Contribution to a thematic issue

Carina Heßeling and Jörg Keller*

Replication study challenges and new number formats for chaotic pseudo random number generators

<https://doi.org/10.1515/itit-2021-0065>

Received November 30, 2021; accepted January 21, 2022

Abstract: Chaotic Pseudo Random Number Generators have been seen as a promising candidate for secure random number generation. Using the logistic map as state transition function, we perform number generation experiments that illustrate the challenges when trying to do a replication study. Those challenges range from uncertainties about the rounding mode in arithmetic hardware over chosen number representations for variables to compiler or programmer decisions on evaluation order for arithmetic expressions. We find that different decisions lead to different streams with different security properties, where we focus on period length. However, descriptions in articles often are not detailed enough to deduce all decisions unambiguously. To address similar problems in other replication studies for security applications, we propose recommendations for descriptions of numerical experiments on security applications to avoid the above challenges. Moreover, we use the results to propose the use of higher-radix and mixed-radix representations to trade storage size for period length, and investigate if exploiting the symmetry of the logistic map function for number representation is advantageous.

Keywords: Chaotic pseudo-random number generator, Replication study, Number representation, Compiler Options

ACM CCS: General and reference → Evaluation, Security and privacy → Systems security, Software and its engineering → Compilers, Hardware → Arithmetic and datapath circuits

Article note: This article is an extended version of [10].

*Corresponding author: Jörg Keller, FernUniversität in Hagen, Faculty of Mathematics and Computer Science, 58084 Hagen, Germany, e-mail: joerg.keller@fernuni-hagen.de

Carina Heßeling, FernUniversität in Hagen, Faculty of Mathematics and Computer Science, 58084 Hagen, Germany, e-mail: carina.hesseling@fernuni-hagen.de

1 Introduction

Chaotic pseudo-random number generators (cPRNGs), in particular the logistic map [1], have been considered as a means for secure PRNGs. Experiments [11, 13] revealed that the period lengths are not satisfactory in relation to state size. When studying the descriptions of these experiments, we found that they are missing some implementation details that also influence the period length, even for fixed parameter settings. Hence, replicating those studies experiences some challenges, which we report and from which we derive recommendations for describing experiments including numeric calculations. We also underline experimentally that the implementation details such as chosen number representation, rounding mode and order of evaluation in a formula influence the results.

Beyond the typical number representations integer, i. e. fixed-point, and (binary) floating-point, we next present number representation formats in-between those two that use radix higher than 2 and possibly mixed radix. We demonstrate experimentally that those formats enable to trade accuracy and storage. Thus, our investigations also exhibit relations to the field of approximate computing. Moreover, we investigate if exploiting the symmetry of the logistic map function, i. e. $f(x) = f(1 - x)$ for $x \in [0; 1]$ can be mirrored in the number representation in order to achieve longer period lengths.

The remainder of this article is structured as follows. Section 2 gives information on pseudo-random number generation for security applications, especially when using chaotic functions, summarizes basic information on number representations for rational numbers, and discusses related work. Section 3 presents the challenges that we encountered in experiments with chaotic pseudo-random number generation based on logistic map and reports on own experiments to demonstrate those challenges. In Section 4 we present number formats between standard floating-point and fixed-point representations, as well as experimental results that indicate that those formats allow a tradeoff between storage size and achievable period lengths. Section 5 investigates how symmetry in the

logistic map function can be exploited in number representations, while Section 6 concludes with recommendations on configuration details to be reported about numerical experiments, and an outlook on future work.

2 Background

2.1 Chaotic pseudo-random number generation

A pseudo-random number generator is a finite state automaton. Upon seeding, it receives input which is transformed into a value for the inner state. When the generator is called from an application, an output is generated by applying an output function on the state, and the state transitions to the follow-up state by applying the state transition function (or update function) on the current state. The follow-up state is unique for each state, as the generator receives no further input until a re-seeding takes place.

The pseudo-random number generator should produce outputs that follow the claimed distribution (e.g. if bits are output with uniform distribution, each value should appear 50 % of the time), pass statistical tests of randomness like Diehard [17] (also 0, 1, 0, 1, 0, 1 has uniform distribution but does not look random), and should have a sufficiently long period before output sequences repeat. The latter is unavoidable as the inner state is finite. A good exposition of random number generators can be found in [12].

For using pseudo-random numbers in security applications, it should not be possible to infer the inner state from outputs, or to predict future outputs from current outputs. Finally, it should not be possible to compute from one state previous states or previous outputs [7].

Therefore, while in general the state could be a counter and the complexity rest in the output function, here the state transition function must be one-way. Thus, so-called chaotic functions seemed attractive for use as state transition functions.

The *logistic map* [1] is a function $f : [0; 1] \rightarrow [0; 1]$ with

$$f(x) = a \cdot x \cdot (1 - x). \quad (1)$$

If parameter a is close to 4, then the range $[0; a/4]$ approaches the domain, so the function can be applied iteratively, and it exhibits chaotic behaviour. As $f(0) = f(1) = 0$, we will identify the two and consider only interval $[0; 1]$ in the sequel. A variant of the logistic map has actually been used in a stream cipher [15].

2.2 Finite-length representations for real numbers

For non-integral numbers, the commonly used representation are floating-point numbers [14], which have been standardized by IEEE in 1985, and revised twice since then [8]. Interval $[0; 1)$ is partitioned into $b - 1$ intervals $I_i = [2^{-i-1}; 2^{-i})$ for $i = 0, \dots, b - 2$, plus one interval $I_{b-1} = [0; 2^{-(b-1)})$.

Thus, a floating-point representation is comprised of a power of 2 times a mantissa, i. e. $(i, M) = 2^{-i} \cdot 0.M$. In order to avoid redundancies, i. e. to ensure that a representation $(i, M) \in I_i$, the mantissa in all intervals except I_{b-1} must be in the range $[0.5; 1)$, i. e. the first mantissa bit must be 1. This 1 is called implicit as it need not be stored explicitly. Those numbers are called normalized. The numbers in interval I_{b-1} , which comprises 0, are called de-normalized.

As IEEE standard 754 also covers negative numbers and numbers larger than 1, a floating-point representation with a sign bit s , mantissa bits M of length m and an exponent E of e bits comprises $t = s + m + e$ bits and represents number

$$(-1)^s \cdot 2^{E-b} \cdot \langle 1.M \rangle,$$

where $b = 2^{e-1} - 1$ is the bias and $\langle \cdot \rangle$ denotes binary representation. Exponent value $E = 2^e - 1$ is used to represent infinity, while exponent value $E = 0$ is used to represent *de-normalized* numbers

$$(-1)^s \cdot 2^{1-b} \cdot \langle 0.M \rangle,$$

the most important of which is 0. Thus interval $[0; 1)$ is represented by all representations with sign bit $s = 0$ and $E < b$. The representations are not evenly spread over this interval, but 2^m representations are evenly spaced in each interval $[2^{-i}; 2^{-i+1})$, for $i = 1, \dots, b - 1$, plus one interval $[0; 2^{-(b-1)})$. Multiplication of floating-point representations is done by adding of exponents and multiplying the mantissas, plus rounding if the result length does not fit into the mantissa length. Rounding modes are round to zero, i. e. remove all bits after bit m , round to infinity, i. e. always round up, and round to nearest, i. e. round to zero if bit $m + 1$ is 0, and round to infinity if bit $m + 1$ is 1.

IEEE Standard 854 [9] generalizes floating-point representations to radix or basis $B > 2$. If the basis is a power of 2, then each mantissa digit in the range $\{0, 1, \dots, B - 1\}$ can be represented by $\log_2 B$ bits. The “implicit 1” cannot be used anymore for higher radix, as the first mantissa digit of a normalized representation will be different from 0, but can have $B - 1$ possible values, and also the binary representation of this digit has no bit which is 1 for all those values.

Also fixed-point representations [14] can be used to represent interval $[0; 1)$. For other real numbers, a combination of their integral part and their rational part is used for representation. A fixed-point representation M of m' bits for interval $[0; 1)$ represents $\langle M \rangle \cdot 2^{-m'}$, i. e. the representations are equally spaced over the interval. Addition of fixed-point representations is done just as for integers. Multiplication of fixed-point representations is done by multiplying the mantissas, plus a final right shift of m' bits that realizes the multiplication by $2^{-m'}$ inside the brackets of the following formula:

$$\langle M_1 \rangle \cdot 2^{-m'} \cdot \langle M_2 \rangle \cdot 2^{-m'} = \langle \langle M_1 \rangle \cdot \langle M_2 \rangle \cdot 2^{-m'} \rangle \cdot 2^{-m'}.$$

Also a rounding can be applied similar to floating-point numbers.

Please note that for $m' = t$, i. e. similar memory requirements for both representations, the fixed-point representation has denser representations than floating-point representation in the interval $[2^{-m+1}; 1)$, while floating-point representations are denser in interval $[0; 2^{-m})$.

2.3 Related work

The investigations by Keller and Wiese [11] and Kneusel [13] focus on the influence of parameter a : even for two close values of $a \approx 4$, the period length can vary notably. They use floating-point numbers, but do not report different rounding modes or order of evaluation when doing the two multiplications in $a \cdot x \cdot (1 - x)$. The thesis by Luber [16] considers different ordering of multiplications for fixed-point representations, but does not consider rounding modes as well. Furthermore, Luber does not investigate period length but statistical properties of output bits derived from state values.

An extension of fixed-point format is described in [4], where two different exponents $-m_1$ and $-m_2 < -m_1$ can be used. The choice of exponent is encoded in one bit. This could also be considered as a floating-point format with one exponent bit and mixed radix, and as such be a special case of our proposal in Section 4. In [21], the authors apply signed digit numbers and allow the use of different radix for different parts of the mantissa, yet they do not generalize their approach from signed digit representations to floating-point or fixed-point representations.

[2] point out that higher radix floating-point format is advantageous in reconfigurable hardware, but that normalization (ensuring that not all of the $\log_2 B$ uppermost mantissa bits are zero after an operation) often is not applied there. In contrast, our implementation applies normalization.

In [6], the necessary mantissa length (bitwidth) for a given application is analyzed. This approach might be used as a post-processing to our approaches in Sections 3.4 and 4.

3 Challenges

In this section, we will investigate challenges for replication of execution results, when evaluating the formula $a \cdot x \cdot (1 - x)$. Those challenges illustrate that even on such microscopic scale, replication is difficult, and that therefore, replication of experiments on a larger scale must work with the assumption that certain differences will be inevitable.

3.1 Order of computations

For real numbers, the order of above multiplications does not matter, as multiplication is commutative and associative. Yet, associativity is not given anymore for finite number representations, because rounding will occur. Hence, depending on the choice, there are three possibilities for choosing the arguments of the first multiplication, when we assume commutativity. Even when the source code is given, we still cannot be sure about the order, as compilers have the possibility to re-order operations, e. g. to improve performance for a given processor architecture. Yet, as long as the compiler product, its version and its compilation flags are specified in detail, this can be avoided. However, when the source code is not given, the implementer of the replication code is free to choose order, and over time several different implementation orders have been seen by the author. While the differences between results from different orders will be tiny, the resulting state graphs will have notable differences, cf. Subsection 3.4.

As an example, we assume a representation with 8 significant binary digits and take $a = 3.921875 = 4 - 2^{-4} - 2^{-6} = 11.111011$ and $x = 0.6875 = 11/16 = 0.1011$, i. e. $1 - x = 0.3125 = 5/16 = 0.0101$. Then, $a \cdot x = 2.6962890625 = 10.1011001001$ and will be rounded to $10.101101 = 2.703125$. Further multiplication with $1 - x$ results in 0.8447265625 .

If we multiply x and $1 - x$ first, this results in $0.21484375 = 0.00110111$, which will not be rounded. Further multiplication with a results in $0.84259033203125 = 0.1101011101101$ which will be rounded to $0.11011000 = 0.84375$.

3.2 Rounding modes

Even if the former challenge has been mastered, there is more in the multiplications. So far, we have assumed that rounding goes to the nearest representable number. Yet IEEE 754 standard defines several rounding modes [8], i. e. round towards zero (the rounding applied so far), round towards infinity (always round up to next representation), and round to nearest, i. e. depending on the value of the first digit to be cut away, one rounds up or down.

All the rounding modes are nowadays implemented in hardware by IEEE754 compliant floating-point units in processor cores. The choice of the rounding mode is done either by the BIOS on power up, but might also be changed by the operating system or even by the application program, e. g. via function `fesetround()` in *fenv.h*.

As an example, when multiplying $x = 0.6875$ and $1 - x$ first and then multiplying with $a = 3.921875$, the result is 0.1101011101101. When rounding to nearest is used, we get 0.11011001, while the result would change to 0.11011000 when rounding down. Thus, the difference between the two results is 2^{-8} , which is a notable 0.33 %. Also here, this difference will lead to different state graph structures, e. g. with respect to cycle lengths.

Unfortunately, rounding modes normally are not mentioned when experimental conditions are described in articles, so everyone seems to assume that round to nearest is used.

3.3 Number representation

So far, we have used the straightforward number representation for reals, i. e. floating-point numbers. However, this has some disadvantage. First, two bits of the available format are lost: we are not using negative numbers, as we are focusing on interval $[0; 1)$, and we only use half of the possible exponent values. Second, the representable numbers in IEEE 754 are not evenly distributed over the interval $[0; 1)$ as mentioned in Section 2.

In contrast, a fixed-point number representation comprising 32 bits has 2^{32} equally spaced representations in $[0; 1)$. Fixed-point arithmetic is simple when we choose `uint32` to implement this scheme. Addition (and subtraction) is done as usual, at least as long as no overflow occurs, which we can exclude here. Multiplication is done by converting the arguments to `uint64`, doing the multiplication, shifting by 32 bits to the right, and converting back to `uint32`. Please note that in addition to a round towards zero (by shifting 32 bits to the right), also other rounding modes can be realized by considering the highest bit to be

Table 1: Minimum, average (with standard deviation) and maximum cycle lengths found over 63 values of a , for 32 bit fixed-point representations with round to nearest and different multiplication orders.

fixed-point	min	avg	std. dev	max
$\cdot a$ last	2493	14755.3	7997.6	34204
$\cdot x$ last	210	31220.3	23147.2	98969
$\cdot (1 - x)$ last	6347	37542.7	21576.7	101665
all	210	27839.5	21151.0	101665

rounded away. Rounding up then is realized by incrementing after the right shift.

Multiplication of $x' = x(1 - x) \leq 0.25$ by a , which is in interval $[3; 4)$, is realized by representing $a = 3 + a'$ with $0 < a' < 1$, doing the multiplication $x' \cdot a'$, and successively adding x' and $2x'$, i. e. x' shifted by one bit.

Please note that this realization, which has been presented and used in [20], prefers a particular order of multiplications, i. e. first multiply x with $1 - x$, to avoid intermediate results larger than 1. However, also other orders are possible with marginal additional effort.

3.4 Experiments

As floating-point implementations have been investigated in the past [11, 13], we will first focus on fixed-point implementations in our experiments. For 63 equi-spaced values of a , ranging from $4 - 2^{-12} = 3.999755859$ to $4 - 2^{-17} = 3.999992371$, we explore the differences between the three different orders of multiplications with respect to the period length of the resulting generator. As there the state graph comprises multiple weakly connected components with differing periods, in each experiment we use 32 equi-spaced states in the range 2^{-6} to 0.5 to start from and report the maximum period. As we expect the largest connected component, which typically also comprises the largest (or second largest) cycle, to comprise about 75 % of all states [5], the chance to miss it with 32 starting points is only 2^{-64} which we can tolerate. While the start states are not chosen randomly, they are chosen independently of the particular a , i. e. their choice should be as good as random. The results are depicted in Table 1.

Please note that differences between the rounding modes have been investigated in [10], and that in all except one scenario, round to nearest lead to longest periods. Hence we will use this rounding mode in the rest of this article.

The order where $x(1 - x)$ is computed first leads to shortest cycles on average. Please note that standard deviation is very high, so the average is not too meaningful. However, due to the chaotic nature of the logistic map, this

Table 2: Average cycle lengths when length of fixed-point representation is shortened by some bits.

reduced bits	avg. cycle length
0	37542.7
1	24319.3
2	16003.9
3	9670.9
4	7107.6
5	6560.0
6	4016.9
7	3310.9
8	2446.8
9	1746.3
10	1069.5

does not improve with more trials, i. e. different values of a in the same range, over which the average is taken. The same situation is encountered for floating-point formats, cf. Section 5. For all variants, the spread of cycle lengths among the different values of a is large: from 210 to 101,665.

To compare with floating-point implementations, we used an implementation based on IEEE754-compliant floats with round to nearest. For them (cf. Section 5 and Table 3), computation of $x(1-x)$ first leads to longest cycles, which however range from 91 to 7,802, with an average of 3,172, i. e. are much shorter than cycles from generators with fixed-point implementations, which have the same state size of 32 bits.

This comparison also serves to illustrate a tradeoff that is possible. We might shorten the number of bits used in the fixed-point representation. Table 2 illustrates how average cycle lengths (averaged over 63 values of a as before, with round to nearest an $1-x$ multiplied last) are reduced by using shorter number representations. We see that after shortening by 8 bits, i. e. only using 24 bit fixed-point representation, the cycles are about as short as for single precision, i. e. 32 bit, floating-point representation. Thus, if state space is very tight, e. g. in very small devices such as RFID devices, this opens a chance to reduce hardware footprint, i. e. number of gates necessary, and reduce energy consumption for multiplications, targeted towards security needs and energy constraints – similar to other approaches of approximate computing.

4 Comparing different number formats

Fixed-point and floating-point representations are just the opposite ends of a whole spectrum of possible number for-

ats. In the following, we describe a generic scheme to specify different number formats. To do so, we partition the interval $[0;1)$ into a sequence of d intervals I_0 to I_{d-1} , where $I_0 = [0;2^{e_0})$, $I_i = [2^{e_{i-1}};2^{e_i})$ for $i \geq 1$, and the e_i are a strictly increasing sequence of integers, with $e_{d-1} = 0$ to enforce that interval I_{d-1} ends with 1. In each interval, we have (up to) 2^m representations that are equi-spaced within the interval by representing multiples of 2^{e_i-m} , i. e. the represented number is

$$r(i,M) = \langle 0.M \rangle \cdot 2^{e_i} = \langle M \rangle \cdot 2^{e_i-m}. \quad (2)$$

Thus, to store a number in this format, we need $e = \lceil \log_2 d \rceil$ bits to represent the interval index, and m bits to represent the mantissa (or $m-1$ in case of binary floating-point representations, see below).

We start with fixed-point representations. They only comprise one interval I_0 , i. e. $d = 1$, and thus the number of exponent bits is zero. The mantissa bits represent a multiple of 2^{-m} , i. e. the represented number is $r(0,M) = \langle 0.M \rangle = \langle M \rangle \cdot 2^{-m}$.

In single precision floating-point representations, $d = 127$, and interval I_0 contains the denormalized numbers. The exponents form a sequence $e_i = i - 126$, i. e. successive exponents only differ by 1. To ensure that $r(i,M) \geq 2^{e_{i-1}}$ for $i \geq 1$, i. e. that $r(i,M) \in I_i$, it follows from Eq. (2) that $\langle 0.M \rangle \geq 0.5$, i. e. that the uppermost bit of mantissa M must be set (so-called implicit 1). Thus, only $m-1$ bits must be stored, or put otherwise, if we store m bits, we really work with a $m+1$ -bit mantissa.

Another example of a possible format is a separation in $d = 4$ intervals, i. e. using $e = 2$ exponent bits, with $(e_i)_i = (-32, -16, -10, 0)$. Thus, no particular regularity is necessary beyond the restrictions above. The mantissa is shorter than for fixed-point representations, but longer than for floating-point representations, so a possible advantage in cycle length might arise.

If all differences $e_i - e_{i-1}$ are equal, then we can consider the representation to be designed with respect to higher radix $2^{e_i - e_{i-1}}$, if they are not equal, they can be considered multi-radix representations.

Hence, for given total storage size of $t = e + m$ bits, there are different numbers $d = 2^e$ of possible exponents, leading to different mantissa lengths m , and for each d the exponents can have a number of possible distances leading to different number formats, which will in turn lead to different cycle lengths in the logistic map application. In this configuration space, we can search for the representation which is best with respect to a particular application.

Alternatively, one can fix the required performance, i. e. cycle length in our application, and search for the representation with shortest storage size that achieves that

performance, taking up the view of approximate computing [19].

Moreover, from all combinations of performance and storage size, the user can draw a plot and identify the Pareto front, i. e. the curve of Pareto-optimal combinations, which is the (typically small) number of combinations that the user can choose from depending on her needs [3].

In general, if $e_i - e_{i-1} \geq m$, then all representations of Eq. (2) (if not all mantissa bits are 0) are in I_i . Otherwise, the uppermost $e_i - e_{i-1}$ mantissa bits may not all be 0 simultaneously. As there is no similar trick as the implicit 1 if $e_i - e_{i-1} \geq 2$, a fraction of $2^{e_i - e_{i-1}}$ of the representations does not belong to I_i but to I_j with $j < i$, i. e. there is a certain redundancy, i. e. reduction in the total number of distinct representations.

To explore the spectrum of possible number formats, we have implemented a custom library where d , M and the sequence of e_i can be specified for a representation type.¹

We investigate total lengths t from 32 down to 24, with mantissa lengths ranging from t (resulting in $e = 0$) to 24 (resulting in $e = t - 24$). For each combination of e and m we explore sequences of e_i with regular distances $e_i - e_{i-1}$ from the set $\{1, 2, 4, 6, 8, 10\}$. So far, only round-to-zero is supported and multiplication with $a = 3 + \varepsilon$ is performed last, by first adding $x(1 - x)$ to itself, by computing $\varepsilon x(1 - x)$ and adding $x(1 - x)$, and finally adding the two intermediate results.

For each number representation, we choose again 63 different values of a as before, and compute the minimum, maximum and average cycle lengths (plus standard deviation).

Like in previous experiments, we see that standard deviation is quite large (ranging 49 % to 69 % of the average) and does not get better if we take larger number of samples. For almost all (31 of 33) combinations of e and m , we find that $e_i - e_{i-1} = 1$ leads to largest average cycle lengths. In the two exceptions, $e_i - e_{i-1} = 2$ performs better: in one case ($e = 1$, $m = 29$) the difference is only 0.32 %, yet in the other case ($e = 1$, m_30) the difference is 30.9 %, i. e. notably, mainly driven by a really large maximum cycle length achieved. Thus that case can be considered an outlier.

In the following, we further investigate the e, m combinations with distance leading to maximum average cycle length. Figure 1 depicts that the average cycle length is mainly determined by the number of mantissa bits, for each m it is largely constant over the different exponent

Figure 1: Dependence of average cycle length on exponent length e for different mantissa lengths m .

Figure 2: Dependence of average cycle length on mantissa length m for different exponent lengths e .

lengths e , and the curves are ordered by growing m . Please note that the cycle lengths for $e = 0$ are taken from Table 2 where different rounding mode and multiplication order are applied, so that the step from $e = 0$ to $e = 1$ is not completely fair, which might explain the unusual shape for $m = 31$.

Figure 2 depicts that for a fixed exponent length e , the average cycle length grows with total length $t = e + m$, i. e. with mantissa length m , and the curves are ordered by shrinking exponent length e . The only exception is a crossing of the curves for $e = 1$ and $e = 2$ at the right end of the curves. Figure 3 completes this view by plotting curves for each total length $t = e + m$. We see that each curve decreases with growing e , i. e. shrinking m , and the curves are ordered by t .

In summary, we find that for our application, distances $e_i - e_{i-1} = 1$ are best, as in binary floating-point representations, and that fixed-point representation leads indeed to longest average cycle lengths, even when we consider the large space of representations between (binary) floating-point and fixed-point representations.

¹ Available from authors upon request.

Figure 3: Dependence of average cycle length on exponent length e for different total lengths $t = e + m$.

Still, our view unifies several proposals for number representations from literature, cf. Section 2.3, and our implementation allows to find the optimal representation, e. g. for a reconfigurable hardware device with fixed application, with low effort.

5 Extending number formats for symmetry

We note that the logistic map as defined in Eq. (1) is symmetric with middle 0.5, i. e. for $z \leq 0.5$ we have $f(0.5 - z) = f(0.5 + z)$ because for $x = 0.5 - z$ we get $1 - x = 0.5 + z$ (and vice versa) and from Eq. (1) we get $f(x) = f(1 - x)$. Hence we can extend previous number formats to respect this symmetry. If z is any of the previous representations, only restricted to $[0; 0.5]$ instead of $[0; 1]$, then (s, z) with $x = 0.5 + (-1)^s \cdot z$ is a representation for $[0; 1]$ where representation densities are symmetric around 0.5, and $1 - x$ can be represented by $(1 - s, z)$, i. e. by inverting the “sign” bit s . The product $x \cdot (1 - x)$ can be expressed as $0.25 - z^2$ independent of s , and

$$\begin{aligned} f(x) &= a \cdot x \cdot (1 - x) \\ &= a \cdot (0.25 - z^2) \\ &= a/4 - a \cdot z^2 \\ &= 0.5 + \underbrace{(a/4 - a \cdot z^2 - 0.5)}_{z'} \end{aligned}$$

If $\tilde{z} = a/4 + a \cdot z^2 \geq 0.5$, then z' is non-negative, and $f(x) \geq 0.5$ can be represented by $(0, z')$. Otherwise, $-z'$ is non-negative and $f(x) < 0.5$ can be represented by $(1, -z')$. Thus, one first computes $\tilde{z}$, and depending on whether $\tilde{z} \geq 0.5$ or not, one computes $\tilde{z} - 0.5$ or $0.5 - \tilde{z}$ to determine z' or $-z'$, respectively. If (s, z) can be represented together

Table 3: Minimum, average (with standard deviation), and maximum cycle length found over 32 values of a , for ordinary and symmetric single precision floating-point numbers and different computation orders.

ordinary	min	avg	std. dev	max
$\cdot a$ last	91	3172.1	2053.1	7802
$\cdot x$ last	141	2089.2	1330.4	5300
$\cdot (1 - x)$ last	49	2281.9	1630.9	6308
all	49	2514.4	1761.8	7802
symmetric	min	avg	std. dev	max
$\cdot a$ last, $-az^2$ last	240	2459.5	1610.0	6803
$\cdot z$ last, $-az^2$ last	1127	2596.8	1710.6	10277
$\cdot a$ last, -0.5 last	306	1734.8	915.3	4316
$\cdot z$ last, -0.5 last	25	1945.4	1316.6	5630
all	25	2184.1	1465.8	10277

as in floating-point format, then one can simply compute $z' \in [-0.5; +0.5]$. Please note that in this case almost twice as many values of IEEE754 floats can be used: on the one hand one value less from the exponent range is used (but still close to half of the values), but on the other hand the sign bit is used.

To see if “symmetric” floating-point representations are advantageous, we did experiments similar to those in Section 3.4: for each a tested, we find the largest cycle reachable from 32 starting points, and we compute minimum, maximum, and average cycle length over 32 values of a starting from $4 - 2^{-12}$ and equi-spaced with distance 2^{-17} . We only use rounding to nearest, but compare different orders of computation. For ordinary single precision (32 bit) IEEE754 floating-point numbers we distinguish multiplication with a last, or x last, or $1 - x$ last. For symmetric single precision floating-point numbers we distinguish multiplication with a last, i. e. computation of z^2 first, or multiplication with z last, i. e. computation of $a \cdot z$ first, and we distinguish subtraction of 0.5 last, or subtraction of $a \cdot z^2$ last. Moreover, we compute minimum, maximum and average over all possibilities.

The results can be seen in Table 3. There is no clear winner. With respect to average cycle length found, ordinary floating-point representation with multiplication order “ a last” is best, but the best variant of symmetric floating-point representations is much better in minimum (and maximum) cycle length found. Moreover, the standard deviations are very large compared to the averages, so that the averages are not very meaningful, and minimum and maximum lengths might provide a better insight. Please note that increasing the number of trials over which averages are computed does not help here, due to the chaotic nature of the logistic map for different values of a close to 4. For 1024 equi-spaced values of a starting from

$4 - 2^{-12}$, the standard deviation does not shrink substantially, e. g. from 1610 to 1523 for symmetric float representations and multiplication with a last. Yet, in some variants standard deviation even increases, e. g. from 1630 to 1688 for conventional floats and multiplication with $1 - x$ last. The biggest difference is that the average cycle length for ordinary floats where multiplication with a is done last reduces to 2459.7, so that the range of averages is much closer together.

6 Conclusions

We have presented challenges that occur with respect to replicability in security or cryptographic applications that involve numeric parts originally devised for real numbers. As example, we have chosen the logistic map, which has been one proposal for a chaotic pseudo-random number generator. We have demonstrated by experiments the relevance of considering the differences that possibly result from not arbitrating the ambiguities underlying the above challenges.

As similar problems might occur in other security applications as well, we derive the following recommendations. First, the rounding mode used in the original experiments should be found out by the experimenters and should be reported. Second, the order of computations used in the original experiments should be reported as well, and possibly should be modified by the first experimenters to check if notable differences in result can be the consequence. Finally, different number representations should be considered by experimenters, and results on their influence and reasons for the final choice of representation should be reported.

Moreover, we have investigated higher and mixed radix representations in between fixed-point and binary floating-point representations, plus symmetric representations, and demonstrated how their tradeoff between storage size and performance (in the sense of period length in the application) can be used by users to tailor their application to their needs, thus connecting this investigation to the field of approximate computing.

Future work will comprise a deeper analysis of our experimental results. Also, the scope of numerical challenges will be extended from chaotic random numbers to other branches of IT security. Moreover, the introduction of redundancy by higher radix representations, e. g. by skipping normalization [2], opens the path for information hiding, e. g. storage covert channels [18], in such representations, which we plan to investigate further.

Funding: This work is supported by project SIMARGL (Secure intelligent methods for advanced recognition of malware, stegomalware & information hiding methods, <https://simargl.eu>), which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 833042.

References

1. Marcel Ausloos and Michel Dirickx, editors. *The Logistic Map and the Route to Chaos*. Springer, Berlin Heidelberg, 2006.
2. Bryan Catanzaro and Brent E. Nelson. Higher radix floating-point representations for FPGA-based arithmetic. In *13th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2005), Proceedings*, Napa, CA, USA, 17–20 April 2005, pages 161–170. IEEE Computer Society, 2005.
3. Matthias Ehrgott. *Multicriteria Optimization*, volume 491 of *Lecture Notes in Economic and Mathematical Systems*. Springer, 2000.
4. Chun Te Ewe. Dual fixed-point: An efficient alternative to floating-point computation for DSP applications. In *Proceedings of the 2005 International Conference on Field Programmable Logic and Applications (FPL)*, Tampere, Finland, August 24–26, 2005, pages 715–716. IEEE, 2005.
5. Philippe Flajolet and Andrew M. Odlyzko. Random mapping statistics. In *Advances in Cryptology – EUROCRYPT '89*, volume 434 of *Lecture Notes in Computer Science*, pages 329–354. Springer, 1989.
6. Altaf Abdul Gaffar, Oskar Mencer, Wayne Luk, Peter Y. K. Cheung, and Nabeel Shirazi. Floating-point bitwidth analysis via automatic differentiation. In *Proceedings of the 2002 IEEE International Conference on Field-Programmable Technology, FPT 2002*, Hong Kong, China, December 16–18, 2002, pages 158–165. IEEE, 2002.
7. Oded Goldreich. *Foundations of Cryptography I: Basic Tools*. Cambridge University Press, Cambridge, UK, 2001.
8. IEEE standard for floating-point arithmetic, 2019. IEEE Standard 754-2019 (Revision of IEEE 754-2008).
9. IEEE standard for radix-independent floating-point arithmetic, 1987. IEEE Standard 854-1987.
10. Jörg Keller. Chaotic pseudo random number generators: A case study on replication study challenges. In *ARES 2021: The 16th International Conference on Availability, Reliability and Security*, Vienna, Austria, August 17–20, 2021, pages 94:1–94:4. ACM, 2021.
11. Jörg Keller and Hanno Wiese. Period lengths of chaotic pseudo-random number generators. In *Proc. IASTED International Conference on Communication, Network, and Information Security (CNIS 2007)*, Calgary, Canada. Acta Press, 2007.
12. Wolfgang Killmann and Werner Schindler. A proposal for: Functionality classes for random number generators. Report ais 20/31 v2.0, Federal Office for Information Security/Bundesamt für Sicherheit in der Informationstechnik (BSI), Bonn, Germany, 2011.

13. Ronald T. Kneusel. *Random Numbers and Computers*. Springer, Cham (CH), 2018.
14. Israel Koren. *Computer Arithmetic Algorithms*. Taylor & Francis, Milton Park, UK, 2nd edition, 2001.
15. Ping Li, Zhong Li, Wolfgang A. Halang, and Guanrong Chen. A stream cipher based on a spatiotemporal chaotic system. *Chaos, Solitons & Fractals*, 32(5):1867–1876, 2007.
16. Gerald Luber. Statistical analysis of outputs from chaotic pseudo random number generators with fixpoint representation (in German: Statistische Analyse der Ausgaben von chaotischen Pseudozufallszahlengeneratoren mit Festkommadarstellung). Master thesis, FernUniversität in Hagen, Germany, 2014.
17. George Marsaglia. The marsaglia random number CDROM including the Diehard battery of tests of randomness. Technical report, Florida State University, Tallahassee, FL, 1995.
18. Wojciech Mazurczyk, Steffen Wendzel, Sebastian Zander, Amir Houmansadr, and Krzysztof Szczypiorski. *Information Hiding in Communication Networks. Fundamentals, Mechanisms, and Applications*. IEEE Series on Information and Communication Networks Security. Wiley-IEEE, 2016.
19. Sparsh Mittal. A survey of techniques for approximate computing. *ACM Comput. Surv.*, 48(4):62:1–62:33, 2016.
20. Lena Oden and Jörg Keller. Improving cryptanalytic applications with stochastic runtimes on GPUs. In *Proc. 11th International Workshop on Accelerators and Hybrid Emerging Systems (AsHES@IPDPS 2021)*, New York, NY, May 2021. IEEE, 2021.
21. Dhananjay S. Phatak and Israel Koren. Hybrid signed-digit number systems: A unified framework for redundant number representations with bounded carry propagation chains. *IEEE Trans. Computers*, 43(8):880–891, 1994.

Bionotes

M. Sc. Carina Heßeling

FernUniversität in Hagen, Faculty of Mathematics and Computer Science, 58084 Hagen, Germany
carina.hesseling@fernuni-hagen.de

M. Sc. Carina Heßeling studied Computer Science at FernUniversität in Hagen, Germany (Bachelor 2017, Master 2021). Since August 2021, she is a PhD student and Research Assistant at FernUniversität in Hagen, Germany. Her research interests are approximate computing and network steganography.

Prof. Dr. Jörg Keller

FernUniversität in Hagen, Faculty of Mathematics and Computer Science, 58084 Hagen, Germany
joerg.keller@fernuni-hagen.de

Prof. Dr. Jörg Keller studied Computer Science and Mathematics at Saarland University, Saarbrücken, Germany (Master 1989, PhD 1992, Habilitation 1996). Since 1996, he is Professor for Computer Engineering at FernUniversität in Hagen, Germany. His research interests are network steganography, energy-efficient and fault-tolerant parallel computing, and virtual laboratories.