PERIOD LENGTHS OF CHAOTIC PSEUDO-RANDOM NUMBER GENERATORS

Jörg Keller   Hanno Wiese
FernUniversität in Hagen
LG Parallelität und VLSI
58084 Hagen, Germany
joerg.keller@fernuni-hagen.de

ABSTRACT
Cryptographic communication protocols frequently employ random numbers to achieve desirable properties. Often, generators for pseudo-random numbers (PRNGs) are employed. A class of PRNGs that currently gains popularity are chaotic PRNGs which are derived from chaotic systems in physics. While their mathematical properties make them promising, typical implementations of such generators use IEEE 754 floating point representations instead of real numbers. We show that this leads to periods that are much shorter than expected, and thus renders these PRNGs potentially insecure.

KEY WORDS
Cryptographic protocols, pseudo-random number generators, random mappings, chaotic systems.

1 Introduction

Many cryptographic protocols need random numbers to achieve their objectives. Instead of random numbers, most generators supply pseudo-random numbers. There is a wealth of pseudo-random number generators (PRNGs) for cryptographic applications. Every PRNG should have a reasonably long period before its output sequence repeats itself, and its output sequence should pass statistical test suites so that it sufficiently resembles a truly random sequence. A cryptographic PRNG additionally should have the property that its future behaviour cannot be predicted from its past output [9].

In recent years, a class of algorithms called *chaotic* PRNGs have appeared in the literature. Those chaotic PRNGs originate from physics. Their state s is a real in the interval $[0 : 1]$, their output bit is computed as a function of the state. In the simplest case, if $s \geq 0.5$, then the output is 1, else the output is 0. The state transition function is a function $f : [0 : 1] \to [0 : 1]$ that exhibits chaotic behaviour. Because of this chaotic behaviour, the output is assumed to have desirable statistical properties.

If such an algorithm is implemented on a contemporary computer, the reals are typically implemented as floating point numbers, normally following the standard IEEE 754. Thus, the set S of representations available for the interval $[0 : 1]$ is finite, e.g. the size n of S is about 2^{30} for single precision floats. The state transition function changes from f to a function $\tilde{f} : S \to S$, where for $x \in S$, the value $\tilde{f}(x)$ corresponds to an evaluation of f with IEEE 754 arithmetic on x. Thus the implementation of a chaotic PRNG is a finite state automaton with state transition function $\tilde{f}$. Thus, it also has a period with a finite length. The period length is an important parameter of a cryptographic PRNG, because if it is too short, then the PRNG is suspectible against attacks.

Restricting a chaotic system to a finite universe always can cause problems because the Lyapunov exponent is not greater than zero anymore. In order to test the quality of a finite-state PRNG, one can apply statistical tests to its output sequence, see e.g. [8] for theoretical underpinnings, or the Diehard, NIST and other suites for practice. However, we will follow a different path, that directly allows to test the period length of implementations.

As f has chaotic properties, one assumes $\tilde{f}$ to look rather "random", i.e. as if chosen equiprobable from the set of all functions from S to S, which has a size of $|S|^{|S|}$. Such a situation has been called a *random mapping*, and expected values have been derived [3] e.g. for the length of the longest cycle, which is the best period to be hoped for in such a mapping. For a random mapping on a set S of size n, the expected length of the longest period is $\Theta(\sqrt{n})$, with a constant factor close to 1, the largest connected component has an expected size of about $0.7 \cdot n$, and the expected number of connected components is about $0.5 \cdot \ln n$ [3]. Hence, also a floating point implementation of a chaotic PRNG should have at least such a behaviour.

We have implemented several several chaotic PRNGs and computed their period experimentally. Our findings are that in all cases, the period lengths are much shorter than expected. Thus, despite the desirable mathematical properties of the chaotic functions, their use in floating-point based implementations of PRNGs cannot be recommended. Our analysis of the logistic map partly explains the experimental results.

The remainder of this paper is organized as follows. In Section 2, we present the investigated chaotic functions. In Section 3, we review our method to compute the periods in random mappings. In Section 4, we present experimental results. In Section 5, we underpin our findings by an analysis of the logistic map, and in Section 6, we conclude.

2 Chaotic Functions

The definitions of a chaotic function $f : R \to R$, where R is a an interval in the reals, vary widely. Most definitions agree on the following properties of a sequence of points $x, f(x), f(f(x)), \ldots$ [2, 5]:

- f reacts sensibly to changes in x, i.e. even small changes to the value of x result in large changes of the sequence.
- f is topologically transitive, i.e. almost every element of R can be connected to almost every other element of R by a finite sequence.
- f is topologically dense, i.e. even small intervals of R contain periodic points[1] of f.

A popular chaotic function, that underlies also some other chaotic functions, is the so-called *logistic map* [1]:

$$\begin{aligned} f_1 &: [0:1] \to [0:1] \\ f_1(x) &= a \cdot x \cdot (1-x),\ a \approx 4\ . \end{aligned}$$

Another, so-called trigonometric chaotic function, was presented in [6]:

$$\begin{aligned} f_2 &: [0:1] \to [0:1] \\ f_2(x) &= \sin^2\left(z \cdot \arcsin\sqrt{x}\right),\ z > 1\ . \end{aligned}$$

A third, k-dimensional function [7], is defined by:

$$\begin{aligned} f_3^k &: [0:1]^k \to [0:1]^k \\ f_3^k(x_1, \ldots, x_k) &= (y_1, \ldots, y_k) \\ y_1 &= (1-\varepsilon) \cdot f_1(x_1) + \varepsilon \cdot f_1(x_k) \\ y_i &= (1-\varepsilon) \cdot f_1(x_i) + \varepsilon \cdot f_1(x_{i-1}) \\ & \quad i \geq 2,\ \varepsilon \in (0:1)\ . \end{aligned}$$

While this function is intended as a stream cipher, we note that a stream cipher is also a kind of pseudo-random number generator that generates a random key stream that is mixed with the plain text to form the cipher text. We will consider variants of f_3 for $k = 2$ and $k = 4$. Note that for $k = 2$, we get a special case if $\varepsilon = 0.5$: After one step, $y_1 = y_2$ and thus f_3 reduces to f_1.

3 Computing Periods of Random Mappings

A finite state machine with state set S of size n and state transition function $\tilde{f}$ can be represented by a directed graph G_f with node set S and edges $(x, \tilde{f}(x))$ for each $x \in S$, i.e. each node has exactly one outgoing edge. Each weakly connected component (wcc) of such a graph consists of one cycle and a number of trees directed towards their roots, where the roots connect the trees and the cycle. An example is depicted in Figure 1.

[1]These are points to which the sequence returns after a finite number of steps when starting from there.

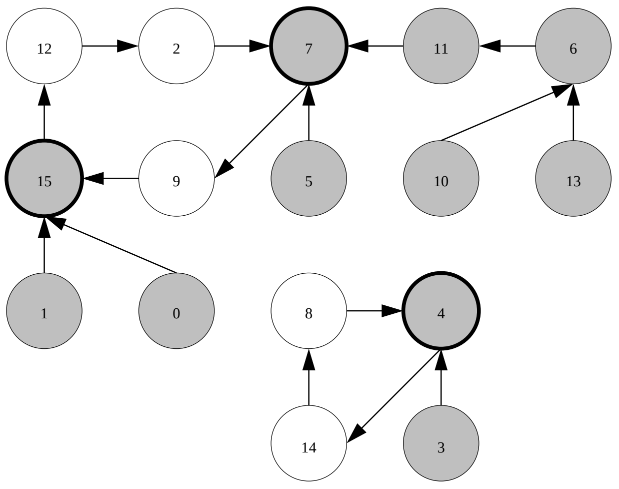

Figure 1. Example of a state transition graph for 16 nodes. Tree nodes are shaded, root nodes are encircled. The cycle lengths in the two weakly connected components are 5 and 3, respectively.

If n is small enough that the adjacency list fits into memory, then one can start at some node x_i not yet visited and follow the path starting in x_i, marking all visited nodes with tag i, until we meet a visited node y for the first time. If the tag of y also is i, then we have found a new cycle, the length of which we can compute by starting from y until we meet it again. Then we increase i by 1. If y's tag is different from i, we have hit on a path already treated, and replace all tags from x_i to y by y's tag. We start with $i = 1$ at an arbitrary x_1, and continue until all nodes are visited. The runtime of this algorithm is $O(n)$, and all cycles and their lengths are found. The number of the cycles corresponds to the number of the weakly connected components. Counting the nodes with tag i computes the size of weakly connected component i.

If n is too large to hold the adjacency list in memory, we also follow paths from some starting nodes, but mark only a small fraction of the nodes on the way. So we will detect when we have hit a new cycle, and can abort a path in a known component early when we hit a marked node. If we use all nodes as starting nodes, then we also get the complete picture. If we use only a random sample of k nodes as starting nodes, then we will at least get the cycles of the larger components, which tend to be the longest ones. As the expected length of a path to a cycle, and the expected length of the longest cycle both are $O(\sqrt{n})$ for a randomly chosen $\tilde{f}$, the expected runtime of this algorithm is $O(k\sqrt{n})$. It can be parallelized on cluster computers to achieve results faster [4].

4 Experimental Results

We have investigated all the chaotic PRNGs from Section 2 with the methods of Section 3. In our implementation, we used floating point arithmetic compliant to the IEEE 754 standard. We have tried to achieve state space sizes of the same order for all four cases. Therefore we have used double precision arithmetic for f_1 and f_2, and single precision arithmetic for f_3^2 and f_3^4.

For single precision arithmetic, there are $2^{23} \cdot (2^7 - 1) + 1 \approx 2^{30}$ representations of reals in the interval $[0:1]$, if denormalized numbers are used as well. There are 23 bits for the mantissa, and with the characteristics values from 0 to 126 we represent the interval $[0:1)$. Finally, we have one representation for 1. For double precision arithmetic, there are $2^{52} \cdot (2^{10} - 1) + 1 \approx 2^{62}$ representations[2] of reals in the interval $[0:1]$, as the mantissa has now 52 bits, and one can use characteristics values from 0 to 1022 for the interval $[0:1)$.

Thus, for f_1 and f_2 we have state spaces of about 2^{62}, and for f_3^2 we have a state space of about $(2^{30})^2 = 2^{60}$. In order to have a similar state space size for f_3^4 as well, we artificially shortened the mantissa in this case so that the state space size was 2^{64}.

Each chaotic function was investigated by the last algorithm of the previous section, with a random sample of starting values, until 2^{28} states had been seen. Our findings are as follows.

The parameter settings below (a, z, ε) are chosen to reflect the known chaotic behaviour of the respective functions.

4.1 Logistic Map

For the logistic map f_1, we would expect a longest cycle of size about $2.4 \cdot 10^9$, and about 22 weakly connected components. In our experiments, which we conducted for $a = 3.80, 3.90, 3.91, \ldots, 3.99$ we only hit one component, which indicates that further components, if they exist, must be very small, because none of the randomly chosen starting points lay in them. The cycle lengths in this component ranged between $8 \cdot 10^6$ ($a = 3.92$) and $67 \cdot 10^6$ ($a = 3.99$), which is about 40 times smaller than expected even for the largest of those cycles.

To find out whether this map contains any further weakly connected components, we re-implemented f_1 with single precision arithmetic, in order to be able to explore the graph G_f completely. Here we would expect about 10 components and a longest cycle of size about $57,000$. For the same values of a as above, we found from 5 ($a = 3.80, 3.94$) to 10 (3.95) components, and longest cycles of lengths from 345 ($a = 3.96$) to $3,630$ ($a = 3.98$). So while there are more components, there is one component that is very large (about 10^9 nodes), and the cycles again are much shorter than expected.

[2]IEEE 754 floating point numbers store the characteristic (exponent minus fixed bias) instead of exponents so that it is a non-negative number.

	$a = 3.90$		$a = 3.99$	
ε	#wcc	longest cycle	#wcc	longest cycle
0.05	3	**2,652,338**	2	2,815,820
0.1	**5**	61,552	4	319,817
0.2	**26**	6,203	7	77,543
0.3	10	4,073	10	**790**
0.4	12	1,621	6	1,956
0.5	10	4,172	11	2,077
0.6	13	**1,222**	10	1,605
0.7	8	1,764	12	1,426
0.8	13	5,762	**14**	391,838
0.9	13	122,783	4	1,998,897
9.95	6	701,680	**3**	1,832,524
0.99	6	10,682	6	**4,677,937**

Table 1. Experimental results for function f_3^2.

4.2 Trigonometric Function

For the trigonometric function f_2, that has the same state space size as the logistic map, we also would expect a longest cycle of size about $2.4 \cdot 10^9$, and about 22 weakly connected components. We performed our experiments for $z = 2, 3, 3.99, 4, 4.11, 20, 150$. In each case, we only hit one component, so that we assume a situation with one overwhelmingly large component, similar to the logistic map. The length of the longest cycle ranges from $1.7 \cdot 10^6$ ($z = 3.99$) to $91.2 \cdot 10^6$ ($z = 4.0$). This is between 25 and $1,400$ times shorter than expected!

4.3 Two-Dimensional Function

For the function f_3^2, we expect about 21 weakly connected components and a longest cycle of length about $0.83 \cdot 10^9$. We used parameter values $\varepsilon = 0.05, 0.1, 0.2, \ldots, 0.9, 0.95, 0.99$ and $a = 3.90, 3.99$. The number of components (#wcc) and the longest cycle lengths found are depicted in Table 1. We see large spans for component count and cycle lengths, but even the longest cycles found are 200 times shorter than expected, while the cycles are very short in the majority of cases.

Astonishingly, while values of ε close to 0 or 1 are said to bring most chaos, for $a = 3.90$ and $\varepsilon = 0.99$ we see a very short cycle.

4.4 Four-Dimensional Function

The expected number of components here is about 22, the expected length of the longest cycle is about $3.36 \cdot 10^9$. We used the same values for ε, but restricted to $a = 3.99$. The number of components ranges from 2 ($\varepsilon = 0.95$) to 52 ($\varepsilon = 0.05$), with half the cases having at most 10 components. The maximum cycle length ranges from 496 ($\varepsilon = 0.7$) to 1,527,499 ($\varepsilon = 0.95$), with all except two being shorter than 50,000. Basically, the same comments as for the two-dimensional function apply, with the difference that the cycle lengths are even farther off from the expected value.

5 Analytical Underpinning

We partly explain the experimental findings for chaotic PRNGs by analyzing the logistic map $f_1(x) = a \cdot x \cdot (1-x)$. As a is typically chosen close to 4, we choose $a = 4$ to simplify the analysis. The derivative of the logistic map is $f_1'(x) = -8x + 4$, which is strictly monotonous decreasing on $[0 : 0.5]$.

The IEEE 754 floating point representation partitions the interval $[0 : 1)$ into sub-intervals $I_i = [2^{-i} : 2^{-i+1})$ of length 2^{-i}, where $1 \leq i \leq 127$, and one sub-interval $I_{128} = [0 : 2^{-127})$, the so-called denormalized numbers. In many implementations, the only denormalized number allowed is zero. Hence, we will not further consider that interval. In each sub-interval I_i, the 2^{23} representations (single precision) are equi-spaced with distance 2^{-i-23}. Obviously,

$$\begin{aligned} f_1(2^{-i}) &= 2^{-i+2} - 2^{-2i+2} \\ f_1(2^{-i+1}) &= 2^{-i+3} - 2^{-2i+4} \ . \end{aligned}$$

We thus see that the sub-interval $I_1 = [0.5 : 1]$ is mapped onto $f_1(I_1) = [0 : 1]$, that the sub-interval $I_2 = [0.25 : 0.5)$ is mapped onto $f_1(I_2) = [0.75 : 1)$, i.e. half of sub-interval I_1, and that for $i \geq 3$, each sub-interval I_i is mapped onto parts of the two neighbouring sub-intervals $I_{i-1} = [2^{-i+1} : 2^{-i+2})$ and $I_{i-2} = [2^{-i+2} : 2^{-i+3})$; from the former, a fraction of $2^{-2i+2}/2^{-i+1} = 2^{-i+1}$ is used, from the latter, a fraction of $(2^{-i+2} - 2^{-2i+4})/2^{-i+2} = 1 - 2^{-i+2}$. Thus, for $i \geq 3$, a sub-interval is mapped onto a fraction of $1 - 2^{-i+1}$ of a sub-interval. Hence, we see a slight form of compression by mapping each I_i, where $i \geq 2$.

As soon as $x \geq 0.5$, the sub-interval is mapped onto the union of all sub-intervals.

By solving $y = 4x(1-x)$ for $x \in [0.5 : 1]$, we obtain $x = (1 + \sqrt{1-y})/2$. We see for $y_i = 2^{-i}$ that

$$x_i = 0.5 + 0.5 \cdot \sqrt{1 - 2^{-i}}$$

and thus a fraction of $a_i = (x_i - x_{i-1})/0.5$ of the 2^{23} representations from interval I_1 is mapped onto sub-interval I_i by function f_1. Table 2 tabulates the fractions for intervals I_1 to I_5. The fractions decrease very quickly.

Therefore, as soon as x has once reached the interval I_1, which it will do starting from I_j in only $j/2$ to j steps, from the following step on only a small fraction of the representations will lead outside sub-interval I_1. Because of the compression mentioned above, and because at most i steps are needed to reach I_1 from I_i, the number of representations in use after I_1 has been reached can be bound from above by

$$2^{23} \cdot \sum_{i \geq 1} i \cdot a_i \approx 1.56 \cdot 2^{23} \ ,$$

i	$a_i = (x_i - x_{i-1})/0.5$
1	0.707106781
2	0.158918623
3	0.069388943
4	0.03283149
5	0.016005148

Table 2. Fractions of I_1 mapped to I_i by f_1.

which is much lower than the 2^{30} representations available in $[0 : 1]$.

For 32-bit fix-point representations, where the representations are equi-spaced over $[0 : 1)$, we can take into consideration the symmetric nature of f_1 between $[0 : 0.5)$ and $[0.5 : 1)$. As f_1' is strictly monotonous decreasing on $[0 : 0.5)$, $f_1'(3/8) = 1$, and f_1 is strictly monotonous increasing on $[0 : 0.5]$, we see that each sub-interval $[a : b]$ with $0 \leq a < b \leq 3/8$ is mapped by f_1 onto a sub-interval $[f_1(a) : f_1(b)]$ with $f_1(b) - f_1(a) > b - a$. In particular, the interval $[0 : 3/8]$ is mapped onto $[0 : 15/16]$, i.e. on average only 40% of the representations in $[0 : 15/16]$ are a target. The interval $[3/8 : 1/2]$ is mapped onto $[15/16 : 1]$, i.e. on average each representation in $[15/16 : 1]$ is twice a target. Taking the symmetry of f_1 into account, the interval $[1/2 : 5/8]$ is also mapped to $[15/16 : 1]$, and the interval $[5/8 : 1]$ is also mapped to $[0 : 15/16]$. Thus, in total, from $15/16$ of the representations about 40% are used (each twice as target), and from $1/16$ of the representations each is used four times as target, and thus about $7/16$ of the available representations ($1.75 \cdot 2^{30}$) are used. This indicates that fix-point representations are better suited than floating point representations to implement chaotic functions.

6 Conclusions

We have investigated implementations of several chaotic pseudo-random number generators (PRNG) on the basis of IEEE 754 floating point numbers. Our focus was on computing the finite period lengths of those implementations. We have found that the periods are much shorter than forecasted by the theory of random mappings. An analysis of the logistic map partly explains this behaviour. The measured values also do not allow to give hints which parameter combinations to choose to obtain somehow acceptable cycle lengths. Thus implementations of chaotic PRNGs based on floating point numbers cannot be recommended for use in cryptographic applications, except if the number of pseudo-random bits to be delivered is very small, in which case however many other methods of good quality exist. Future investigations will investigate fix-point implementations of chaotic PRNGs, because our analysis indicates that they are better suited to implement chaotic functions.

References

[1] M. Ausloos, editor. *The Logistic Map and the Route to Chaos*. Springer, 2006.

[2] R. L. Devaney. *An Introduction to Chaotic Dynamical Systems*. Addison-Wesley, 1998.

[3] P. Flajolet and A. M. Odlyzko. Random mapping statistics. In *Proc. Eurocrypt 89*, pages 329–354, 1989.

[4] J. Heichler, J. Keller, and J. F. Sibeyn. Parallel storage allocation for intermediate results during exploration of random mappings. In *Proc. 20th Workshop Parallel Algorithms, Computing Structures and System Software (PARS '05)*, pages 126–134, 2005.

[5] T. Kapitaniak and S. R. Bishop. *The Illustrated Dictionary of Nonlinear Dynamics and Chaos*. Wiley, 1999.

[6] Z. Kotulski and J. Szczepanski. On constructive approach to chaotic pseudorandom number generators. In *Proc. Regional Conf. on Military Communication and Information Systems (RCMIS 2000) vol. 1*, pages 191–203, 2000.

[7] P. Li, Z. Li, W. A. Halang, and G. Chen. A stream cipher based on a spatiotemporal chaotic system. *Chaos, Solutions & Fractals*, accepted for publication, 2005.

[8] U. Maurer. A Universal Statistical Test for Random Bit Generators. *J. Cryptology*, 5:89–105, 1992.

[9] A. Menezes, P. van Oorshot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.