2014 International Symposium on Nonlinear Theory and its Applications
NOLTA2014, Luzern, Switzerland, September 14-18, 2014

Introducing the truly chaotic finite state machines and theirs applications in security field

Christophe Guyeux[†], Qianxue Wang[‡], Xiaole Fang[§] and Jacques M. Bahi[†]

†Femto-St Institute, University of Franche-Comté, France
‡College of Automation, Guangdong University of Technology, China
§Land and Resources Technology Center of Guangdong Province, China
Email: christophe.guyeux@univ-fcomte.fr, wangqianxue@gdut.edu.cn, xiaole.fang@gmail.com, jacques.bahi@univ-fcomte.fr

Abstract—The truly chaotic finite machines introduced by authors in previous research papers are presented here. A state of the art in this discipline, encompassing all previous mathematical investigations, is provided, explaining how finite state machines can behave chaotically regarding the slight alteration of their inputs. This behavior is explained using Turing machines and formalized thanks to a special family of discrete dynamical systems called chaotic iterations. An illustrative example is finally given in the field of hash functions.

1. Introduction

The use of chaotic dynamics in cryptography is often disputed as a finite state machine is reputed to always enter into a cycle. Even though such a regular behavior is not completely opposed to almost all definitions of chaos in mathematics, constituting a kind of border situation in case of discrete sets, this situation appears as problematic to cryptologists that consider periodic dynamics and chaos as antithetical. This problem can be solved by introducing truly chaotic finite machines. Our proposal is to constitute them, that is, finite machines that can be rigorously proven as chaotic, as defined by Devaney [4], Knudsen [7], and so on. The key idea is to consider that the finite machine is not separated from the outside world but that it must interact with it in order to be useful. At each iteration, the new input provided to the finite machine can be used together with its current state to produce the next output. By doing so, the finite machine iterates on the finite cartesian product of its possible states multiply by all the possible inputs. This idea is formalized theoretically using Turing machines and explained practically thanks to the so-called chaotic iterations. An example of use is finally provided in the field of hash functions.

2. The So-called Chaotic Iterations

Our proposal in creating chaotic finite machines is to take a new input at each iteration. This process can be realized using a tool called chaotic iterations.

2.1. Review of Basics

In the whole document, to prevent from any conflicts and to avoid unreadable writings, we have considered the following notations, usually in use in discrete mathematics:

• The n−th term of the sequence s is denoted by s^n.
• The i−th component of vector v is v_i.
• The k−th composition of function f is denoted by f^k. Thus $f^k = f \circ f \circ \ldots \circ f$, k times.
• The derivative of f is f'.

$\mathcal{P}(X)$ is the set of subsets of X. On the other hand $\mathbb{B}$ stands for the set $\{0;1\}$ with its usual algebraic structure (Boolean addition, multiplication, and negation), while $\mathbb{N}$ and $\mathbb{R}$ are the usual notations of the following respective sets: natural numbers and real numbers. $\mathcal{X}^{\mathcal{Y}}$ is the set of applications from $\mathcal{Y}$ to $\mathcal{X}$, and thus $\mathcal{X}^{\mathbb{N}}$ means the set of sequences belonging in $\mathcal{X}$. We will use the notation $\lfloor x \rfloor$ for the integral part of a real x, that is, the greatest integer lower than x. Finally, $[\![a;b]\!] = \{a, a+1, \ldots, b\}$ is the set of integers between a and b.

2.2. Introducing chaotic iterations

Definition 1. *Let* $f : \mathbb{B}^\mathsf{N} \longrightarrow \mathbb{B}^\mathsf{N}$ *and* $S \in \mathcal{P}([\![1,\mathsf{N}]\!])^{\mathbb{N}}$. Chaotic iterations $(f,(x^0,S))$ *are defined by:*

$$\left\{\begin{array}{l} x^0 \in \mathbb{B}^\mathsf{N} \\ \forall n \in \mathbb{N}^*, \forall i \in [\![1;\mathsf{N}]\!], x_i^n = \left\{\begin{array}{ll} x_i^{n-1} & \text{if } i \notin S^n \\ f(x^{n-1})_i & \text{if } i \in S^n \end{array}\right. \end{array}\right.$$

We have regarded whether these chaotic iterations can behave chaotically, as it is defined for instance by Devaney, and if so, in which application context this behavior can be profitable. To do so, chaotic iterations have first been rewritten as simple discrete dynamical systems, as follows.

2.3. The Study of Iterative Systems

We have firstly stated that [6] (for the definitions of well-known mathematical properties of chaos, readers are referred to [4,5,7]):

Theorem 1. G_{f_0} *is regular and transitive on* $(\mathcal{X}, d)$*, thus it is chaotic according to Devaney. Furthermore, its constant of sensibility is greater than* $\mathsf{N}-1$.

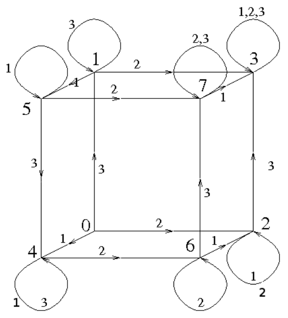

Figure 1: Example of an asynchronous iteration graph

Thus the set C of functions $f : \mathbb{B}^{\mathsf{N}} \longrightarrow \mathbb{B}^{\mathsf{N}}$ making the chaotic iterations of Definition 1 a case of chaos according to Devaney, is a nonempty set. To characterize functions of C, we have firstly stated that transitivity implies regularity for these particular iterated systems [3]. To achieve characterization, we then have introduced the following graph.

Let f be a map from $\mathbb{B}^{\mathsf{N}}$ to itself. The *asynchronous iteration graph* associated with f is the directed graph $\Gamma(f)$ defined by: the set of vertices is $\mathbb{B}^{\mathsf{N}}$; for all $x \in \mathbb{B}^{\mathsf{N}}$ and $i \in [\![1;\mathsf{N}]\!]$, the graph $\Gamma(f)$ contains an arc from x to $F_f(i,x)$. The relation between $\Gamma(f)$ and G_f is clear: there exists a path from x to x' in $\Gamma(f)$ if and only if there exists a strategy s such that the parallel iteration of G_f from the initial point (s,x) reaches the point x'. Figure 1 presents such an asynchronous iteration graph. We thus have proven that [3].

Theorem 2. *G_f is transitive, and thus chaotic according to Devaney, if and only if $\Gamma(f)$ is strongly connected.*

This characterization makes it possible to quantify the number of functions in C: it is equal to $\left(2^{\mathsf{N}}\right)^{2^{\mathsf{N}}}$. Then the study of the topological properties of disorder of these iterative systems has been further investigated, leading to the following results.

Theorem 3. *$\forall f \in C$, $Per\left(G_f\right)$ is infinitely countable, G_f is strongly transitive and is chaotic according to Knudsen. It is thus undecomposable, unstable, and chaotic as defined by Wiggins.*

Theorem 4. *$\left(\mathcal{X}, G_{f_0}\right)$ is topologically mixing, expansive (with a constant equal to 1), chaotic as defined by Li and Yorke, and has a topological entropy and an exponent of Lyapunov both equal to $ln(\mathsf{N})$.*

At this stage, a new kind of iterative systems that only manipulates integers have been discovered, leading to the questioning of their computing for security applications. In order to do so, the possibility of their computation without any loss of chaotic properties has first been investigated. These chaotic machines are presented in the next section.

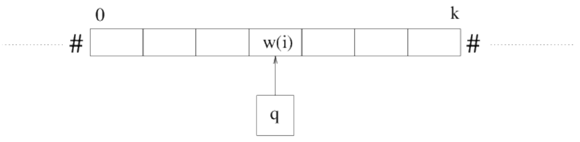

Figure 2: Turing Machine

3. Chaotic Turing Machines

3.1. General presentation

Let us consider a given algorithm. Because it must be computed one day, it is always possible to translate it as a Turing machine, and this last machine can be written as $x^{n+1} = f(x^n)$ in the following way. Let (w,i,q) be the current configuration of the Turing machine (Figure 2), where $w = \sharp^{-\omega}w(0)\ldots w(k)\sharp^{\omega}$ is the paper tape, i is the position of the tape head, q is used for the state of the machine, and δ is its transition function (the notations used here are well-known and widely used). We define f by:

- $f(w(0)\ldots w(k),i,q) = (w(0)\ldots w(i-1)aw(i+1)w(k), i+1, q')$, if $\delta(q,w(i)) = (q',a,\rightarrow)$,
- $f(w(0)\ldots w(k),i,q) = (w(0)\ldots w(i-1)aw(i+1)w(k), i-1, q')$, if $\delta(q,w(i)) = (q',a,\leftarrow)$.

Thus the Turing machine can be written as an iterate function $x^{n+1} = f(x^n)$ on a well-defined set $\mathcal{X}$, with x^0 as the initial configuration of the machine. We denote by $\mathcal{T}(S)$ the iterative process of the algorithm S.

Let τ be a topology on $\mathcal{X}$. So the behavior of this dynamical system can be studied to know whether or not the algorithm is τ-chaotic. Let us now explain how it is possible to have true chaos in a finite state machine.

3.2. Practical Issues

Up to now, most of computer programs presented as chaotic lose their chaotic properties while computing in the finite set of machine numbers. The algorithms that have been presented as chaotic usually act as follows. After having received its initial state, the machine works alone with no interaction with the outside world. Its outputs only depend on the different states of the machine. The main problem which prevents speaking about chaos in this particular situation is that when a finite state machine reaches the same internal state twice, the two future evolution are identical. Such a machine always finishes by entering into a cycle while iterating. This highly predictable behavior cannot be set as chaotic, at least as expressed by Devaney. Some attempts to define a discrete notion of chaos have been proposed, but they are not completely satisfactory and are less recognized than the notions evoked in a previous section.

The next stage was then to prove that chaos is possible in finite machine. The two main problems are that: (1) Chaotic sequences are usually defined in the real line whereas define real numbers on computers is impossible.

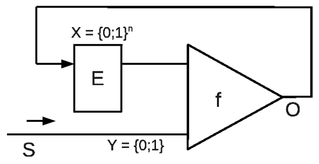

Figure 3: A chaotic finite-state machine. At each iteration, a new value is taken from the outside world (S). It is used by f as input together with the current state (E).

(2) All finite state machines always enter into a cycle when iterating, and this periodic behavior cannot be stated as chaotic.

The first problem is disputable, as the shadow lemma proves that, when considering the sequence $x^{n+1} = trunc_k\left(f(x^n)\right)$, where $(f, [0,1])$ is a chaotic dynamical system and $trunc_k(x) = \dfrac{\lfloor 10^k x \rfloor}{10^k}$ is the truncated version of $x \in \mathbb{R}$ at its $k-$th digits, then the sequence (x^n) is as close as possible to a real chaotic orbit. Thus iterating a chaotic function on floating point numbers does not deflate the chaotic behavior as much. However, even if this first claim is not really a problem, we have prevent from any disputation by considering a tool (the chaotic iterations) that only manipulates integers bounded by N.

The second claim is surprisingly never considered as an issue when considering the generation of randomness on computers. However, the stated problem can be solved in the following way. The computer must generate an output O computed from its current state E *and* the current value of an input S, which changes at each iteration (Figure 3). Therefore, it is possible that the machine presents the same state twice, but with two future evolution completely different, depending on the values of the input. By doing so, we thus obtain a machine with a finite number of states, which can evolve in infinitely different ways, due to the new values provided by the input at each iteration. Thus such a machine can behave chaotically.

4. Application to Hash Functions

4.1. Definitions

This section is devoted to a concrete realization of such a finite state chaotic machine in the computer science security field. We will show that, given a secured hash function, it is possible to realize a post-treatment on the obtained digest using chaotic iterations that preserves the security of the hash function. Furthermore, if the media to hash is obtained frame by frame from a stream, the resulted hash machine inherits the chaos properties of the chaotic iterations presented previously. For the interest to add chaos properties to an hash function, among other things regarding their diffusion and confusion [8], reader is referred to the following experimental studies: [1, 2, 6].

Let us firstly introduce some definitions.

Definition 2 (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 $h : K \times \Gamma^+ \to \Sigma^l$ 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 $h_k : \Gamma^+ \to \Sigma^l$ is defined by $h_k(m) = h(k, m)$, i.e., h_k generates a message digest of length l.*

Definition 3 (Collision resistance). *For a keyed hash function $h : \mathbb{B}^k \times \mathbb{B}^* \longrightarrow \mathbb{B}^n$, define the advantage of an adversary A for finding a collision as*

$$Adv_{\mathsf{A}} = Pr\left[\begin{array}{c} K \overset{\$}{\leftarrow} \mathbb{B}^k \\ (m, m') \leftarrow \mathsf{A}(K) \end{array} : \begin{array}{c} m \neq m' \\ h(K, m) = h(K, m') \end{array}\right] \tag{1}$$

where \$ means that the element is pick randomly. The insecurity of h with respect to collision resistance is

$$InSec_h(t) = \max_{\mathsf{A}} \{Adv_{\mathsf{A}}\} \tag{2}$$

when the maximum is taken over all adversaries A with total running time t.

In other words, an adversary should not be able to find a collision, that is, two distinct messages m and m' such that $h(m) = h(m')$.

Definition 4 (Second-Preimage Resistance). *For a keyed hash function $h : \mathbb{B}^k \times \mathbb{B}^* \longrightarrow \mathbb{B}^n$, define the advantage of an adversary A for finding a second-preimage as*

$$Adv_{\mathsf{A}}(m) = Pr\left[\begin{array}{c} K \overset{\$}{\leftarrow} \mathbb{B}^k \\ m' \overset{\$}{\leftarrow} \mathsf{A}(K) \end{array} : \begin{array}{c} m \neq m' \\ h(K, m) = h(K, m') \end{array}\right] \tag{3}$$

The insecurity of h with respect to collision resistance is

$$InSec_h(t) = \max_{\mathsf{A}} \left\{ \max_{m \in \mathbb{B}^k} \{Adv_{\mathsf{A}}(m)\} \right\} \tag{4}$$

when the maximum is taken over all adversaries A with total running time t.

That is to say, an adversary given a message m should not be able to find another message m' such that $m \neq m'$ and $h(m) = h(m')$. Let us now give a post-operative mode that can be applied to a cryptographically secure hash function without loosing the cryptographic properties recalled above.

Definition 5. *Let*

- $k_1, k_2, n \in \mathbb{N}^*$,
- $h : (k, m) \in \mathbb{B}^{k_1} \times \mathbb{B}^* \longmapsto h(k, m) \in \mathbb{B}^n$ *a keyed hash function,*
- $S : k \in \mathbb{B}^{k_2} \longmapsto \left(S(k)^i\right)_{i \in \mathbb{N}} \in [\![1, n]\!]^{\mathbb{N}}$*:*

  - *either a cryptographically secure pseudorandom number generator (PRNG),*
  - *or, in case of a binary input stream $m = m^0 \| m^1 \| m^1 \| \dots$ where $\forall i, |m^i| = n$, $\left(S(k)^i\right)_{i \in \mathbb{N}} = \left(m^k\right)_{i \in \mathbb{N}}$.*

- $\mathcal{K} = \mathbb{B}^{k_1} \times \mathbb{B}^{k_2} \times \mathbb{N}$ *called the* key space,
- *and $f : \mathbb{B}^n \longrightarrow \mathbb{B}^n$ a bijective map.*

We define the keyed hash function $\mathcal{H}_h : \mathcal{K} \times \mathbb{B}^* \longrightarrow \mathbb{B}^n$ by the following procedure

```
Inputs:  $k = (k_1, k_2, n) \in \mathcal{K}$
         $m \in \mathbb{B}^*$
Runs:    $X = h(k_1, m)$, or $X = h(k_1, m^0)$ if $m$ is a stream
         for $i = 1, \dots, n$ :
             $X = G_f(X, S^i)$
         return $X$
```

$\mathcal{H}_h$ is thus a chaotic iteration based post-treatment on the inputted hash function h. The strategy is provided by a secured PRNG when the machine operates in a vacuum whereas it is redetermined at each iteration from the input stream in case of a finite machine open to the outside. By doing so, we obtain a new hash function $\mathcal{H}_h$ with h, and this new one has a chaotic dependence regarding the inputted stream.

4.2. Security proofs

The two following lemma are obvious.

Lemma 1. *If $f : \mathbb{B}^n \longrightarrow \mathbb{B}^n$ is bijective, then $\forall S \in [\![1, n]\!]$, the map $G_{f,S} : x \in \mathbb{B}^n \to G_f(x, S)_1 \in \mathbb{B}^n$ is bijective too.*

Proof. Let $y = (y_1, \dots, y_n) \in \mathbb{B}^n$ and $S \in [\![1, n]\!]$. Thus

$$G_{f,S}(y_1, \dots, y_{S-1}, f^{-1}(y_S), y_{S+1}, \dots, y_n)_1 = y.$$

So $G_{f,S}$ is a surjective map between two finite sets. □

Lemma 2. *Let $S \in [\![1, n]\!]^{\mathbb{N}}$ and $N \in \mathbb{N}^*$. If f is bijective, then $G_{f,S,N} : x \in \mathbb{B}^n \longmapsto G_f^N(x, S)_1 \in \mathbb{B}^n$ is bijective too.*

Proof. Indeed, $G_{s,f,n} = G_{f,S^n} \circ \dots \circ G_{f,S^0}$ is bijective as a composition of bijective maps. □

We can now state that,

Theorem 5. *If h satisfies the collision resistance property, then it is the case too for $\mathcal{H}_h$. And if h satisfies the second-preimage resistance property, then it is the case too for $\mathcal{H}_h$.*

Proof. Let $A(k_1, k_2, n) = (m_1, m_2)$ such that $\mathcal{H}_h((k_1, k_2, n), m_1) = \mathcal{H}_h((k_1, k_2, n), m_2)$. Then $G_{f,S(k_2),n}(h(m_1)) = G_{f,S(k_2),n}(h(m_2))$. So $h(m_1, k_1) = h(m_2, k_1)$.

For the second-preimage resistance property, let $m, k \in \mathbb{B}^* \times \mathcal{K}$. If a message $m' \in \mathbb{B}^*$ can be found such that $\mathcal{H}_h(k, m) = \mathcal{H}_h(k, m')$, then $h(k_1, m) = h(k_1, m')$: a second-preimage for h has thus be found. □

Finally, as $\mathcal{H}_h$ simply operates chaotic iterations with strategy $\mathcal{S}$ provided at each iterate by the media, we have:

Theorem 6. *In case where the strategy $\mathcal{S}$ is the bitwise xor between a secured PRNG and the input stream, the resulted hash function $\mathcal{H}_h$ is chaotic.*

5. Conclusion

In this article, the research we have previously done in the field of truly chaotic finite machines are summarized and clarified to serve as an introduction to our approach. This approach consists in considering a specific family of discrete dynamical systems that iterate on a set having the form $\mathcal{X} = \mathcal{P}([\![1, N]\!])^{\mathbb{N}} \times \mathbb{B}^{\mathbb{N}}$, making it possible to obtain pure, non degenerated chaos on finite machines. These particular dynamical systems are called chaotic iterations. Our method consists in considering the left part of $\mathcal{X}$ as the tape of the Turing machine whereas the right part is the state register of the machine. Chaos implies here that if the initial tape and the initial state are not known exactly, then for some transition function the evolution of the iterates of the Turing machine cannot be predicted.

References

[1] J. Bahi, J-F. Couchot, and C. Guyeux. Performance analysis of a keyed hash function based on discrete and chaotic proven iterations. In *INTERNET 2011*, pages 52–57, Luxembourg, Luxembourg, June 2011. Best paper award.

[2] J. Bahi, J-F. Couchot, and C. Guyeux. Quality analysis of a chaotic proven keyed hash function. *International Journal On Advances in Internet Technology*, 5:26–33, 2012.

[3] J. Bahi, J-F Couchot, C. Guyeux, and A. Richard. On the link between strongly connected iteration graphs and chaotic boolean discrete-time dynamical systems. In *FCT'11*, volume 6914, pages 126–137, Oslo, Norway, August 2011.

[4] R. L. Devaney. *An Introduction to Chaotic Dynamical Systems*. Addison-Wesley, 2nd edition, 1989.

[5] Enrico Formenti. *Automates cellulaires et chaos : de la vision topologique à la vision algorithmique*. PhD thesis, École Normale Supérieure de Lyon, 1998.

[6] C. Guyeux and J. Bahi. A topological study of chaotic iterations. application to hash functions. In *CIPS*, volume 394, pages 51–73. Springer, 2012. Best paper.

[7] Knudsen. Chaos without nonperiodicity. *Amer. Math. Monthly*, 101, 1994.

[8] C. E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28:656–715, 1949.