A Brief Cryptanalysis of the Talos Encryption Scheme

Caleb Alexander

1 Problem Construction

The proposed algorithm can be formalized as below:

- Accept a plain-text bit string of length $n > 1$ and split the bit string into blocks of length 256. If needed pad the final block with suitably random noise.
- Use a given 32 bit key to generate two 256-bit keys; a transposition key T_0 and a shift key S_0
  - The key expansion algorithm is a Cellular Automata (CA) which uses an evolution rule that resembles the rule used in John Conway's "Game of Life".
- Obtain each key at the ith step, S_i and T_i, by applying the chosen evolution rule to the previous keys, S_{i-1} and T_{i-1}, 11 times.
- Take each block, B_i, and compute $E_i = B_i \oplus S_i$.
- Use a scrambling function V, which as its input takes E_i and T_i, on each E_i. The concatenation of the outputs of V on each E_i is our final cipher text, C.

This paper will consider the key generation algorithm, the CA, as the primary vector of attack. We do this mostly because we consider it to be the primary novel contribution of the proposed algorithm. Thus, we will discard the particulars of certain surrounding details.

1.1 Cellular Automata

In the context of this algorithm we can define a cellular automata by a pairing (B, r). We say that B is a set with $|B| = 256$, such that for every element $p \in B$, we have that $p \in GF(2)$. Visually, B is a 16×16 grid such that each grid position is considered a neighbor to all 8 surrounding positions and the edges of the grid are identified in the manner of a genus-1 torus.

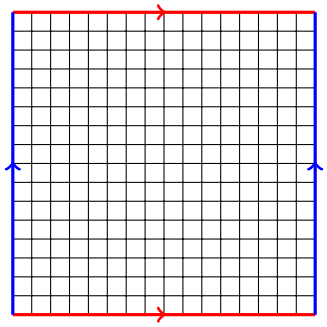


Figure 1: A 16×16 grid where sides of the same color are identified.

Generally we can consider r, which we call a rule, as a function $r : GF(2)^9 \to GF(2)$. In practice a rule decides the value of a given position for the next step of evolution. The rule for the proposed algorithm is:

1. If a cell is a '1' and has 2-4 neighboring '1' cells it stays a '1'; otherwise it becomes a '0'.
2. If a cell is a '0' and has 2-6 neighboring '1' cells it becomes a '1'; otherwise it becomes a '0'.

Notice that in this case we may simplify our idea of a rule to being a function $r : [9] \times GF(2) \to GF(2)$ (where $[9] = \{0, 1, 2, \ldots, 8\}$) since this particular rule depends only on the count of surrounding bits which are '1'.

See that any particular rule r_x, and particular board B_x, induces a function $R_x : GF(2)^{256} \to GF(2)^{256}$. This rule can most trivially be described as the application of r_x to each position individually:

$$R_x(B_x) = \big(r_x(p_1), r_x(p_2), \ldots, r_x(p_{256})\big).$$

Thus, our rules induce functions in the function space $\mathcal{F}_{GF(2)^{256}}$. Furthermore, the rules which induce a linear function in this function space are exactly those which will be representable by a 16×16 matrix. We will refer to the functions induced by a particular rule on a particular board as the "global rule".

2 Defining The Space

Before, evaluating specific security attributes of the algorithm; we ought to do some initial analysis on our hash function. The set of all possible states for a 16×16 grid based CA, is of course $S = GF(2)^{256}$ and has $|S| = |GF(2)^{256}| = 2^{256}$. However, the grid which Talos uses has 128 pre-set values (as shown below) and thus, not every 256-bit string is a valid start state for our CA. We will let I_S be the set of possible initial states. An easy mistake to make would be to assume that $|I_S| = 2^{128}$ since there are 128 bits which depend on our key. However, our key is only 32 bits. To fill in all 128 missing bits of our grid, Talos simply adds each bit of the chosen key multiple times. Thus $|I_S| = 2^{32} = |K|$ (where K is our keyspace). Below is given the initial state of our shift grid, where numbers in parentheses represent a corresponding index of our key. So (15) in our grid tells us that the value in that position will be equal to the 16th (we use zero-indexing) bit of our key.

$$\begin{bmatrix}
(15) & 1 & (14) & 1 & (13) & 1 & (12) & 1 & (11) & 1 & (10) & 1 & (9) & 1 & (8) & 1 \\
1 & (11) & 1 & (10) & 0 & (9) & 1 & (8) & 0 & (7) & 0 & (6) & 1 & (5) & 0 & (7) \\
(16) & 0 & (3) & 1 & (2) & 1 & (1) & 1 & (0) & 1 & (31) & 1 & (30) & 1 & (4) & 1 \\
1 & (12) & 0 & (23) & 1 & (22) & 0 & (21) & 0 & (20) & 0 & (19) & 0 & (29) & 1 & (6) \\
(17) & 0 & (4) & 0 & (7) & 1 & (6) & 0 & (5) & 1 & (4) & 0 & (18) & 1 & (3) & 0 \\
1 & (13) & 1 & (24) & 0 & (19) & 1 & (18) & 0 & (17) & 0 & (3) & 1 & (28) & 0 & (5) \\
(18) & 0 & (5) & 0 & (8) & 1 & (27) & 1 & (26) & 0 & (16) & 1 & (17) & 1 & (2) & 0 \\
1 & (14) & 0 & (25) & 1 & (20) & 0 & (31) & 1 & (25) & 1 & (2) & 0 & (27) & 1 & (4) \\
(19) & 1 & (6) & 1 & (9) & 0 & (28) & 0 & (30) & 1 & (15) & 0 & (16) & 0 & (1) & 1 \\
1 & (15) & 1 & (26) & 0 & (21) & 1 & (29) & 0 & (24) & 1 & (1) & 0 & (26) & 0 & (3) \\
(20) & 0 & (7) & 1 & (10) & 0 & (22) & 0 & (23) & 1 & (14) & 1 & (15) & 0 & (0) & 0 \\
1 & (16) & 0 & (27) & 1 & (11) & 0 & (12) & 0 & (13) & 0 & (0) & 1 & (25) & 0 & (2) \\
(21) & 0 & (8) & 0 & (28) & 1 & (29) & 0 & (30) & 1 & (31) & 0 & (14) & 1 & (31) & 0 \\
1 & (17) & 0 & (9) & 0 & (10) & 1 & (11) & 0 & (12) & 0 & (13) & 0 & (24) & 1 & (1) \\
(22) & 0 & (18) & 1 & (19) & 0 & (20) & 1 & (21) & 1 & (22) & 0 & (23) & 0 & (30) & 1 \\
1 & (23) & 0 & (24) & 0 & (25) & 0 & (26) & 1 & (27) & 0 & (28) & 0 & (29) & 0 & (0)
\end{bmatrix}$$

We define B_S to be the set containing all possible board states reachable, given our initial board constraints and our local rule. Formally it is,

$$B_S = \{b : b \in I_S \quad \vee \quad b = R_n(i_S)\}$$

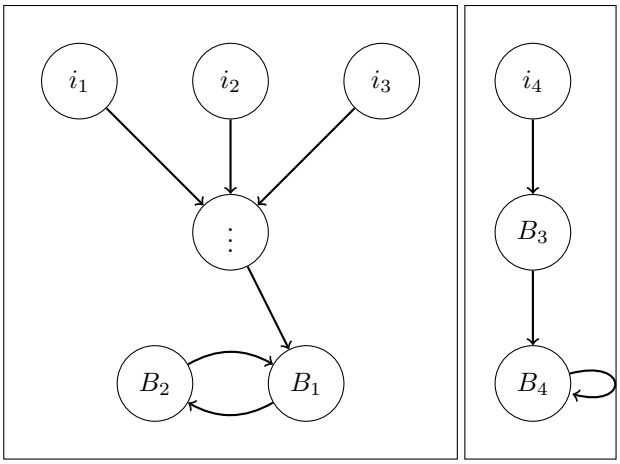
where $R_n(i_S)$ is n applications of our global rule to some initial board state.

2.1 Collision Classes

Next, we define so-called *collision classes* with respect to a rule. First see that,

$$E_n = \{(b_1, b_2) : b_1, b_2 \in I_S \quad \wedge \quad R_i(b_1) = R_j(b_2)\}$$

with $i, j \in \mathbb{N}$ and $0 \leq i, j \leq n$. In english, this relation considers two initial states to be related, under n evolutions, iff there is a collision between the two initial states after some number of evolutions less than n. Note that this does not mean that they will be equivalent after some exact number of evolutions, but just that they will have both touched the same board state prior to, or at, n evolutions. See that this relation is an equivalence relation and thus induces a partition on B_S, denoted as B_S/E_n, such that each part is an equivalence class. These equivalence classes are meaningful because our rule function is deterministic. If two board states collide at all, past that point they will collide forever. Indeed, the overall structure of our board state could be described as a graph.



See that in this graph, once boards collide, they will be trapped within some cycle. We can think of these cycles as the "end states" of our board under our rule and define another equivalence relation, E, which identifies elements with whichever cycle they converge. In the graph above, all board states in the left box would be identified with the cycle formed by B_1 and B_2. Everything in the right box would be identified with the cycle formed by B_4 and itself.

An easy cycle to identify is that of the grid of all 0s. Under our rule, a board of all 0s stays all 0s after an application of our rule. Thus this board forms a cycle unto itself. Furthermore, any board with exactly 1 or 2 1s will also be identified with the 0 board cycle, since regardless of where the 1(s) is/are placed; an application of our rule will always result in the 0-board.

Ideally, we would find all of the cycles which exist in our space and thereby substantially shrink the number of "unique" CA which can arise in this algorithm. However, in practice this is very hard to do since B_S is at least of size 2^{32}, and is very likely much larger.

Lastly, a question which may arise from the construction of our algorithm is "Are there any initial states which, after an application of our rule, become a different initial state?" in other words, does there exist $i_1, 1_2 \in I_S$ so that $R_1(i_1) = i_2$?

Claim 1. There does **not** exist $i_1, 1_2 \in I_S$ so that $R_1(i_1) = i_2$.

Proof. Consider the bit which is at row 7 and column 4 in our seeded matrix. It is the central bit as shown below:

$$\begin{bmatrix} 1 & (24) & 0 \\ (5) & 0 & (8) \\ 0 & (25) & 1 \end{bmatrix}.$$

Notice that regardless of our key, this bit will have 2-6 surrounding 1 bits. And thus, by our rule, it will always change from a 0 to a 1. This bit is a static bit of our initial states, meaning that for a board to be a

valid initial state, this bit must be 0. However, regardless of key, we have seen that this static bit will always change after an evolution. Hence, we cannot find an initial board which will become a different initial board after a single evolution. □

3 General Analysis

Below we examine Talos in a more general manner. None of the properties claimed will be proven; but any claims made should be clear.

3.1 Security Analysis

Talos has a generally good security analysis, but it is fundamentally limited by key size. Despite that limitation, the evolving nature of a CA allows for an effective expansion of the key space. In particular we can say that the space of possible expanded keys (our possible board states B_S) is

$$2^{32} < B_S \leq 2^{256}.$$

The tightest lower bound we could get for our key space is strictly greater than 2^{32}. But in all likelihood the expanded key space is most likely **substantially** larger than 2^{32}.

In regards to confusion, the 128 fixed bits of the initialization matrix allows for the chosen key bits to be effectively obfuscated. Diffusion is similarly enhanced by the high number of 1s which are fixed, and the manner in which they are spread. Additionally, the rule chosen favors the continued life of a bit (as shown in the theoretical evaluation section), thus increasing the likelihood that a bit will have the opportunity to affect another bit across time.

3.2 Efficiency Analysis

A common historical reason for the use of CAs for cryptography is they're massive potential for parallelization. Because each bits next state relies only on its neighbors, each bits next state can be computed in parallel. This property is clearly enhanced as we are more restrictive with neighbors (often referred to as the diameter). However, as we shrink the diameter of considered neighbors many cryptographic properties grow much weaker. For flat CAs [1] showed that a diameter of at least 3 was desirable and to achieve truly ideal cryptographic properties, the diameter should be equivalent to the size of our array. Of course, such a large diameter almost completely destroys the efficiency of a CA.

Talos benefits from being 2-dimensional and thus having a more complicated neighbor relationship. Despite this, Talos may still benefit from expanding its diameter

3.3 Key Management and Scalability

Two potential large concerns with Talos are as follows:

- We have no guarantee that a given initial state will not quickly fall into a short cycle. In the worst case scenario a CA may completely "die off" meaning that it has reached the 0-board and will stay there indefinitely. It should be clear that the 0-board has very bad cryptographic properties.
- The collision classes may be very large. If it is shown that the number of collision classes for our CA is small then in effect we are losing much of our keyspace. Once they have reached their end cycle, the only difference between two CAs in the same collision class is how many evolutions they have undergone. If you manage to align them under the rule, then you have effectively collapsed two separate keys into the same CA.

4 Theoretical Evaluation

Three important properties which we often desire from hash function families (taken from [2]) are:

- **Pre-Image Resistance**. We say that a hash function, $H : D \to R$, is pre-image resistant iff for some $y \in H(D)$ it is computationally *hard* to find an $x \in D$ so that $H(x) = y$.
- **2nd Pre-Image Resistance**. We say that H is 2nd pre-image resistant iff for any fixed $x \in D$, it is computationally *hard* to find another $x' \in D$, such that $H(x) = H(x')$.
- **Collision Resistance**. We consider H collision resistant iff it is computationally *hard* to find any $x, x' \in D$ so that $H(x) = H(x')$.

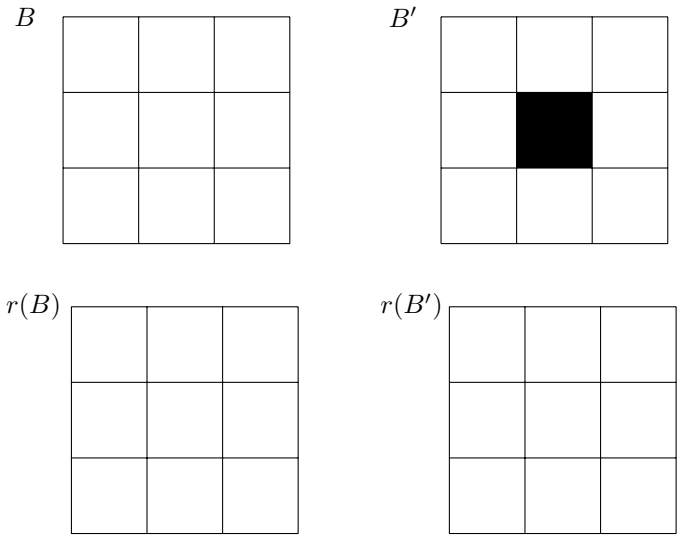
Note that Collision Resistance is strictly stronger than 2nd Pre-Image Resistance and 2nd Pre-Image Resistance is strictly stronger than Pre-Image Resistance so:

$$\textbf{Collision Resistance} \implies \textbf{2nd Pre-Image Resistance} \implies \textbf{Pre-Image Resistance}.$$

Historically, collision resistance has been a highly desired trait for a hash function. However, that is to say that the property is considered **necessary** but not always **sufficient** for a hash function to be usable. As noted by [3], at least three hash functions were proposed in [1] which were proven to be "collision free"(Similar to collision resistant) which were then shown to not be ideal for hashing purposes.

Claim 2. Talos is not collision resistant.

Proof. If we allow for any board (ignoring that some bits of Talos are fixed), then the proof becomes trivial. Choose B to be the trivial board, then choose B' to be any board with exactly one "on" bit. Notice that an application of the Talos rule will "kill" the on bit (since it must have no neighbors) and none of the off bits have any chance to become on without any neighbors.



If we include only boards which are valid CA, then the proof is more complex. Consider some board $R_n(i_1)$. We can proceed to construct the pre-image space as follows. Consider a single bit, b, of our board. There are 2^9 possible previous states for the bit and its neighbors. Of these 2^9 states we can partition them based on whether or not our considered bit is 0 or 1 and whether or not it stayed the same or changed.

if $b = 0$ and $b^{-1} = 0$ there are $\binom{8}{0} + \binom{8}{1} + \binom{8}{7} + \binom{8}{8}$ possible previous states.

if $b = 0$ and $b^{-1} = 1$ there are $\binom{8}{0} + \binom{8}{1} + \binom{8}{5} + \binom{8}{6} + \binom{8}{7} + \binom{8}{8}$ possible previous states.

if $b = 1$ and $b^{-1} = 0$ there are $\binom{8}{2} + \binom{8}{3} + \binom{8}{4} + \binom{8}{5} + \binom{8}{6}$ possible previous states.

if $b = 1$ and $b^{-1} = 1$ there are $\binom{8}{2} + \binom{8}{3} + \binom{8}{4}$ possible previous states.

Altogether, when considering a bit which is 0, there are

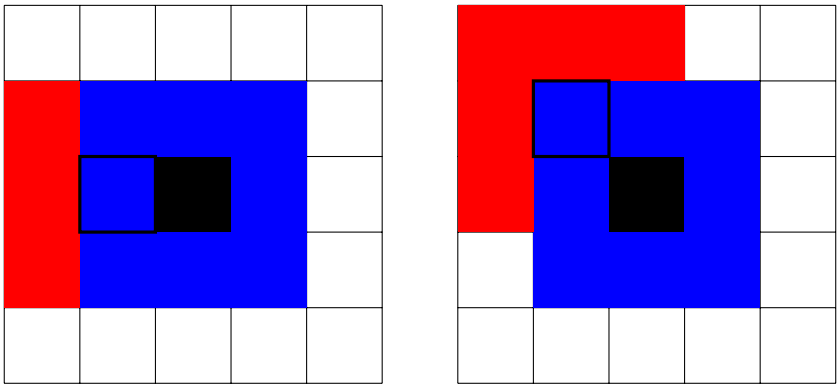
$$\begin{aligned}
&\binom{8}{0} + \binom{8}{1} + \binom{8}{7} + \binom{8}{8} + \binom{8}{0} + \binom{8}{1} + \binom{8}{5} + \binom{8}{6} + \binom{8}{7} + \binom{8}{8} \\
&\quad = 1 + 8 + 8 + 1 + 1 + 8 + 56 + 28 + 8 + 1 \\
&\quad = 120
\end{aligned}$$

possible previous states. For a bit which is 1, there are

$$\begin{aligned}
&\binom{8}{2} + \binom{8}{3} + \binom{8}{4} + \binom{8}{5} + \binom{8}{6} + \binom{8}{2} + \binom{8}{3} + \binom{8}{4} \\
&\quad = 28 + 56 + 70 + 56 + 28 + 28 + 56 + 70 \\
&\quad = 392.
\end{aligned}$$

possible previous states. Consider the space of possible previous states as a tree. We let the root of our tree be some 0 bit which has 120 branches which lead down. Then we have a choice; assuming that we next

examine the possible previous neighbors of a bit which was adjacent to our root bit, we can choose either a corner bit (in which case there are 2^5 previous states) or a corner bit (in which case there are 2^3 previous states).



In the above image we consider the black position to be the bit chosen as our root; the blue bits to be the bits which are given a value after one step; and the red bits to be those which are given a value based on which bit choose as our next center. If we consider b to be the branching factor after the first level, we can see that $b \leq 5$. Furthermore, the number of levels in our tree will be $L \approx 2 + \frac{247}{b}$. So $L \approx 83$ or $L \approx 50$. If we were to examine all possible leaves of the tree, we have to search through something like,

$$120 \times 5^{50} \text{ or } 120 \times 3^{83} \text{ possible states.}$$

But, many of these states will not be valid states of our CA because they may not be reachable via an initial state and our chosen rule. Furthermore, to prove that Talos is not collision resistant; we do not need to compute *all* possible previous states. But merely find a single valid previous state. Below is psuedo-code for such an algorithm.

Notice however, that this algorithm does not guarantee that the found previous state is one which is reachable by our CA. We could not find an algorithm which could verify that in any reasonable time (Though we do not claim that such an algorithm does not exist). In this way, we can claim that our CA is not collision resistant with respect to the chosen rule, however it *might* be with respect to its chosen rule and its valid board space B_S. □

5 Future Examinations

Due to time constraints and our relative inexperience in this field, we were unable to examine Talos in every way that we would have liked to. We see the following questions as good motivators for future research and analysis:

- Is there an efficient algorithm for deciding whether a given board state is a member of B_S?

- Is Talos 2nd pre-image resistant?
- Is Talos pre-image resistant?
- Is Talos universal, or approximately universal?
- Can we find families of rules such that they exhibit similar cryptographic properties?
- Can we gain further characterization of the collision classes induced by our rule, or a given rule?
- Are there any topological ways of characterizing our CA such as examining its simplicial homology groups or digital topology group(s)? (see [4])

Additionally, in the future we plan to investigate a number of these questions ourselves.

References

[1] Ivan Bjerre Damgård. A design principle for hash functions. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 416–427, New York, NY, 1990. Springer New York.

[2] Miodrag Mihaljević, Yuliang Zheng, and Hideki Imai. A cellular automaton based fast one-way hash function suitable for hardware implementation. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography*, pages 217–233, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[3] Joan Daemen, René Govaerts, and Joos Vandewalle. A framework for the design of one-way hash functions including cryptanalysis of damgård's one-way function based on a cellular automaton. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology — ASIACRYPT '91*, pages 82–96, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[4] Dae-Woong Lee and P. Christopher Staecker. Digital topological groups. *Topology and its Applications*, 338:108644, 2023.