The MECA Block Cipher

John Holly

November 17, 2025

Abstract

In an effort of moving symmetric cipher abstractions up a level and simplifying implementation, a novel cipher was developed that metamorphically evolves and has data-dependent operations. The cipher is dependent on first and second-order cellular automata and metamorphic engines for encryption/decryption and key-scheduling with careful security considerations. The cipher was developed and made in an easily templated fashion to accomodate larger block-sizes and variable sized architectures.

1 Introduction

In reality, I wanted to see if something I read about in a thriller novel could be done. Dan Brown's Digital Fortress mentioned a "rotating ciphertext" which I took as a data-dependent encryption, where the same operation isn't being used for individual blocks. My thesis was to simplify symmetric encryption and accomplish said goal using what has been used for S-boxes in the past.

MECA (term is taken from A. H. Encinas and Dios 2008 as a shorthand for second-order cellular automata; a higher-order automata) is a novel block cipher utilizing second-order cellular automata (MECA) and metamorphic engines in both encryption/decryption rounds and key scheduling.

1.1 Cellular automata

Cellular automata are discrete systems that undergo state transitions by iterating over each cell within the given dimensionality by means of a local transition rule. The operating area for a local transition is the cell itself and a specified number of neighbors. This sliding window across the state can be unbalanced but generally has an equal number of neighbors on either side of the target cell. For the purpose of this paper, the cellular space is one-dimensional and the neighborhood is the cell with its left and right neighbors. All CAs within this paper are two-state CAs, that is each cell is either a 1 or a 0. When the neighborhood reaches the boundary of the cellular space, a boundary condition must be put in place. For the purpose of this paper, the boundary is periodic (cyclic); borrowing a neighbor from the opposite side of the cellular state.

1.1.1 First-order (CA)

When talking about orders of CAs, this is simply referring to the number of state transitions (timesteps) considered during a transition. First-order CAs are only aware of the current timestep.

A CA can be described as a quadruple $(\mathcal{L}, \mathcal{S}, \mathcal{N}, f)$, where $\mathcal{L}$ is the cellular space (state size) in one dimension, $\mathcal{S}$ is the finite set of states, $\mathcal{N} = (\vec{v_1}, \vec{v_2}, ..., \vec{v_m})$ is the association of a single cell's neighborhood of m cells (including the cell itself) within $\mathcal{L}$, $f : \mathcal{S}^m \to \mathcal{S}$ is the local transition rule of the CA.

Let r be the rule number selected where r is between 0 and 255, $\mathcal{P}$ is the set of all permutations for a given neighborhood size where $\mathcal{N} \ni \mathcal{P}$, i is the index of $\mathcal{N}$ in $\mathcal{P}$.

$$\mathcal{S}^m \to \mathcal{S} = f(\mathcal{N}, r)$$

The local transition is defined as the selected rule number bit-shifted right by the index of the $\mathcal{N}$ in $\mathcal{P}$ ANDed with 1.

$$f(\mathcal{N}, r) = r \gg i \,\&\, 1$$

1.1.2 Second-order (MECA)

A second-order CA is defined similarly, with the exception of it tracking two timesteps $\mathcal{S}_t$ and $\mathcal{S}_{t-1}$. The second-order local transition is an extension of the definition of the first-order f above. If the cell in $\mathcal{S}_{t-1} \neq f$ then the cell is set to 1, otherwise 0. By swapping the end state and preceding state the evolution can be reversed.

1.2 Metamorphic Engines (CLUs)

Metamorphism is a process that occurs in rocks where transformation to a rock of a different composition occurs. The data-dependent transformations in CA and MECA make this seem like a fitting nomenclature... dude.

There are two metamorphic engines in use, one using irreversible first-order CA rules as a one-way hashing function for the key-generating-key (unscheduled, the one of b bytes specified as a parameter), and one used during encryption/decryption rounds selecting the second-order rules per round. The first-order engine selects from 3 class 4 (Dhar et al. 1994) elementary CA rules. The second-order engine selects from 7 cherry-picked rules that also display globally chaotic behavior (A. H. Encinas and Dios 2008).

2 Design and Motivation

The design of MECA began with inspiration from a handful of other well known ciphers: RC6 (Ronald L. Rivest and Yin 1998) for its small size and simplicity, speed, and elegant extensibility to different word sizes, rounds, and key length, Stone Cipher-192 (SC-192) (Saeb 2009) for its crypto logic unit (metamorphic engine, or CLU), and a variety of papers I've read on novel ciphers utilizing first-order elementary cellular automata.

The MECA cipher was designed to be simplistic and easily understood to those without a deep knowledge of mathematics and cryptography from an implementation standpoint. The goals in mind were simplistic chaos that can be easily visualized, extensibility, efficiency in both hardware and software implementations, and exploitation of the reversibility of second-order cellular automata in place of traditional feistel networks. This is accomplished by having the current/previous or next timestep encoded in the encrypted data's upper/lower half of blocks - this also simplifies encryption/decryption logic by requiring simply a swap of said states.

3 Details

Similarly to RC5 and RC6, MECA is fully parameterized and can be specified as MECA-$w/r/b$ with the word size (registers) being w bits, r encryption rounds, and b bytes in the encryption key. Metamorphic logic units and cellular automata (both first-order and second-order) are the primitive building blocks throughout the cipher. Much like feistel networks can be run in reverse with the same logic as forwards, MECAs can be run in reverse by simply swapping the pre-initial and initial states (the previous and current timesteps [state] of the cellular automata) the forward pass (encryption) converged to.

3.0.1 First-order CLU

The first-order CLU selects a class 4 elementary CA rule based on the remainder (mod 3) of the key-generating-key word at index i of a specific iteration in the scheduling algorithm XORed with the current key schedule key at index j. The mappings are shown in table 1 and the rules are shown evolved with random initial conditions in Figure 1, Figure 2, Figure 3.

The metamorphic engine for CA rule selection is defined by the piecewise function below.

$$m(x) = \begin{cases} 54 & x \; mod \; 3 = 0 \\ 110 & x \; mod \; 3 = 1 \\ 137 & x \; mod \; 3 = 2 \end{cases}$$

Table 1: First-order CLU mapping

Remainder	**Rule**
0	54
1	110
2	137

Figure 1: Rule 54

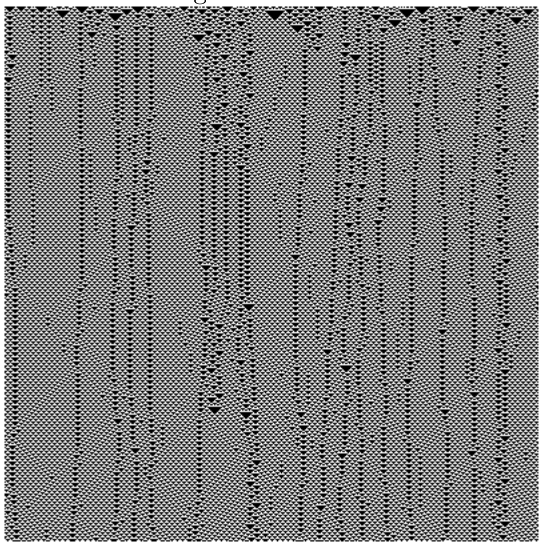


Figure 2: Rule 110

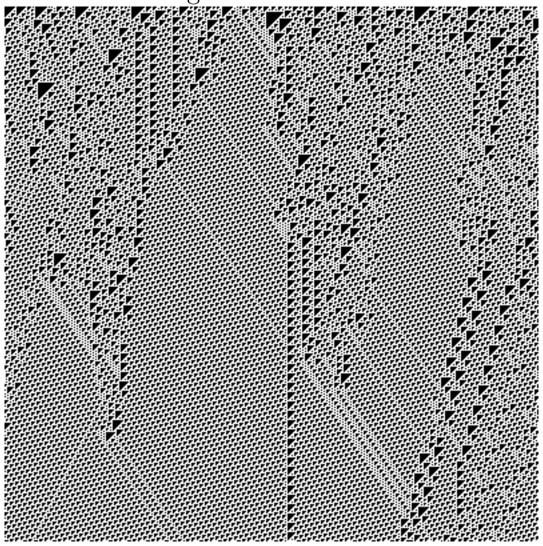
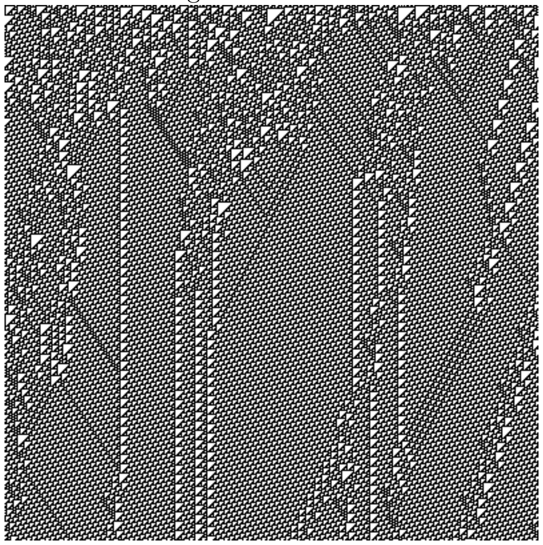


Figure 3: Rule 137



3.0.2 Second-order CLU

The second-order CLU selects a chaotic second-order rule based on the remainder (*mod* 7) of the current state of the MECA (timestep t). The rules are shown evolved with random initial conditions in Figure 4, Figure 5, Figure 6, Figure 7, Figure 8, Figure 9, Figure 10.

The metamorphic engine for MECA rule selection is defined by the piecewise function below.

$$m(x) = \begin{cases} 75 & x\ mod\ 7 = 0 \\ 86 & x\ mod\ 7 = 1 \\ 89 & x\ mod\ 7 = 2 \\ 149 & x\ mod\ 7 = 3 \\ 166 & x\ mod\ 7 = 4 \\ 173 & x\ mod\ 7 = 5 \\ 229 & x\ mod\ 7 = 6 \end{cases}$$

Figure 4: Rule 75

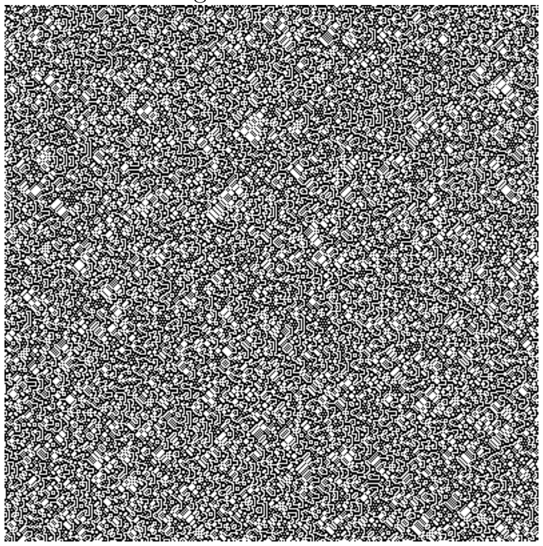


Figure 5: Rule 86

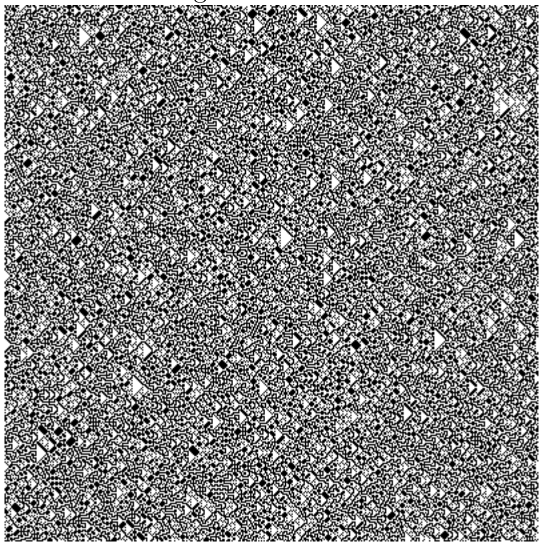


Figure 6: Rule 89

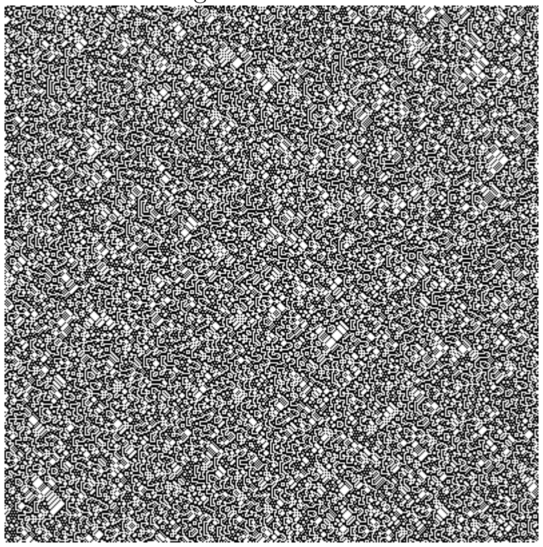


Figure 7: Rule 149

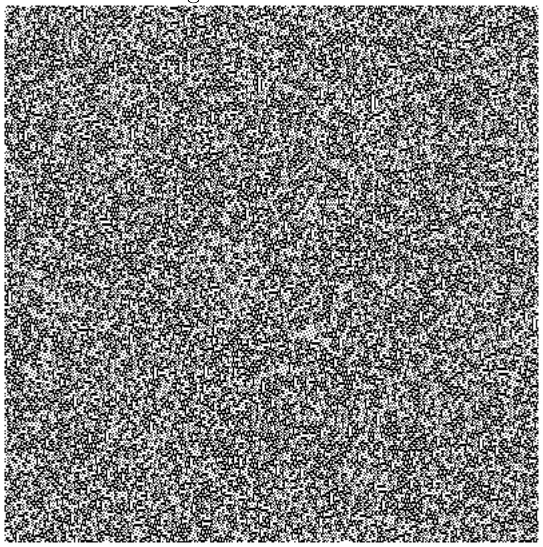


Figure 8: Rule 166

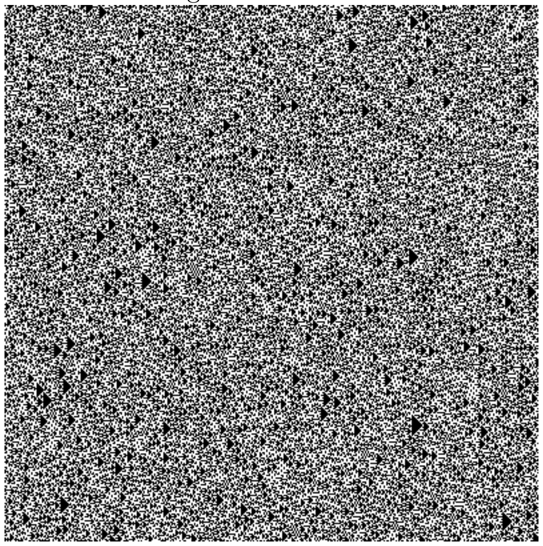


Figure 9: Rule 173

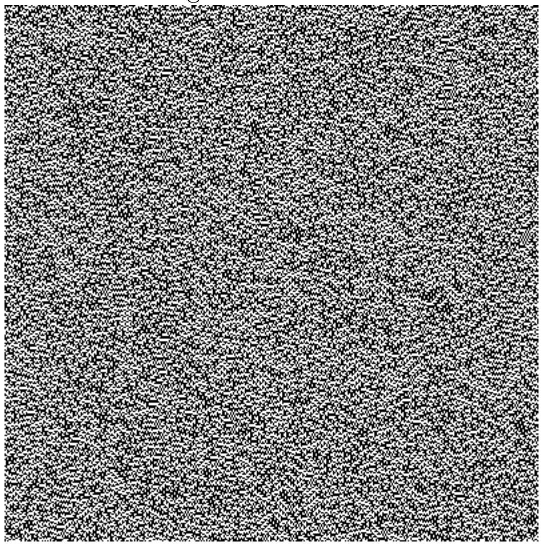
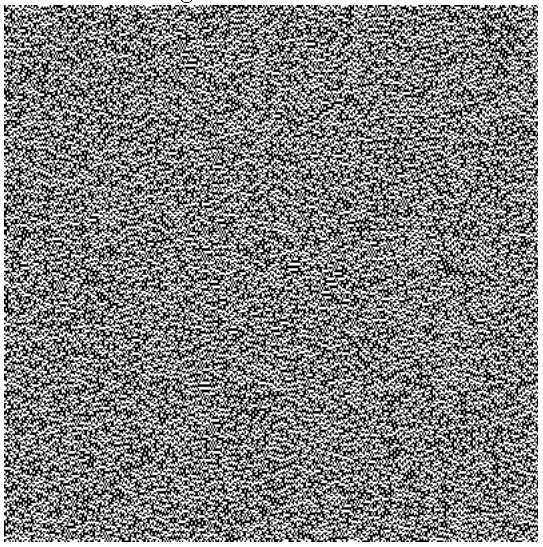


Figure 10: Rule 229



3.1 Key Schedule

The key schedule uses the same magic constants for initialization as RC6. Let P be the binary expansion of $e-2$ and Q is the binary expansion of $\varphi - 1$, where φ is the golden ratio.

The function for the outcome of a single CA timestep with a given rule is defined by the function below. Let $m(x)$ be the desired metamorphic rule and $c(S_i, m(x))$ is the execution of the elementary CA, where S_i is the resulting round key (for a given iteration).

$$S_i = c(S_i, m(x))$$

Algorithm 1 Key schedule for MECA-$w/r/b$

```
Input
    L    $b$ byte key preloaded into $c$-word array
    r    number of rounds
Output
    $S$    $w$-bit round keys $S_{0,\ldots,2r-1}$
$S_0 \leftarrow P$
for $i \leftarrow 1$ to $2r-1$ do
    $S_i \leftarrow S_{i-1} + Q$
end for
$j \leftarrow 0$
$i \leftarrow 0$
for $s \leftarrow 0$ to max(c, $2r$) do
    $S_i \leftarrow c(S_i, m(L_j \oplus S_i))$
    $i \leftarrow i + 1 \ mod \ 2r - 1$
    $j \leftarrow j + 1 \ mod \ c - 1$
end for
```

3.2 Encryption

The encryption processes takes in 4 w-bit words of plaintext and creates two w-bit MECAs (A and B) by taking the first/last two words and using them as the initial/pre-initial states.

The function for the outcome of a single MECA timestep with a given rule is defined by the function below. Let $m(x)$ be the desired metamorphic rule and $z(X, m(x))$ is the execution of the MECA.

$$X = z(X, m(x))$$

Algorithm 2 Encryption (forward evolution) for MECA-$w/r/b$

```
Input
    $P$    4 $w$-bit words of plaintext
    $S$    key schedule of $2r$ $w$-bit words
Output
    $C$    4 $w$-bit words of ciphertext
$A_{t-1} \leftarrow P_0$
$A_t \leftarrow P_1$
$B_{t-1} \leftarrow P_2$
$B_t \leftarrow P_3$
$j \leftarrow 0$
for $i \leftarrow 0$ to $r - 1$ do
    $A_t \leftarrow A_t \oplus L_j$
    $A \leftarrow z(A, m(A))$
    $B_t \leftarrow B_t \oplus L_{j+1}$
    $B \leftarrow z(B, m(B))$
    $j \leftarrow j + 2$
end for
$C_0 \leftarrow A_{t-1}$
$C_1 \leftarrow A_t$
$C_2 \leftarrow B_{t-1}$
$C_3 \leftarrow B_t$
```

3.3 Decryption

The decryption processes takes in 4 w-bit words of ciphertext and creates two w-bit MECAs (A and B) by taking the first/last two words and using them as the initial/pre-initial states, they are reversed during decryption.

The function for the outcome of a single MECA timestep in reverse with a given rule is defined by the function below. Let $m(x)$ be the desired metamorphic rule and $z(X, m(x))$ is the execution of the MECA.

$$X = z(X, m(x))$$

Algorithm 3 Decryption (reverse evolution) for MECA-$w/r/b$

```
Input
    $C$   4 $w$-bit words of ciphertext
    $S$   key schedule of $2r$ $w$-bit words
Output
    $P$   4 $w$-bit words of plaintext
$A_{t-1} \leftarrow C_1$
$A_t \leftarrow C_0$
$B_{t-1} \leftarrow C_3$
$B_t \leftarrow C_2$
$j \leftarrow 2r - 1$
for $i \leftarrow 0$ to $r - 1$ do
    $A \leftarrow z(A, m(A))$
    $A_{t-1} \leftarrow A_{t-1} \oplus L_j$
    $B \leftarrow z(B, m(B))$
    $B_{t-1} \leftarrow B_{t-1} \oplus L_{j-1}$
    $j \leftarrow j - 2$
end for
$P_0 \leftarrow A_{t-1}$
$P_1 \leftarrow A_t$
$P_2 \leftarrow B_{t-1}$
$P_3 \leftarrow B_t$
```

4 Performance

At optimal throughput, here are is the runtime for encryption and decryption for various word sizes. These tests were conducted from a C++ implementation running on an 12th Gen Intel(R) Core(TM) i5-1235UCPU @ 2.50GHz under Linux kernel 6.17.7-1.

Table 2: Performance of various word sizes

Word size	**Encryption Execution Time**	**Decryption Execution Time**
16-bit	31 us	22 us
32-bit	36 us	36 us
64-bit	65 us	65 us

5 Implementation Considerations

Implementing the MECA cipher can pose its own challenges in terms of performance. Firstly, when creating the second-order automata A and B during half-rounds, you will need to efficiently divide the block in half which (depending on word size) can be up to 128-bits each. Finding a suitable container for the state is key to performance and avoiding strings at all cost. My solution for the reference implementation was using C++'s STL bitset. This way, you may efficiently create two 128-bit wide bitsets and left-shift/OR the two 64-bit words efficiently into the state of A and B.

A future direction may be to use arrays of function pointers for metamorphic engines and compare performance, use optimized processor instructions/instruction sets, or implement this in hardware.

6 Test Vectors

6.1 16-bit words (64-bit block)

Plaintext

0 1	2 3	4 5	6 7
1337	1337	1337	1337

Key

0 1	2 3	4 5	6 7	8 9	10 11	12 13	14 15	16 17	18 19	20 21	22 23	24 25	26 27	28 29
4d79	2063	7269	6d65	2069	7320	7468	6174	206f	6620	6375	7269	6f73	6974	7900

Encrypted

0 1	2 3	4 5	6 7
ee1c	fa14	68e1	51c8

Decrypted

0 1	2 3	4 5	6 7
1337	1337	1337	1337

6.2 32-bit words (128-bit block)

Plaintext

0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
00001337	00001337	00001337	00001337

Key

0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15	16 17 18 19	20 21 22 23	24 25 26 27	28 29 30 31
4d792063	72696d65	20697320	74686174	206f6620	63757269	6f736974	79000000

Encrypted

0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
d93041ec	e3fa5b94	01a2df58	71ca05ab

Decrypted

0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
00001337	00001337	00001337	00001337

6.3 64-bit words (256-bit block)

Plaintext

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15		
0000000000001337	0000000000001337	0000000000001337	0000000000001337

Key

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
4d79206372696d65	2069732074686174	206f662063757269	6f73697479000000

Encrypted

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15		
e72d26f7318d3da2	0ec874868d14ac9b	7d91e0e34ee12d46	0fd1e08f367ad342

Decrypted

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15		
0000000000001337	0000000000001337	0000000000001337	0000000000001337

7 Security

First and foremost, when considering security - I wanted a large key size. The goal was to use a first-order cellular automata - whether second-order or first-order for everything. With this in mind, without using an SMT solver, reversing a first-order cellular automata that is chaotic - such as the chosen rules is quite difficult and computationally expensive. A key size of up to 255 bytes is possible much like in RC6, as much of the key derivation function was adapted from it. The key schedule, however, is metamorphically evolved using globally chaotic (class 4) rules of elementary cellular automata fulfilling the desire of the previous sentence. This makes them mostly irreversible outside of plausible SMT solver attacks as demonstrated in PagedOut!'s second issue (Ascherman 2019).

To make this block-cipher's operations data-dependent and thus harder to reverse the algorithm in use at a given time, a methodology was chosen from the Stone Cipher (Saeb 2009) paper that selectively chooses globally chaotic rules to evolve from identified as the "metamorphic engine" in this paper. As such, the encryption process evolves during encryption and in key-scheduling, increasing confusion/diffusion. During encryption/decryption - both the current and previous/next timestep are metamorphically evolved.

In terms of implementation security - the entire goal was to create a simple algorithm that is elegant and easy to understand with a concise implementation, heavily focusing on boolean algebra rather than heavy math.

8 Flexibility and Future Direction

The implementation of this cipher was designed to be templated as an easy means of accomodating higher block-sizes as future-proofing requires and architecture word sizes grow, rules are also configurable and I have done auxiliary testing on rules greater than 8 bits that have shown even more globally chaotic options in higher bit-sizes once linear and other rules have been filtered away.

Another future direction may be to add a third metamorphic engine for the binary operation of combining key with plaintext.

As cellular automata are highly efficient in hardware, I'd like to see FPGA and other implementations that can remove the slowness of certain software executions to have real competition for existing block-ciphers.

9 Conclusions

I was able to accomplish data-dependent encryption by adapting methodologies from the Stone Cipher (Saeb 2009) paper and even extending that to key-scheduling with first-order cellular automata. This accomplished a simplification of symmetric ciphers in an adaptable and extensible manner for future architecture or block-size needs. While not the fastest, in its core implementation in C++ without special processor instructions or hardware acceleration, the goal of this paper was satisfied.

10 Acknowledgements

Friends, family, employers, being raised on the internet, tough times, and those who put up with me.

References

A. H. Encinas A. M. d. Rey, J. L. P. Iglesias G. R. Sánchez and A. Q. Dios (2008). "Cryptographic Properties of Second-Order Memory Elementary Cellular Automata". In: *Third International Conference on Availability, Reliability and Security, 2008, pp. 741-745.* DOI: `10.1109/ARES.2008.114`.

Ascherman, Cornelius (2019). *Traveling Back in Time (in Conway's Game of Life).* URL: `https://pagedout.institute/download/PagedOut_002_beta2.pdf` (visited on 11/14/2025).

Dhar, Avinash et al. (1994). "Universal Cellular Automata and Class 4". In.

Ronald L. Rivest M.J.B. Robshaw, R. Sidney and Y.L. Yin (1998). "The RC6™Block Cipher". In: *First Advanced Encryption Standard (AES) Conference, USA.*

Saeb, Magdy (2009). "The Stone Cipher-192 (SC-192): A Metamorphic Cipher". In: *(IJCNS) International Journal of Computer and Network Security.*