On New Schemes for Cryptography

Mark Carney

Affils: To be provided once peer reviewed.

March 26, 2016

Abstract

In this paper we hope to demonstrate new ways of thinking about potential new cryptographic functions, both with uses for hashing, as well as cryptography in general. We will introduce a new scheme for hashing and key-stream generation that does not make use of any external PRNG's, instead opting for ECA's. We will rely on an algorithm's mathematical content, in this case the satisfaction of the Devaney Criteria for a chaotic function. Through this, we try to guarantee the cryptographic strength we have become accustomed to, as well as permitting new features in cryptographic functions - such as variable length (non-truncating) and variable iteration count (without loss of strength).

1 Overview

In this article, we will give an overview of some new approaches the author would like to propose regarding new ways of choosing and implementing potential candidates for cryptographic functions. The ideas we consider for inclusion in this are;

- Pseudo-Random Number Generation
- Hashing functions
- Key-Stream Generation

We will allow the standard definitions for all of the above. We will, however, need to rigidly define our theoretical bases; namely Elementary Cellular Automata, and Chaos. Other terms we will define as required.

1.1 Definitions

We define a *string* as the concatenation of symbols form a set. A *binary string* is the concatenation of symbols from the set $\{0, 1\}$. Let Σ^n denote the set

$$\Sigma^n = \{\sigma : len(\sigma) = n \ \& \ \sigma \text{ is a binary string}\}$$

We denote the set of all binary strings of length $< \omega$ as Σ, where ω is the cardinality (*i.e.* the first ordinal in bijection with all other ordinals) of $\mathbb{N}$. [1]

Definition 1 (Elementary Cellular Automata)**.** *We define an Elementary Cellular Automata as an onto function f such that*

$$f : \Sigma^3 \mapsto \{0, 1\}$$

This function is iterated sequentially across a string known as the state*. Suppose we have a state of symbols from $\{0, 1\}$*

$$\ldots, x_{-2}, x_{-1}, x_0, x_1, x_2, x_3, \ldots$$

then, starting with $f(x_{-1}, x_0, x_1) = x'_0$, we iterate sequentially along the string.

These mappings were first described by Wolfram [REF!!!], and were subsequenty found to have a rich mathematical and computational structure. It is the subject of current research, to categorize the effects of different boundary conditions, for example.[2] Here we will only concern ourselves with the 'wrap around' concept of boundary (see footnote).

Their usual format is as follows:

111	110	101	100	011	010	001	000
s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0

ECA's are usually categorized by means of the concatenation of $s_7 \ldots s_0$ into a binary number. This number is then converted into decimal for conversational reasons.

An example of this notation for ECA's is exemplified by the rule we will use here; Rule 30. 30 in decimal is 00011110. This then converts to the following values for our automata:

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

Table 1: Rule 30 Automaton

1.2 Chaotic Dynamics

Our only aim in this subsection is to list the particular chaotic criteria we plan to use to justify our claims regarding the behaviour of the Rule 30 ECA (described later). We use the definition of Devaney Chaos from [REF!!!][3]

[1] The author did consider restricting this to the ordinal ω, but this seemed superfluous.

[2] Here, we could set, for some finite state of size n, $x_n = x_{-1}$, and $x_{n+1} = x_0$, thus creating a circular topology. There are many interesting effects, including stability in the chaotic Rule 30 ECA, which occur when you manipulate the boundary conditions.

[3] Hasselblatt, Boris; Anatole Katok (2003). A First Course in Dynamics: With a Panorama of Recent Developments. Cambridge University Press. ISBN 0-521-58750-6.

Definition 2 (Devaney Chaos)**.** *For a system to be described as Devaney Chaotic, it must be;*

1. *sensitive to initial conditions*
2. *topologically transitive (mixing)*
3. *have dense periodic orbits*

It is very interesting that Crannell [REF!!!][4] deftly demonstrates that sensitivity to initial conditions and density of periodic points are both implied by transitivity alone. [5]

Crannell's definitions of Chaos are as follows; a function is chaotic if:

1. it is *transitive* - that is, for any pair of non-empty open sets U and V in M, there is some $k > 0$ with $f^k(U) \cap V \neq \emptyset$
2. the periodic points of f are dense in M; and
3. f displays the famous condition of *sensitive dependence to initial conditions*: there is a number $\delta > 0$ depending only on M and f, so that in every non-empty open subset of M one can find a pair of points whose eventual iterates under f are separated by a distance of at least δ.

We shall discuss the importance of these conditions in the next section.

1.3 Rule 30 ECA and Chaos

Theorem 1.1. *(Cattaneo et al.) The Rule 30 ECA is chaotic.*

For a proof, see Cattaneo, Gianpiero; Finelli, Michele; Margara, Luciano (2000). [6] In fact, this meets the strict Devaney and Knudson definitions of a Chaotic Iterator.

We hope to give some reasons as to why this is of interest to our purposes in cryptography. Firstly, the fact that Rule 30, by Devaney's criteria, is sensitive to initial conditions is itself very promising - we want to detect a small change in the input file, and a chaotic function *by definition* will end up far away from wherever it might have done in the first instance by means of a single bit change, thanks to the requirement for sensitivity to initial conditions.

Secondly, transitivity implies that we will 'touch' every possible output (with some restrictions granted), given enough iterations. More strictly, we are given that for any two $\sigma_1, \sigma_2 \in \Sigma^n$ there is some k s.t.

$$f^k(\sigma_1) = \sigma_2$$

[4] Annalisa Crannell, The American Mathematical Monthly, Vol. 102, No. 9 (Nov., 1995), pp. 788-793

[5] The author notices that, given Rule 30 is left-permutative, that is, if two strings differ in position i, after iteration under Rule 30, they are guaranteed to differ in position $i + 1$ (Cattaneo), it is transitive, and so we get our full result with a minimum of effort!

[6] "Investigating topological chaos by elementary cellular automata dynamics". Theoretical Computer Science 244 (1–2): 219–241. doi:10.1016/S0304-3975(98)00345-4. MR 1774395.

as well as the existence of a converse k' satisfying the opposite action. It is a natural consequence that there exists for every n some i s.t. for every $\sigma_1, \sigma_2 \in \Sigma^n$ we have

$$f^i(\sigma_1) = \sigma_2 \wedge f^i(\sigma_2) = \sigma_1$$

as a fixed point theorem. This natrually arises from the fixed length of all strings in Σ^n giving a fixed periodic orbit size, although the potential algebraic properties of this system have not, to the author's knowledge, been much explored.

Thirdly, by Sharkovskii's theorem [REF!!!], we will have stable and unstable periodic points with orbits of any possibly value (again, with some restrictions granted for a discrete finite system). The exact bounds are yet to be calculated, but initial experimental results indicate that they are sufficiently large for our purposes.

We may ask; why are these last two points relevant? Essentially, they provide us the knowledge that in the system we have described, we are guaranteed to have resultant distances between two inputs of any magnitude, without the possbility of prior-computation of these points.

1.4 Rule 30 Cryptosystems: A Review of Previous Work

There is precedence for thinking that the Rule 30 ECA can be used for hashing, indeed, Zinuel 2006 [REF!!!][7] proposes such a scheme, but this only works within the Wolfram Mathematica framework, and in the author's opinion, fails to check adequately for stable points and periodicity.

Jamil *et al.* (2012) [REF!!!][8] have produced some wonderful work on mixing strategies, and sample analyses of different permutations regarding Rule 30 with other ECA rules (notably Rule 134 and Rule 126). This author's work, taking this as a starting point, disregards the apparatus of other hashing algorithms, seeking instead to explore the raw algorithmic randomness, in line with the NIST Statistical Tests [REF!!!].[9]

The final result is that it would appear to be strong enough by itself to withstand any manner of block-based permutation or block-chaining that could be desired! We also present the Rule 30 ECA as a PRNG and key-stream candidate. This work has been performed on FPGA hardware, presented in an excellent paper by Kim and Umeno [REF!!!][10], and the author here hopes to demonstrate a software-based approach.

As always, this would would not have been possible without S. Wolfram, who started much of this research off in 1985 [REF!!!][11]

[7] http://www.zimuel.it/talks/Nks2006.pdf

[8] http://www.ipcsit.com/vol27/32-ICICN2012-N10006.pdf

[9] http://www.ipcsit.com/vol27/32-ICICN2012-N10006.pdf

[10] http://arxiv.org/pdf/nlin/0412028.pdf

[11] S. Wolfram: Cryptography with cellular automata, Lecture Notes in Computer Science, Vol. 0218, (CRYPTO'85), pp. 429–432.

2 Kaos–Hash

Given this chaotic sensitivity to initial conditions, Rule 30 seems a good candidate for generating strong hashes. We propose the following implementation of the *Kaos-Hash* funciton

$$KH(\text{input, ouput length, no. of iterations})$$

2.1 An Implmentation

The implmentation is an algorithm as follows:

First we implement the circular topology function to work on a state of length 32-bits (4 bytes). Once it has completed a full pass from

$$f(x_{-1}, x_0, x_1), \ldots, f(x_{n-1}, x_n, x_{n+1})$$

we call this one iteration. We allow this function to permit multiple iterations in sequence, and then returns the resultant 32-bit state.

We call this function buffer_quatuor($input, iterations$). We build this into our hashing function as follows:

1. Receive an input file of size s, a number of iterations i, and a length l
2. Load the input file into memory buffer B
3. Pass bytes $B[0]$[12] through $B[3]$ to buffer_quatuor, along with i
4. Retrieve the output, and overwrite bytes $B[0]$ through $B[3]$
5. continue for every $(B[i], \ldots, B[i+1])$ quadruplet.
6. when we reach the end, go backwards until you reach $(B[j], \ldots, B[j+4])$, where j is the size of the file minus the desired length of the output.
7. Output bytes $B[j]$ through to the end of the buffer.

There are a number of considerations taken into account with this implementation. Chief amongst these, the Rule 30 ECA stabilises on the left of its output, and remains chaotic on the right. This is easily observed after several thousand iterations on a starting state with a single 1 symbol. Hence, we drop the left hand side of our results to prevent this from ever being a problem, and to keep things 'as chaotic as possible'.[13]

It must also be said that careful attention must be paid to the implmentation of boundaries - as discussed before, should things be off by one symbol/bit, the whole computation can stabilise. In this case, it stabilises to a string of hex value 0xAA. *Cavete Aedificator!!*

[12] The notation here is that $B[0]$ is the first byte (8-bits) of the buffer, $B[1]$ the second byte, and so forth.

[13] '...but no more so', to paraphrase Einstein, quoted by R. Sessions, inspired by William of Ockham.[REF!!!]

2.1.1 Mathematical and Experimental Notes on $KH()$

During testing, the author has noticed a significant difference between an input that varies by 1 bit, 1 byte, and many bytes. This is promising, although current entropic analysis does not seem to be able to delineate between this and other hashing algorithms.

Although Devaney Chaos guarantees transitivity, the periodicity of this particular property seems to be very sparse. This is promising, in terms of avoiding collisions, even at short lengths.

3 Kaos–Crypt

The author did not set about, originally, to create more than a hashing algorithm, the construction of the iterator lends itself to the generation of a strongly random keystream for simple encryption. The result is named by the author as *Kaos-Crypt*, or *KC* for short.

The proposed algorithm is as follows:

1. Receive an input file of size s, and a seed k
2. Load the input file into memory buffer B
3. Create a null buffer K, of size $s + 16$[14] with the first four bytes set to the binary coding of k.
4. Pass $(K[0], K[1], K[2], K[3])$ to a function kaos_nascitur, with the iteration value set to 16 times. Store the return value at the starting 4 bytes of K.
5. Check if the filled length of $K > s$. If not, pass the first four bytes into kaos_nascitur, iterate for 16, and store the result in the next four bytes of K.
6. If the filled length of $K > s$, continue to 7, else, pass the current 4 bytes back into kaos_nascitur for another 16 iterations. Store the result in the next 4 bytes of K, and go to 6.
7. If the buffer is the required length, XOR the buffer K with B to encrypt the file.

The decyption process is identical. We can also extend this with an extra iteration parameter, both of which can be derived from a DHE-like exchange process. It is not intended as a replacement for $RC4$, but the author has had some success in implementing it as such.

4 Summary and Analysis

4.1 Important Points

We provide the following list of things we hope readers will agree have been achieved, at least in part.

[14] The extra 4 bytes cover the sequential nature of the iteration, of $s \mod 4$

1. Kaos-Hash and Kaos-Crypt are multi-parameter algorithms, whereas all other known hashing and keystream algorithms are single input. (with noted exceptions such as PBKGFv2 etc., which provided a basis during the early implementations of this scheme)

2. Due to the ECA base, the complexity of computational operations (i.e. the physical steps taken to perform a single operation) is dramatically reduced, and as such, computational time should be dramatically diminished, or in the same timeframe, the overall strength of the operation should increase. In essence, we achieve through bitshifts what other algorithms achieve through lookups of large data-sets or complex on-the-fly computations.

3. Our algorithm satisfies the requirements of the operations (hashing, key stream generation) that are expected of it by virtue of its mathematical properties.

4. it is experimentally comparable to other algorithms designed to achieve the same result.

The author wishes to point out specifically that the multi-parameter possibilities with these algorithms is where lies their potential true strength - especially with respect to hashing:

- **Variable Output Length** - This allows the resulting hashes to be of any and varible length. If implemented properly (see example code) a variation in length can also change the hash itself. This is in contrast to the comparably weak methods that are the simply truncated hashing functions, such as SHA-384 and SHA-224.

- **Variable Iteration Count** - given the strong resistance to reversing the algorithm (see Zimuel [REF!!!]), we are assured that the recovery of the original input approaches asymptotically the enumeration of all possible inputs very quickly. As such, after a small fixed number of iterations, they can be varied. Experimentally, this has generated *significantly* different hash outputs with a difference of only ± 1 in the value for the iterations.

4.2 Algorithmic Improvements

Here we will describe some of the more practical considerations.

4.2.1 Entropic Maintenance

To ensure entropic maintenance, we allow that any null bytes that are input into our iteration function are reassigned a value of 2155905152. This places a 1 in the MSB for each byte of a 4-byte array, resulting in a state

[10000000 10000000 10000000 10000000]

This is advantageous, as it allows us to use the chaotic right-hand side of the algorithm more effectively. It also helps to seed the input in a useful and, as demonstrated, strong way.

4.2.2 Bit-shift and Comparator Implementation

Some internet sources [REF!!!][15] favoured the use of a bit-shift implementation - the author found this to be much more efficient on memory, and has included a version here with better boundary conditions:

```
#define B(x) (1UL << (x))
        for (state = i = 0; i < 32; i++){
                if (30 & B(7 & (st>>(i-1) | st<<(32-i))))
                        state |= B(i);
}
```

4.3 Motivations

It was found that modern crytographic PRNG's had fallen victim to the inclusion of a Discrete Logarithm Solution to the NIST SP800-90 PRNG. [REF!!!] This was supposed in 2007 by [GET THE NAME AND REF!!!], as no one was able to identify precisely how P and Q were constructed. The current consensus is that these algorithms are potentially compromised mathematically.

In light of these issues, it seemed prudent to create an Open Source inspired solution to the problem. By this, the author intends the following project;

> Modern cryptography should look to easily verifiable algorithms that use direct implementations of known strong phenomena, like ECA Chaos, that do not rely on any unknowns, whatsoever.

There is also a rich possibility for new analytical techniques and methods to be created and formulated for analysis and attack on broader classes of cryptosystems, and recent work in algorithmic randomness, in the author's opinion, anticipates such developments.

5 Example Implementations

We present the following example code. This code is inspired by sources found on the internet, such as Rosetta Code[16], with some alterations to bring it in line mathematically made by the author.

The code we present is written in C, and successfully compiles and runs on Linux-based architectures. [17]

(For PoC Code, please see the github.com page.)

[15] `http://rosettacode.org/wiki/Elementary_cellular_automaton`

[16] RosettaCode page for Rule30, C implementation.

[17] We apologise for the quality of the code - the author is a Mathematics Ph.D. student and Penetration Tester by day; the existence of this code should be considered a bonus.