EURASIP Journal on Embedded Systems
a SpringerOpen Journal

RESEARCH Open Access

Pseudorandom recursions II

Laszlo Hars[1*] and Gyorgy Petruska[2]

Abstract

We present our earlier results (not included in Hars and Petruska due to space and time limitations), as well as some updated versions of those, and a few more recent pseudorandom number generator designs. These tell a systems designer which computer word lengths are suitable for certain high-quality pseudorandom number generators, and which constructions of a large family of designs provide long cycles, the most important property of such generators. The employed mathematical tools could help assessing the bit-mixing and mapping properties of a large class of iterated functions, performing only non-multiplicative computer operations: SHIFT, ROTATE, ADD, and XOR.

Keywords: pseudorandom number generator, recursive function, invertible functions, matrix, binary modular polynomial, extended GCD algorithm

1. Introduction

Security applications, simulations, randomized algorithms, gambling, etc. need good quality random numbers. They can often be substituted with pseudorandom numbers, which are generated by software and behave like true random numbers in many statistics. When these pseudorandom numbers are generated in embedded microprocessors, speed and memory requirements pose constraints, limiting the choice of algorithms.

The quality of the generated sequences is crucial. Randomness tests can verify desired statistical properties for the targeted applications. One of the desired properties of such sequences is the length of the unavoidable cycles. The main point of our investigations is the invertibility of the generator function of such pseudorandom sequences, which can ensure very long cycles in certain operation modes.

Many more characterizations of the generated sequences are possible, like the distribution of blocks of bits. Our corresponding results in this regard have to be deferred to a future publication. This article represents the first step in the investigations of random properties of the sequences generated by a large class of iterated functions, performing only non-multiplicative computer operations: SHIFT, ROTATE, ADD, and XOR.

1.1 Prior work

In our original study [1], we presented many small and fast pseudorandom number generators, which pass the most common randomness tests. They repeatedly call simple bit-mixing functions that perform only a few non-multiplicative operations for each generated number, and require very little memory. Therefore, they are ideal for embedded- or time-critical applications. In [1], we also presented general methods to ensure very long cycles in repeated calls of the mixing functions, and showed how to use these algorithms as cryptographic building blocks.

In 2005 (unpublished submission to the CHES'06 workshop), we proved that a necessary condition for the invertibility of a rotate-XOR chain is that the *number of rotations is odd*. This result later appeared in [1]. In this article, we presented our previously unpublished results of 2005/2006, together with some newer results and useful tools, which would help resolving the invertibility in concrete general cases.

A similar class of functions turned out to be very useful in cryptography and pseudorandom number generation, the T-functions. They have been extensively studied [2-11]. A T-function is a mapping from n-bit input to n-bit output in which each bit i of the output depends only on bits 0,1,..., i of the input. All the logical operations, such as XOR, AND, OR, NOT, and most of the arithmetic operations modulo 2^n, such as addition, multiplication, subtraction, negation, as well as left shift

* Correspondence: lhars@cputech.com
[1]CPU Technology, Pleasanton, CA 94588, USA
Full list of author information is available at the end of the article

and their compositions, are T-functions. However, rotations and right shift operations are not.

1.2 This work

The most important property of the considered bit-mixing functions is long period length, related to the invertibility of their generating function. For invertible functions, a counter can be included in the input, assuring that no output value repeats before the counter wraps around. Even when the output is truncated or its bits are mixed together, there will still be no short cycle. A large part of this study below deals with this invertibility, which is present in many pseudorandom number generator modes we have proposed.

In the era of synthesizable processor cores unusual word lengths are easy to implement. Our results tell a systems designer which ones allow efficient pseudorandom number generators, and which constructions could work. It can save design and experimentation work. The employed mathematical tools are easy to use and powerful, and they can aid investigating large classes of iterated functions.

This article comprises three major sections. In Section 2, we describe and analyze several recent random number generator designs, and include some characteristic code segments. In Sections 3 and 4, we discuss the existence of inverses of rotate-add functions and rotate-XOR functions, respectively. Our experience shows that rotate-add methods are usually inferior to rotate-XOR methods.

2. New random number generator modes

Recall our notation in [1]: Counter mode (of pseudorandom number generators) is defined as $x_i = f(i)$, where the counter i is incremented before each call of the function f. Hybrid counter mode uses a function of several variables, one of them is a similar counter as above: $x_i = f(i, x_{i-1}, x_{i-2},..., x_{i-k})$. Multi-stage generators are based on this kind of iterations, but several calls are performed to such type of functions for one set of output values.

The apparent pseudo-randomness of the counter mode and hybrid counter mode can be improved by incrementing the counter by a large odd constant c (instead of 1), because many more bits change at such addition than at incrementing by 1, most of the time. Although a (loop) counter i is sometimes available for free, and this number c needs extra storage, we found that the pseudo-randomness improves significantly, and so ultimately computation can be saved. We call these new modes offset counter mode and offset hybrid counter mode.

Note that the function f could compute the modified counter k from a regular one i, as $k = i \cdot c \bmod 2^{32}$ (in case of 32-bit machine words), but we excluded multiplication from the admissible operations (because they need large hardware cores and multiple clock cycles at high clock frequencies).

2.1 MIX permutations

It is an intriguing idea to design some small additional hardware to embedded processors for rearranging the bits of a register. With the help of a few extra gates (or just wires) the performance of our pseudorandom number generator might be improved.

A MIX operation has to be a permutation of bits, not to reduce the range of the outputs. At repeated application of the MIX permutation a bit gets back to an already occupied position after at most 32 steps. Odd rotations are maximal permutations in every bit position (when the machine word is 2^w bits). This is advantageous for random number generation, where we must not have short cycles.

Bit or byte reversals are sometimes available as CPU operations, but they are not very good mixers, as they define permutations with short cycles. Similarly, bit-swap, byte swap, or a rotation followed by swapping neighbor bits all proved to be less effective mixers, than simple rotations. This explains why our best constructions are based on rotations, not on complicated MIX permutations.

2.2 MIX-XOR circuits

As compared to our earlier designs, a little more complex bit mixing hardware still proved to be advantageous. It could be implemented with very few gates and wires. For example, in such operations each output bit can be the XOR of two (or more) different input bits. An example is the offset hybrid counter mode generator, which passes all Diehard tests:

$$x = \mathrm{rot}(x, 5) \,\hat{}\, \mathrm{rot}(x, 24) \,\hat{}\, (k \mathrel{+}= 0x37798849)$$

where (x, k) represent the state of the random number generator, updated during each invocation of the mixing function. The output, the generated random number, is x.

In hardware, the rotations need not actually be performed, only the corresponding bits of the machine word x are XOR-ed, so one iteration can provide 32 bits output in 2 clock cycles.

2.3 Statistical randomness tests

We wrote simple C programs for creating 10 MB binary data files for every variant of our pseudorandom number generators and applied statistical tests to them, to assess their quality. Many randomness tests have been published, for example [12-14]. In [15], there is a survey. A recent test suite for testing randomness of sequences for cryptographic applications is the NIST 800-22 Randomness tests

[14], provided as C-99 source code. Unfortunately, it contains errors (acknowledged by its publisher), which were not fixed at the time of this writing.

We found the classic Diehard test suite the most stable and reliable. It was published by Marsaglia [12] and performs 15 different groups of statistical randomness tests. Many different properties are tested and the protocol of the results is 17 pages long. The randomness measures are 250 p-values. We employed the standard way for accepting a single p-value: checked if it was in a certain interval, like [0.001, 0.999].

2.4 Offset hybrid counter mode

We assume 32-bit machine words. The smallest case is of stage-2: These random number generators have two parameters (which can be treated as two internal state variables), one is recursively updated by a mixing function, while the other one (an offset counter) is incremented by a large, odd constant before each call.

Surprisingly, for satisfying the Diehard randomness tests, loading an operand with its bits rotated by a fixed amount proved to be sufficiently random.

$$x = \mathrm{rot}(x, 9) \,\hat{}\, (k \mathrel{+}= 0x37798849).$$

This generator passes all Diehard tests, with one near fail of p-value = 0.9995. Rotation by 7 works, too, with one p-value = 0.9998.

Rotation to the right works even better (because the carry propagation is better utilized):

$$x = \mathrm{rot}(x, 23) \,\hat{}\, (k \mathrel{+}= 0x49A8D5B3).$$

(Rotate by 23 to the left is the same as rotate by 9 to the right.) This generator passes all Diehard tests, with no p-value > 0.999. Rotation by 25 bits (or 7 bits to the right) is equally good.

Because these generators are already good enough with the minimum number of operations, there is no need for considering more stages (stored in more internal variables).

2.5 Offset counter mode

Offset counter mode is the one-stage version of the above-discussed offset hybrid counter mode, that is, there is no state variable, except the counter k to be incremented by a large odd constant before each call. It can be supplied as input, and the output is computed directly from k. This mode can be used as a component for data scrambling, hashing, and encryption. Note that invertible functions are needed to map the input to the full range of machine words as output.

We use the notation ROL/ROR(x, k) for rotation of the unsigned integer x to the left/right, respectively, by k positions.

32-bit words

The generators are defined as

(L,R) = (4,9), no Diehard test fails, or nearly fails (rotate left).

x = (k += 0x37798849).

x = (x ^ ROL(x,L) ^ ROL(x,R)) + 0x49A8D5B3.

x = (x ^ ROL(x,L) ^ ROL(x,R)) + 0x6969F969.

x = (x ^ ROL(x,L) ^ ROL(x,R));

(L,R) = (4,9), no Diehard test fails, or nearly fails (rotate right).

x = (k += 0x37798849).

x = (x ^ ROR(x,L) ^ ROR(x,R)) + 0x49A8D5B3.

x = (x ^ ROR(x,L) ^ ROR(x,R)) + 0x6969F969.

x = (x ^ ROR(x,L) ^ ROR(x,R)).

Here x is the output, used also for storing intermediate values. Its value is not retained between calls. These generators work even with structured constants for both adders (e.g., 0x55555555, with only one near fail), so we can safely replace these constants with parameters, to diversify the generators.

64-bit Words

One could think that 64-bits need more iterations to get full distribution of bits, but the process above proved to have enough reserve that it still works adapted for long machine words.

The direct dispersion of any bit is to 3 × 3 × 3 = 27 positions. With the two long added constants, most of the time, the carry makes the majority of the 64 bits changed when a single bit is flipped in the counter. Of course, when two initial values are close (e.g., k = 0 and 1, or generally at small counter increments), this 1-to-27 dispersion effect is not sufficient. That is why we increment k with a large odd value, ensuring that many input bits change between consecutive calls.

If these increment values (considered as 64-bit keys), have no blocks of 20 identical bits, the scheme was found to work well, so we are reasonable safe against accidental weak keys. Nevertheless, these keys should be tested for blocks of more than 12 zeros or 12 ones, and reject such numbers.

The generators are defined as

(L,R) = (4,9), no fail, no near fail in Diehard (rotate left).

x = (k += 0x3779884922721DEB).

x = (x ^ ROL(x,L) ^ ROL(x,R)) + 0x49A8D5B369 69F969.

x = (x ^ ROL(x,L) ^ ROL(x,R)) + 0x6969F96949A 8D5B3.

x = (x ^ ROL(x,L) ^ ROL(x,R)).

(L,R) = (4,9), no fail, no near fail in Diehard (rotate right).

x = (k += 0x3779884922721DEB).

$x = (x \wedge \mathrm{ROR}(x,L) \wedge \mathrm{ROR}(x,R)) + 0x49A8D5B36969F969$.

$x = (x \wedge \mathrm{ROR}(x,L) \wedge \mathrm{ROR}(x,R)) + 0x6969F96949A8D5B3$.

$x = (x \wedge \mathrm{ROR}(x,L) \wedge \mathrm{ROR}(x,R))$.

If we set both other additive constants in the rotate-left version to the structured $0x3333333333333333$ or $0x7777777777777777$, only one Diehard test fails. With $0x7E7E7E7E7E7E7E7E$ all tests pass (with one near fail). Experiments with many similar values show that we have a safety margin for weak constants, therefore these numbers can serve as further 64-bits keys.

2.6 Data expansion

For ciphers, e.g., of unbalanced Feistel networks [16,17], we often need to scramble and to expand short, e.g., 32-bit numbers to long values. We can do this really fast in hardware: perform several of these offset counter mode mix operations with different additive constants, in parallel, maybe with varying rotate directions and distances. To get the expanded data just concatenate the results.

If the additive constants are treated as secret keys, or they are derived from a secret key, we get a primitive cipher. With sufficiently many iterations of varying constants, it could be secure.

3. Invertibility of rotate-add functions

Since we observed thorough mixing properties in the offset counter mode generators, we could be tempted to simplify the generator function by tweaking their code lines. In [1], we showed that XOR-ing two (instead of three) rotated entries breaks the invertibility of the function, so we tried this idea with addition instead of XOR: $x \leftarrow x + \mathrm{ROT}(x,k)$. (It represents a reduction from two rotates and two XOR operations to one rotate and one addition.) The Diehard tests still pass with a rotation by 7 or 11, in either a left- or the right-rotating variant.

The rationale of investigating this function is that adding to the input its rotated version causes larger changes in the output than a rotate-XOR operation had: a flipped bit in the input influences at least two output bits, but usually much more, dependent on the carry propagation. In this sense the *dispersion* of input changes is larger than at the rotate-XOR type functions, so better mixing properties are expected.

Unfortunately, most of the time this simplified function cannot be inverted, that is, we cannot solve the equation $y = x + \mathrm{ROL}(x, k) = \mathrm{floor}(x \cdot (1 + 2^k + 2^{k-32})) \bmod 2^{32}$ for x (assuming 32-bit machine words). For many y values, there is no solution, or there are more than one possible x values. Therefore, we should better *avoid these functions in counter mode, in hybrid counter mode of random number generators, or in ciphers.*

The problems are easily seen by computing $x + \mathrm{ROL}(x, k)$ for all values of x, and sorting the results. For example, for $w = 16$-bit machine words, and rotation by $k = 3$, the sequence of the sorted y values starts as:

0, 2,2,2, 5,5,5, 8,8, 9, 11,11,11, 14,14,14, 17,17, 18, 20,20,20, 23...

To the best of the authors' knowledge, the corresponding general problem has not been addressed in the literature.

Claim: *the Rotate-Add functions defined below do not attain all y values*, with any fixed k: $0 < k < w$, when x goes over all possible values in $[0, 2^w - 1]$.

$$y(x) = x + \mathrm{ROL}(x,k) = \mathrm{floor}(x \cdot (1 + 2^k + 2^{k-w})) \bmod 2^w.$$

In the rest of this section, we are going to validate this claim. We will use a little more convenient way to write $y(x)$, by first partitioning x into its least significant k bits (v), and the remaining bits (u), such that $x = 2^{w-k} \cdot u + v$, (with $0 \le u < 2^{w-k}$ and $0 \le u < 2^k$). Our rotate-add function now expressed as

$$y(x) = (2^k + 1)u + (2^{w-k} + 1)v \bmod 2^w.$$

3.1 Word lengths of 2^w

3.1.1. Common factors

For the ordinary sizes of machine words, the coefficients of u and v are not relative primes. Below we list their common factors as k ranges through 0 to w:

$w = 16$:

1, 3, 5, 3, 17, 3, 5, 3, 257, 3, 5, 3, 17, 3, 5, 3, 1

$w = 32$:

1, 3, 5, 3, 17, 3, 5, 3, 257, 3, 5, 3, 17, 3, 5, 3, 65537, 3, 5, 3, 17, 3, 5, 3, 257, 3, 5, 3, 17, 3, 5, 3, 1

$w = 64$:

1, 3, 5, 3, 17, 3, 5, 3, 257, 3, 5, 3, 17, 3, 5, 3, 65537, 3, 5, 3, 17, 3, 5, 3, 257, 3, 5, 3, 17, 3, 5, 3, 4294967297, 3, 5, 3, 17, 3, 5, 3, 257, 3, 5, 3, 17, 3, 5, 3, 65537, 3, 5, 3, 17, 3, 5, 3, 257, 3, 5, 3, 17, 3, 5, 3, 1

As we can see, there is no proper rotation of $0 < k < w$ distance, which does not suffer from common factors. It is a remarkable experience, that all the common factors are Fermat numbers, that is integers of form $F_n = 2^{2^n} + 1$. There are deep and age old open problems concerning Fermat numbers. Computational evidence supports the following conjecture, which is important, because the length of machine words in all practical cases is a power of 2 (8, 16, 32, 64...).

Conjecture 1: If w is a power of two and $0 < k < w$, then $\mathrm{GCD}(2^k + 1, 2^{w-k} + 1)$ is a Fermat number $2^{2n} + 1$.

Notes:

• The first few Fermat numbers are $F_0 = 3$, $F_1 = 5$, $F_2 = 17$, $F_3 = 257$, $F_4 = 65537$, $F_5 = 4294967297$, $F_6 = 18446744073709551617$,

F_7 = 340282366920938463463374607431768211457.

• Only the first five Fermat numbers F_0, F_1, F_2, F_3, F_4 are known to be prime. The next three we listed are products of two primes:

4294967297 = 641 × 6700417

18446744073709551617 = 274177 × 67280421310721

340282366920938463463374607431768211457 = 59649589127497217 × 5704689200685129054721

• Conjecture 1 has been numerically verified for $w = 2^2, 2^3..., 2^{20}$. Up to $w = 2^{18}$ just minutes of PC computing time was used, $w = 2^{19}$ took 3 h, and verifying the conjecture for $w = 2^{20}$ needed 22 h at light CPU load. The cases w = 16, 32, 64 are demonstrated by the tables presented above.

Though Conjecture 1 eludes a rigorous proof, we can prove a somewhat weaker statement, sufficient for our investigations: the GCD in question is at least divisible by a Fermat number:

Theorem 3.1: If W is a power of two, then GCD(2^K + 1, 2^{W-K} + 1) is divisible by a Fermat number for any K integer, $0 < K < W$.

Proof: We change the notation showing that the exponents are symmetrically positioned around $w = W/2$, again a power of 2 say, $w = 2^p$. Also, we denote $k = w - K$ and using the new notations we are to show that GCD (2^{w+k} + 1, 2^{w-k} + 1) is a multiple of a Fermat number. We put $k = 2^q \cdot r$, $0 \le q < p$ integer, and r is an odd number. Now we obtain

$w + k = 2^p + 2^q r = 2^q \cdot (2^{p-q} + r) = 2^q a$ for the first, and

$w - k = 2^p - 2^q r = 2^q \cdot (2^{p-q} - r) = 2^q b$ for the second exponent, where a and b are odd integers.

With the notation $u = 2^{2q}$ we have GCD(2^{w+k} + 1, 2^{w-k} + 1) = GCD(u^a + 1, u^b + 1). Since a and b are odd, u + 1 (that is Fermat number F_q) is a divisor of both numbers u^a + 1 and u^b + 1. □

Corollary 3.2: If Conjecture 1 holds true, then the Fermat number F_q we found in the above proof is the greatest common divisor in question.

Indeed, it is well known that the Fermat numbers are pair-wise relative primes, thus a Fermat number cannot be the divisor of another Fermat number. Note that for $q = 0$ we have $F_q = 3$, which explains the occurrence of 3 in every second position in the above tables of common divisors. □

3.1.2. Overflow

If the addition of x to ROL(x, k) does not cause overflow, we have $y(x) = (2^k+1)u + (2^{w-k} + 1) \cdot v$. For the investigated word lengths of 2^w, $y(x)$ is a multiple of one of the common factors granted by Theorem 3.1, and so $y(x)$ does not take all possible values.

The situation is not much more complicated when there is an overflow (which can only be 1):

$$y(x) = (2^k + 1)u + (2^{w-k} + 1)v - 2^w$$

In this case, dividing $y(x)$ by the above discussed common factor the remainder is determined by 2^w.

Note that when we divide by the Fermat number we found above as a common factor, the remainder is always 1. This is explained by the well known and fairly obvious product formula of Fermat numbers:

$$F_{p+1} - 2 = F_0 \cdots F_p.$$

3.1.3. Missing words

As we just saw, $y(x)$ is a multiple of a Fermat number 3, 5, 17, 257, 65537, 4294967297...; or 1 less than such a multiple, in all practical computing systems. Thus, numbers in at least one residue class modulo a Fermat number (at least a third of the possible output values [0, 2^w - 1]) never get generated.

3.2 Uncommon word lengths

There are machine word lengths, which do give relative prime coefficients of u and v, for certain rotation lengths. These machine words are almost never used in real-life computing systems, but in the age of synthesizable processor cores special hardware could easily be built for them, if they were advantageous. Unfortunately, as our negative results show below, they are not much better regarding invertibility than the more common word sizes. This knowledge can save a lot of futile work.

24-bit words

We can list the common factors of the coefficients of u and v, as k goes from 0 to w:

w = 24: 1, 3, 5, 9, 17, 3, 65, 3, **1**, 9, 5, 3, 4097, 3, 5, 9, **1**, 3, 65, 3, 17, 9, 5, 3, 1

Here, rotations by 8 and 16 could be good candidates for mixing functions, but when there is an overflow, many y values get repeated. With a simple PC program we counted the number of missing words: at rotation by 8 or 16 there are 4,210,688 missing words, which represents over 25% of all 24-bit words.

25-bit words

The odd word length 25 makes *each* pair of the coefficients of u and v relative prime, and still all rotation-add options leave out many words. The best cases are with rotations by 12 or 13 (0.024%: 8191 missing words), the worst cases are with rotations by 1 or 24 (one-third of the words: 11,184,811 are missing).

31-bit words

One can drop one bit of the most common 32 bit machine words. All the pairs of multipliers become relative primes, and still every rotation-add option leaves out many words. A PC program found the best cases at rotations by 15 or 16 (65,535 = 0.003% missing words),

and the worst cases at rotations by 1 or 30 (one third of the words: 715,827,883 are missing).

Note that the relatively few missing words at rotations by 15 or 16 do make this scheme useable for Feistel-style encryption, but other constructions (like rotate-XOR) are still better.

3.3 Arbitrary word sizes

We can show in general that *no* rotate-add function is invertible:

Theorem 3.3: At any word length w and rotation distance k the corresponding rotate-add function repeats at least one word (and so at least one output word is always missing).

Proof: (a) If there is a common factor $d > 1$ dividing both the coefficients of u and v in $(2^k + 1)\cdot u + (2^{w-k} + 1)\cdot v$, it is odd, therefore at least 3. Thus, $y(x) \equiv 0$ or $-2^w \bmod d$, hence numbers in the remaining (at least one) mod d residue classes are not generated.

(b) If $\mathrm{GCD}(2^k + 1, 2^{w-k} + 1) = 1$, the extended GCD algorithms find u' and v' integers (one of them negative), such that $(2^k + 1)\cdot u' + (2^{w-k} + 1)\cdot v' = 1$. Multiplying this equation with 2^w we obtain $(2^k + 1)\cdot(u'\cdot 2^w) + (2^{w-k} + 1)\cdot(v'\cdot 2^w) = 2^w$, thus the Diophantine equation $(2^k + 1)\cdot u + (2^{w-k} + 1)\cdot v = 2^w$ admits a solution.

Note that for any integer m, $u = u' + m\cdot(2^{w-k} + 1)$, and $v = v' - m\cdot(2^k + 1)$ represent another solution for the equation above. At a suitable m value there is a solution (u'',v''), such that $0 < u'' < 2^{w-k}$.

Because of the symmetry, we may assume that $k \le w - k$. Substituting the minimum and maximum u'' values into the equation $(2^k + 1)\cdot u + (2^{w-k} + 1)v = 2^w$ we find that $0 < v'' < 2^k$. These (u'',v'') values, therefore, can be concatenated to form a machine integer $x \neq 0$, of length w. Our mod 2^w rotate-add function transforms this x into 0. Because 0 is a fix point, we found two machine integers (x and 0), which are both transformed to 0. □

4. Invertibility of rotate-XOR functions

For many applications of random number generator constructions presented in Section 2 of this article (and of the ones in [1]) we needed the recursions to be invertible. In [1], we proved the following

Lemma: The determinant of M, the sum of k powers of unit circulant matrices is divisible by k.

Its corollary is that even number of rotations XOR-ed together does not define invertible recursions.

In the rest of the article, we investigate the invertibility problem in more details. Two (equivalent) models of the iterated functions are employed, namely, matrix and binary polynomial representations.

4.1 Elementary results

Let N denote the length of the machine word where we perform rotate-XOR computations. We denote by C the corresponding unit circulant matrix of size $N \times N$ (all entries are 0, except the 1s above the main diagonal and in the lower left corner). C is the cyclic permutation matrix performing a circular left-shift (rotation) on the elements of an N-vector. Its kth power C^k performs a rotation by k places.

The parity of the determinant of the $N \times N$ (composite circulant) matrix $M = C^{k_1} + C^{k_2} + \ldots + C^{k_m}$ decides the solvability of the linear system of equations on the individual bits in the recursions defined by rotations (by $k_1, k_2..., k_m$ positions) and bitwise XOR (with possibly a known number added to the result). Therefore, the matrix entries can be taken modulo 2 (0 or 1). Adding a matrix of all even entries to M does not change the parity of $\det(M)$.

Note that $C^N = I$, and $\det(C^k) = 1$. By $M = C^{k_1}\left(I + C^{k_2-k_1} + \ldots + C^{k_m-k_1}\right)$ we may always assume $0 = k_1 < k_2 < ... < k_m < N$. Since the system of parameters $\{k_1,k_2...,k_m; N\}$ fully determines the invertibility of the recursion represented by the corresponding circulant matrix M, we may call the system $\{k_1, k_2,..., k_m; N\}$ itself *regular* for $\det(M) = 1$, or *singular* for $\det(M) = 0$. We state the result mentioned in the introductory remark of this section as

Theorem 4.1 [1]. If for a system $\{k_1, k_2,...,k_m; N\}$ m is an even number, then the system is singular. That is, for regular systems m is necessarily odd. □

It is well known that a matrix **A** has an inverse over any field *iff* (if and only if) its determinant is non-zero ($\det(\mathbf{A}) \neq 0$). The inverse $\mathbf{A}^{-1}$ can be explicitly written as a matrix of cofactors.

Note that the determinant is multiplicative in general: $\det(AB) = \det(A)\ \det(B)$, and hence

$\det(M) \equiv \det^k(M) \equiv \det(M^k) \bmod 2$, for any integer $k > 0$.

The case $N = 2^n$

If we expand $M^2 = \left(C^{k_1} + C^{k_2} + \ldots + C^{k_m}\right)^2$, the double products contribute only even (~0) entries:

$$M^2 \equiv C^{2k_1} + C^{2k_2} + \ldots + C^{2k_m} \bmod 2.$$

If $N = 2^n$, squaring matrix M n-times gives $M^N \equiv C^{Nk_1} + C^{Nk_2} + \ldots + C^{Nk_m} \bmod 2$. Because $C^{Nk_1} = \left(C^N\right)^{k_1}$, and $C^N = I$ (the unit matrix), $M^N \equiv m\cdot I \bmod 2$. This proves the following

Theorem 4.2: If $N = 2^n$, M is invertible mod 2, that is the system $\{k_1, k_2,..., k_m; N\}$ is regular *iff* m, the number of non-zero diagonals is an odd number. □

Theorem 4.2 is important because it covers almost all practical cases in computer systems, where the word length is 8, 16, 32, or 64 bits, even the extended precision of 128 and 256 bits.

The case $N = q\cdot 2^n$, with odd q

After n squaring operations, two terms become equal: $C^{u2^n} = C^{v2^n}$, *iff* the exponents are congruent mod N: $2^n u \equiv$

$2^n v \bmod q{\cdot}2^n$, or equivalently $u \equiv v \bmod q$. These terms cancel each other; therefore, it is enough to consider those $M = C^{k_1} + C^{k_2} + \ldots + C^{k_m}$ matrices, where $k_1, k_2, \ldots, k_m$ are all different mod q. In particular, the following cancellation law holds true: if we add (or remove) $C^u + C^v$ where $u \equiv v \bmod q$, the parity of det(M) does not change. Thus we obtain the following useful

Corollary 4.3: If $N = q{\cdot}2^n$ and $u \equiv v \bmod q$, then replacing C^u by C^v in M does not change the parity of det(M). In particular, we can restrict our investigations to systems $\{k_1, k_2, \ldots, k_m; q{\cdot}2^n\}$ such that $0 \le k_i < q$, or $-(q\text{-}1)/2 \le k_i \le (q\text{-}1)/2$. □

Now the construction of a regular system $\{k_1, k_2, \ldots, k_m; q{\cdot}2^n\}$ (q odd) is easy as shown in

Corollary 4.4: The system $\{k_1, k_2, \ldots, k_m; q{\cdot}2^n\}$ (q odd) is regular, if $k_1, k_2, \ldots, k_m$ are chosen such that one residue class mod q contains an odd number of k_i values, and every other residue class contains an even number of k_i values. In this case det(M) is odd. □

We remark that if with the above notations $N = q$ (that is, $n = 0$) the statement of Corollary 4.3 reduces to a triviality: det(M) is odd if it is derived from a single rotation.

The sub-case $N = 3 \cdot 2^n$

This case has practical relevance for digital systems with a word length of 12, 24, 48... bits.

Theorem 4.5: If $N = 3 \cdot 2^n$ and $M = C^{k_1} + C^{k_2} + \ldots + C^{k_m}$ is an $N \times N$ matrix, then det(M) is odd *iff* one of the three residue classes mod 3 contains an odd number of k_i values, and each of the other two residue classes contains an even number of k_i values.

Proof: Corollary 4.4 shows that these determinants are indeed odd. As for the other direction, according to the cancellation law in Corollary 4.3, the following systems are to be considered:

$\{0; 3{\cdot}2^n\}$, $\{1; 3{\cdot}2^n\}$, $\{2; 3{\cdot}2^n\}$, $\{0,1; 3{\cdot}2^n\}$, $\{0,2; 3{\cdot}2^n\}$, $\{1,2; 3{\cdot}2^n\}$, $\{0,1,2; 3{\cdot}2^n\}$.

The first three are regular (obvious), the next three are singular (Theorem 4.1). In order to verify Theorem 4.5 we have to show that the last system is also singular. For this we manipulate the corresponding matrix. We do not change the determinant, if we add all the rows of index 4, 7..., $(4 + 3k)$,... to the first row, and add all the rows 5, 8..., $(5 + 3k)$,... to the second row. We obtain a matrix such that all the entries in the first two rows are 1, and hence the determinant is 0. □

4.1.1. Consecutive diagonals

In practice, the most important non-trivial invertible recursions (the fastest to compute) have *three* rotations. We can fully characterize the cases, when the rotation displacements are next to each other. We will revisit this case later, and prove a more general theorem with the help of binary polynomials.

Theorem 4.6: det($C^0 + C^1 + C^2$) = 0 mod 2 *iff* N is divisible by 3. That is, the system $\{0,1,2; N\}$ is singular *iff* $N = 3n$.

Proof: For $N \ge 6$: The top and bottom rows of the matrix look like:

1 1 1 0 0 0...
0 1 1 1 0 0...
0 0 1 1 1 0...
.....
1 0 0 0 *x*...
1 1 0 0 0...

We add rows 1 and 2 to the second but last row, and rows 1 and 3 to the last row, and obtain

0 0 0 1 *x*...
0 0 0 1 1...

in the last two rows. The first column of the matrix has now only a single leading 1 entry, so we can remove it together with the first row (Laplace's formula). In the new matrix the first column still has just a single leading 1, so this row/column removal can be done, all together 3 times. The result is a matrix of the original type, only its dimension decreased by 3. Repeat these reduction steps until the size of the matrix is reduced to ≤5. The result is one of three small matrices, and their determinants D_3, D_4, and D_5 are easily computed, completing the proof:

$$D_3 = 0,\ D_4 = 3,\ D_5 = 3.$$

Note: The above described reduction method works for *any* number ($m \ge 3$) of consecutive cyclic diagonals. We assume $N \ge 2m$ and, as usual, we can suppose $k_1 = 0$. We denote the sum of row 1 and row j of the matrix by s_j, we obtain the following modified rows:

$$S_2 = 1\underbrace{0\ldots0}_{m-1}10\ldots,\quad S_3 = 11\underbrace{0\ldots0}_{m-2}110\ldots,\quad \ldots,\quad S_m\underbrace{1\ldots1}_{m-1}0\underbrace{1\ldots1}_{m-1}0\ldots$$

When each of these rows is added to the corresponding row of index $N - m + 2$, $N - m + 3$,..., N, respectively, in the bottom section of the matrix the 1 entries in the leftmost m columns are effectively moved m positions to the right. We can apply Laplace's formula for column 1, then for column 2,... up to column $m - 1$, to reduce the matrix to an m-diagonal matrix of size $N - m$. The reduction process does not change the determinant.

These steps can be repeated until the matrix becomes too small for any further reduction. In the end m small matrices of size $m \times m$,..., $(2m - 1) \times (2m - 1)$ remain to be evaluated. The smallest one is of size $m \times m$. This matrix, having all its entries = 1, has 0 determinant. The other determinants can easily be computed and their parity may vary. The exact characterization of consecutive diagonals will be completed in Section 5, Theorem 5.2.

4.2 Modular binary polynomials

Using a polynomial model and arithmetic, we may obtain better insight to the problem of inverses and prove more general results.

4.2.1. Polynomial representation of circulant matrices

There is a one-to-one correspondence between mod 2 circulant matrices of size $n \times n$, and binary polynomials mod $x^n + 1$: Replace the unit circulant matrix C in the matrix equation with x, and replace the $(+,\times)$ matrix operations with their polynomial counterparts. The unit matrix I corresponds to the polynomial identically 1, and the matrix equation $C^n = I$ translates to the polynomial equation $x^n = 1 \bmod x^n + 1$ (note that $x^n - 1 = x^n + 1 \bmod 2$).

Proposition 4.7: If M^{-1}, the inverse of the circulant matrix M over a finite field exists, it is also a circulant matrix.

Proof: For the given size $n \times n$, there are only a finite number of circulant matrices over a finite field, so $M^a = M^b$ for some $a > b \geq 0$ integers. Multiply this equation $(b + 1)$-times with M^{-1} to get $M^{a-b-1} = M^{-1}$. The left-hand side is a non-negative power of a circulant matrix, so it is circulant. □

Corollary 4.8: A circulant matrix is mod 2 invertible *iff* the corresponding binary polynomial has an inverse, such that

$$p(x) \cdot q(x) = 1 \bmod x^n + 1.$$

The following lemma is well known.

Lemma 4.9: The inverse polynomial $q(x)$ of $p(x)$ exists *iff* $\mathrm{GCD}(p(x), x^n + 1) = 1$.

Proof: (a) The extended Euclidean algorithm computes the inverse $q(x)$ if $\mathrm{GCD}(p(x), x^n + 1) = 1$.

(b) If $\mathrm{GCD}(p(x), x^n + 1) = h(x) \neq 1$, then $p(x) = p_1(x) \cdot h(x)$ and $x^n + 1 = u_1(x) \cdot h(x)$ with some $p_1(x)$ and $u_1(x)$ polynomials. If there was an inverse, $q(x)$, then there is $u(x)$ polynomial such that $p_1(x) \cdot h(x)\, q(x) = 1 + u(x) \cdot h(x)$. It is equivalent to the impossible equation $[p_1(x) \cdot q(x) - u(x)] \cdot h(x) = 1$. □

The following result shows that the singularity of systems is "stable" for multiplied dimensions. Unfortunately no such stability holds for regular systems, even under stronger conditions.

Theorem 4.10: If the system $\{k_1, k_2, \ldots, k_m; N\}$ is

(i) singular, then for all integer $j > 0$ the system $\{k_1, k_2, \ldots, k_m; j \cdot N\}$ is also singular
(ii) regular and $d > m$ is a divisor of N, then $\{k_1, k_2, \ldots, k_m; d\}$ is also regular.

Proof: Note that (i) and (ii) are equivalent. Let $p(x)$ be the polynomial representation of the system. By Corollary 4.8 and Lemma 4.9, we have $\mathrm{GCD}(p(x), x^N + 1) \neq 1$. Since $x^N + 1$ divides $x^{j \cdot N} + 1$, $GCD(p(x), x^{j \cdot N} + 1) \neq 1$ and statement (i) follows. □

Note: Even if both systems $\{k_1, k_2 \ldots, k_m; N_1\}$ and $\{k_1, k_2 \ldots, k_m; N_2\}$ are regular, the system $\{k_1, k_2 \ldots, k_m; N_1 \cdot N_2\}$ is not necessarily regular. We find the following counterexample: {0,1,6; 7} and {0,1,6; 9} are regular systems, however {0,1,6; 63} is singular. Indeed,

$\mathrm{GCD}(x^6 + x + 1, x^7 + 1) = \mathrm{GCD}(x^6 + x + 1, x^9 + 1) = 1$, but $x^6 + x + 1$ divides $x^{63} + 1$. □

Theorem 4.11 [18]. Every irreducible binary polynomial of degree k divides $x^{2^k} + x \bmod 2$. If $k > 1$ then $p(x) \neq x$, so $p(x)$ has a constant term, and we have $p(x) | x^{2^k - 1} + 1 \bmod 2$. □

Corollary 4.12: For any $p(x)$ binary polynomial with a constant term, there exists an exponent t, such that $p(x) \mid x^t + 1 \bmod 2$.

Proof: Write the polynomial $p(x)$ (not divisible by x) as a product of powers of irreducible factors. Each irreducible factor has a corresponding multiple of form $x^u + 1$.

$x^u + 1$ and $x^v + 1$ both divide $x^{u \cdot v} + 1$ (where $u = v$ allowed), from an elementary identity. Because $x^{2uv} + 1 = (x^{uv} + 1)^2 \bmod 2$, $(x^u + 1) \cdot (x^v + 1)$ divides $x^{2uv} + 1$. Repeating this for all the factors of $p(x)$, we see that there always exists an exponent t, such that $p(x) \mid x^t + 1 \bmod 2$. □

Definition: We call the smallest of such t values the *characteristic exponent* of p.

Note that if $p(x) \mid x^u + 1 \bmod 2$, then u is a multiple of the characteristic exponent t. Indeed, we write $u = kt + r$ $(r < t)$ and we get $x^{kt+r} + 1 = ((x^t)^k - 1)x^r + x^r + 1$, and hence $p(x) \mid x^r + 1 \bmod 2$, a contradiction to the minimum choice of t.

Theorem 4.13: Given a $p(x)$ binary polynomial, let $t > 0$ denote its characteristic exponent. Then, $p(x)$ is invertible mod $x^n + 1$ *iff* it is invertible mod $x^{n+t} + 1$, or assuming $n > t$, *iff* it is invertible mod $x^{n-t} + 1$.

Proof: Since $p(x)$ divides $x^t + 1 \bmod 2$, it also divides $x^{t+n} + x^n \bmod 2$. Adding this to $x^n + 1$, we get $x^{t+n} + 1$, which is relative prime to $p(x)$ *iff* $x^n + 1$ is relative prime to $p(x)$. Suppose $n > t$, then we can apply the first part of Theorem 4.13 for $n - t$ in place of n, which completes the proof. □

Theorem 4.13 plays a fundamental role in testing the regularity of systems.

Corollary 4.14: Let p denote the polynomial associated to the system $\{0 = k_1, k_2, \ldots, k_m; N\}$, and t the characteristic exponent of p. If $N_1 = N_2 \bmod t$, the systems $\{k_1, k_2, \ldots, k_m; N_1\}$ and $\{k_1, k_2 \ldots, k_m; N_2\}$ are both regular or both singular. That is, the regularity of a system depends on the mod t residue class of the dimension. □

Note that by the above corollary, the notion of regularity/singularity of a system $\{0 = k_1, k_2, \ldots, k_m; N\}$ is meaningful for any dimension N, even if the N-dimensional matrix is too small to accommodate the rotations in the

system: the dimension is to be considered mod t, that is if N is too small we may always replace it by $N + t$.

4.2.2. Testing procedure

Based upon Corollary 4.14 above, we established the following testing procedure: if we want to know if a circulant matrix of a fixed set of diagonals, but of arbitrary size N, is invertible, we determine the characteristic exponent t, and the residue class $q = N \bmod t$. Now we have to compute the determinant of the circulant matrix of size q. In particular, if we know the "regular" residue classes mod t, we know every dimension numbers for which the system is regular or singular. Also, rather than computing determinants, we can deal with GCD($p(x)$, $x^q + 1$) to check regularity.

Corollary 4.15: A system is singular if the dimension is the characteristic exponent t, or any of its multiples. In greater generality, if q is a singular residue class mod t, then the system is singular in any dimension nq (n = 1,2,...). If q is regular residue class mod t and $d > m$ is a divisor of q, then the system is regular in dimension d as well. □

Lemma 4.16. For any binary polynomial $p(x)$, $x + 1 \mid p(x)$ mod 2 holds *iff* p has an even number of terms.

Proof: Indeed, adding pair-wise the terms of p, each such sum $x^a + x^b = x^a(x^{b-a} + 1)$ is divisible by $x + 1$. In case of even number of terms these pairs add up to the polynomial, making it a multiple of $x + 1$, while for an odd number of terms there remains a single term x^a, clearly not a multiple of $x + 1$. □

Theorem 4.17: Let $p(x)$ be the polynomial associated to a system $\{0 = k_1, k_2,..., k_m; N\}$ and let t denote the characteristic exponent.

(i) If $x + 1 \mid p(x)$ mod 2 then the system is completely singular, that is singular for any dimension N.
(ii) If $x + 1$ is not a divisor of $p(x)$ mod 2, then t - 1 and $t + 1$ are regular dimensions (in particular, the system is not completely singular). Moreover, if t happens to be a prime number, then the system is completely regular, that is regular for any dimension except the multiples of t.

Proof: Since $x + 1$ divides $x^n + 1$ mod 2 for any n, GCD($x^n + 1$, $p(x)$) = 1 cannot hold true, verifying (i). Next consider $x^{t+1} + 1 - (x^t + 1) = x^t(x + 1)$ mod 2 and we obtain GCD($x^{t+1} + 1$, $x^t + 1$) = GCD($x^{t-1} + 1$, $x^t + 1$) = $x + 1$. Since $p \mid x^t + 1$ and $x + 1$ is not a divisor of p, GCD($x^{t+1} + 1$, p) = GCD($x^{t-1} + 1$, p) = 1, showing that t - 1 and $t + 1$ are indeed regular. If, in addition, t is a prime number, then the multiples of a non-zero residue class run through all the residue classes' mod t, thus a single singular dimension would imply complete singularity which cannot be the case, proving (ii) and the theorem. □

Note that by Lemma 4.16 above, statement (i) is the polynomial version of Theorem 4.2. By the lemma and Theorem 4.17, the following corollary is immediate.

Corollary 4.18: The statements below are equivalent:

(i) A system is completely singular
(ii) A system has an even number of rotations
(iii) $x + 1$ is a divisor of the associated polynomial. □

Several examples are given below illustrating the frequent case of complete regularity.

5. Examples

5.1 Three non-zero diagonals

For certain fixed sets of diagonals (~polynomial coefficients, corresponding ultimately to the rotation distance in a rotate-XOR bit mixing function) we determined with a computer algebra system, at which word sizes n are the function invertible. We call n "invertible" or "regular".

The computation takes two steps. First, with a search loop we determine the smallest t, such that the corresponding binary polynomial $p(x)$ divides $x^t + 1$ mod 2 (the smallest characteristic exponent). Then, we check for which $n < t$, $p(x)$ is invertible. The computation for each case takes only a fraction of a second. (Recall, that we can transform the system to have $k_1 = 0$, that is, $p(x)$ to have a constant term 1.)

(1) For $p(x) = x^2 + x + 1$, $t = 3$. Only $0 = n \bmod 3$ is singular.

(2) For $p(x) = x^3 + x + 1$, $t = 7$. Only $0 = n \bmod 7$ is singular.

(3) For $p(x) = x^3 + x^2 + 1$, $t = 7$. Only $0 = n \bmod 7$ is singular.

(4) For $p(x) = x^4 + x + 1$, $t = 15$. Only $0 = n \bmod 15$ is singular.

(5) For $p(x) = x^4 + x^2 + 1 = (x^2 + x + 1)^2$, $t = 6$. The 0 and 3 residue classes mod 6 are singular.

(6) For $p(x) = x^5 + x^4 + 1 = (1 + x + x^2)(1 + x + x^3)$, $t = 21$. The singular residue classes mod 21: 0, 3, 6, 7, 9, 12, 14, 15, 18; (not the ones relative prime to 21).

(7) For $p(x) = x^6 + x + 1$, $t = 63$. Only $0 = n \bmod 63$ is singular.

5.2. Consecutive non-zero diagonals

The above techniques work for any number of nonzero matrix diagonals:

For $p(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 = (1 + x + x^3)(1 + x^2 + x^3)$, $t = 7$. Only $0 = n \bmod 7$ is singular.

The sum of k (odd) consecutive powers like the above case can be fully characterized. The degree of the polynomial is $k - 1$, thus multiplying the sum with $(x + 1)$ gives $x^k + 1$, and we have $t = k$ for the characteristic exponent. Hence, k is a singular dimension. In general, there can be other singular cases, like

GCD($x^3 + 1$, $x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$) = $x^2 + x + 1$ mod 2

GCD($x^6 + 1$, $x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$) = $x^2 + x + 1$ mod 2.

Having multiplied $p(x)$ with $(x + 1)$ the condition of invertibility is GCD($x^n + 1$, $x^k + 1$) = $x + 1$, because $x + 1 \mid x^n + 1$.

Lemma 5.1: If d = GCD(n, k) then GCD($x^n + 1$, $x^k + 1$) = $x^d + 1$.

Proof: We assume $k < n$ and put $n = q\,k + r$. Since $x^n + 1 = (x^k)^q \cdot x^r + 1 = [(x^k)^q - 1]\cdot x^r + x^r + 1$, and the term in the brackets is divisible by $x^k + 1$ mod 2, we have GCD($x^n + 1$, $x^k + 1$) = GCD($x^k + 1$, $x^r + 1$). This is the $(n, k) \rightarrow (k, r)$ reduction the Euclidean algorithm performs in computing GCD(n, k), and the algorithm ends at d and $x^d + 1$, respectively. □

Theorem 5.2: Let k be odd and $p(x) = 1 + x + ... + x^{k-1}$.

(i) $p(x)$ is regular mod $x^n + 1$ *iff* GCD(n, k) = 1. Or, equivalently, the system {0, 1..., $k - 1$; n} is regular *iff* GCD(n, k) = 1.

(ii) If $p(x)$ is irreducible then the characteristic exponent k is a prime number and the system is completely regular.

Proof: (i) If $p(x)$ is regular mod $x^n + 1$ then GCD($p(x)$, $x^n + 1$) = 1, therefore GCD($(1 + x)\cdot p(x)$, $x^n + 1$) = GCD($x^k + 1$, $x^n + 1$) = $1 + x$. By Lemma 5.1 we have GCD(n,k) = 1.

As for (ii), suppose $k = ab$, where $a > 1$, $b > 1$. Now $p(x) \mid x^{ab} + 1 = (x^a)^b + 1 = (x^a + 1)((x^a)^{b-1} + ... + 1)$ and $p(x)$ must be a divisor of one of the factors on the right hand side. This cannot hold, since for the degrees: $a < k$ - 1, and k - $a < k$ - 1. □

This Theorem settles the procedure left open in the Note following Theorem 4.6.

Note that the statement in (ii) cannot be reversed as we have seen the counterexample above:

$$p(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 = (1 + x + x^3)\quad(1 + x^2 + x^3),$$
$$t = k = 7,\ \text{prime number}.$$

5.3. Further notes

(1) If $p(x) = q^k(x)$, and $q(x)$ is an irreducible binary polynomial, the singular residue classes are 0 mod t_q, or 0, k, $2k$... mod t_p (we have $t_p = k\cdot t_q$).

(2) For the computations we needed a polynomial irreducibility test. There have been several such tests published. One of them is the Ben-Or test: a polynomial $p(x)$ of degree d is reducible if $GCD(x^{2^k} + x) \bmod p(x);\ p(x)) \neq 1$ for any $k < d/2$ (see [19]).

(3) There are a huge number of irreducible binary polynomials available (see [19]). For example:

d = 32: 134,215,680; d = 40: 27,487,764,474

This number is roughly doubling when d is incremented by 1. More precisely, for large degrees d the probability that a randomly chosen polynomial is irreducible is about $1/d$. These show that for machine word size $n \geq 32$, one has a very large choice of sets of diagonals to get an invertible binary circulant matrix.

(4) Irreducible binary *trinomials* of the form $1 + x^k + x^d$ can be listed with a computer algebra system:

k = 1: The primitive trinomials of the form $1 + x + x^d$ for $d \leq 400$ are those with d = 2, 3, 4, 6, 7, 15, 22, 60, 63, 127, 153

k = 2: $1 + x^2 + x^d$, $(d > 2)$ is irreducible for the following:

d = 3, 5, 11, 21, 29, 35, 93, 123, 333, 845, 4125

k = 3: $1 + x^3 + x^d$, $(d > 3)$ is irreducible for the following:

d = 4, 5, 6, 7, 10, 12, 17, 18, 20, 25, 28, 31, 41, 52, 66, 130, 151, 180, 196, 503, 650, 761, 986

k = 4: $1 + x^4 + x^d$, $(d > 4)$ is irreducible for the following:

d = 7, 9, 15, 39, 57, 81, 105

k = 5: $1 + x^5 + x^d$, $(d > 5)$ is irreducible for the following:

d = 6, 9, 12, 14, 17, 20, 23, 44, 47, 63, 84, 129, 236, 278, 279, 297, 300, 647, 726, 737

(5) Let $q(x)$ be an irreducible polynomial of degree $d > 1$ over a prime field F_p. The order of q is the smallest positive integer n such that $q(x)$ divides $x^n - 1$. It is also the multiplicative order of any root of q, and a divisor of $p^d - 1$. q is called a primitive polynomial if $n = p^d - 1$.

The smallest degree non-primitive binary irreducible polynomial is $x^4 + x^3 + x^2 + x + 1$. Its order is 5.

There is no degree 5 non-primitive binary irreducible polynomial, because 2^5 - 1 = 31, a prime.

There are three degree six non-primitive binary irreducible polynomials:

Ord($x^6 + x^3 + 1$) = 9

Ord($x^6 + x^4 + x^2 + x + 1$) = 21

Ord($x^6 + x^5 + x^4 + x^2 + 1$) = 21

There is no degree 7 non-primitive binary irreducible polynomial, because 2^7 - 1 = 127, a prime.

There are 14 degree 8 non-primitive binary irreducible polynomials.

6. Conclusion

We proposed new pseudorandom number generator modes of iterative algorithms built from non-multiplicative computer operations: the offset counter mode and offset hybrid counter mode. They are somewhat better than simple counter- or hybrid counter-mode generators described in [1]. Long cycle lengths of these and some other generators can be assured when the generator function is invertible. We showed that two-term rotate-add functions are never invertible, but many classes of rotate-XOR functions are. In particular, when the length of the computer word is a power of 2 (8, 16, 32, 64...),

any rotate-XOR function of an odd number of terms is invertible. For other word lengths, we presented simple algorithms that decides the invertibility of any given set of rotate-XOR terms, and listed the full answers for many classes of fixed terms. These pieces of information could help a system designer.

Author details
[1]CPU Technology, Pleasanton, CA 94588, USA [2]Purdue University, Fort Wayne, IN, USA

Competing interests
The authors declare that they have no competing interests.

Received: 25 July 2011 Accepted: 1 February 2012
Published: 1 February 2012

References

1. Hars L, Petruska G: **Pseudorandom recursions-small and fast pseudorandom number generators for embedded applications.** *EURASIP J Embed Syst* 2007, Article ID 98417, 13 (2007). doi:10.1155/2007/98417.
2. Anashin V: **Uniformly distributed sequences of p-adic integers.** *Math Notes* 1994, **55**:109-133.
3. Anashin V: **Uniformly distributed sequences of p-adic integers, II.** *Discrete Math Appl* 2002, **12**:527-590.
4. Anashin V: **Pseudorandom number generation by p-adic ergodic transformations.** *arXiv: Cryptography and Security* 2004 [http://arxiv.org/abs/cs/0401030/].
5. Anashin V: **Wreath products in stream cipher design.** *arXiv: Cryptography and Security* 2006 [http://arxiv.org/abs/cs/0602012/].
6. Anashin V, Khrennikov A: In *Applied Algebraic Dynamics. De Gruyter Expositions in Mathematics. Volume 49.* Walter de Gruyter, Berlin; 2009.
7. Klimov A, Shamir A: **A new class of invertible mappings.** *Workshop on Cryptographic Hardware and Embedded Systems 2002* 2003, **2523**:470-483, Lecture Notes in Computer Science.
8. Klimov A, Shamir A: **Cryptographic applications of T-functions.** *Selected Areas in Cryptography (SAC) 2003* 2004, **3006**:248-261, Lecture Notes in Computer Science.
9. Klimov A, Shamir A: **New cryptographic primitives based on multiword T-functions.** *Fast Software Encryption 2004* 2004, **3017**:1-15, Lecture Notes in Computer Science.
10. Klimov A, Shamir A: **New applications of T-functions in block ciphers and hash functions.** *Fast Software Encryption 2005* 2005, **3557**:18-31, Lecture Notes in Computer Science.
11. Klimov A: **Applications of T-functions in cryptography.** *Thesis for the degree of Ph.D., Weizmann Institute of Science* 2005.
12. Marsaglia G: **A current view of random number generators.** *Computer Science and Statistics: The Interface* Elsevier Science; 1985, 3-10.
13. Maurer U: **A universal statistical test for random bit generators.** *J Cryptogr* 1992, **5(2)**:89-105.
14. NIST Special Publication 800-22: **A statistical test suite for random and pseudorandom number generators for cryptographic applications.** 2008 [http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22b.pdf].
15. Ritter T: **Randomness tests: a literature survey.** 1996 [http://www.ciphersbyritter.com/RES/RANDTEST.HTM].
16. Menezes A, van Oorschot P, Vanstone S: *Handbook of Applied Cryptography* CRC Press; 1996.
17. Morris B, Rogaway P, Stegers T: **How to encipher messages on a small domain.** *Advances in Cryptology. CRYPTO* 2009 [http://www.cs.ucdavis.edu/~rogaway/papers/thorp.pdf].
18. Koblitz N: **A Course in Number Theory and Cryptography.**, 238, Proposition II.1.8 (Springer, Graduate Text in Mathematics 114, 1994), ISBN-13: 978-0387942933.
19. Arndt J: **Matters Computational: Ideas, Algorithms, Source Code.**, (Springer, 2010), ISBN: 3642147631.

doi:10.1186/1687-3963-2012-1
Cite this article as: Hars and Petruska: Pseudorandom recursions II. *EURASIP Journal on Embedded Systems* 2012 **2012**:1.