Engineering Nonlinear Pseudorandom Number Generators

Samuel Neves and Filipe Araujo

CISUC, Dept. of Informatics Engineering
University of Coimbra, Portugal
`sneves@dei.uc.pt`, `filipius@uc.pt`

Abstract. In the era of multi and many-core processors, computer simulations increasingly require parallel, small and fast pseudorandom number generation. Although linear generators lend themselves to a simpler evaluation that ensures favorable properties like guaranteed period, they may adversely affect the result of simulations or be quite large. Conversely, nonlinear generators may provide apparently random sequences, but are either very slow or difficult to analyze regarding their period.
This is the case of our previous functions, Tyche and Tyche-i. Despite being among the fastest in their class and having average periods of 2^{127}, they may contain smaller cycles of arbitrary size. To overcome this limitation, in this paper we explore different forms of counters impacting either the state or the speed of the generator. We also introduce two number-theoretic generators that use 2×127 bits for periods of 2^{116} and 2^{125} and low to moderate computational costs. We experimentally demonstrate the efficiency of our new generators and observe that they exchange speed for period guarantees in a tradeoff that seems widespread in state-of-the-art random number generators.

Keywords: Nonlinear pseudorandom generator, Elliptic curves, Elliptic curve linear congruential generator

1 Introduction

Pseudorandom number generators (PRNGs) attempt to generate sequences that have similar properties to that of genuinely random sequences, while simultaneously being deterministic and reproducible. Most commonly-used generators are based on number-theoretic constructions: linear congruential generators (LCGs) work over the ring of integers modulo m, linear feedback shift registers from the ring of polynomials modulo p, and so on. Working in such mathematical structures makes it easier to reason about period, equidistribution, and other desirable properties of PRNGs. However, they are often *linear*, which adversely affects real-world simulation results, and speed, due to the required arithmetic.

Nonlinear generators can be divided in two main camps. On one hand, we have the number-theoretic generators, which rely on number theory to demonstrate their properties, such as period and distribution. On the other hand, there are the cryptography-based generators, which rely on the avalanche effect of cryptographic primitives to mask any relationship from one state to the next.

Recently, non-number-theoretic, nonlinear generators have been proposed with some success [20,24]. These often try to "mix" the bits of the PRNG's state as best as possible across iterations, thus obtaining quality random sequences. Despite passing statistical tests, these generators tend to rely on heuristic assumptions, making it impossible to guarantee their period or distribution properties.

Some generators, most notably the Mersenne Twister [17], use very large states to thwart some of the drawbacks of their linearity. The SIMD-oriented version of the Mersenne Twister [23] is indeed quite faster, but still uses a state of similar size as the original MT, requiring hundreds of 128-bit words. Smaller-state generators, like Xorshift [16], have small state but struggle to provide adequate statistical quality [22].

Meanwhile, computer architecture is shifting. It is becoming harder to put more transistors within the same chip area, and manufacturers are often forced to choose between memory and execution units. For now, general-purpose chips, such as `x86-64` and ARM, tend to favor fast memory (cache), while special-purpose chips (e.g., GPUs) lean toward more computational power. The current trend appears to be towards more cores, and therefore one can expect less memory per running thread. Future PRNGs should therefore save memory and share no state between threads, to avoid contention. By decreasing order of relevance, modern PRNGs should have the following properties:

High quality Any proposed generator must pass stringent statistical tests, such as TestU01's "Big Crush" battery [13].

Large period While opinions vary about the minimum acceptable period, we aim for a minimum of 2^{128}.

Small state The size of the state should not be significantly larger than the binary logarithm of its period, which we assume to be roughly 2^{128}.

Fast The generator must be as quick as possible, because it is often in the critical path of simulations.

Linearity enables design of fast generators with provable properties, such as period, statistical distribution, and small state [14,26]. However, linear congruential generators often possess a lattice structure that can skew the results of a simulation [15,9]. To run away from linear generators' drawbacks, we add another requirement to a good generator: *nonlinearity.* This means that a generator should not be representable as an affine transformation in $\mathbb{F}_2, \mathbb{Z}_n$, or any such ring. Since provable properties such as period often affect the speed of PRNGs, in this paper we propose a number of different algorithms, including elliptic curve generators, to explore the speed/period tradeoff. We believe that our generators provide excellent state-of-the-art speeds for their properties.

Our contribution in this paper is twofold: firstly, we present two nonlinear number-theoretic generators based on elliptic curves over a prime field. Secondly, we introduce two new tweaks to the Tyche generator [20], representing two different tradeoffs between period guarantee and speed. We analyze and discuss the results in Section 3.

2 Small nonlinear generators

Generators based on number-theoretic structures are among the most common in the literature, and include the linear congruential generator, the linear feedback shift register and its many variants, the inversive congruential generator, Blum-Blum-Shub [5], and others. Generally, generators in this category that are linear are reasonably fast, while nonlinear ones tend to underperform and are less commonly used in real applications.

The most general construction for a generator is of the form

$$\begin{aligned} S_i &= f(S_{i-1}) \\ x_i &= g(S_i) \end{aligned}$$

The functions f and g are known as the *transition* and *output* functions. This construction allows for much freedom in the design process, and encompasses the vast majority of the existing generators. For example, in the common linear congruential generator [14], f is the $S_i = aS_{i-1} + b \bmod m$ recurrence, while g is usually a truncation of the current value. In this case, the computational effort is skewed almost completely towards f. Conversely, a generator created from, e.g., a block cipher E_K (e.g., AES-128) in counter mode [8] has $S_i = S_{i-1} + 1$, and $x_i = E_K(S_i)$. This case is computationally skewed towards the output function.

There are advantages and disadvantages on each of these extremes. It is easier to find a simple function that achieves full period — for example $f(x) = x + 1$ — and use the output function to produce random-looking outputs, than it is to find a random-looking transition function that also has large guaranteed period. However, the latter case is often computationally better than the former; simple functions often do not have good statistical properties, and the output function has to do a large amount of work to achieve enough bit diffusion.

To achieve full period in the least possible space, the transition function must be invertible, i.e., a permutation. Non-invertible functions have an expected period of $2^{n/2}$ for n bits of state ; such functions therefore need $2n$ bits to achieve 2^n period, which breaks our space-efficiency requirement. On the other hand, there are not many restrictions on the output function.

2.1 Elliptic curves

Elliptic curves have found many uses in cryptography [18,10], and are the current leading candidate for public-key key-exchanges and digital signatures. The set of solutions (x, y) to the Weierstrass equation over some field F

$$E(F) : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \quad x, y, a_i \in F \tag{1}$$

together with the "point at infinity" P_∞ and an point addition operation $+$ form an Abelian group. The order of this group, $\#E$, follows the famous Hasse bound:

$$\#F + 1 - 2\sqrt{\#F} < \#E(F) < \#F + 1 + 2\sqrt{\#F} \tag{2}$$

These two facts allow one to define a linear congruential-style generator of the form

$$P_i = G + P_{i-1} \tag{3}$$

which will then have an order equal to that of the generator G. Apart from having guaranteed period, such generators are also known to have desirable statistical properties [11,6], have small state (i.e., one point), and can skip ahead quickly by noticing that

$$P_n = nG + P_0 \tag{4}$$

where nG means repeated addition and can be done in $O(\log n)$ additions using standard techniques. For a generator of this kind to be fast, two things are required: *i*) fast arithmetic over F; *ii*) fast point addition.

Despite all elliptic curves being representable in affine coordinates satisfying Equation 1, several other representations exist that present certain performance tradeoffs: Jacobian projective coordinates [7], Montgomery curves [19], Jacobi quartics [7], Hessian curves [7], Edwards curves [4], Twisted Edwards curves [2], and others[1]. Of these, there are two main representations that are of interest to us, when it comes to cheap point addition: Montgomery curves and twisted Edwards curves.

Algorithm 2.1: MontgomeryDouble(X, Z)

```
comment: K is Constant for a given curve
$K \leftarrow (a_2 + 2)/4$
$A \leftarrow (X + Z)^2$
$B \leftarrow (X - Z)^2$
$C \leftarrow A - B$
return $(AB, C(B + KC))$
```

Twisted Edwards curves have recently been used to break speed records in elliptic curve scalar multiplication [3]. However, these curves require rather large point representations (4 field elements) to achieve the best speeds, and even then have *relatively* high addition and doubling costs. In comparison, Montgomery

[1] Most known efficient formulas for various curves and point representations are found in the Explicit-Formulas Database: `http://hyperelliptic.org/EFD/index.html`

curves only require two field elements but do not support full addition, instead requiring *differential addition* to perform scalar multiplication. Point doubling of Montgomery curves, listed in Algorithm 2.1, has the best operation count, at 4 multiplications, 4 additions, and 1 multiplication by a constant. This fact motivates our new generator mode, vaguely resembling the Blum-Blum-Shub [5] generator:

$$P_i = P_{i-1} + P_{i-1} = 2P_{i-1}. \tag{5}$$

This new recurrence has two advantages: it only requires point doubling, and requires less fixed constants by removing the need for G. This recurrence computes the sequence $2P, 4P, \ldots, 2^iP$ and is therefore also possible to skip ahead by computing

$$P_n = (2^n \bmod \operatorname{ord}(P_0))P_0 \tag{6}$$

and the period of this sequence is given by $\operatorname{ord}(2) \bmod \operatorname{ord}(P_0)$. The order of this generator is thus dependent not only on the number of points in the elliptic curve, but also on the order of 2 in the ring of integers modulo the order of the initial point.

2.2 The `M127` generator

Our first concrete proposal of a generator that achieves near 2^{128} period consists of the following Montgomery curve over the integers modulo the Mersenne prime $2^{127} - 1$:

$$y^2 = x^3 + 131074x^2 + x. \tag{7}$$

This curve has order $4 \cdot p_1$, $p_1 = 42535295865117307934202406649106774733$[2]. The constant 131074 was chosen so that $32769 = (131074 + 2)/4$ has Hamming weight 2 and multiplication can be performed via one shift and one addition: $32769x = x \ll 15 + x$.

Points are represented in the traditional Montgomery curve fashion (X, Z). This representation only contains information about the X coordinate of a given point; the Y coordinate is ignored, and thus points are in reality an equivalence class of (X, Y) and $(X, -Y)$. The (X, Z) representation avoids costly inversions by storing X as a fraction, i.e., $X = X/Z$.

The underlying field, $\mathbb{F}_{2^{127}-1}$, was chosen to minimize the cost of modular reduction. Modular reduction by $2^n - 1$ is known to be achievable by the divisionless expression

$$x \bmod (2^n - 1) \equiv (x \bmod 2^n + \lfloor x/2^n \rfloor) \bmod (2^n - 1). \tag{8}$$

To select a starting point from a (say) 128-bit seed s, one can compute $P_0 = s(2, 1)$. Note that $(2, 1)$ has order p_1, and thus there is no chance that an

[2] The order of Montgomery curves is always a multiple of 4.

unlucky seed will get stuck in a small order point forever. The only seed that must be avoided is, of course, p_1 itself, since this would result in the point-at-infinity as the starting point.

This generator is appropriate for architectures where integer multiplication is fast. There are 4 $\mathbb{F}_{2^{127}-1}$ multiplications per iteration, each of which requires 4 $64 \times 64 \to 128$-bit multiplications. While there is plenty of exploitable parallelism in both field and curve arithmetic to attenuate the effect of high-latency multiplication instructions, it may not be enough.

2.3 The `M31x4` generator

The curve from the previous section relied on fast arithmetic over the underlying field. While this can be reasonably expected from large general purpose processors, it is often the case that smaller or specialized processors are unable to perform multiple-precision arithmetic very quickly. Furthermore, small integer multiplication has quadratic complexity, and for CPUs with small register sizes that complexity grows quickly. For this reason we propose the following generator, which only requires 32-bit arithmetic.

This generator uses not one, but 4 Montgomery curves in parallel over the Mersenne prime $2^{31} - 1$:

$$y^2 = x^3 + v_i x^2 + x \tag{9}$$

where $v = \{904572996, 1467357171, 1043599384, 1244578513\}$. The 4 curves have orders of respectively $4 \cdot 536871259$, $4 \cdot 536872363$, $4 \cdot 536872907$, and $4 \cdot 536873203$. At the end of each iteration, the generator outputs the *combination* of the x-coordinates of the points:

$$g(x_0, x_1, x_2, x_3) = x_0 \oplus (x_1 \lll 7) \oplus (x_2 \lll 11) \oplus (x_3 \lll 29) \tag{10}$$

where $\oplus$ means XOR and $\lll$ means rotation towards the most significant bits.

The 4 v_i parameters were chosen to maximize the order of 2 in the respective fields. This makes the overall period of this generator $536871258 \cdot 536872362 \cdot 536872906 \cdot 536873202 \approx 2^{116}$.

This generator allows many different implementation approaches, and is suitable for large CPUs (where it can be implemented using general purpose instruction or SIMD) and GPUs alike.

2.4 Tweaking Tyche with a counter

The Tyche generator [20] is a generator based on the ChaCha core permutation [1], which works in a mode similar to OFB mode in block ciphers. We refer to [20] for the complete description of Tyche. While it shows great performance across many architectures, due to its use of simple 32-bit instructions, it has several drawbacks:

No provable period Treating the core permutation `MIX` as a random permutation allows us to estimate the expected period of a sequence to be roughly 2^{127}. However, this says nothing about the actual cycle structure of Tyche, and unlikely as it may be, there may be some hidden pitfalls in this generator.

No random access While Tyche provides some higher level parallelism support by defining different stream starting points, it is impossible to jump ahead inside a single stream. This may be inconvenient in some situations.

In this section we propose a tweak to Tyche [20], named Tyche-CTR-R, to change the mode of operation of Tyche. Once the initial state is set up, the least significant 64 bits are used as a counter incremented by the odd constant 5871781008561895865[3], while the most significant 64 bits remain constant, serving as identifier (a *nonce*) of the current stream. Then, this state is processed R times by the `MIX` function, and the least significant word is returned. Algorithm 2.2 describes Tyche-CTR-R.

Algorithm 2.2: TYCHE-CTR-$R(a, b, c, d)$

```
$(b, a) \leftarrow (a + 2^{32}b) + 5871781008561895865$
$(a', b', c', d') \leftarrow (a, b, c, d)$
for $i \leftarrow 0$ to $R$
  do $(a', b', c', d') \leftarrow MIX(a', b', c', d')$
return $(a')$
```

Our experiments have suggested that 5 rounds, i.e., Tyche-CTR-5, are sufficient to achieve enough diffusion to pass known statistical tests. It is easy to see that the period of Tyche-CTR-R is 2^{64} and it is easy to jump ahead arbitrarily within a stream, by adding an appropriate multiple of the constant used in Algorithm 2.2. This also implies the generator is massively paralellizable. One should notice that Tyche-CTR-5 provides 2^{64} distinct streams with a guaranteed period of 2^{64}, and still enables further tweaks to the lengths of the counter and nonce.

2.5 Tyche as a counter-dependent generator

The tweak presented in the previous section was fairly aggressive: instead of one `MIX` call per iteration, we now require $R \geq 5$ calls to achieve the same effect. This is a massive slowdown, even though it does enable some desirable properties, and the higher latency may be hidden by computing several values in parallel.

We propose in this section another tradeoff: a 2^{32} guaranteed minimum period, 2^{159} average period, and 160 bits of state. The approach we follow is known as *counter-dependent* generators [25], and is pictured in Algorithm 2.3.

[3] Counters that add an odd constant different from 1 are often known as Weyl generators.

Table 1: Timing and period information of Section 2 generators, a 128-bit LCG, and a 128-bit EC-RNG.

Generator	State bits	Cycles/Iteration	Average period	Min period	Jump ahead
LCG-128	127	32	$2^{127}-1$	$2^{127}-1$	Yes
EC-LCG	3×127	238	$\approx 2^{127}$	$\approx 2^{127}$	Yes
XORWOW [21]	192	7	$2^{192}-2^{32}$	$2^{192}-2^{32}$	Yes
`M127`	2×127	96	$\approx 2^{125}$	$\approx 2^{125}$	Yes
`M31x4`	2×127	38	$\approx 2^{116}$	$\approx 2^{116}$	Yes
Tyche [20]	128	12	$\approx 2^{127}$	1	No
Tyche-i [20]	128	6	$\approx 2^{127}$	1	No
Tyche-CD-32	160	12	$\approx 2^{159}$	2^{32}	No
Tyche-CTR-5	128	44	2^{64}	2^{64}	Yes

Algorithm 2.3: TYCHE-CD-32(a, b, c, d, e)

```
$e \leftarrow e + (e^2 \vee 5) \bmod 2^{32}$
$(a, b, c, d) \leftarrow MIX(a, b, c, d)$
return $(b + e)$
```

Since this tweak does not enable random stream access, we opted to use the T-function $x + (x^2 \vee 5) \pmod{2^n}$, proven by Klimov and Shamir to be invertible and single-cycled [12]. This function is executed in parallel with `MIX`, and is not expected to significantly slow down the generator. Additionally, the greater complexity of this function provides some more diffusion than a simpler counter.

3 Results and discussion

The generators described in Section 2 have passed the TestU01 [13] "BigCrush" battery of tests. Therefore, to evaluate their performance, we include timings for a 128-bit LCG with modulus $2^{127} - 1$ and multiplier 43, and an EC-LCG using projective coordinates over a Weierstrass curve of prime order over the same prime as `M127`, $2^{127} - 1$. Additionally, we also compare the original Tyche algorithm against the variants proposed in Section 2.4 and 2.5.

Table 1 shows the number of cycles per iteration of the aforementioned generators. The timings were obtained on an Intel Core-i7 2630QM "Sandy Bridge" processor, with Turbo Boost and hyperthreading disabled.

It is apparent from Table 1 that `M127`, despite much optimization effort, is quite far from the performance of a similar-period LCG. Nevertheless, it is more than twice as fast as an EC-LCG of similar period, most of this stemming from the faster Montgomery doubling operation. The speed of the `M31x4` generator is on the same order of magnitude as the LCG, due to the lack of large integer arithmetic in favor of parallel small elliptic curves and a simple combiner.

One pattern that emerges is that the number-theoretic generators do not seem to have significant advantages over the counter-based generator. The counter-based generator Tyche-CTR-5 not only is faster than the nonlinear number-theoretic generators, but also contains every feature found in the latter. Note that if storage space is not an issue, it is possible to, e.g., run 4 or 8 parallel instances of the Tyche-CTR in SIMD registers, resulting in a significant speedup. This would be impossible in recursive generators.

Finally the Tyche-CD-32 generator strikes a balance between features and speed. It is as fast as Tyche, hiding the extra instructions in between Tyche's critical path, guarantees a reasonable minimum period of 2^{32}, and does not require vectorization tricks to be extremely quick.

Ultimately, this is the tradeoff generators seem to make: linear generators can be fast, small, and provably periodic, but suffer in statistical properties; nonlinear generators can be fast and small, but their period will not be provable and are purely sequential; nonlinear number theoretic generators will have provable period, but will suffer in speed due to the heavy arithmetic.

Future work involves investigating additional number-theoretic constructs to obtain faster generators; perhaps elliptic curves are not the optimal choice, despite their popularity in cryptography.

Acknowledgments

This work has been supported by the project CMU-PT/RNQ/0015/2009, TRONE — Trustworthy and Resilient Operations in a Network Environment.

References

1. Bernstein, D.J.: ChaCha, a variant of Salsa20. Workshop Record of SASC 2008: The State of the Art of Stream Ciphers (January 2008)
2. Bernstein, D.J., Birkner, P., Joye, M., Lange, T., Peters, C.: Twisted Edwards curves. In: AFRICACRYPT'08: Proceedings of the Cryptology in Africa 1st international conference on Progress in cryptology. pp. 389–405. Springer-Verlag, Berlin, Heidelberg (2008)
3. Bernstein, D.J., Lange, T.: Analysis and optimization of elliptic-curve single-scalar multiplication. IACR Cryptology ePrint Archive 2007, 455 (2007)
4. Bernstein, D.J., Lange, T.: Faster addition and doubling on elliptic curves. In: ASIACRYPT'07: Proceedings of the Advances in Crypotology 13th international conference on Theory and application of cryptology and information security. pp. 29–50. Springer-Verlag, Berlin, Heidelberg (2007)
5. Blum, L., Blum, M., Shub, M.: A simple unpredictable pseudo-random number generator. SIAM J. Comput. 15(2), 364–383 (1986)
6. Chen, Z., Gomez, D., Pirsic, G.: On lattice profile of the elliptic curve linear congruential generators. Periodica Mathematica Hungarica (Accepted), – (2012)
7. Chudnovsky, D.V., Chudnovsky, G.V.: Sequences of numbers generated by addition in formal groups and new primality and factorization tests. Adv. Appl. Math. 7(4), 385–434 (1986)

8. Dworkin, M.: Recommendation for Block Cipher Modes of Operation — Methods and Techniques. Special Publication 800-38A, National Institute of Standards and Technology, Gaithersburg, MD 20899-8930 (December 2001)
9. Ferrenberg, A.M., Landau, D.P., Wong, Y.J.: Monte Carlo simulations: Hidden errors from "good" random number generators. Phys. Rev. Lett. 69, 3382–3384 (Dec 1992)
10. Hendrik W. Lenstra, J.: Elliptic curves and number-theoretic algorithms. In: Gleason, A.M. (ed.) Proceedings of the International Congress of Mathematicians, volume 1. pp. 99–120. American Mathematical Society, Providence (1987)
11. Hess, F., Shparlinski, I.E.: On the linear complexity and multidimensional distribution of congruential generators over elliptic curves. Des. Codes Cryptography 35(1), 111–117 (Apr 2005), `http://dx.doi.org/10.1007/s10623-003-6153-0`
12. Klimov, A., Shamir, A.: Cryptographic Applications of T-Functions. In: Matsui, M., Zuccherato, R.J. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 3006, pp. 248–261. Springer (2003)
13. L'Ecuyer, P., Simard, R.: TestU01: A C library for empirical testing of random number generators. ACM Trans. Math. Softw. 33(4), 22 (2007)
14. Lehmer, D.: Mathematical methods in large-scale computing units. In: Proceedings of the 2nd Symposium on Large-Scale Digital Calculating Machinery. pp. 141–146. Harvard University Press, Cambridge, Massachusetts (1949)
15. Marsaglia, G.: Random numbers fall mainly in the planes. PNAS 61(1), 25–28 (September 1968), `http://dx.doi.org/10.1073/pnas.61.1.25`
16. Marsaglia, G.: Xorshift RNGs. Journal of Statistical Software 8(14), 1–6 (7 2003)
17. Matsumoto, M., Nishimura, T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul. 8(1), 3–30 (1998)
18. Miller, V.S.: Use of elliptic curves in cryptography. In: Williams, H.C. (ed.) Advances in cryptology: CRYPTO '85. Lecture Notes in Computer Science, vol. 218, pp. 417–426. Springer, Berlin (1986)
19. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computation 48, 243–264 (1987)
20. Neves, S., Araujo, F.: Fast and Small Nonlinear Pseudorandom Number Generators for Computer Simulation. In: 9th International Conference on Parallel Processing and Applied Mathematics (PPAM 2011). Torun, Poland (September 2011)
21. NVIDIA Corporation: CURAND Library (July 2013), `http://docs.nvidia.com/cuda/curand/`
22. Panneton, F., L'ecuyer, P.: On the Xorshift Random Number Generators. ACM Trans. Model. Comput. Simul. 15(4), 346–361 (Oct 2005)
23. Saito, M., Matsumoto, M.: SIMD-oriented fast Mersenne Twister: a 128-bit pseudorandom number generator. In: Monte Carlo and Quasi-Monte Carlo Methods 2006, pp. 607–622. Springer (2008)
24. Salmon, J.K., Moraes, M.A., Dror, R.O., Shaw, D.E.: Parallel random numbers: as easy as 1, 2, 3. In: Lathrop, S., Costa, J., Kramer, W. (eds.) SC. p. 16. ACM (2011), `http://doi.acm.org/10.1145/2063384.2063405`
25. Shamir, A., Tsaban, B.: Guaranteeing the diversity of number generators. Inf. Comput. 171(2), 350–363 (Jan 2002)
26. Tausworthe, R.C.: Random Numbers Generated by Linear Recurrence Modulo Two. Mathematics of Computation 19, 201–209 (1965)