 1997. With a foreword by P. M. Cohn.

[OS10] Alina Ostafe and Igor E. Shparlinski. On the degree growth in some polynomial dynamical systems and nonlinear pseudorandom number generators. *Mathematics of Computation*, 79(269):501–511, 2010.

Massachusetts Institute of Technology
Email address: clariseh@mit.edu