

# CELLULAR AUTOMATA: IS RULE 30 RANDOM?

DUSTIN GAGE  
UNIVERSITY OF MAINE - FARMINGTON

ELIZABETH LAUB  
SUSQUEHANNA UNIVERSITY

BRIANA MCGARRY  
CENTRAL MICHIGAN UNIVERSITY

DR. KEN SMITH - FACULTY ADVISER  
DEPARTMENT OF MATHEMATICS  
CENTRAL MICHIGAN UNIVERSITY

















ABSTRACT. Dr. Stephen Wolfram, developer of *Mathematica*, claims that Rule 30 can be used as an effective encryption scheme due to its random qualities. We investigate this claim by using a battery of statistical tests as well as identify properties that help characterize its security if used for encryption. The results provide evidence that Rule 30 shows adequate randomness for a high level of security with weaknesses isolated to even window sizes.

## 1. INTRODUCTION

Cellular automata is a relatively new development in modern science. It consists of simple progressions from algorithms, or Rules, over time. The purpose of this research was to investigate a claim made by Dr. Stephen Wolfram. Dr. Wolfram believes that the use of a particular algorithm, which he referred to as Rule 30, produces a binary sequence that is sufficiently random and can be used as a secure encryption system.

Rules are formed through a definition of the  $2^3 = 8$  possible progressions of three cells (the cell, the cells left-hand neighbor, and the cells right-hand neighbor). Each of these progressions gives a single output, producing a new cell and creating a three to one mapping. The Rules are then named using these progressions as shown in Fig. 1. The name of the Rule can be found by arranging the progressions, starting from the left with seven base two  $(111)_2$ , descending to zero  $(000)_2$ , and converting this base two number to base ten. In doing this, there are  $2^8 = 256$  possibilities, and therefore 256 possible Rules.

---

111	110	101	100	011	010	001	000
							
							
0	0	0	1	1	1	1	0

$(00011110)_2 = (30)_{10}$

**Figure 1: The name of each rule is given by the base 10 representation of their output. This is the set of parameters and outputs for Rule 30.**

---

Rule 30 is a function that maps three bits to one bit according to the equation:

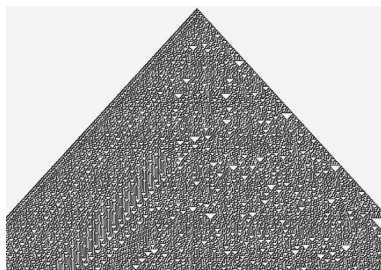
$$f(a, b, c) = a + b + c + bc.$$

Another way to execute Rule 30 is to “look at each cell and its right hand neighbor. If both of these were white on the previous step, then take the color of the cell to be whatever the previous

color of its left-hand neighbor was. Otherwise, take the new color to be opposite of the left hand neighbor” [4]. A visual correspondence with how Rule 30 behaves can be seen in Fig. 2 where Rule 30 acts on every cell in a row of zeros and ones to produce a new row. This process is then repeated.

The practical applications of cellular automata can be seen in multiple fields of science and social science. They range from computer science, technology, physics, biology, and math, to economics, psychology, philosophy, and even art [4]. The research we have focused on applies mainly to cryptology and random number generation.

Before any further mention of randomness, a definition must be established. The concept of “random” can be described in a variety of ways. The definition we have referred to in our research can be credited to Dr. Solomon Golomb, a professor at the University of Southern California. Dr. Golomb proposed three postulates for randomness, which he classified as preliminary steps. If a binary string passes all three postulates, the string can be considered as pseudo-noise and qualifies for further inspection. The



**Figure 2: This is the progression of a configuration with one value one site using Rule 30. Random behavior is observable and worthy of investigation.[5]**

---

first postulate states “If  $p$  is even then the cycle of length  $p$  shall contain an equal number of zeros and ones. If  $p$  is odd then the number of zeros shall be one more or less than the number of ones” [1]. The second postulate says “In the cycle of length  $p$ , for each  $i$  for which there are at least  $2^{i+1}$  runs,  $\frac{1}{2^i}$  of the runs have length  $i$ . Moreover, for each of these lengths, there are equally many gaps and blocks” [1]. Finally, the third postulate asserts “The out-of-phase autocorrelation is a constant” [1]. Essentially, the third postulate is simply the notion of independent trials. To pass this postulate, it must be impossible to predict the next value of the strip from previous values.

**Definition 1.** A cell’s **right-hand neighbor** is the cell immediately to its right, and a cell’s **left-hand neighbor** is the cell immediately to its left.

**Definition 2.** **Window Size** is the number of binary bits for a given automata.

**Remark.** For finite window sizes, the boundary condition for the first and last bit of a configuration is to make them adjacent and effectively “wrap” the window.

**Definition 3.** A **Seed** is the combination of 0’s and 1’s where the automaton begins its iterations.

**Definition 4.** A **State Diagram** is a structure that contains all  $2^N$  binary configurations, where  $N$  is the window size. The diagram is a digraph with each configuration as a vertex and directed edges showing their progression under the Rule, (Fig. 3).

**Definition 5.** A **Cycle** is a loop in the state diagram.

**Definition 6.** A **Tail** is a sequence of configurations in the state diagram to which the progression will not return. In other words, something that is part of a tail is not in a cycle.

**Definition 7.** An **End Point** is a configuration on the tail that has no predecessors in the progression.

**Definition 8.** A **Branch Point** is a configuration which has two or more predecessors in the progression.

**Definition 9.** A **Test Strip** is a binary strip found by selecting one cell in the seed and following the cell straight down through a given number of iterations. (This is chosen by the tester.)

**Remark.** Test strips are used for statistical testing. The seed typically used in this research to produce test strips was a single one and the rest zeros. The bit with a value of one in the seed was the bit that was followed to generate the test strip.

**Definition 10.** Let  $G$  be a permutation group acting on a set  $X$ . Let  $C$  be a collection of colorings of  $X$ . Then  $c_1, c_2 \in C$  are equivalent provided there is a permutation  $f \in G$  such that  $f * c_1 = c_2$ . Thus, two colorings are **inequivalent** provided that they are not equivalent [6].

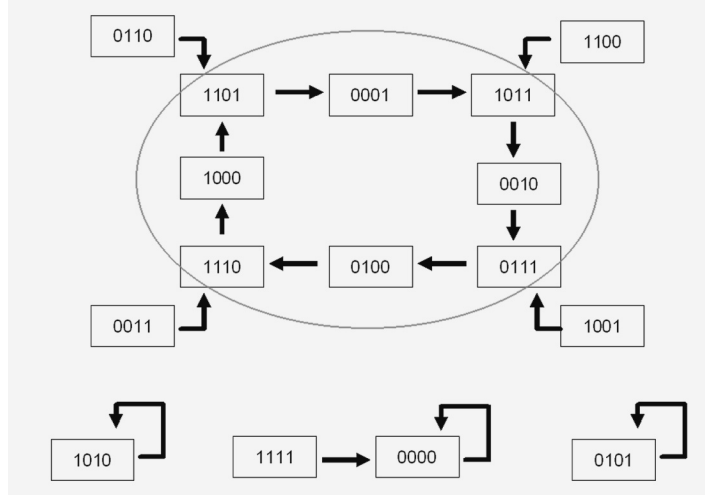


Figure 3: This is the state diagram for window size 4 under Rule 30. The arrows denote the direction of progression, and the cycle of maximum length is circled. Branch points have two or more arrows pointing at them(i.e. 1110), and end points have none(i.e. 0011).

Our research focused on both the local and global aspects of Rule 30 which consist of single strips of binary data and the overall structure of the automata, respectively. The local approach involved performing statistical tests for randomness on test strips of different lengths and different window sizes. The global approach involved inspecting the complete state diagram of the automata for different window sizes for which many different characteristics were examined including sizes of cycles, amounts of cycles, and amounts of tails.

## 2. PROCEDURE

**2.1. Local Structure.** Looking at a single strip of data and using statistics to evaluate it allows for a local look at different window sizes and Rules. Two computer programs were used to investigate and quantify properties of Rules, including Rule 30, for different window sizes. The first program output test strips with each bit separated by commas, as well as the size of the cycle, and the size of the tail that preceded the cycle. This program allowed for a decision of what Rule and what seed to start with at the time of compilation, and capped out on a window size of 36 bits. The other program output a stream with line breaks between every bit, and allowed for any window size. The program allowed for a decision of the starting seed, a Rule, how many bits to run, and the window size all at run time.

Using these two programs, we were able to gather a working collection of strips on which to run statistical tests. The main source of tests was a statistical battery package distributed by the National Institute of Standards and Technology (NIST). From this package, we were able to run tests on the collection of strips previously gathered. The tests that correspond to Dr. Golomb's three postulates of randomness are the frequency test, the runs test, and the spectral test, respectively. All of the tests were compared to a chi-square distribution in order to find p-values necessary for hypothesis testing [3]. In addition to statistical tests, the NIST program included an option to test binary streams created by a variety of random number generators (RNG's) [3].

Another program utilized for running statistical tests was MiniTab. This provided another runs test that used a normal distribution comparison to find the corresponding p-values for test strips. This program was only used for smaller window sizes where the cycle lengths were explicitly known. This ensured that the strips tested were aperiodic by keeping their length shorter than the cycle length. The MiniTab runs test was also used to test strip sizes of 1,000 bits and 50,000 bits.

The NIST Maurer’s Universal Test stream length requirement of 387,840 was too large for small window sizes to be tested, so a third program was written by Dr. John Daniels, a statistics professor at Central Michigan University. This program was written in SAS, a statistical programming language, and was used for smaller strip sizes. The use of the NIST package and Dr. Daniel’s program allowed us to test strips for their compressibility across all window sizes.

**2.2. Global Structure.** If the level of local randomness is found to be at an acceptable level, we must then turn to the global perspective of Rule 30 to identify properties that either help or hinder its use as an encryption system. Most of this work was done using computers with reliance on empirical evidence due to the exponential nature and computational intractability of Rule 30[5]. Dr Wolfram focused on the maximum cycle lengths for increasing window size, but there is only minimal data published about the tails on those cycles. Keeping track of both the cycles and tails is computationally more difficult, so an algorithmic approach was required to gather such data.

In order to meet and extend the published information gathered about Rule 30, the computer programs we developed used inequivalent colorings to increase the computational efficiency to produce results in a timely manner. Colorings in this context refer to the unique sequences of zeros and ones that make up a configuration. Since the boundary conditions effectively wrap the window, each configuration has a cyclic nature and we can partition all of the possible configurations, or “colorings,” into equivalence classes for window size  $N$  based on the symmetries of the cyclic group of order  $N$ ,  $C_N$ . This property is sometimes denoted as *shift invariance*, and by examining only one of the elements in each class, the number of necessary computations is brought down to approximately  $\frac{2^N}{N}$ . As the size of the window grew, identifying the inequivalent colorings themselves became a computational challenge, and so the Gray Code[6] was utilized to increase this efficiency.

**Remark.** *Given a configuration  $X$  on window size  $N$ , when  $X$  is written in base 10, all of its equivalent colorings are of the form  $X * 2^n \pmod{2^N - 1}$  for  $(n \geq 0)$ .*

The Gray Code[6] exhibits symmetries within the inequivalent colorings that improve the computational efficiency of searching for them spread throughout  $2^N$  possible configurations for window size  $N$ . A consequence of the above remark is that the equivalent colorings will be separated by powers of two and thus, half of the configurations do not need to be checked. This means that the Gray Code can produce the list of inequivalent colorings by searching  $2^{N-2}$  configurations. The searching process can be improved further by writing the configurations in base 10 and using the modular relationship from above to check equivalence. The use of Polya’s Counting Theorem[6] which generates the number of inequivalent colorings with a specific number of ones and zeros was used to improve the performance as well.

The Gray Code describes an ordering of  $n$ -tuples which can be used as a consistent ordering across all window sizes in the case of cellular automata. The ordering of the Gray Code is useful later on when looking for weaknesses across window sizes in regard to security and reliability as an encryption system. Using the above methodology, we produced and compiled data in its entirety for window sizes less than 25 and corrected previously published data for window sizes less than 36.

### 3. RESULTS

**3.1. Local Results.** Using the variety of programs to which we had access, we were able to create a large battery of statistical tests and a large table of p-values. Appendix 1 includes information about various window sizes of Rule 30, as well as Rule 45, Rule 89, and RNGs. Our null hypothesis for the statistical tests was that the strip was random. Highlighted [Appendix 1] are p-values that indicate failure of the null hypothesis at 95% and 99% confidence intervals. ( $\alpha = 0.05$  and  $\alpha = 0.01$ )

**3.2. Global Results.** Table 1 shows the number of each cycle length present in the state diagram for window size 5 to 24 with the max cycle separated from the rest. The table also shows the percent of all configurations that show up in tails versus cycles, and the dominance by tails is easily seen even for small window sizes. Since a good encryption system is dependent on the percent of total vertices evolving to the max cycle, the trend in those vertices was calculated and found to be approximately  $2^{(0.928N-0.1017)}$  for window size  $N$  using the last column.

Window Size	Length of Max Cycle	Tails in Max Cycle	Other Cycles	% in Tails	% in Cycles	% in Max
5	5	5	1x1	81.25	18.75	93.75
6	1	12	2x1	95.31	4.69	96.88
7	63	2	7x4, 1x1	28.13	71.88	60.16
8	40	15	1x8, 3x1	80.08	19.92	87.50
9	171	27	1x72, 1x1	52.34	47.66	80.86
10	2x 15	42	1x5, 3x1	96.29	3.71	82.03
11	154	53	11x17, 1x1	83.30	16.70	75.73
12	4x 102	127	1x8, 4x3, 3x1	89.48	10.52	93.46
13	832	66	1x260, 1x247, 1x91, 1x1	82.53	17.47	31.74
14	1428	265	2x133, 1x112, 1x63, 1x14, 7x4, 3x1	88.29	11.71	84.56
15	1455	432	5x30, 5x9, 15x7, 4x5, 1x1	94.58	5.42	92.70
16	6016	316	1x4144, 3x40, 1x8, 3x1	84.30	15.70	90.87
17	10846	2237	1x1632, 1x867, 1x306, 1x136, 1x17, 1x1	89.47	10.53	95.65
18	2844	1721	6x186, 1x171, 1x72, 6x24, 3x1	98.34	1.66	81.61
19	3705	500	1x247, 1x133, 1x38, 1x1	99.21	0.79	27.82
20	2x 6150	2492	4x3420, 4x1715, 1x580, 5x68, 4x30, 1x15, 1x8, 1x5, 3x1	96.76	3.24	73.40
21	2793	2201	7x597, 21x409, 1x63, 21x44, 1x42, 7x4, 1x1	99.21	0.79	9.62
22	2x 3553	240	1x3256, 2x781, 1x154, 2x77, 11x17, 3x1	99.63	0.37	94.90
23	38249	10489	1x4784, 1x138, 1x1	99.49	0.51	99.85
24	185040	15601512	1x5448 8x366 2x312 24x20 4x102 1x40 1x8 4x3 2x1 1x1	98.84	1.16	94.10

TABLE 1. Cumulative Data for window sizes 5 to 24 under Rule 30.

Table 2 shows the maximum cycle length,  $\Pi_N$ , for window size  $N$  from 4 to 36. A similar table was published[5], so the work provided to produce these results was to confirm the previously published data. The rows shown in bold differ from the previously published data where the numbers for maximum cycle were lower than the ones found during our research. The trend in maximum cycle calculated using the data below with Dr. Wolfram’s data for window sizes 37 to 54 was found to be  $2^{(0.621x+0.5081)}$ .

N	$\Pi_N$	$\log_2(\Pi_N)$
4	8	3
5	5	2.32
6	1	0
7	63	5.98
8	40	5.32
9	171	7.42
10	15	3.91
11	154	7.27
12	102	6.68
13	832	9.70
14	1428	10.48
15	1455	10.51
16	6016	12.55
17	10846	13.40
18	2844	11.47
19	3705	11.86
20	6150	12.59
21	2793	11.45
<b>22</b>	<b>3553</b>	<b>11.79</b>
23	38249	15.22
24	185040	17.50
25	588425	19.17
26	312156	18.25
<b>27</b>	<b>240300</b>	<b>17.88</b>
28	249165	17.93
29	1466066	20.48
<b>30</b>	<b>374265</b>	<b>18.51</b>
31	2841150	21.44
32	2002272	20.93
33	2038476	20.96
34	5656002	22.43
35	18480630	24.14
36	2237472	21.10

TABLE 2. Maximum cycle lengths  $\Pi_N$  found for cellular automata under Rule 30 for window sizes 4 to 36. The rows in bold signify the data that differs from[5].

An important consequence of the window’s shift invariance involves the presence of configurations with cyclic order less than the window size. This is a result of symmetry due to the divisors of the window size. For a given window size  $N$ , there exist configurations that can be partitioned into  $d$  identical configurations of length  $\frac{N}{d}$ . This means that the order of the entire configuration must be  $\frac{N}{d}$ . Since the boundary conditions will be satisfied by each partition as if they were in an  $\frac{N}{d}$  window, the entire configuration will evolve as if it were a window size of length  $\frac{N}{d}$ , and for this reason, we call them *Divisor Cycles*. When Rule 30 mimics a window size of  $\frac{N}{d}$  on a true window size of  $N$ , the Divisor Cycle formed must also be identical as seen in Fig. 4. Divisor Cycles consist of configurations that are effectively “trapped” since they are fully determined by a smaller window size. They, in turn, play an important role in producing large amounts of bad seeds that could cause an entire window size to be considered unacceptable for encryption.

---

1101 + 1101 → 11011101
0001 + 0001 → 00010001
1011 + 1011 → 10111011
0010 + 0010 → 00100010
0111 + 0111 → 01110111
0100 + 0100 → 01000100
1110 + 1110 → 11101110
1000 + 1000 → 10001000
1101 + 1101 → 11011101

---

**Figure 4: An example of a Divisor Cycle on a window size of 8. The Divisor Cycle mimics a cycle on a window size of 4.**

(000..011000..011), it is always a branch point. This means that for any given window size  $N$ , the cycle that the above configuration evolves to will produce more tails that evolve to Divisor Cycles in window size  $2N$ .

It is easily shown that the fraction of configurations in Divisor Cycles tends exponentially to zero as window size increases, but there is a caveat in that observation. The end-point of a state diagram in a window size  $N$  is not necessarily going to produce another end point when doubled and trapped in a window size of  $2N$ . The easiest example to observe is the sequence of  $N - 2$  cells with value zero and 2 adjacent cells with value one, (000..011). It can be shown that there is no configuration that maps to this in any window size, but it can also be shown that when doubled,

Window Size	Lost Tails	% Added to Divisor Cycles
6	54	100
8	16	8
10	150	27.03
12	180	17.39
14	862	6.47
16	5648	10.26
18	39582	15.72
20	172230	32.97
22	66321	3.13
24	240750	1.52

**TABLE 3.** Trapped configurations caused by the conversion of an end point to a branch point on a Divisor Cycle

The number of tails trapped by this situation is not necessarily negligible as seen in Table 3. In the case of window size 20, for example, over 30 percent of the tail vertices in the state diagram evolve to Divisor Cycles. If Dr. Wolfram's observation that the only configuration with more than two predecessors is the all 1's configuration on window sizes divisible by 3, the trapped new tails must be limited to only the doubling of window sizes [5]. This is true since the conversion of an end point to a branch point from a trapped configuration caused by three or more identical partitions would give rise to a branch point with three or more predecessors.

#### 4. DISCUSSION AND CONCLUSIONS

**4.1. Local.** The statistics suggest that these strings show adequate local randomness to be used as a RNG and possibly used for encryption. When looking at their cycle lengths, one would expect them to fail all tests that exceed this cycle length. Appendix 1 shows this to be the case most of the time. In the instances that a window size fails a test associated with one of Dr. Golomb's postulates, the other p-values are irrelevant (for that size strip). A failure of one of these tests indicates that the strip was not random; consequently no further testing is necessary.

Dr. Wolfram believes Rule 30, with a window size of 200, is adequate for a RNG. Only failing one statistical test out of the entire battery (not corresponding



to one of Dr. Golomb's postulates), it appears to follow a window size of 200 will be adequate, as far as local randomness is concerned.

Although various window sizes of Rule 30 fail some statistical tests, some of the RNG's also fail a few statistical tests. Therefore, Rule 30 is at least as good as some currently accepted RNG's. However, in order to know that this will be secure for encryption, different seeds must be taken into account, and evaluating global properties will do this.

**4.2. Global.** The inability to easily determine whether a configuration evolves to the max cycle means that the explicit knowledge of which configurations do not evolve to the max cycle, ("bad seeds"), might never be achieved, so with security in regard to encryption, a suitable replacement of this knowledge is empirical evidence suggesting the fraction of bad seeds tends exponentially to zero. The percent of allowable bad seeds in practice is application-specific, but it would be a safe estimate to consider 99% of the total configurations evolving to the max cycle as a lower bound on that requirement. Table 2 shows only one window size, 23, that meets this 99% requirement. The smaller window sizes would have no true application as an encryption system, so this table is mainly used to identify trends and show the cumulative data of their respective state diagrams. The empirical trend in vertices evolving to the max cycle is approximately  $2^{(0.928N-0.1017)}$  which bodes well for sufficiently large window sizes to meet the 99% requirement.

Table 3 does introduce a source for large amounts of bad seeds that may be present in window sizes too large to be fully analyzed. The conversion of an end point to a branch point on the Divisor Cycles is presumably restricted to even window sizes, and empirical evidence suggests that there is no apparent trending to identify when a Divisor Cycle will produce large amounts of bad seeds. Unless a pattern in this trapping of new tails is discovered, it is safer to consider all even window sizes to be suspect in regard to encryption security and reliability.

## 5. ACKNOWLEDGEMENTS

This research was done with assistance from faculty adviser Dr. Ken Smith. There was also assistance given by Dr. John Daniels with SAS program writing. Funding was provided by the National Science Foundation Research Experience for Undergraduates program and Central Michigan University's Summer Scholars Program. Programs written during our research process are available upon request: please contact Dustin Gage at [lorenzgage@hotmail.com](mailto:lorenzgage@hotmail.com) or Briana McGarry at [blmcgarry@gmail.com](mailto:blmcgarry@gmail.com).

## 6. FURTHER RESEARCH

- (1) Examine other fields (such as GF(4))
- (2) Examine window sizes larger than 24 (which will involve either a faster computer or more efficient algorithms)
- (3) Finding a pattern for the best window sizes
- (4) Examine Rules similar to Rule 30 in more detail (i.e. Rule 45 and Rule 89)

## REFERENCES

- [1] Beker, Henry and Fred Piper. *Cipher Systems, The Protection of Communications*. New York: Wiley-Interscience Publication (1982)
- [2] Maurer, Ueli. "A Universal Statistical Test for Random Bit Generators." *Journal of Cryptology* 5, no. 2, 1992, pp. 89-105.
- [3] Rukhin, Andrew, et al. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptologic Applications*. NIST special publication 800-822; Revised 15 May 2001. <http://nist.gov>.
- [4] Wolfram, Stephen. *A New Kind of Science*. Illinois: Stephen Wolfram LLC, 2002.
- [5] Wolfram, Stephen. "Random Sequence Generation by Cellular Automata." *Advances in Applied Mathematics* 7, 1986, pp. 123-126.
- [6] Brualdi, Richard. *Introductory Combinatorics Third ed.*. New Jersey: Prentice Hall Inc, 1999.