LCAHASH-1.1: A New Design of the LCAHASH System for IoT

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

***Abstract*—The present paper represents an extension of LCAHASH system. LCAHASH is a previously developed lightweight hash algorithm. It is based on cellular automata. It was developed as an alternative to existing hash functions to ensure data integrity and to meet the security requirements of the Internet of Things devices. Due to the limited amount of storage and the limited computation capabilities of these devices, the algorithms used by these devices should be as efficient and as robust as possible. In this contribution, we propose an enhanced version of the original LCAHASH algorithm to improve its efficiency and its robustness. A description of the system proposed along with a security analysis and the results of a statistical battery of tests (Dieharder) are included. These results show that the system proposed exhibits good statistical and cryptographic features.**

Keywords—Information security; hash function; cellular automata; IoT; data integrity

I. Introduction

In the past few years, our societies have witnessed a rapid growth in technology due to the development of the Internet and the technological advances in electronics and software. Soon, every device we possess will be connected to the Internet. These devices range from toothbrushes to homes and include cameras, smartphones and other appliances.

These advances help to improve our daily lives. However, most of these connected devices suffer from a lack of security. This can be due to either a negligence of the manufacturer or the lack of adapted security standards for these constrained devices. Examples of these security breaches are man-in-the-middle attacks, denial of service (DoS), data forging or physical attacks. This explains the increasing need to develop lightweight cryptographic primitives adapted to the computing power and the storage capacity of these devices.

In this context, LCAHASH, a lightweight hash system based on cellular automata, was previously developed to ensure the integrity of RFID tags data. In this paper, we modified the original LCAHASH design to make it more robust, more efficient and more suited for the Internet of Things (IoT) devices [1].

This paper is organized as follows: In Section II, we present cellular automata. In Section III, we mention some related works. In Section IV, we describe the new design for LCAHASH. Afterwards, we present the experimental results in Section V. Finally, we conclude with the conclusion in Section VI.

II. Background on Cellular Automata

Cellular automata (CA) are dynamic systems made of cells that take a state from a defined set of states [2]. In the case of $S=\{0,1\}$, the CA is called a Boolean CA. Cellular automata evolve according to a local transition rule [2]. The local transition rules can be represented by a truth a table or a logical function defining the relation between the present cell's state and its next state in relation with its neighbor(s). The whole system evolves according to a global function. Depending on the number of neighbors, the boundary conditions, the rules applied and the ruleset, a cellular automaton can be one-dimensional or more, linear or chaotic, uniform or hybrid, etc.

Cellular automata are simple structures that yield a complex and unpredictable behavior. This feature makes them easy to implement both in hardware and software and makes them good candidates to use in the context of lightweight cryptography [2].

III. Related Works

We present here some other existing lightweight hash functions and some existing cellular automata based hash functions.

With the exception of DM-PRESENT [3], which is based on the Davies-Meyer construction, most of the existing lightweight hash function are based on a sponge construction [4] rather than a Merkle-Damgård construction. Using the sponge construction, a fixed length output or digest is obtained from an arbitrary length input. This is made possible by the use of fixed length permutation function. Examples of these lightweight hash functions are KECCAK [5] (winner of the SHA-3 competition), PHOTON [6], SPONGENT [7], QUARK [8], GLUON [9] and Hash-One [10].

In [11], Damgård proposed three methods to design collision resistant hash functions. Among these methods, one uses cellular automata as a building block. This method was later on attacked in [12]. Daemen et. al also proposed two hash functions based on cellular automata in [12] and [13]. Those constructions were shown to be vulnerable by Chang in [14]. A family of hash functions was proposed by Mihaljevic in [15]. However, in this proposition, no rules and no neighborhood configuration were specified. Newer constructions comprise [16], [17] and [18]. 2D cellular automata were used by Hirose and Yoshida in [19].

IV. Description of LCAHASH 1.1

In this section, we describe the new design of LCAHASH.

The LCAHASH 1.1 process starts with splitting the input M into blocks of 128 or 256 bits, depending on the version of the algorithm. Padding is applied to the last block M_n if necessary. In the next step, a block is chosen at random (M_{index}) and XORed with IV_1, an initial vector of size 128 or 256 bits generated randomly. Following this step, a 7 or 13 bits prime number N is generated at random and R_is are computed following:

$$R_i = M_i \bmod N \quad (1)$$

The R_is are then concatenated into M'. M' is in turn split into blocks of 128 or 256 bits and padding is applied to the last block M_k' if necessary. M_{evol} is then obtained as follows:

$$M_{evol} = IV_2 \oplus M_1' \oplus M_2' \ldots \oplus M_k' \quad (2)$$

where IV_2 an initial vector of size 128 or 256 bits generated randomly. Finally, using the hybrid cellular automaton with the rule set {30,90}, M_{evol} is evolved for 128 or 256 iterations depending on the version of the algorithm. The obtained sequence is the output or the digest of our hash function system.

Algorithm 1. LCAHASH 1.1 Algorithm

Input: M, IV_1, IV_2, *index*, N
Output: Digest

1 Split M into n blocks of 128 or 256 bits;
2 **If** M_n is not a multiple of 128 or 256 **then**
3 Pad M_n;
4 End If
5 $M_{index} \leftarrow M_{index} \oplus IV_1$
6 **For** i=1 to n **do**
7 $R_i \leftarrow M_i \bmod N$
8 End For
9 $M' \leftarrow R_1 \| R_2 \| \ldots \| R_n$
10 Split M' into k blocks of 128 or 256 bits;
11 **If** M_k' is not a multiple of 128 or 256 **then**
12 Pad M_k';
13 End If
14 $M_{evol} \leftarrow IV_2 \oplus M_1' \oplus M_2' \oplus \ldots \oplus M_k'$
15 $Digest \leftarrow Evol_{\{30,90\}}(M_{evol}$, 128 or 256 iterations)

Fig. 1 shows the different steps of LCAHASH 1.1.

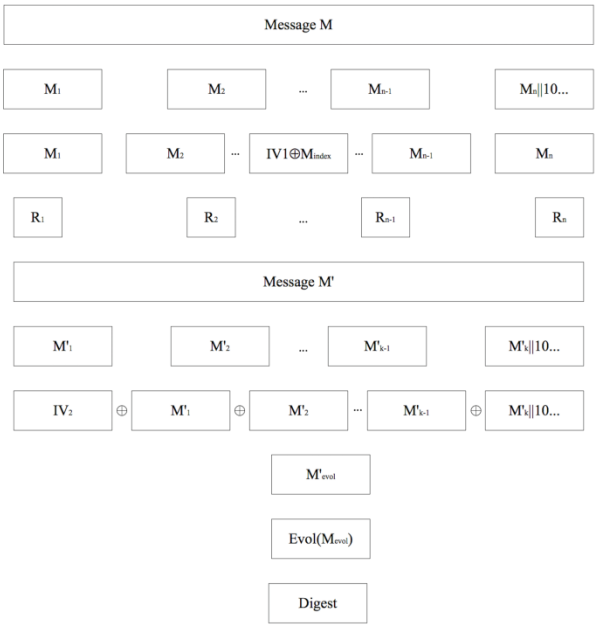

Fig. 1. LCAHASH 1.1 steps

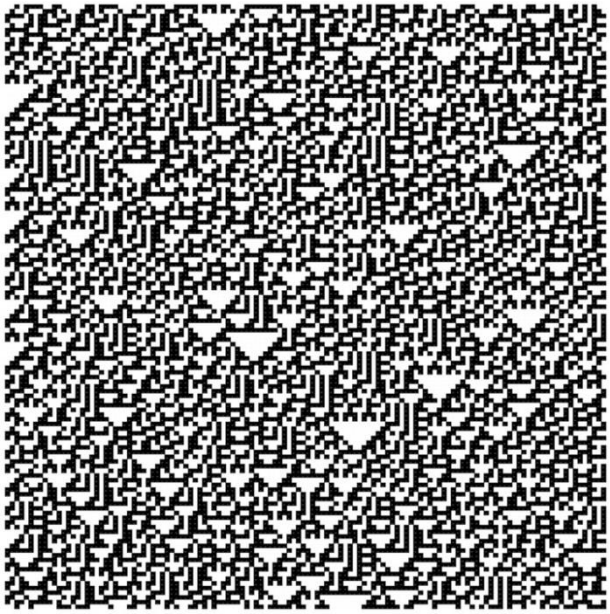

Fig. 2. Sample Evolution of M_{evol}.

Fig. 2 shows a sample evolution of M_{evol} using the non-uniform cellular automaton with ruleset {30,90}.

V. Security Analysis, Statistical Tests and Performance

Before proceeding to the security analysis of our modified design of LCAHASH, we consider first the major properties expected from a cryptographic hash function. These properties represent theoretical measures and are used in the field of cryptography to measure the level of security of hash functions and their ability to resist the most known cryptanalytic attacks.

Ideally, a cryptographic hash function is [20]:

- Deterministic: the same output (*h*) is always produced from the same input (*m*).
- Easy and fast to compute.
- Pre-image resistant: given an output *h*, it is computationally hard to find any input *m* such that $h = f(m)$. This property prevents an attacker who has a hash value *h* from finding the input *m*.
- Second pre-image resistant: given an input *m*, it is computationally hard to find another input *m'* such that $f(m) = f(m')$. This property prevents an attacker who has an input (*m*) and its corresponding output (*h*) from replacing the original input with another input (*m'*).
- Collision resistant: it is computationally hard to find two inputs *m* and *m'* that generate the same output such that $h = h'$. This property prevents an attacker to find two inputs with the same output. Note that a hash function which is collision resistant is also second pre-image resistant.

A. Security Analysis

1) Complexity: In general, a hash function should be easy and fast to compute. In order to evaluate this property, we try to approximate the complexity of our algorithm.

Splitting the message M into n blocks and applying padding requires at most $s - \frac{L}{s}$ steps (where *s* is the block size (128 or 256 bits) and *L* is the message length). XORing IV_1 and M_{index} requires *s* modulo 2 additions. Calculating the R_is requires $n \times s$ modulo 2 divisions. Splitting the message M' into *k* blocks and applying padding requires at most $s - \frac{L'}{s}$ steps (where *L'* is the length of the message *M'*). XORing M'_is with IV_2 requires *k* modulo 2 additions. Finally, evolving M_{evol} requires s^2 steps.

Overall, the complexity of our algorithm is $O(s^2)$.

2) Pre-image and second pre-image resistance: The security of the proposed system lies in the global transition rule of the cellular automaton as IV_1, IV_2, *index* and *N* are known parameters. Depending on the version of the algorithm, 128×2^{128} or 256×2^{256} operations are needed in order to find the global transition rule. Therefore, our proposed system is pre-image and second pre-image resistant.

3) Collision resistance: For each of the cellular automaton evolutions, a new state is obtained with $2^s - 1$ possible sequences. If we consider the birthday attack [21] , the complexity upper bound for breaking the collision resistance of our system is $O(2^{s/2})$. It means that depending on the version used, $2^{128/2}$ or $2^{256/2}$ operations are required to find a collision.

4) Avalanche effect: In cryptography, a function that displays the avalanche effect property is a function for which a small change in the input causes a greater change in the output [22]. Preferably, one bit changed in the input should change half of the bits in the output (strict avalanche criterion) [23].

To evaluate the avalanche effect property of our system, we took a sample of a hundred 1024 bits messages. For each message, we changed the original message bit per bit and calculated the Hamming distance between the hash value of the original sample message and the modified messages. We then calculated the average percentage values based on the hundred sample messages. We evaluated the avalanche effect using the 128-bit version of our system. The results of our setting are shown in Fig. 3 and Table I.

B. Statistical Tests

In order to test LCAHASH against statistical attacks, we used the DIEHARD test suite [24]. This test suite consists of a series of tests that evaluate the randomness of an algorithm. The benefit of this test suite is to show that the output of our algorithm is statistically indistinguishable from the output of a true random source and that predicting the output is computationally hard.

Table II shows the results of the DIEHARD test suite.

From the table below, we can see that LCAHASH 1.1 128-bit version passed all the tests of the DIEHARD test suite. LCAHASH 1.1 has then a good random behavior and its outputs are statistically indistinguishable from those generated by a true random function.

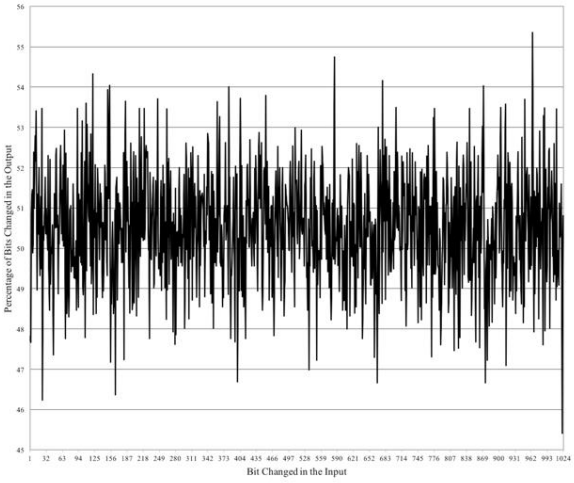

Fig. 3. Avalanche Effect for LCAHASH (128-Bit Version).

TABLE. I. Min, Max and mean Average Hamming Distances

Min	45.41%
Max	55.35%
Mean	50.51%

TABLE. II. DIEHARD TEST SUITE

Test	P-values	Interpretation
Diehard birthdays	0.39440811	Passed
Diehard operm5	0.53313565	Passed
Diehard_rank_32x32	0.57604853	Passed
Diehard_rank_6x8	0.72206931	Passed
Diehard_bitstream	0.94834578	Passed
Diehard_opso	0.13756532	Passed
Diehard_opso	0.92199512	Passed
Diehard_dna	0.63962068	Passed
Diehard_count_1s_str	0.49117533	Passed
Diehard count1s byt	0.93944197	Passed
Diehard parking lot	0.22547510	Passed
Diehard 2d sphere	0.64639589	Passed
Diehard 3d sphere	0.79712585	Passed
Diehard squeeze	0.83155292	Passed
Diehard sums	0.17939475	Passed
Diehard runs	0.79737984	Passed
Diehard craps	0.47732566	Passed
Marsaglia tsang gcd	0.85305736	Passed
Sts monobit	0.95959990	Passed
Sts_runs	0.64144586	Passed
Sts_serial	0.50158333	Passed
Rgb_bitdist	0.46256434	Passed
Rgb_minimum_distance	0.63795066	Passed
Rgb_permutations	0.59225124	Passed
Rgb_lagged_sum	0.49443699	Passed
Rgb_kstest_test	0.69428829	Passed
Dab_bytedistrib	0.56457052	Passed
Dab_dct	0.42650444	Passed
Dab_filltree	0.41658555	Passed
Dab_filltree2	0.54588999	Passed
Dab_monobit2	0.67956689	Passed

C. Performance

To measure the software implementation of LCAHASH 1.1, the Java source code was turned on an Intel Core i3-4010 32-bit processor clocked at 1.7 GHz with 4 Go of RAM. The results are presented in Table III. The performance of LCAHASH 1.1 is compared to LCAHASH 1.0 and other lightweight hash functions [9]. Those results show that LCAHASH 1.1 has a better performance than LCAHASH 1.0. Furthermore, LCAHASH has a satisfying performance when compared to other well established lightweight hash functions.

TABLE. III. SOFTWARE PERFORMANCE OF LCAHASH 1.1

Hash function	Output size (bit)	Cycle per byte (cpb)	Clock (GHz)
LCAHASH 1.0	128	324	1.7
	256	374	1.7
LCAHASH 1.1	128	995	1.7
	256	2844	1.7
GLUON	112	1951	2.66
U-QUARK	128	43373	2.66
D-QUARK	160	53103	2.66
S-QUARK	224	25142	2.66
PHOTON	80	1243	2.66

VI. CONCLUSIONS

In this paper, we presented a modified version of LCAHASH, a previously developed lightweight hash function. In our proposed design, we used a non-uniform cellular automaton with the rule set {30,90} to evolve the variable length input of our algorithm in order to generate a 128-bit or a 256-bit digest.

The new design preserves the security and statistical properties of the original design and has a better performance.

In future work, we should investigate non-uniform cellular automata more deeply and we should implement our algorithm in constrained devices to evaluate its hardware performance.

VII. CONFLICT OF INTEREST

The authors declare no conflict of interest.

REFERENCES

[1] 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.

[2] S. Wolfram, A New Kind of Science. Champaign, Ill: Wolfram Media, 2002.

[3] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An Ultra-Lightweight Block Cipher," Cryptographic Hardware and Embedded Systems - CHES 2007 Lecture Notes in Computer Science, pp. 450–466.

[4] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "On the Indifferentiability of the Sponge Construction," Advances in Cryptology – EUROCRYPT 2008 Lecture Notes in Computer Science, pp. 181–197, 2008.

[5] E. B. Kavun and T. Yalcin, "A Lightweight Implementation of Keccak Hash Function for Radio-Frequency Identification Applications," Radio Frequency Identification: Security and Privacy Issues Lecture Notes in Computer Science, pp. 258–269, 2010.

[6] 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.

[7] 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.

[8] J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia, "Quark: A Lightweight Hash," Journal of Cryptology, vol. 26, no. 2, pp. 313–339, Oct. 2012.

[9] 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.

[10] P. M. Mukundan, S. Manayankath, C. Srinivasan, and M. Sethumadhavan, "Hash-One: a lightweight cryptographic hash function," IET Information Security, vol. 10, no. 5, pp. 225–231, Jan. 2016.

[11] Damgård, I. B. (1989). A Design Principle for Hash Functions. Advances in Cryptology — CRYPTO' 89 Proceedings Lecture Notes in Computer Science, 416-427. doi:10.1007/0-387-34805-0_39.

[12] Daemen, J., Govaerts, R., & Vandewalle, J. (1991). 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,82-96. doi:10.1007/3-540-57332-1_7.

[13] Daemen, J., Govaerts, R., & Vandewalle, J. (1992). A Hardware Design Model for Cryptographic Algorithms. In: Deswarte Y., Eizenberg G., Quisquater JJ. (eds) Computer Security — ESORICS 92,648, 419-434. doi: https://doi.org/10.1007/BFb0013911.

[14] Chang, D. (2006). Preimage Attacks on CellHash, SubHash and Strengthened Versions of CellHash and SubHash (Vol. 2006, p. 412, Rep. No. 2006/412). IACR Cryptology ePrint Archive.

[15] Mihaljevic, M., Zheng, Y., & Imai, H. (1999). A Family of Fast Dedicated One-Way Hash Functions Based on Linear Cellular Automata Over GF (q). IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 40-47.

[16] Jeon, J. (2012). One-Way Hash Function Based on Cellular Automata. IT Convergence and Security 2012 Lecture Notes in Electrical Engineering, 21-28. doi:10.1007/978-94-007-5860-5_3.

[17] Kuila, S., Saha, D., Pal, M., & Chowdhury, D. R. (2014). CASH: Cellular Automata Based Parameterized Hash. Security, Privacy, and Applied Cryptography Engineering Lecture Notes in Computer Science, 59-75. doi:10.1007/978-3-319-12060-7_5.

[18] Hanin, C., Echandouri, B., Omary, F., & Bernoussi, S. E. (2017). L-CAHASH: A Novel Lightweight Hash Function Based on Cellular Automata for RFID. Ubiquitous Networking Lecture Notes in Computer Science, 287–298. doi: 10.1007/978-3-319-68179-5_25.

[19] Hirose, S., & Yoshida, S. (1997). A One-Way Hash Function Based on A Two-Dimensional Cellular Automaton. The 20th Symposium on Information Theory and Its Applications (SITA97), 213-216.

[20] P. Rogaway and T. Shrimpton, "Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance," Fast Software Encryption Lecture Notes in Computer Science, pp. 371–388, 2004.

[21] M. Bellare and T. Kohno, "Hash Function Balance and Its Impact on Birthday Attacks," Advances in Cryptology - EUROCRYPT 2004 Lecture Notes in Computer Science, pp. 401–418, 2004.

[22] A. F. Webster and S. E. Tavares, "On the Design of S-Boxes," Lecture Notes in Computer Science Advances in Cryptology — CRYPTO '85 Proceedings, pp. 523–534.

[23] R. Forrié, "The Strict Avalanche Criterion: Spectral Properties of Boolean Functions and an Extended Definition," Advances in Cryptology — CRYPTO' 88 Lecture Notes in Computer Science, pp. 450–468.

[24] G. Marsaglia, DIEHARD Statistical Tests: http://www. stat. fsu. edu/pub/diehard/.