HCAHF: A New Family of CA-based Hash Functions

Anas Sadak[1], Fatima Ezzahra Ziani[2], Bouchra Echandouri[3], Charifa Hanin[4], Fouzia Omary[5]
Faculty of Science, Mohammed V University
Rabat, Morocco

***Abstract*—Cryptographic hash functions (CHF) represent a core cryptographic primitive. They have application in digital signature and message authentication protocols. Their main building block are Boolean functions. Those functions provide pseudo-randomness and sensitivity to the input. They also help prevent and lower the risk of attacks targeted at CHF. Cellular automata (CA) are a class of Boolean functions that exhibit good cryptographic properties and display a chaotic behavior. In this article, a new hash function based on CA is proposed. A description of the algorithm and the security measures to increase the robustness of the construction are presented. A security analysis against generic and dedicated attacks is included. It shows that the hashing algorithm has good security features and meet the security requirements of a good hashing scheme. The results of the tests and the properties of the CA used demonstrate the good statistical and cryptographic properties of the hash function.**

Keywords*—Hash function; boolean function; cellular automata; cryptography; information security; avalanche; nist statistical suite; DIEHARDER battery of tests; generic attacks; dedicated attacks

I. Introduction

Cryptographic hash functions are of great importance in cryptography and have a major role in modern communication. Most security applications and cryptographic protocols rely on them. They can be used within other schemes (digital signature or message authentication codes (MAC)) or as standalone primitives (password or key generation). They are generally used to ensure data integrity and to provide user authentication. The principle behind a hashing scheme is to compute a digest or hash value of fixed length starting from an arbitrary length message. The digest is considered as a compact identifier of the message [1].

Hashing schemes can be classified according to the nature of the compression function used. Three main categories of hash functions emerge from that classification: hash functions based on block ciphers, hash functions based on modular arithmetic and dedicated hash functions [2]. The hash function proposed in this article falls in this last category. It uses cellular automata in the process of producing the digest. Cellular automata are a class of dynamic systems that are simple in principle but produce chaotic and complex behaviors.

In this article, a new hash function based on cellular automata is presented. It was designed with security in mind. The algorithm described in this paper comprises three phases: a preprocessing phase, a processing phase and a transformation phase. Each of these phases contains elements that provide the algorithm with security measures that help prevent or lower the risk of cryptanalytic attacks against cryptographic hash functions.

The article is organized as follows: in Section 2, a background on cellular automata and hash functions is included. In Section 3, some related works are presented. Section 4 details the hashing scheme proposed. In Section 5, the result of the different statistical tests and the cryptographic properties of cellular automata are provided. Next, in Section 6 a security analysis is performed. Finally, Section 7 summarizes the article.

II. Background

A. Cryptographic Hash Functions (CHF)

Cryptographic hash functions with good security properties represent a significant part of cryptography. They are the basis of other cryptographic primitives and protocols. Their major tasks are to ensure data integrity and message authentication. Their major use is hence in digital signature schemes and message authentication protocols [3].

A hash function is a one-way function that maps an arbitrary finite length m-bits input to a fixed length n-bits output, called a hash value or a digest (m>n). The digest is thought of as the unique identifier of the input string.

In general, hash functions are built by iterating a compression function [4]. Depending on the nature of its internal compression function, the hash function can be categorized as either a hash function based on a block cipher, a hash function based on an arithmetic primitive or a dedicated hash function. In this article, the authors are interested in the latter category as the hash function described is based on cellular automata [5].

A hash function must ensure data compression and be easily computable. In addition, to be labeled as "cryptographic" it should guarantee the following basic security criteria [3]:

- Preimage: Given the hash value y = h(x), it is hard to find the message x' such that h(x') = y
- Second preimage: Given x, it is hard to find x' such that h(x') = h(x)
- Collision resistance: It is hard to find x and x' such that h(x) = h(x')

B. Cellular Automata (CA)

Cellular automata are dynamic mathematical models used for modeling physical and real-life phenomena. A cellular automaton is an array of cells arranged as a network that evolves in discrete time and space. Depending on the nature of the network arrangement, the cellular automaton can be one-dimensional or n-dimensional. Each cell assumes a state and evolves in time according to some local rule *f* and the states of

its neighbors. The states of all the cells at a given time represent the configuration of the CA. In an *m*-state, *k*-neighborhood cellular automaton, each cell can assume *m* states, the local rule depends on up to *k* neighbors and m^{2^k} rules are possible [6].

They were first studied by von Neumann [7] in the 1960s for self-reproducing systems. They were first used in cryptography by Wolfram [8]. He used elementary cellular automata (ECAs) which are one-dimensional, two-state, three-neighborhood CAs. In this article, the authors are only considering ECAs.

A cellular automaton is said to be uniform if the same rule *f* is applied to all the cells. Otherwise, if two or more rules are used alternately, then the cellular automaton is called non-uniform or hybrid.

The rules can be represented either by a Boolean function or by a truth table. Table I gives the Boolean expression and the truth table of elementary rule 30. In Table I, s_i^{t+1} is the next state of cell *i* and s_{i-1}^{t}, s_i^{t} and s_{i+1}^{t} are the states of cells *i-1*, *i* and *i+1* at time *t* respectively.

If the Boolean expression of the local rule(s) involves only the operation, the cellular automaton is called to be linear. Otherwise, it is called non-linear.

How the states of the leftmost and rightmost cells are chosen determines the boundary configuration of the CA. Cellular automata can have a fixed or periodic boundary configuration. In fixed boundary CAs, some fixed states are assigned. An example of fixed boundary is null boundary. In periodic boundary CAs, the boundary cells are neighbors of other boundary cells. For example, in the case of one-dimensional cellular automata, the rightmost and leftmost cells are neighbors [6].

Cellular automata attractiveness lies on their ability to display a complex global behavior from simple local computations and interaction and the possibility of parallel update of the states of the cells.

TABLE. I. RULE 30 BOOLEAN EXPRESSION AND TRUTH TABLE

Algebraic Normal Form							
$s_i^{t+1} = s_{i-1}^{t} \oplus (s_i^{t} + s_{i+1}^{t})$							
Truth Table							
111	110	101	100	011	010	001	000
0	0	0	1	1	1	1	0

III. RELATED WORK

Damgård [9] introduced a method for constructing a hash function that is collision resistant and provided three examples of possible use of his construction, one of which is based on cellular automata. Daemen, Govaerts, and Vandewalle [10] showed the vulnerabilities of the construction proposed by Damgård [9] and presented a framework for constructing practical hash functions. Along with their framework, a newly developed CA-based hash function was introduced: CellHash. The same authors proposed an enhanced version of CellHash called SubHash [11]. Both those constructions were broken by Chang [12]. Mihaljevic, Zheng, & Imai [13] presented a family of fast CA-based hash function over GF(q) without specifying the rules used and the neighborhood configuration used. A hash function based on a cellular automaton using both linear and non-linear rules is proposed by Jeon [14]. Kuila, Saha, Pal, and Chowdhury [15] promoted a hash construction based on cellular automata and inspired by the sponge construction. This construction proved to be as efficient and as secure against known attacks as other well-known hash functions [16] such as SPONGENT [17], GLUON [18], QUARK [19], PHOTON [20] and SHA-3 [21]. In this article, the construction used by the authors is inspired by the wide pipe Merkle-Damgård construction. Therefore, it is an alternative to the work proposed by Kuila et al. [15]. More recently, Hanin, Echandouri, Omary and El Bernoussi [22] proposed a CA-based hash function that uses the MD-construction. The hash algorithm proposed by Hanin et al. [22] lacks however security measures such as a strong padding scheme and the use of hybrid cellular automata and therefore do not have good cryptographic properties and a good pseudorandom behavior. The use of two-dimensional cellular automata for constructing a hash function was explored by Hirose and Yoshida [23]. However, the rule space for two-dimensional cellular automata and the complexity of this kind of cellular automata are beyond the scope of this article.

IV. DESCRIPTION OF HCAHF-256

In this section, the proposed hashing scheme is described. It takes inspiration from the wide pipe hash construction [24], which is a modified Merkle-Damgård construction. The general version with a digest size of *m* bits is named HCAHF while HCAHF-256 designates the 256-bit version.

The algorithm is made of three phases. First, the message is pre-processed. The pre-processed message is then compressed using cellular automata evolutions and the XOR function during the compression phase. Finally, a final transformation is applied before generating the output of HCAHF.

The three phases of HCAHF-256 are detailed in the following subsections. The pseudo-code for the padding scheme and for the digest generation mechanism is also presented.

A. Preprocessing Phase

1) Padding scheme: The first step in the preprocessing of a message M of arbitrary length consists of applying a padding scheme. Padding is always applied even in the case |M| is a multiple of 256.

The padding scheme used in HCAHF-256 is the technique known as Merkle-Damgård strengthening [25]. A single '1' bit is appended to *M*. It is followed by '0' bits and the size of *M* (before padding is applied) encoded in 64 bits. The necessary number of '0' bits must make the padded message size a multiple of 256.

Padding is used as a mean to avoid attacks such as the length-extension attack [3].

2) Message splitting: After the message *M* is padded, it is split into blocks of size 256 bits.

3) Salt: In the last step of the preprocessing phase, a salt value *salt* is computed using a pseudorandom number generator. It is then prepended to the padded message.

Using a salt value is a security measure that reduces the risk of collisions and prevents pre-computation attacks such as the dictionary attack as hashing the same message with different salt values $salt_1$ and $salt_2$ yields two different hash values [3].

The salt value is generated using the class ThreadedSeedGenerator of the Bouncy Castle Java library.

B. Compression Phase

During the compression phase, a compression function *f* is repeatedly used to produce a pre-digest value. *f* takes as inputs a chaining variable h_i from the previous step and message block M_i [16]. The initial chaining variable IV is a 256-bit block also generated using the class ThreadedSeedGenerator of the Bouncy Castle Java library. The final chaining variable $M_{compressed}$ is the pre-digest value.

Starting from (n+1) 256-bit blocks at the beginning of the compression phase, a single 256-bit block is obtained; hence the use of the compression term.

f combines a cellular automaton and the XOR function. The cellular automaton is designated as the function E and the XOR function as $\oplus$.

1) The Function E: The function *E* is a 3-neighborhood cellular automaton with periodic boundary conditions.

Each block M_i is evolved 128 times using a rule R_i as follows:

$$e_0 = E(salt, R_0)$$

$$e_1 = E(M_1, R_1)$$

$$e_n = E(M_n, R_n)$$

The rule R_i used is obtained from the first 8 bits of each block M_i. For example, if the first 8 bits of a block are 00011110, then the rule to be used is rule 30 ($00011110_2 = 30_{10}$). However, if R_i does not belong to the Wolfram classes 3 and 4 as classified in [26], it is discarded and a rule with good cryptographic properties (non-linearity, algebraic degree, balancedness, resiliency…) is selected at random from the set $R_{process}$. This set has been selected according to the recommendations and cryptographic properties found in [27] and [28]. For a detail of the rules included in this set and their cryptographic properties, the Appendix can be consulted.

2) The XOR Function: At each step, the result of the previous steps is XORed with the evolution of the block M_i.

3) The Function f: The compression function *f* of this construction can be expressed as:

$$h_0 = IV$$

$$h_1 = f(e_0, h_0)$$

$$h_i = f(e_{i-1}, h_{i-1})$$

$$h_{n+1} = f(e_n, h_n)$$

C. Transformation Phase

During the transformation phase, a function *T* is applied to $M_{compressed}$, the 256-bit output of the previous phase.

The function *T* is a 3-neighborhood hybrid cellular automaton with periodic boundary conditions. It evolves $M_{compressed}$ for 128 evolutions using the ruleset R_{digest} = {30, 90, 150, 30, 135, 30, 90, 150}. This ruleset has been chosen using the recommendation found in [27] and a detail of the selection process can be found in the Appendix.

The digest obtained can be expressed as:

$$digest = T(M_{compressed}, R_{digest})$$

Fig. 1 summarizes the different steps of HCAHF.

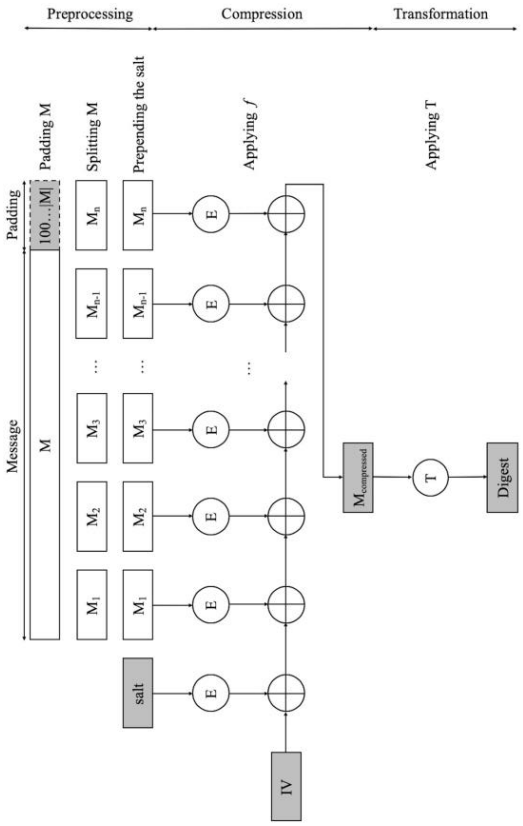

Fig. 1. HCAHF Design.

D. HCAHF-256 Algorithm

1) Pseudo-Code for The Padding Scheme of HCAHF-256

Algorithm 1 details the padding scheme used for HCAHF-256.

Algorithm 1. Pseudo-code of the padding scheme of HCAHF-256

Input: M the message to pad
Output: paddedM

```
begin
    m ← sizeOf(M) mod 256
    x ← 256 - m
    if sizeOf(M) is multiple of 256 then
        NbZero ← 191
    else if sizeOf(M) is not a multiple of 256 then
        if x < 65 then
            y ← x + 256
            NbZero ← y - 65
        else
            NbZero ← x - 65
        end if
    end if
    paddedM ← M || 1 || 0^NbZero||sizeOf(M) in {0,1}^64
    return paddedM
end
```

2) Pseudo-Code for The Generation Mechanism of HCAHF-256

Algorithm 2 shows the different steps by which a 256-bit digest is generated by HCAHF-256.

ALGORITHM 2. PSEUDO-CODE OF HCAHF-256

Input: M the message to hash
Output: digest

```
begin
    Let IV ← randomSequence(256)
    Let salt ← randomSequence(256)
    Let ruleSet← {30, 90, 150, 30, 135, 30, 90, 150}
    paddedM ← padding(M)
    splittedM ← split(paddedM, 256)
    saltedM ← insert (salt, splittedM)
    Xor ← IV
    For each block b in saltedM do
        If IntegerValue(8firstBits(b)) is in S the set of class
        3 and 4 rules then
            rules_i ←IntegerValue(8firstBits(b))
        Else
            rules_i ← randomRule(S)
        End if
        E←evolv(b, rules_i, 128, periodicBoundaries)
        Xor ← Xor ⊕ E
    End for
        s←Xor
    For it from 1 to 128
        For i from 1 to 256
            k ← i - 1 modulo sizeOf(ruleSet)
            If ((k == 0) || (k == 3) || (k == 5)) then
              evolution[i]←s[i-1] ⊕ (s[i]+s[i+1])
            Else if ((k == 1) ||( k == 6)) then
              evolution[i]←s[i-1] ⊕ s[i+1]
            Else if ((k == 2) || (k == 7)) then
              evolution[i] ← s[i-1] ⊕ s[i] ⊕ s[i+1]
            Else
              evolution[i] ←1⊕ s[i-1] ⊕ (s[i].s[i+1])
            End if
        End for
    End for
    digest ← evolution
    return digest
End
```

In this article, the authors chose to use $\frac{n}{2}$ or 128 evolutions in the case of HCAHF-256 for the number of evolutions when cellular automata are used. This is a general rule for guaranteeing a greater period of the cellular automaton [26].

V. RESULTS

In this section, the results of some statistical tests performed on the proposed hash algorithm are presented. Passing those tests does not guarantee the resistance of HCAHF to attacks targeted at hash functions. However, passing those tests is a good indicator of the good security level of a hash function. Pseudorandom behavior and statistical independence between the input and the output are some of the properties desired in cryptographic hash functions. In addition, the complexity of the algorithm is estimated to show the easy computation and implementation of the algorithm proposed.

A. Avalanche Test

The first test performed on HCAHF is the avalanche test. The avalanche effect was first used in cryptography by Feistel [29]. It was introduced as a property of S-boxes and Substitution-Permutation Networks (SPNs). It states that a very small difference in the input, generally a single bit change, produces a substantial difference on the output. More formally, if a function *f* exhibits the avalanche effect, then the Hamming distance between the outputs obtained from M and M', that differ by a single bit, is on average half the digest size. This definition is closely related to the concept of nonlinearity. A function displaying the avalanche effect can be considered as highly non-linear. Mathematically, this concept can be described as follows:

$f:\{0,1\}^m \to \{0,1\}^n$ has the avalanche effect if:

$$\forall M, M' \in \{0,1\}^m : Hamming(M, M') = 1$$

$$\Rightarrow average(Hamming(f(M), f(M')) = \frac{n}{2}$$

In order to conduct the avalanche test on HCAHF, 100 1024-bit messages were generated. For each message M_i, the hash value of the original message and the hash values of its one-bit change replicas

($Hamming(M_i, M'_{i,1\le j\le 1024}) = 1$) are generated by HCAHF. Then, the hamming distances between the hash values are calculated ($Hamming(f(M_i), f(M'_{i,1\le j\le 1024}))$).

The results of the test are presented in Fig. 2. Here the average values for each bit position *j* were taken for the 100 messages and expressed as percentages. The figure shows that the avalanche values are concentrated around 50. This indicates that HCAHF has the avalanche effect and thus its outputs are statistically independent from its inputs.

B. Statistical Tests

In practice, cryptographic hash functions should behave like a random oracle to prevent statistical attacks. Thus, to

check this pseudorandom behavior, standard statistical tests are commonly applied. The NIST Statistical Test Suite (STS) and the DIEHARDER battery of tests were used.

1) NIST Statistical Test Suite (STS): The National Institute of Standards and Technology developed a set of tests called the NIST Statistical Test Suite (STS). This test suite is useful to check the randomness property of some cryptographic primitives like hash functions. p-values are computed and evaluated for the generated sequences to verify if the sequences are random. A more detailed description can be found in NIST special publication 800-22 [30].

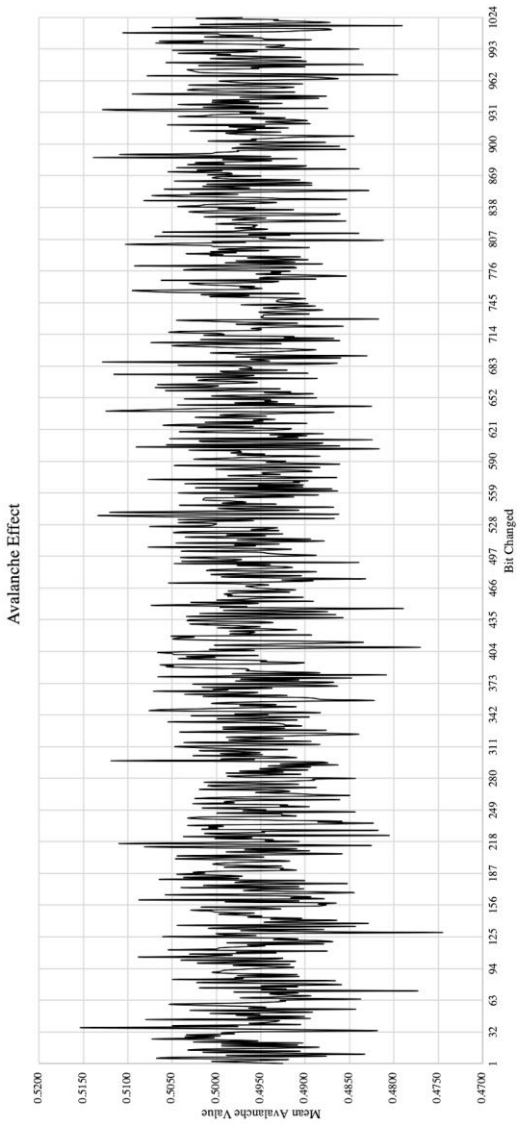

Fig. 2. Avalanche Test Results.

A 10 MB sequence is obtained by concatenating the hash values generated by HCAHF-256. This sequence is used as an input to the test suite. The results of the different tests are presented in Table II. A test is passed if the p-value is between 0.001 and 0.999. Some tests cannot be applied because the size of the output (256 bits) is too short compared to the test requirements (e.g. The Binary Matrix Rank Test requires a sequence of 1000000 bits). HCAHF-256 passes all the applicable tests.

2) Dieharder battery of tests: DIEHARDER is a stronger test battery developed by Brown, Eddelbuettel and Bauer [31] to test all types of random number generators as well as other cryptographic primitives (block ciphers, stream ciphers, hash functions, …). It is a collection of tests that include tests from the DIEHARD battery of tests and from the NIST STS as well as other tests designed by Brown, Bauer, Marsaglia and Tsang. It is an extensible test suite that includes more and more tests with each update. The p-values must be in the range [α ,1- α] to pass a test, α being the significance level. Table III shows the results of the DIEHARDER tests. The significance level used here is α = 0.005. HCAHF-256 passes all the tests.

The results of these two batteries of statistical tests show that HCAHF-256 has a good pseudorandom behavior, which is one of the essential characteristics of a secure hash function. The output generated by HCAHF-256 can then be considered as indistinguishable from the output of a true random number generator.

TABLE. II. NIST STS RESULTS

Test name	p-value	Interpretation
The Frequency (Monobit) Test	0.4963	PASS
Frequency Test within a Block	0.4888	PASS
The Runs Test	0.4927	PASS
Tests for the Longest-Run-of-Ones in a Block	0.4954	PASS
The Binary Matrix Rank Test	-	NOT APPLICABLE
The Discrete Fourier Transform (Spectral) Test	0.4815	PASS
The Non-Overlapping Template Matching Test	0.6973	PASS
The Overlapping Template Matching Test	-	NOT APPLICABLE
Maurer's "Universal Statistical" Test	-	NOT APPLICABLE
The Linear Complexity Test	0.5644	PASS
The Serial p-value1 Test	0.4915	PASS
The Serial p-value2 Test	0.4993	PASS
The Approximate Entropy Test	0.4914	PASS
The Cumulative Sums (Cusums) Forward Test	0.5140	PASS
The Cumulative Sums (Cusums) Reverse Test	0.5132	PASS
The Random Excursions Test	-	NOT APPLICABLE
The Random Excursions Variant Test	-	NOT APPLICABLE

TABLE. III. DIEHARDER TEST SUITE RESULTS

Test name	p-value	Interpretation
Diehard birthdays	0.8559	PASS
Diehard OPERM5	0.9309	PASS
Diehard 32x32 Binary Rank	0.3301	PASS
Diehard 6x8 Binary Rank	0.8151	PASS
Diehard_bitstream	0.7112	PASS
Diehard OPSO	0.9008	PASS
Diehard OQSO	0.1161	PASS
Diehard DNA	0.6635	PASS
Diehard Count the 1s (stream)	0.0808	PASS
Diehard Count the 1s (byte)	0.0082	PASS
Diehard Parking Lot	0.7936	PASS
Diehard Minimum Distance (2d Circle)	0.0254	PASS
Diehard 3d Sphere (Minimum Distance)	0.8868	PASS
Diehard Squeeze	0.9114	PASS
Diehard Sums	0.0138	PASS
Diehard Runs	0.4253	PASS
Diehard Craps	0.5868	PASS
Marsaglia and Tsang GCD	0.3477	PASS
STS Monobit	0.7161	PASS
STS Runs	0.2438	PASS
STS Serial Test (Generalized)	0.5342	PASS
RGB Bit Distribution	0.5511	PASS
RGB Generalized Minimum Distance	0.7152	PASS
RGB Permutations	0.5	PASS
RGB Lagged Sum	0.5330	PASS
RGB Kolmogorov-Smirnov	0.5784	PASS
DAB Byte Distribution	0.5964	PASS
DAB DCT (Frequency Analysis)	0.0541	PASS
DAB Fill Tree	0.7417	PASS
DAB Fill Tree 2	0.4559	PASS
DAB Monobit 2	0.0432	PASS

C. Cryptographic Properties of the Rules used in Function T

In addition to the avalanche effect displayed by HCAHF-256 and the good statistical features of HCAHF-256 proven by the results of the NIST STS and the DIEHARDER statistical tests, the cryptographic properties of the hybrid ruleset used in the transformation phase are presented here. The Appendix provides more clarifications about the selection process of this ruleset and the meaning of each property.

The following tables (Tables IV to VIII) present the cryptographic properties of the hybrid ruleset R_{digest} = {30, 90, 150, 30, 135, 30, 90, 150} for eight cells, assumed to be unknown Boolean values x_i, and up to three clock cycles.

From these tables, it can be noted that with the exception of correlation immunity and resiliency, all the other cryptographic properties (algebraic degree, nonlinearity and balancedness) increase with each iteration (algebraic degree, nonlinearity) or remain true (balancedness). The decrease of the values of correlation immunity and resiliency can be explained by the increase in the values of the algebraic degree and the nonlinearity, as some cryptographic properties are self-contradicting [27]. Therefore, a compromise should be found. For a cryptographic hash function, the algebraic degree and the nonlinearity can be judged as more important properties to achieve than the correlation immunity and resiliency properties.

D. Complexity

As stated before, basic requirements for a hash function is the ease of computation and the simplicity of implementation. In this subsection, the complexity of HCAHF is estimated in order to establish those two requirements.

TABLE. IV. ALGEBRAIC DEGREE

Iteration	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	2	1	1	2	2	2	1	1
2	3	2	2	3	3	3	2	2
3	4	3	3	4	5	4	3	3

TABLE. V. NONLINEARITY

Iteration	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	2	0	0	2	2	2	0	0
2	8	8	8	8	4	8	8	8
3	32	48	48	48	16	36	48	48

TABLE. VI. CORRELATION IMMUNITY

Iteration	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	1	2	0	0	0	1	2
2	0	2	2	0	0	0	2	2
3	0	0	0	0	0	0	1	1

TABLE. VII. RESILIENCY

Iteration	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	1	2	0	0	0	1	2
2	0	2	2	0	0	0	2	2
3	0	0	0	0	0	0	1	1

TABLE. VIII. BALANCEDNESS

Iteration	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	✓	✓	✓	✓	✓	✓	✓	✓
2	✓	✓	✓	✓	✓	✓	✓	✓
3	✓	✓	✓	✓	✓	✓	✓	✓

The padding requires at most *n+64* steps, where *n* is the block size. Splitting the message requires $m=\frac{L}{n}$ steps, where *L* is the message size after padding. The complexity of generating the salt and the *IV* is *O(n)*. Applying the function *E* to blocks involves $(m+1) \times n \times \frac{n}{2}$ steps. The compression phase requires $(m+1) \times n \bmod 2$ additions. The last transformation T requires $n \times \frac{n}{2}$ steps. The overall complexity of HCAHF is then $O(m \times n^2)$, *n* being the digest size and *m* the number of blocks.

VI. Security Analysis

In this section, a security analysis of the proposed hash algorithm HCAHF is performed. Formally proving the resistance of a given cryptographic primitive to attacks targeted against it is not an easy task. However, since the HCAHF is a wide-pipe MD construction, it can be assumed that its security can be reduced to that of its compression (E and XOR) and transformation (T) functions.

A. Brute Force Attacks against Hash Functions

Brute force attacks targeted at hash functions are attacks that depend only on the length of the hash value and not on the hash algorithm itself. In the case of keyless hash functions, three brute force attacks are possible: the preimage attack, the second preimage attack and the collision attack.

1) Preimage and second preimage attack: The number of trials required for an adversary to find the original message *M* from a given hash value *h* (preimage attack) or to find a second message *M'* given a pair (*M*,*h*) (second preimage attack) is 2^n [16], where n is the digest size in bits. In the case of HCAHF-256, the minimum amount of work required for such attacks to be successful is 2^{256} operations. Therefore, HCAHF-256 can be considered as robust against those two attacks.

2) Collision attack: The number of trials required for an adversary to find two messages *M* and *M'* that produce the same hash value *h* (collision attack) is $2^{\frac{n}{2}}$ [16]. In the case of HCAHF-256, the minimum amount of work required for an attacker to find a collision is 2^{128} operations. Thus, HCAHF-256 can be considered as robust against the collision attack.

B. Cryptanalytic Attacks

Cryptanalytic attacks are attacks targeted at the hash algorithm itself. The goal of these attacks is to reduce the complexity of the algorithm and thus reduce the complexity of the brute force attacks. Many different types of cryptanalytic attacks exist. Some examples of cryptanalytic attacks are the length-extension attack [32], the fixed-point attacks by Dean [33] and Kelsey and Schneier [34] and the herding attack [35].

As it is not possible to prevent all kinds of cryptanalytic attacks, the authors chose different techniques to prevent or lower the risk of these attacks. The following paragraphs detail those tools.

The first measure to prevent cryptanalytic attacks is the adoption of MD-strengthening as a padding scheme when preprocessing the message. MD-strengthening was used in HCAHF as a measure to prevent length-extension attacks.

Another measure adopted to avoid cryptanalytic attacks is the use of a salt value. The use of a generated salt value prepended to the message after applying padding helps in preventing attacks [3] such as the length-extension attack [32], the multi-collision attack, the fix point attacks by Dean [33] and by Kelsey and Schneier [34] and the herding attack [35].

The use of cellular automata with linear and non-linear rules (E function), of the bit-by-bit addition modulo 2 (XOR function) and of a non-linear cellular automaton (T function) during the compression and transformation phases of HCAHF also help in preventing cryptanalytic attacks. Differential and linear attacks are avoided by the use of cellular automata. The diffusion property is provided by the XOR operation and the use of cellular automata and was proved by the good avalanche effect of HCAHF. The confusion property is provided by the use of the non-linear cellular automaton (T function) and the non-linear rules cellular automata (E function). It was proved by the good results of the statistical tests performed on HCAHF. Moreover, the cryptographic properties of cellular automata presented in the Appendix point also in that direction.

VII. Conclusion

In this paper, a new family of CA-based hash functions is proposed. The hash algorithm described in the article consists of three phases: a preprocessing phase, a processing phase and a transformation phase. In the preprocessing phase, the input is padded using the MD-strengthening padding scheme, split into blocks of the same size *n* and prepended with a salt value. Applying a padding scheme and prepending a salt value to the input are measures used to prevent some cryptanalytic attacks targeted at cryptographic hash functions. During the processing phase, each block is first evolved using a cellular automaton then XORed with a chaining variable from a previous step. The first chaining variable is a precomputed *n*-bit block called *IV*. The use of cellular automata and the XOR operator during this phase provide the confusion and diffusion properties. The last phase consists of a final transformation by means of a hybrid cellular automaton with a ruleset carefully chosen. The use of the hybrid cellular automaton provides the confusion property and the pseudorandom behavior. In addition to the description of the algorithm and the security measures adopted in design of HCAHF, several statistical tests were performed. The result of these tests proves that HCAHF has good statistical features. These tests show that HCAHF displays the pseudorandom behavior, the statistical independence between the input and the output as well as the sensitivity of the algorithm to changes in the input. Finally, it can be noted that HCAHF has been implemented in software and that it can be presumed to perform better if implemented in hardware due to the simplicity of implementing cellular automata in hardware. Also, two additional properties, the hiding property and the puzzle friendliness property [36], can be verified if HCAHF is to be used in the blockchain technology in a future work. Finally, the number of evolutions used for cellular automata can be the subject of another study in order to maximize the period of the cellular automata used [26].

References

[1] D. R. Stinson and M. B. Paterson, Cryptography: theory and practice. Boca Raton: CRC Press, 2019.

[2] H. C. A. van Tilborg and S. Jajodia, Encyclopedia of cryptography and security. New York: Springer, 2011.

[3] E. Biham and O. Dunkelman, “A framework for iterative hash functions — HAIFA,” IACR Cryptology ePrint Archive 2007: 278, 2007.

[4] B. Preneel, "MACs and hash functions: State of the art," Information Security Technical Report, vol. 2, no. 2, pp. 33-43, 1997.

[5] B. Preneel, “The first 30 years of cryptographic hash functions and the NIST SHA-3 competition,” Topics in Cryptology - CT-RSA 2010 Lecture Notes in Computer Science, pp. 1–14, 2010.

[6] K. Bhattacharjee, N. Naskar, S. Roy, and S. Das, “A survey of cellular automata: types, dynamics, non-uniformity and applications,” Natural Computing, 2018.

[7] J. V. Neumann, Theory of self-reproducing automata. Urbana (U.S.A.): University of Illinois Press, 1966.

[8] S. Wolfram, “Cryptography with cellular automata,” Lecture Notes in Computer Science Advances in Cryptology — CRYPTO ’85 Proceedings, pp. 429–432, 1985.

[9] I. B. Damgård, “A design principle for hash functions,” Advances in Cryptology — CRYPTO’ 89 Proceedings Lecture Notes in Computer Science, pp. 416–427, 1989.

[10] J. Daemen, R. Govaerts, and J. Vandewalle, “A framework for the design of one-way hash functions including cryptanalysis of Damgård’s one-way function based on a cellular automaton,” Advances in Cryptology — ASIACRYPT 91 Lecture Notes in Computer Science, pp. 82–96, 1991.

[11] J. Daemen, R. Govaerts, and J. Vandewalle, “A hardware design model for cryptographic algorithms,” Computer Security — ESORICS 92 Lecture Notes in Computer Science, pp. 419–434, 1992.

[12] D. Chang, “Preimage attacks on CellHash, SubHash and strengthened versions of CellHash and SubHash,” IACR Cryptology ePrint Archive 2006: 412, 2006.

[13] M. Mihaljević, Y. Zheng, and H. Imai, “A cellular automaton based fast one-way hash function suitable for hardware implementation,” Public Key Cryptography Lecture Notes in Computer Science, pp. 217–233, 1998.

[14] J.-C. Jeon, “One-way hash function based on cellular automata,” IT Convergence and Security 2012 Lecture Notes in Electrical Engineering, pp. 21–28, Nov. 2012.

[15] S. Kuila, D. Saha, M. Pal, and D. R. Chowdhury, “CASH: Cellular automata based parameterized hash,” Security, Privacy, and Applied Cryptography Engineering Lecture Notes in Computer Science, pp. 59–75, 2014.

[16] W. Stallings, Cryptography and network security: principles and practice. Hoboken, NJ: Pearson Education, Inc., 2019.

[17] A. Bogdanov, M. Knežević, G. Leander, D. Toz, K. Varıcı, and I. Verbauwhede, “SPONGENT: A lightweight hash function,” Cryptographic Hardware and Embedded Systems – CHES 2011 Lecture Notes in Computer Science, pp. 312–325, 2011.

[18] T. P. Berger, J. D’Hayer, K. Marquet, M. Minier, and G. Thomas, “The GLUON Family: A lightweight hash function family based on FCSRs,” Progress in Cryptology - AFRICACRYPT 2012 Lecture Notes in Computer Science, pp. 306–323, 2012.

[19] J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia, “Quark: A lightweight hash,” Cryptographic Hardware and Embedded Systems, CHES 2010 Lecture Notes in Computer Science, pp. 1–15, 2010.

[20] J. Guo, T. Peyrin, and A. Poschmann, “The PHOTON family of lightweight hash functions,” Advances in Cryptology – CRYPTO 2011 Lecture Notes in Computer Science, pp. 222–239, 2011.

[21] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, “The making of KECCAK,” Cryptologia, vol. 38, no. 1, pp. 26–60, Feb. 2014.

[22] C. Hanin, B. Echandouri, F. Omary, and S. E. Bernoussi, “L-CAHASH: A novel lightweight hash function based on cellular automata for RFID,” Ubiquitous Networking Lecture Notes in Computer Science, pp. 287–298, 2017.

[23] S. Hirose and S.Yoshida, “A one-way hash function based on a two-dimensional cellular automaton,” The 20th Symposium on Information Theory and Its Applications (SITA97), pp. 213–216, 1997.

[24] S. Lucks, “A failure-friendly design principle for hash functions,” Lecture Notes in Computer Science Advances in Cryptology - ASIACRYPT 2005, pp. 474–494, 2005.

[25] X. Lai and J. L. Massey, “Hash functions based on block ciphers,” Advances in Cryptology — EUROCRYPT’ 92 Lecture Notes in Computer Science, pp. 55–70, 1992.

[26] S. Wolfram, A new kind of science. USA: Wolfram Media, 2002.

[27] K. Chakraborty and D. R. Chowdhury, “CSHR: Selection of cryptographically suitable hybrid cellular automata rule,” Lecture Notes in Computer Science Cellular Automata, pp. 591–600, 2012.

[28] S. Karmakar, D. Mukhopadhyay, and D. R. Chowdhury, “D-monomial tests of nonlinear cellular automata for cryptographic design,” Lecture Notes in Computer Science Cellular Automata, pp. 261–270, 2010.

[29] H. Feistel, “Cryptography and computer privacy,” Scientific American, vol. 228, no. 5, pp. 15–23, 1973.

[30] A. Rukhin, J. Sota, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo, “A statistical test suite for random and pseudorandom number generators for cryptographic applications,” Gaithersburg, MD: U.S. Dept. of Commerce, Technology Administration, National Institute of Standards and Technology, 2000.

[31] R. Brown, D. Eddelbuettel, and D. Bauer, “Dieharder: A random number test suite,” webhome.phy.duke.edu. [Online]. Available: https://webhome.phy.duke.edu/~rgb/General/dieharder/dieharder.abs. [Accessed: Nov-2019].

[32] D. Gligoroski, “Length Extension Attack on Narrow-Pipe SHA-3 Candidates,” Communications in Computer and Information Science ICT Innovations 2010, pp. 5–10, 2011.

[33] R. D. Dean, “Formal aspects of mobile code security,” (Unpublished doctoral dissertation). Princeton University, Princeton, NJ., 1999.

[34] J. Kelsey and B. Schneier, “Second preimages on n-bit hash functions for much less than 2 n work,” Lecture Notes in Computer Science Advances in Cryptology – EUROCRYPT 2005, pp. 474–490, 2005.

[35] J. Kelsey and T. Kohno, “Herding hash functions and the nostradamus attack,” Advances in Cryptology - EUROCRYPT 2006 Lecture Notes in Computer Science, pp. 183–200, 2006.

[36] M. Wang, M. Duan, and J. Zhu, “Research on the security criteria of hash functions in the blockchain,” Proceedings of the 2nd ACM Workshop on Blockchains, Cryptocurrencies, and Contracts - BCC 18, 2018.

[37] C. Carlet, “Boolean functions for cryptography and error-correcting codes,” Boolean Models and Methods in Mathematics, Computer Science, and Engineering, pp. 257–397, 2010.

APPENDIX

Cellular automata were used for the generation mechanism of HCAHF in both the processing phase (function E) and the transformation phase (function T) as they provide some interesting features desired in cryptographic primitives. The randomness property, the maximum period and the high nonlinearity are properties are among the properties provided by cellular automata that ate well suited for cryptographic hash functions.

In this Appendix, some definitions of cryptographic properties are given. In addition, the properties of some of the rules from Wolfram class 3 and 4 used for the function *E* are presented. Finally, the selection process of the rules of the ruleset R_{digest}, used for the function *T* of the transformation phase, is detailed.

A. Definitions

Before reporting on the properties of the rules used in the function *E* and describing the selection process of the ruleset R_{digest}, cryptographic properties are first defined [37].

1) Affine function: An affine function is a Boolean function where only the XOR operator is allowed.

2) Hamming weight: The Hamming weight of a Boolean function corresponds to the number of 1’s in a Boolean function’s truth table.

3) Balancedness (BAL): An *n* variables Boolean function *f* is said to be balanced if its Hamming weight is 2^{n-1}.

4) Hamming distance: The Hamming distance between two functions f_1 and f_2 is defined as the Hamming weight of $f_1 \oplus f_2$.

5) Nonlinearity (NL): The nonlinearity of an *n* variables Boolean function *f* is defined as the minimum Hamming distance between *f* and all *n* variables affine functions.

6) Algebraic Degree (AD): The number of variables of the highest order term of a Boolean function determines its algebraic degree.

7) Correlation Immunity (CI): An *n* variables function *f* has correlation immunity of order *k* if its values are statistically independent of any subset of *k* input variables.

8) Resiliency (RES): A Boolean function that is both balanced and has a correlation immunity of order k is said to be a *k*-resilient function.

B. Cryptographic Properties of Class 3 and 4 ECAs

In [25], Wolfram defined four classes of elementary cellular automata. Within these four classes, rules from classes 3 and 4 are assumed to display chaotic and complex behaviors. These behaviors are well suited for cryptographic primitives and thus for cryptographic hash functions. However, some of these rules can be discarded according to their cryptographic properties. Table IX shows all the rules from class 3 and 4 along with their cryptographic properties.

TABLE. IX. CRYPTOGRAPHIC PROPERTIES OF CLASS 3 AND CLASS 4 ECAS

Rule	NL	CI	AD	RES	BAL
18	2	0	2	-1	F
22	1	0	3	-1	F
30	2	0	2	0	T
41	1	0	3	-1	F
45	2	0	2	0	T
60	0	1	1	1	T
75	2	0	2	0	T
86	2	0	2	0	T
89	2	0	2	0	T
90	0	1	1	1	T
101	2	0	2	0	T
102	0	1	1	1	T
105	0	2	1	2	T
106	2	0	2	0	T
110	1	0	3	0	F
120	2	0	2	0	T
121	1	0	3	0	F
122	1	0	3	0	F
124	1	0	3	0	F
126	2	1	2	1	F
128	1	0	3	0	F
135	2	0	2	0	T
137	1	0	3	0	F
146	1	0	3	0	F
147	2	0	2	0	T
149	2	0	2	0	T
150	0	2	1	2	T
151	1	0	3	0	F
161	1	0	3	0	F
165	0	1	1	1	T
169	2	0	2	0	T
182	1	0	3	0	F
183	1	0	2	0	F
193	1	0	3	0	F
195	0	1	1	1	T
225	2	0	2	0	T

From Table IX, the following set has been selected to be used within the processing phase in function *E*:

$R_{process}$ = {18, 22, 30, 41, 45, 60, 75, 86, 89, 90, 101, 102, 105, 106, 110, 120, 121, 122, 124, 126, 128, 135, 146, 147, 149, 150, 151, 161, 165, 169, 182, 183, 193, 195, 225}

C. Selection Process of R_{digest}

A selection procedure for choosing the right ruleset to construct a strong hybrid cellular automaton is presented in [27]. Some linear and nonlinear rules that have good cryptographic properties are shown in Table X. The rules preselected in this table were taken from a preselection process mentioned in [27]. The selection procedure guidelines are summarized in Table XI. This guideline has been proposed in [27].

TABLE. X. CRYPTOGRAPHIC PROPERTIES OF RULES

Rule	AD	NL	BAL	CI
22	7	45	No	1
30	5	40	Yes	1
37	7	49	No	0
41	7	44	No	1
43	5	44	Yes	0
45	4	40	Yes	1
60	1	0	Yes	3
90	1	0	Yes	3
91	7	49	No	0
102	1	0	Yes	3
105	1	0	Yes	4
110	6	38	No	1
120	5	48	Yes	0
135	5	48	Yes	2
150	1	0	Yes	4
165	1	0	Yes	3
180	4	32	Yes	1
195	1	0	Yes	3
210	4	32	Yes	0

TABLE. XI. SELECTION PROCESS FOR THE RULESET

Input: Set L = {60,90,102,105,150,165,195} of linear rules and
set NL = {22,30,37,41,43,45,91,110,120,135,180,210} of nonlinear rules

Principle	Choice
For the first cell, pick a nonlinear rule with good cryptographic properties.	Rule 30 selected **[30]**
An equal number of linear and nonlinear rules should be selected for the remaining cells to find a compromise between the algebraic degree and the correlation immunity.	Rules 60, 90 and 135 selected **[30,60, 135,90]**
Two or three nonlinear rules should appear consecutively to increase the algebraic degree.	Rule 30 followed by rule 135 **[30,60,30,135,90]**
One nonlinear rule should be followed by more than one linear rule to increase the order of correlation immunity.	Rule 30 followed by rules 60 and 90 **[30,60,90,30,135,30,60,90]**
Rules with larger period should be chosen to increase the period of the hybrid CA.	Rules 90 and 150 selected **[30,90,150,30,135,30,90,150]**

OUTPUT: A RULESET WITH RULES HAVING GOOD CRYPTOGRAPHIC PROPERTIES [30,90,150,30,135,30,90,150].