Performance Analysis of a Keyed Hash Function based on Discrete and Chaotic Proven Iterations

Jacques M. Bahi, Jean-François Couchot, and Christophe Guyeux*
University of Franche-Comté, Computer Science Laboratory (LIFC)
Belfort, France
Email: {jacques.bahi, jean-francois.couchot, christophe.guyeux}@univ-fcomte.fr

***Abstract*—Security of information transmitted through the Internet is an international concern. This security is guaranteed by tools like hash functions. However, as security flaws have been recently identified in the current standard in this domain, new ways to hash digital media must be investigated. In this document an original keyed hash function is evaluated. It is based on chaotic iterations and thus possesses various topological properties as uniform repartition and sensibility to its initial condition. These properties make our hash function satisfy the requirements in this field. This claim is verified qualitatively and experimentally in this research work, among other things by realizing simulations of diffusion and confusion.**

***Keywords*-Keyed Hash Function; Internet Security; Mathematical Theory of Chaos; Topology.**

I. Introduction

Hash functions are fundamental tools to guarantee the quality and security of data exchanges through the Internet. For instance, they allow to store passwords in a secure manner or to check whether a download has occurred without any error. SHA-1 is probably the most widely used hash functions. It is present in a large panel of security applications and protocols through the Internet. However, in the last decade, security flaws have been detected in SHA-1. As the SHA-2 variants are algorithmically close to SHA-1 and produce finally message digests on principles similar to the MD4 and MD5 message digest algorithms, a new hash standard based on original approaches is then eagerly awaited. In this context, we have proposed a new hash function in [1]. Based on chaotic iterations, this function behaves completely different from approaches followed until now.

However chaos insertion to produce hash functions is sometimes disputed [2], [3]. Indeed existing chaos-based hash functions only include "somewhere" some chaotic functions of real variables like logistic, tent, or Arnold's cat maps. It is then supposed that the final hash function preserves these properties [4], [5], [6], [7]. But, in our opinion, this claim is not so evident. Moreover, even if these algorithms are themselves proven to be chaotic, their implementations on finite machines can result to lost of chaos property. Among other things, the main reason is that chaotic functions (embedded in these researches) only manipulate real numbers, which do not exist in a computer. In [1], the hash function we have proposed does not simply integrate chaotic maps into algorithms hoping that the result remains chaotic; we have conceived an algorithm and have mathematically proven that it is chaotic. To do both our theory and our implementation are based on finite integer domains and chaotic iterations.

* Authors in alphabetic order

Chaotic iterations (CIs) were formerly a way to formalize distributed algorithms through mathematical tools [8]. Thanks to these CIs, it was thus possible to study the convergence of synchronous or asynchronous programs over parallel, distributed, P2P, grid, or GPU platforms, in a view to solve linear and non-linear systems. CIs have recently revealed numerous interesting properties of disorder formalized into the mathematical topology framework. These studies lead to the conclusion that the chaos of CIs is very intense and that chaos class can tackle the computer science security field [9]. As CIs only manipulate binary digits or integers, we have shown that they are amenable to produce truly chaotic computer programs. Among other things, CIs have been applied to pseudo-random number generators [10] and to an information hiding scheme [11] in the previous sessions of the International Conference on Evolving Internet. In this paper, the complete unpredictable behavior of chaotic iterations is capitalized to produce a truly chaotic keyed hash function.

The remainder of this research work is organized in the following way. In Section II, basic recalls concerning chaotic iterations and Devaney's chaos are recalled. Our keyed hash function is presented, reformulated, and improved in Section III. Performance analyses are presented in the next two sections: in the first one a qualitative evaluation of this function is outlined, whereas in the second one it is evaluated experimentally. This research work ends by a conclusion section, where our contribution is summarized and intended future work is given.

II. Discrete and Chaotic Proven Iterations

This section gives some recalls on topological chaotic iterations. Let us firstly discuss about domain of iterated functions. As far as we know, no result rules that the chaotic behavior of a function that has been theoretically proven on $\mathbb{R}$ remains valid on the floating-point numbers, which is the implementation domain. Thus, to avoid loss of chaos this research work presents an alternative, namely to iterate boolean maps: results that are theoretically obtained in that domain are preserved during implementations.

Let us denote by $[\![a;b]\!]$ the following interval of integers: $\{a, a+1, \ldots, b\}$. A *system* under consideration iteratively modifies a collection of n components. Each component $i \in [\![1;n]\!]$ takes its value x_i among the domain $\mathbb{B} = \{0,1\}$. A *configuration* of the system at discrete time t (also said at *iteration* t) is the vector $x^t = (x_1^t, \ldots, x_n^t) \in \mathbb{B}^n$. In what follows, the dynamics of the system is particularized with the negation function $\neg : \mathbb{B}^n \to \mathbb{B}^n$ such that $\neg(x) = (\overline{x_i}, \ldots, \overline{x_n})$ where $\overline{x_i}$ is the negation of x_i.

In the sequel, the *strategy* $S = (S^t)^{t\in\mathbb{N}}$ is the sequence defining which component is updated at time t and S^t denotes its $t-$th term. We introduce the function $F_\neg$ that is defined for the negation function by:

$$\begin{array}{rcl} F_\neg : [\![1;n]\!] \times \mathbb{B}^n & \rightarrow & \mathbb{B}^n \\ F_\neg(s,x)_j & = & \left\{ \begin{array}{l} \overline{x_j} \text{ if } j = s \\ x_j \text{ otherwise.} \end{array} \right. \end{array}$$

With such a notation, configurations are defined for times $t = 0, 1, 2, \ldots$ by:

$$\left\{ \begin{array}{l} x^0 \in \mathbb{B}^n \text{ and} \\ x^{t+1} = F_\neg(S^t, x^t) \ . \end{array} \right. \quad (1)$$

Finally, iterations defined in (1), called "chaotic iterations" [8], can be described by the following system

$$\left\{ \begin{array}{rcl} X^0 & = & ((S^t)^{t\in\mathbb{N}}, x^0) \in [\![1;n]\!]^{\mathbb{N}} \times \mathbb{B}^n \\ X^{k+1} & = & G_\neg(X^k) \end{array} \right. , \quad (2)$$

such that

$$G_\neg\left(((S^t)^{t\in\mathbb{N}}, x)\right) = \left(\sigma((S^t)^{t\in\mathbb{N}}), F_\neg(S^0, x)\right)$$

where σ is the function that removes the first term of the strategy (*i.e.*, S^0). Let us remark that the term "chaotic" in the name of this tool is just an adjective, which has a priori no link with the mathematical theory of chaos.

In the space $\mathcal{X} = [\![1;n]\!]^{\mathbb{N}} \times \mathbb{B}^n$ we define the distance between two points $X = (S, E), Y = (\check{S}, \check{E}) \in \mathcal{X}$ by

$$\begin{array}{rcl} d(X,Y) & = & d_e(E,\check{E}) + d_s(S,\check{S}), \text{ where} \\ d_e(E,\check{E}) & = & \displaystyle\sum_{k=1}^{n} \delta(E_k, \check{E}_k), \text{ and} \\ d_s(S,\check{S}) & = & \displaystyle\frac{9}{n}\sum_{k=1}^{\infty} \frac{|S^k - \check{S}^k|}{10^k}. \end{array}$$

If the floor value $\lfloor d(X,Y) \rfloor$ is equal to j, then the systems $E, \check{E}$ differ in j cells. In addition, $d(X,Y) - \lfloor d(X,Y) \rfloor$ is a measure of the differences between strategies S and $\check{S}$. More precisely, this floating part is less than 10^{-k} if and only if the first k terms of the two strategies are equal. Moreover, if the k^{th} digit is nonzero, then the k^{th} terms of the two strategies are different.

With this material it has been already proven that [9]:

- $G_\neg$ is a continuous function on a suitable metric space $(\mathcal{X}, d)$,
- iterations as defined in Equ. 2 are regular (*i.e.*, periodic points of $G_\neg$ are dense in $\mathcal{X}$),
- $(\mathcal{X}, G_\neg)$ is topologically transitive (*i.e.*, for any pair of open sets $U, V \subset \mathcal{X}$, there exists some natural number $k > 0$ s. t. $G_\neg^k(U) \cap V \neq \varnothing$),
- $(\mathcal{X}, G_\neg)$ has sensitive dependence on initial conditions (*i.e.*, there exists $\delta > 0$ s.t. for any $X \in \mathcal{X}$ and any neighborhood V of X, there exist $Y \in V$ and $k \geqslant 0$ with $d(G_\neg^k(X), G_\neg^k(Y)) > \delta$).

To sum up, we have previously established that the three conditions for Devaney's chaos hold for chaotic iterations. So CIs behave chaotically, as it is defined in the mathematical theory of chaos [12], [13].

III. A Chaos-Based Keyed Hash Function

This section first recalls an informal definition [14], [15] of Secure Keyed One-Way Hash Function. We next present our algorithm. Finally, we establish relations between the algorithm properties inherited from topological results and requirements of Secure Keyed One-Way Hash Function.

A. Secure Keyed One-Way Hash Function

Definition 1 (Secure Keyed One-Way Hash Function) *Let Γ and Σ be two alphabets, let $k \in K$ be a key in a given key space, let l be a natural numbers which is the length of the output message, and let $h : K \times \Gamma^+ \rightarrow \Sigma^l$ be a function that associates a message in Σ^l for each pair of key, word in $K \times \Gamma^+$. The set of all functions h is partitioned into classes of functions $\{h_k : k \in K\}$ indexed by a key k and such that $h_k : \Gamma^+ \rightarrow \Sigma^l$ is defined by $h_k(m) = h(k,m)$ i.e., h_k generates a message digest of length l.*

A class $\{h_k : k \in K\}$ is a Secure Keyed One-Way Hash Function *if it satisfies the following properties:*

1) *the function h_k is keyed one-way. That is,*
   a) *Given k and m, it is easy to compute $h_k(m)$.*
   b) *Without knowledge of k, it is hard to find m when $h_k(m)$ is given and to find $h_k(m)$ when only m is given.*
2) *The function h_k is keyed collision free, that is, without the knowledge of k it is difficult to find two distinct messages m and m' s.t. $h_k(m) = h_k(m')$.*
3) *Images of function h_k has to be uniformly distributed in Σ^l in order to counter statistical attacks.*
4) *Length l of produced image has to be larger than* 128 *bits in order to counter birthday attacks.*
5) *Key space size has to be sufficiently large in order to counter exhaustive key search.*

Let us now present our hash function that is based on chaotic iterations as defined in Section II. The hash value message is obtained as the last configuration resulting from chaotic iterations of $G_\neg$.

We then have to define the pair $X^0 = ((S^t)^{t\in\mathbb{N}}, x^0)$, *i.e.*, the strategy and the initial configuration x^0.

B. Computing x^0

The first step of the algorithm is to transform the message in a normalized $n = 256$ bits sequence x^0. This size n of the digest can be changed, mutatis mutandis, if needed. Here, this first step is close to the pre-treatment of the SHA-1 hash function, but it can easily be replaced by any other compression method.

To illustrate this step, we take an example, our original text is: "*The original text*".

Each character of this string is replaced by its ASCII code (on 7 bits). Following the SHA-1 algorithm, first we append a "1" to this string, which is then

```
10101001 10100011 00101010 00001101 11111100
10110100 11100111 11010011 10111011 00001110
11000100 00011101 00110010 11111000 11101001.
```

Next we append the block 1111000, which is the binary value of this string length (120), and finally another "1" is added:

```
10101001 10100011 00101010 00001101 11111100
```

```
10110100 11100111 11010011 10111011 00001110
11000100 00011101 00110010 11111000 11101001
11110001.
```

The whole string is copied, but in the opposite direction:

```
10101001 10100011 00101010 00001101 11111100
10110100 11100111 11010011 10111011 00001110
11000100 00011101 00110010 11111000 11101001
11110001 00011111 00101110 00111110 10011001
01110000 01000110 11100001 10111011 10010111
11001110 01011010 01111111 01100000 10101001
10001011 0010101.
```

The string whose length is a multiple of 512 is obtained, by duplicating enough this string and truncating at the next multiple of 512. This string, in which the whole original text is contained, is denoted by D. Finally, we split our obtained string into blocks of 256 bits and apply to them the exclusive-or function, from the first two blocks to the last one. It results a 256 bits sequence, that is in our example:

```
11111010 11100101 01111110 00010110 00000101
11011101 00101000 01110100 11001101 00010011
01001100 00100111 01010111 00001001 00111010
00010011 00100001 01110010 01000011 10101011
10010000 11001011 00100010 11001100 10111000
01010010 11101110 10000001 10100001 11111010
10011101 01111101.
```

The configuration x^0 is the result of this pre-treatment and is a sequence of $n = 256$ bits. Notice that some distinct texts lead to the same string.

Let us build now the strategy $(S^t)^{t\in\mathbb{N}}$ that depends on both the original message and a given key.

C. Computing $(S^t)^{t\in\mathbb{N}}$

To obtain the strategy S, an intermediate sequence $(u^t)^{t\in\mathbb{N}}$ is constructed from D, as follows:

1) D is split into blocks of 8 bits. Let $(u^t)^{t\in\mathbb{N}}$ be the finite sequence where u^t is the decimal value of the t^{th} block.
2) A circular rotation of one bit to the left is applied to D (the first bit of D is put on the end of D). Then the new string is split into blocks of 8 bits another time. The decimal values of those blocks are added to (u^t).
3) This operation is repeated again 6 times.

Because of the function $\theta \longmapsto 2\theta \ (mod\ 1)$ is known to be chaotic in the sense of Devaney [12], we define the strategy $(S^t)^{t\in\mathbb{N}}$ with:

$$S^t = (u^t + 2 \times S^{t-1} + t) \mod 256,$$

which is then highly sensitive to initial conditions and then to changes of the original text. On the one hand, when a keyed hash function is desired, this sequence $(S^t)^{t\in\mathbb{N}}$ is initialized with the given key k (*i.e.*, $S^0 = k$). On the other hand, it is initialized to u^0 if the hash function is unkeyed.

D. Computing the digest

To construct the digest, chaotic iterations of $G_\neg$ are realized with initial state $X^0 = ((S^t)^{t\in\mathbb{N}}, x^0)$ as defined above. The result of these iterations is a $n = 256$ bits vector. Its components are taken 4 per 4 bits and translated into hexadecimal numbers, to obtain the hash value:

```
63A88CB6AF0B18E3BE828F9BDA4596A6
A13DFE38440AB9557DA1C0C6B1EDBDBD.
```

As a comparison if we replace "*The original text*" by "*the original text*", the hash function returns:

```
33E0DFB5BB1D88C924D2AF80B14FF5A7
B1A3DEF9D0E831194BD814C8A3B948B3.
```

We then investigate qualitative properties of this algorithm.

IV. Qualitative Analysis

We show in this section that, as a consequence of recalled theoretical results, this hash function tends to verify desired informal properties of a secure keyed one-way hash function.

A. The avalanche criteria

Let us first focus on the avalanche criteria, which means that a difference of one bit between two given medias has to lead to completely different digest. In our opinion, this criteria is implied by the topological properties of sensitive dependence to the initial conditions, expansivity, and Lyapunov exponent. These notions are recalled below.

First, a function f has a constant of expansivity equal to ε if an arbitrarily small error on any initial condition is *always* magnified till ε. In our iteration context and more formally, the function $G_\neg$ verifies the *expansivity* property if there exists some constant $\varepsilon > 0$ such that for any X and Y in $\mathcal{X}$, $X \neq Y$, we can find a $k \in \mathbb{N}$ s.t. $d(G_\neg^k(X), G_\neg^k(Y)) \geqslant \varepsilon$. We have proven in [16] that, $(\mathcal{X}, G_\neg)$ is an expansive chaotic system. Its constant of expansivity is equal to 1.

Next, some dynamical systems are highly sensitive to small fluctuations into their initial conditions. The constants of sensibility and expansivity have been historically defined to illustrate this fact. However, in some cases, these variations can become enormous, can grow in an exponential manner in a few iterations, and neither sensitivity nor expansivity are able to measure such a situation. This is why Alexander Lyapunov has proposed a new notion being able to evaluate the amplification speed of these fluctuations we now recall:

Definition 2 (Lyapunov Exponent) *Let be given an iterative system $x^0 \in \mathcal{X}$ and $x^{t+1} = f(x^t)$. Its* Lyapunov exponent *is defined by:*

$$\lim_{t\to+\infty} \frac{1}{t} \sum_{i=1}^{t} \ln \left| \ f' \left(x^{i-1}\right) \right|$$

By using a topological semi-conjugation between $\mathcal{X}$ and $\mathbb{R}$, we have proven in [9] that For almost all X^0, the Lyapunov exponent of chaotic iterations $G_\neg$ with X^0 as initial condition is equal to $\ln(n)$.

Let us now explain why the topological properties of our hash function lead to the avalanche effect. Due to the sensitive dependence to the initial condition, two close media can possibly lead to significantly different digests. The expansivity property implies that these similar medias mostly lead to very different hash values. Finally, a Lyapunov exponent greater than 1 lead to the fact that these two close media will always finish to have very different digests.

B. Preimage Resistance

Let us now discuss about the first preimage resistance of our unkeyed hash function denoted by h. Indeed, as recalled previously, an adversary given a target image D should not be able to find a preimage M such that $h(M) = D$. One reason (among many) why this property is important is that on most computer systems user passwords are stored as the cryptographic hash of the password instead of just the plaintext password. Thus an attacker who gains access to the password file cannot use it to then gain access to the system, unless it is able to invert target message digest of the hash function.

We now explain why, topologically speaking, our hash function is resistant to preimage attacks. Let m be the message to hash, (S, x^0) its normalized version (*i.e.*, the initial state of our chaotic iterations), and $M = h(m)$ the digest of m by using our method. So chaotic iterations with initial condition (S, M) and iterate function $G_\neg$ have x^0 as final state. Thus it is impossible to invert the hash process with a view to obtain the normalized message by using the digest. Such an attempt is equivalent to try to forecast the future evolution of chaotic iterations by only using a partial knowledge of its initial condition. Indeed, as M is known but not S, the attacker has an incertitude on the initial condition. he only knows that this value is into an open ball of radius 1 centered at the point M, and the number of terms of such a ball is infinite.

With such an incertitude on the initial condition, and due to the numerous chaos properties possessed by the chaotic iterations (as these stated in Section IV-A), this prediction is impossible. Furthermore, due to the transitivity property, it is possible to reach all of the normalized medias, when starting to iterate into this open ball. Indeed, it is possible to establish that, all of these possible normalized medias can be obtained in at most 256 iterations, and we iterate at least 519 times to obtain our hash value (*c.f.* Proposition 2 below). Finally, to find the normalized media does not imply the discovery of the original plain-text.

V. Quantitative and Experimental Evaluations

Let us first give some examples of hash values before discussing about the algorithm complexity.

A. Hash values

Let us now consider our hash function with $n = 128$. To give illustration of the confusion and diffusion properties, we will use this function to generate hash values in the following cases:

Case 1. The original text message is the poem *Ulalume* (E.A.Poe), which is constituted by 104 lines, 667 words, and 3754 characters.

Case 2. We change *serious* by *nervous* in the verse "*Our talk had been serious and sober*"

Case 3. We replace the last point '.' with a coma ','.

Case 4. In "*The skies they were ashen and sober*", skies becomes Skies.

Case 5. The new original text is the binary value of the Figure 1.

Case 6. We add 1 to the gray value of the pixel located in position (123,27).

Case 7. We substract 1 to the gray value of the pixel located in position (23,127).

The corresponding hash values in hexadecimal format are:

Case 1. C0EA2325BBF956D27C3561977E48B3E1,
Case 2. 6C4AC2F8579BCAB95BAD68468ED102D6,
Case 3. A538A76E6E38905DA0D35057F1DC1B14,

Figure 1: The original plain-image.

Case 4. 01530A057B6A994FBD3887AF240F849E,
Case 5. DE188603CFE139864092C7ABCD21AE50,
Case 6. FF855E5A626532A4AED99BACECC498B1,
Case 7. 65DB95737EFA994DF37C7A6F420E3D07.

These simulation results are coherent with the topological properties of sensitive dependence to the initial condition, expansivity, and Lyapunov exponent: any alteration in the message causes a substantial difference in the final hash value.

B. Algorithm Complexity

In this section is evaluated the complexity of the above hash function for a size l of the media (in bits).

Proposition 1 *The stages of initialization (Sections III-B and III-C) need $\mathcal{O}(l)$ elementary operations to be achieved.*

Proof: In this stage only linear operations over binary tables are achieved, such as: copy, circular shift, or inversion. ■

Let us consider the digest computation stage (Section III-D).

Proposition 2 *The digest computation stage requires less than $2l + 2\log_2(l+1) + 515$ elementary operations.*

Proof: The cost of an iteration is reduced to the negation operation on a bit, which is an elementary operation. Thus, the second stage is realized in t elementary operations, where t is the number of terms into the sequence S. But S has the same number of terms than u, and u and D have the same size (indeed, to build u, D has been copied 8 times, and bits of this sequence have been regrouped 8 per 8 to obtain the terms of u). To sum up, the size of D is equal to the total number of elementary operations of the digest computation stage.

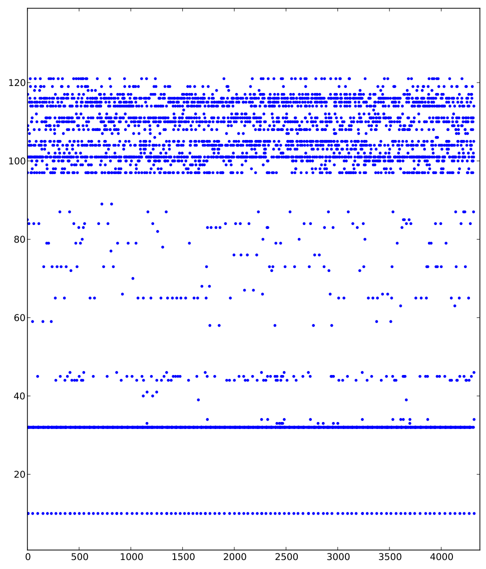
The following operations are realized to obtain D.

1) The digit 1 is added: D has $l + 1$ bits.
2) The binary value of the size is added, followed by another bit: D has $l + 2 + \log_2(l+1)$ bits.
3) This string is copied after inversion: D has now $2 \times (l + 2 + \log_2(l+1))$ bits.
4) Lastly, this string is copied until the next multiple of 512: in the worst situation, 511 bits have been added, so D has in the worst situation $2l + 2\log_2(l+1) + 515$ bits.

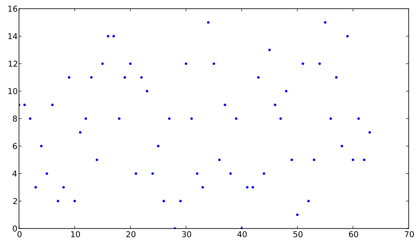
■

We can thus conclude that:

Theorem 1 *The computation of an hash value is linear with the hash function presented in this research work.*

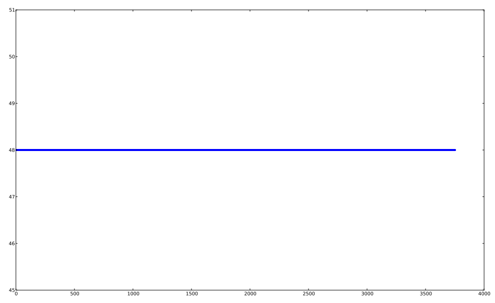


(a) Original text (ASCII)

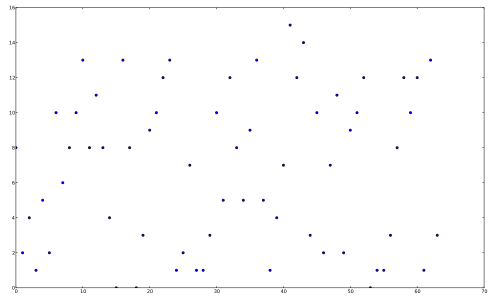


(b) Digest (Hexadecimal)

Figure 2: Values repartition of Ulalume poem



(a) Original text (ASCII)



(b) Digest (Hexadecimal)

Figure 3: Values repartition of the "*00000000*" message

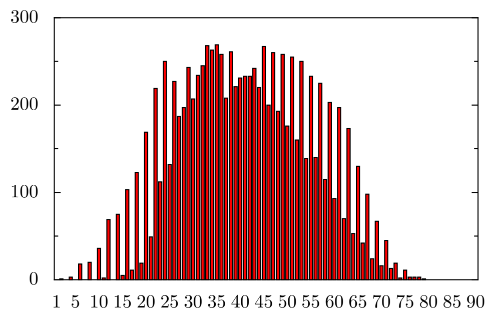


Figure 4: Histogram

C. *Experimental Evaluation*

We focus now on the illustration of the diffusion and confusion properties [17]. Let us recall that confusion refers to the desire to make the relationship between the key and the ciphertext as complex and involved as possible, whereas diffusion means that the redundancy in the statistics of the plaintext must be "dissipated" in the statistics of the ciphertext. Indeed, the avalanche criterion is a modern form of the diffusion, as this term means that the output bits should depend on the input bits in a very complex way.

1) Uniform repartition for hash values: To show the diffusion and confusion properties verified by our scheme, we first give an illustration of the difference of characters repartition between a plain-text and its hash value when the original message is again the Ulalume poem. In Figure 2a, the ASCII codes are localized within a small area, whereas in Figure 2b the hexadecimal numbers of the hash value are uniformly distributed.

A similar experiment has been realized with a message having the same size, but which is only constituted by the character "*0*". The contrast between the plain-text message and its digest are respectively presented in Figures 3a and 3b. Even under this very extreme condition, the distribution of the digest still remains uniform. To conclude, these simulations tend to indicate that no information concerning the original message can be found into its hash value, as it is recommended by the Shannon's diffusion and confusion.

2) Behavior through small random changes: We now consider the following experiment. A first message of 100 bits is randomly generated, and its hash value of size 80 bits is computed. Then one bit is randomly toggled into this message and the digest of the new message is obtained. These two hash values are compared by using the hamming distance, to compute the number B_i of changed bits. This test is reproduced 10000 times. The corresponding distribution of B_i is presented in Figure 4.

As desired, Figure 4 show that the distribution is centered around 40, which reinforces the confidence put into the good capabilities of diffusion and confusion of the proposed hash algorithm.

3) Statistic analysis of diffusion and confusion: Finally, we generate 1000 sequences of 1000 bits, and for each of these sequences, we toggle one bit, thus obtaining a sequence of 1000 couples of 1000 bits. As previously, the two digests of each couple i are obtained, and the hamming distance B_i

	B_{min}	B_{max}	$\overline{B}$	$P(\%)$	ΔB	$\Delta P(\%)$
N = 256	50	92	67.57	52.78	8.89	6.95
N = 512	47	82	65.13	51.11	7.65	5.87
N = 1024	47	81	63.01	52.10	7.51	5.71

Table I: Statistical performance of the proposed hash function

between these digests are computed. To analyse these results, the following common statistics are used.

- Mean changed bit number $\overline{B} = \frac{1}{N}\sum_{i=1}^{N} B_i$.
- Mean changed probability $P = \frac{\overline{B}}{128}$.
- $\Delta B = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(B_i - \overline{B})^2}$.
- $\Delta P = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(\frac{B_i}{128} - P)^2}$.

The obtained statistics are listed in Table I. Obviously, both the mean changed bit number $\overline{B}$ and the mean changed probability P are close to the ideal values (64 bits and 50%, respectively), which illustrates the diffusion and confusion capability of our algorithm. Lastly, as ΔB and ΔP are very small, these capabilities are very stable.

VI. Conclusion

MD5 and SHA-0 have been broken in 2004. An attack over SHA-1 has been achieved with only 2^{69} operations (CRYPTO-2005), that is, 2000 times faster than a brute force attack (that requires 2^{80} operations). Even if 2^{69} operations still remains impossible to realize on common computers, such a result based on a previous attack on SHA-0 is a very important one: it leads to the conclusion that SHA-2 is not as secure as it is required for the Internet applications. So new original hash functions must be found.

In this research work, a new hash function has been presented. The security in this case has been guaranteed by the unpredictability of the behavior of the proposed algorithms. The algorithms derived from our approach satisfy important properties of topological chaos such as sensitivity to initial conditions, uniform repartition (as a result of the transitivity), unpredictability, and expansivity. Moreover, its Lyapunov exponent can be as great as needed. The results expected in our study have been experimentally checked. The choices made in this first study are simple: compression function inspired by SHA-1, negation function for the iteration function, *etc.* The aim was not to find the best hash function, but to give simple illustrated examples to prove the feasibility in using the new kind of chaotic algorithms in computer science. Finally, we have shown how the mathematical framework of topological chaos offers interesting qualitative and qualitative tools to study the algorithms based on our approach.

In future work, we will investigate other choices of iteration functions and chaotic strategies. We will try to characterize topologically the diffusion and confusion capabilities. Other properties induced by topological chaos will be explored and their interest for the realization of hash functions will be deepened.

References

[1] J. M. Bahi and C. Guyeux, "Hash functions using chaotic iterations," *Journal of Algorithms & Computational Technology*, vol. 4, no. 2, pp. 167–181, 2010.

[2] C. song Zhou and T. lun Chen, "Extracting information masked by chaos and contaminated with noise: Some considerations on the security of communication approaches using chaos," *Physics Letters A*, vol. 234, no. 6, pp. 429 – 435, 1997.

[3] W. Guo, X. Wang, D. He, and Y. Cao, "Cryptanalysis on a parallel keyed hash function based on chaotic maps," *Physics Letters A*, vol. 373, no. 36, pp. 3201 – 3206, 2009.

[4] X. M. Wang, J. S. Zhang, and W. F. Zhang, "One-way hash function construction based on the extended chaotic maps switch," *Acta Phys. Sinici.*, vol. 52, No. 11, pp. 2737–2742, 2003.

[5] D. Xiao, X. Liao, and Y. Wang, "Improving the security of a parallel keyed hash function based on chaotic maps," *Physics Letters A*, vol. 373, no. 47, pp. 4346 – 4353, 2009.

[6] ——, "Parallel keyed hash function construction based on chaotic neural network," *Neurocomputing*, vol. 72, no. 10-12, pp. 2288 – 2296, 2009, lattice Computing and Natural Computing (JCIS 2007) / Neural Networks in Intelligent Systems Designn (ISDA 2007).

[7] D. Xiao, F. Y. Shih, and X. Liao, "A chaos-based hash function with both modification detection and localization capabilities," *Communications in Nonlinear Science and Numerical Simulation*, vol. 15, no. 9, pp. 2254 – 2261, 2010.

[8] D. Chazan and W. Miranker, "Chaotic relaxation," *Linear algebra and its applications*, pp. 199–222, 1969.

[9] C. Guyeux, "Le désordre des itérations chaotiques et leur utilité en sécurité informatique," Ph.D. dissertation, Université de Franche-Comté, 2010.

[10] Q. Wang, J. Bahi, C. Guyeux, and X. Fang, "Randomness quality of CI chaotic generators. application to internet security," in *INTERNET'2010. The 2nd Int. Conf. on Evolving Internet*. Valencia, Spain: IEEE Computer Society Press, Sep. 2010, pp. 125–130, best Paper award.

[11] C. Guyeux and J. Bahi, "An improved watermarking algorithm for internet applications," in *INTERNET'2010. The 2nd Int. Conf. on Evolving Internet*, Valencia, Spain, Sep. 2010, pp. 119–124.

[12] R. L. Devaney, *An Introduction to Chaotic Dynamical Systems*, 2nd ed. Redwood City, CA: Addison-Wesley, 1989.

[13] Knudsen, "Chaos without nonperiodicity," *Amer. Math. Monthly*, vol. 101, 1994.

[14] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk, "Keyed hash functions," in *Cryptography: Policy and Algorithms*, ser. Lecture Notes in Computer Science, E. Dawson and J. Golic, Eds. Springer Berlin / Heidelberg, 1996, vol. 1029, pp. 201–214.

[15] J. Zhang, X. Wang, and W. Zhang, "Chaotic keyed hash function based on feedforward-feedback nonlinear digital filter," *Physics Letters A*, vol. 362, pp. 439–448, 2007.

[16] C. Guyeux, N. Friot, and J. Bahi, "Chaotic iterations versus spread-spectrum: chaos and stego security," in *IIH-MSP'10, 6-th Int. Conf. on Intelligent Information Hiding and Multimedia Signal Processing*, Darmstadt, Germany, Oct. 2010, pp. 208–211.

[17] C. E. Shannon, "Communication theory of secrecy systems," *Bell Systems Technical Journal*, vol. 28, pp. 656–715, 1949.