International Journal of Engineering Trends and Technology
ISSN: 2231 – 5381 /doi:10.14445/22315381/IJETT-V69I2P206
Volume 69 Issue 2, 39-41, February 2021


# About a Fast Cryptographic Hash Function Using Cellular Automata Ruled by Far-Off Neighbours

Vincent Manuceau

*Makis Research, La Loupe, Eure-et-Loir, France*

vincent@manuceau.net

***Abstract*** *— This paper describes a 256 bits output cryptographic hashing algorithm based upon a specific kind of 2D cellular automata, adapted from Conway's Game of Life, where each cell is ruled by its far-off neighbors. The notion of Moore's far neighborhood of a cell on a 2D circular grid is defined, and more specifically, a far neighborhood of 8 cells is exploited to drive the cellular automaton. The obtained hash function is lightweight and fast by design, with a nice avalanche effect; it can be used as an effective and sustainable algorithm for Proof-of-Work blockchains, IoT, and many other applications.*

**Keywords** *— Cryptography, Hash function, Cellular automata, Moore's far neighborhood.*

## I. INTRODUCTION

Hash functions became omnipresent in many aspects of people's digital life [1][2], like data integrity, digital signatures, secure communications, or financial transactions processing.

A cryptographic hash function [4] transforms a message of variable length into a fixed-length digest in a deterministic and irreversible fashion. Additionally, two different messages can't have the same digest (collision-free), and tiny changes in the message lead to massive changes in the digest (avalanche effect).

This paper introduces a simple cryptographic hash function based upon specific cellular automata [5], defined as far neighbor-driven cellular automata. Each cell is ruled by its far-off neighbors.

## II. CELLULAR AUTOMATA

### A. Definitions

A cellular automaton is a dynamical system represented as a finite lattice network of state-changing cells, whose evolution is ruled by a transition function applied to each cell [6]. The next state of each cell is determined by its current state and the state of its neighborhood (Von Neumann's or Moore's neighborhood [7]). Conway's Game of Life [5] is a 2D cellular automaton, where each cell's state is either dead or alive and whose transition function is applied on Moore's neighborhood with 3 simple rules:

- an alive cell with 2 or 3 alive neighbors lives,
- a dead cell with exactly 3 live neighbors becomes alive,
- in any other case, the cell dies or remains dead.

In this particular automaton, each end meets as it is a circular state grid. Thus a 3D representation of the lattice can be figured as a toroidal shape. Conway's game of life is graded Class IV on Wolfram's classification scale [9], as it has complex behavior and produces patterns with local structures that move through space and time.

### B. Far-neighbourhood of a cell

Extending Moore's neighborhood definition on a $N\times N$ circular grid [7] $N\geq 3$, Moore's far-neighborhood of a cell C is defined as the set of cells belonging to the edges of a square of size $(N-(N+1) \bmod 2)\times(N-(N+1) \bmod 2)$ centered on C.

Thus, each of these cells is likely to belong to C's farthest cells, with a Manhattan distance [3] D comprised between $\lfloor N/2\rfloor-(N+1) \bmod 2$ and $2\times(\lfloor N/2\rfloor-(N+1) \bmod 2)$.

***a) Moore's 8 cells far-neighborhood:*** Let $N\in\mathbb{N}, N\geq 3$ *and let G a* $N\times N$ circular state grid. Let $n=\lfloor N/2\rfloor-(N+1) \bmod 2$ *and* $M=(0..N-1)$. Let $C\in G$ a cell of coordinates $(x,y)\in M\times M$ and (f,g) two functions defined as

$$f: M\to M / \forall\ x\in M, f(x)=(x-n+N) \bmod N$$

$$g: M\to M / \forall\ x\in M, g(x)=(x+n+N) \bmod N$$

Moore's 8-cells far neighbourhood of C is defined as the following 8 cells of coordinates: (f(x),f(y)), (x,f(y)), (g(x),f(y)), (f(x),y), (g(x),y), (f(x),g(y)), (x,g(y)) and (g(x),g(y)).

***b) Far neighbor cellular automata definition:*** An 8-cells far neighbor driven cellular automaton is defined as any cellular automaton that uses Moore's 8 cells far neighborhood in its transition function. More generally, a far neighbor-driven cellular automaton is defined as any cellular automaton that uses Moore's far neighborhood in its transition function.

## III. DESCRIPTION OF THE HASH FUNCTION

### A. Simple description

The hashing algorithm will use an 8-cells far neighbor-driven cellular automaton with a 16 x 16 circular state grid and a transition function based upon Conway's game of life rules. The state grid will be represented as one dimension 256 bits array.

After compressing and padding the input message to a 256 bits array, a 16 x 16 circular state grid is obtained, then the cellular automaton will be run for 7 rounds. If the resulting state grid is null (all cells died), the input message is recompressed and sent again to the hash function.

The resulting state grid will be considered as the 256 bits digest of the input message.

### *B. Input message compression and padding*

In order to process the message into a cell grid compatible with the CA, compression and padding to a 256 bits array are needed. A fairly simple compression and padding method are used, based upon xoring and bitwise rotations:

**Algorithm 1** Compression and padding of the message

```
function compress(message)
  bits← 0, comp ←Array(256), state ←Array(256)
  for i in message do
    n ← comp[i mod 256] + message[i]
    comp[i mod 256] ← n mod 2
    bits ←bits + message[i]
  end for
  for j in comp do
    comp[j] ← (comp[j] + j) mod 2
    state[(j + bits) mod 256] ← comp[j]
  end for
  return state
end function
```

The resulting output is 256 bits compressed and padded array that will now be used as a 16 x 16 circular grid, the initial state of the cellular automaton.

### *C. Processing far-neighbors of each cell*

For each cell of the CA, it is now necessary to determine the states of Moore's far neighborhood. To that end, the function below takes for argument the current state of the CA and processes the number of alive far-neighbors of each cell as follows:

**Algorithm 2** Determination of far-neighbors states

```
function neighbours(state)
  bits ← 0, ngb ← Array(256), Δ ←[-7; 0; 7]
  for i in state do
    x ← i / 16,  y ← i mod 16
    for (α,β)∈ Δ× Δ do
      if Δ[α]≠ 0 or Δ[β]≠ 0 then
        γ ← (x + Δ[α] + 16) mod 16
        δ ← (y + Δ[β] + 16) mod 16
        id ← 16× γ+δ
        ngb[id] ← ngb[id] + state[i]
      end if
    end for
    bits ← bits + state[i]
  end for
  return (ngb, bits)
end function
```

This algorithm returns the sum of alive neighbors for each cell and the bit sum of the current state.

### *D. Transition function and bitwise rotation*

The core part for each round is to apply the transition function to the current state. Conway's game of life rules will be applied and can be compacted as follows:

- any cell with 3 alive neighbors or any alive cell with 2 alive neighbors, lives on next state,
- in any other case, the cell dies or remains dead in the next state.

During this process, a bitwise rotation of the current state to the next state is applied in parallel. This transition function takes for argument the current state, the sum of alive neighbors for each cell (ngb), and the bit sum of the current state (bits).

**Algorithm 3** Transition function of the CA

```
function transition(state, ngb, bits)
  next ← Array(256)
  for i in state do
    if ngb[i]=3 or (state[i]=1 and ngb[i]=2) then
      next[(i + bits) mod 256] ← 1
    else
      next[(i + bits) mod 256] ← 0
    end if
  end for
  return next
end function
```

This transition function returns the next state of the cellular automaton by applying Conway's Game of Life rules on each cell.

### *E. Implementation of the hash function*

A last function is needed before the implementation of the proposed hash function. Let bitsum(state) a function that returns the bit sum of a given state:

**Algorithm 4** Bit sum processing of a state

```
function bitsum(state)
  bits ← 0
  for i in state do
    bits ← bits + state[i]
  end for
  return bits
end function
```

All the required functions are now defined to process the input message and to run the cellular automaton. The proposed hash function is implemented as follows:

**Algorithm 5** Implementation of the hash function

```
function hash(message)
  state ← compress(message)
  for round = 1 to 7 do
    (neighb, bits) ← neighbours(state)
    state ← transition(state, neighb, bits)
  end for
  if bitsum(state) = 0 then
    return hash(compress(message))
  end if
  return state
end function
```

## IV. RESULTS AND FURTHER DISCUSSION

This hash function has been implemented in OCaml, and preliminary tests were run in terms of randomness, avalanche effect, and throughput:

The average Hamming distance between a random 256 bits input and its corresponding hash is above 128. On a sample of two randomly sized inputs, differing by only one bit, the average Hamming distance between their corresponding hashes is above 90, which is a nice avalanche effect, but it does not meet the Strict Avalanche Criterion of Webster and Tavares [8], i.e., an average Hamming distance of 128, further work will be done to get closer and match this criterion.

Although the Ocaml implementation was made for a single thread, the proposed hash function performs with an average hash rate of 5.3 kHash/sec/GHz (18 kHash/sec in average measured on an Intel core i3-1005G1 CPU, single-core, single thread). Further work will also be done to implement a multi-threaded CPU version, a massively parallel CUDA version, and a distributed version using the IPFS Publish-Subscribe system.

For each cell, only 8 neighbors were taken into account to match Conway's game of life rules and to keep it fast and simple; in fact, it is possible to exploit Moore's far neighborhood principle up to $4\times(N-1-(N+1) \bmod 2)$ cells on a N x N circular grid, which makes up to 56 far neighbors of a cell in a 16 x 16 circular grid.

The number of rounds of our CA was arbitrarily set to 7 for speed considerations, and further analysis will also be done to determine the best-suited number of rounds on single-threaded, upcoming parallel, and distributed versions. Concerning this hash function, the impact of various compression methods to process the input message will also be analyzed.

## V. CONCLUSION

In this paper, a simple and fast cryptographic hash function using a specific cellular automaton exploiting the notion of Moore's far neighborhood was proposed and described. A definition was made for Moore's far neighborhood of a cell in a circular 2D grid as the set of cells belonging to the edges of the greatest fitting square centered on this cell. The obtained algorithm is elegant and lightweight with a nice avalanche effect, although not sufficient yet to meet the Strict Avalanche Criterion [8].

The fact of using far neighbours instead of close ones for the cellular automaton provides very interesting behavior and patterns of global structures moving through space and time that will be thoroughly described in further works.

A multi-threaded, massively parallel CUDA and an IPFS distributed version of this algorithm will also be further implemented.

The impact of adding more neighbors, rules, transition rounds, and different compression methods to this algorithm will also be investigated in terms of security analysis and performance.

## ACKNOWLEDGMENT

This paper is dedicated to Jerome Manuceau, Eric Bernard, Spiros Manuceau, and Giovanni Bouder. I would like to thank my close family, Nadine, Julien, Helene, Ianis, and Olivier, for all their help and encouragement during this research, whom without this would not have been possible. I also appreciated all the support I received from the rest of my family and friends, with special thanks to my friends Elkana Lesmond and Michel Pedurand. Lastly, I would like to thank the French Government for their social grants that allowed me to conduct this self-funded and independent research.

## REFERENCES

[1] J. Andress, The Basics of Information Security: Understanding the Fundamentals of InfoSec in Theory and Practice, 2nd ed., Syngress Publishing, (2014) ISBN 9780128008126.

[2] H. F. Atlamab and G. B. Willsa, Role of Blockchain Technology in IoT Applications. Advances in Computers, 115(2019) 1-39 Elsevier, doi : https://doi.org/10.1016/bs.adcom.2018.10.006.

[3] P.-E. Black, Manhattan distance in Dictionary of Algorithms and Data Structures, NIST, (2019), [Online], Available: https://www.nist.gov/dads/HTML/manhattanDistance.html.

[4] D. E. Knuth, The Art of Computer Programming: 3: Sorting and Searching, 115, Pearson Education, 527(1998) ISBN 9780321635785.

[5] J.-P. Rennard, Vie artificielle: Ou la biologie rencontre l'informatique, 1st ed., Paris, France: Vuibert Informatique, (2002) 63-82, ISBN 9782711786947.

[6] J. L. Schiff, Cellular Automata: A Discrete View of the World, Wiley, (2011) 70-80, ISBN 9781118030639.

[7] T. Toffoli and N. Margolus, Cellular Automata Machines: A New Environment for Modeling, Cambridge, MA, USA: MIT Press, (1987), ch. 7, ISBN 0262200600.

[8] A. F. Webster and S. E. Tavares, On the Design of S-Boxes, Lecture Notes in Computer Sciences; 218 on Advances in Cryptology, Santa Barbara, California, USA: Springer-Verlag, (1986) 523–534, ISBN 0387164634.

[9] S. Wolfram, A New Kind of Science, Wolfram Media, (2002) 231-249, ISBN 9781579550080.

[10] N Bhushana Babu D, E V Krishna Rao, K.S.N.Murthy Inter-Gateway Handoff Management Using Ant Colony Optimization (ACO) for Wireless Mesh Networks, International Journal of Engineering Trends and Technology 68(11) (2020) 63-71.