A Cellular Automaton Based Fast One-Way Hash Function Suitable for Hardware Implementation

Miodrag Mihaljević[1,3], Yuliang Zheng[2] and Hideki Imai[3]

[1]Academy of Science and Arts, Kneza Mihaila 35, Belgrade Yugoslavia
Email: emihalje@ubbg.etf.bg.ac.yu
[2]Monash University, McMahons Road, Frankston, Melburne, VIC 3199, Australia
Email: yzheng@fcit.monash.edu.au
[3]The University of Tokyo, 7-22-1 Roppongi, Minato-ku, Tokyo 106, Japan,
Email: imai@iis.u-tokyo.ac.jp

Abstract. One-way hash functions are an important tool in achieving authentication and data integrity. The aim of this paper is to propose a novel one-way hash function based on cellular automata whose cryptographic properties have been extensively studied over the past decade or so. Furthermore, security of the proposed one-way hash function is analyzed by the use of very recently published results on applications of cellular automata in cryptography. The analysis indicates that the one-way hash function is secure against all known attacks. An important feature of the proposed one-way hash function is that it is especially suitable for compact and fast implementation in hardware, which is particularly attractive to emerging security applications that employ smart cards, such as digital identification cards and electronic cash payment protocols,

1 Introduction

Cryptographic hash functions play an important role in modern cryptography. The basic idea of cryptographic hash functions is that a hash-value serves as a compact representative image (sometimes called an imprint, digital fingerprint, or message digest) of an input string, and can be used as if it were uniquely identifiable with that string.

Following [1], at the highest level, cryptographic hash functions may be classified into two classes: hash functions, whose specification dictates a single input parameter - a message (unkeyed hash functions); and keyed hash functions, whose specification dictates two distinct inputs - a message and a secret key. This paper is concerned with unkeyed hash functions which are also called one-way hash functions.

A typical usage of one-way hash functions for data integrity is as follows. The hash-value corresponding to a particular message M is computed at time t_1. The integrity of this hash-value (but not the message itself) is protected in some manner. At a subsequent time t_2, the following test is carried out to determine whether the message has been altered, i.e., whether a message M' is

the same as the original message. The hash-value of M' is computed and compared to the protected hash-value; if they are identical, one accepts that the inputs are also equal, and thus that the message has not been altered. The problem of preserving the integrity of a potentially large message is thus reduced to that of a small fixed-size hash-value. Since the existence of collisions is guaranteed in many-to-one mappings, the unique association between the inputs and hash-values can, at best, be in a computational sense. A hash-value should be uniquely identifiable with a single input in practice, and collisions should be computationally infeasible to find (essentially never occurring in practice).

In this paper, a novel and fast one-way hash function is proposed and analyzed. The proposed one-way hash function is based on a quite different approach than these employed in other one-way hash functions in that it is based on programmable cellular automata. Advantages of one-way hash functions that employ cellular automata include: it is fast and suitable for hardware implementation, and its security can be analyzed by borrowing some of the well-established research results in cellular automaton theory.

In Sections 2 - 4 relevant background about one-way hash functions and cellular automata is summarized. In Section 5 the novel one-way hash function is proposed. Its security together with efficiency is analyzed in Section 6. Some concluding remarks are made in Section 7.

2 One-Way Hash Functions

A hash function (in the unrestricted sense) is a function $hash(\cdot)$ which has, as minimum the following two properties (see [1], for example):

- *compression* - $hash$ maps an input M of arbitrary finite bit-length, to an output $hash(M)$ of a fixed bit-length n.
- *ease of computation* - given $hash(\cdot)$ and an input M, $hash(M)$ is easy to compute.

In addition, for a one-way hash function $hash(\cdot)$ with inputs M, M' and outputs Z, Z', the following three properties are expected to hold (see [1], for example):

1. *preimage resistance* - for essentially all pre-specified outputs, it is computationally infeasible to find any input which hashes to the output, i.e., to find any preimage M' such that $hash(M') = Z$ when given any Z for which a corresponding input is not known.
2. *2nd-preimage resistance* - it is computationally infeasible to find any second input which has the same output as any specified input, i.e., given M, to find a 2nd-preimage $M' \neq M$ such that $hash(M) = hash(M')$.
3. *collision resistance* - it is computationally infeasible to find two distinct inputs M and M' which hash to the same output, i.e., such that $hash(M) = hash(M')$.

Note that the following relationships between the three properties of a one-way hash function hold:

- 2nd-preimage resistance implies preimage resistance.
- Collision resistance implies both 2nd-preimage resistance and preimage resistance.

2.1 General Model for Iterated Hash Functions

Most one-way hash functions $hash(\cdot)$ are designed as iterative processes which hash arbitrary-length inputs by processing successive fixed-size blocks of the input (see [1], for example). A hash input M of arbitrary finite length is divided into fixed-length ℓ-bit blocks M_i. This preprocessing typically involves appending extra bits (padding) as necessary to attain an overall bit-length which is a multiple m of the block-length ℓ and often includes (for security reasons, see [3] and [4]) a block indicating the bit-length of the unpadded input. Each block M_i then serves as input to an internal fixed-size function h, the *compression function* of $hash$, which computes a new intermediate result of bit-length n for some fixed n, as a function of the previous n-bit intermediate results and the next input block M_i. Let H_i denote the partial result after Stage i Then the general process for an iterated one-way hash function with inputs $M = (M_1, M_2, ..., M_m)$ can be modeled as follows:

$$\begin{aligned} H_0 &= IV \ , \\ H_i &= h(H_{i-1}, M_i) \ , \ \ 1 \leq i \leq m \ , \\ hash(M) &= g(H_m) \ . \end{aligned} \tag{1}$$

H_{i-1} serves as the n-bit *chaining variable* between Stage $i-1$ and Stage i, and H_0 is a pre-defined starting value or initial value IV. An optional transformation g is used in a final step to map the n-bit chaining variable to an n'-bit result $g(H_m)$; g is often the identity mapping $g(H_m) = H_m$.

Specific one-way hash functions proposed in the literature differ from one another in preprocessing, compression function, and output transformation.

2.2 Dedicated One-Way Hash Functions

¿From a structural viewpoint, one-way hash functions may be categorized based on the nature of the operations comprising their internal compression functions. From this viewpoint, the three broadest categories of iterated one-way hash functions studied to date are:
- one-way hash functions based on block ciphers,
- one-way hash functions based on modular arithmetic, and
- dedicated one-way hash functions.

Dedicated one-way hash functions are those designed specifically for hashing, with speed in mind and independent of other system subcomponents (e.g., block cipher or modular multiplication subcomponents which may already be present for non-hashing purposes).

The following one-way hash functions, all based on the so called MD4 initially proposed in [11] have received the greatest attention:
- MD5, [12],
- SHA-1, [13],
- RIPEMD-160, [14],
- HAVAL, [15] .

A quite different class of dedicated hash functions based on a particular linear finite state machines - cellular automata have been reported in [4], [16] and [17].

The one-way hash function to be proposed below belongs to the class of dedicated one-way hash functions, it is a development of the cellular automata approach, and it is suitable for hardware implementation.

2.3 Security of the One-Way Hash Function

Given a specific one-way hash function, it is desirable to be able to prove a lower bound on the complexity of attacking it under specified scenarios, with as few and weak assumptions as possible. However, such results are scarce. Typically, the best guidance available regarding the security of a particular one-way hash function is the complexity of the (most efficient) applicable known attack, which gives an upper bound on security. An attack of complexity 2^n is one which requires approximately 2^n operations, each being an appropriate unit of work. The storage complexity of an attack should also be considered.

Assuming that the hash-code approximates a uniform random variable it is well known that the following holds:

- For an n-bit hash function h, one may expect a guessing attack to find a preimage or second preimage within 2^n hashing operations.
- For an adversary able to choose messages, a birthday attack, [2], allows colliding pairs of messages M, M' with $hash(M) = hash(M')$ to be found in about $2^{n/2}$ operations and a reasonable amount of memory.

An n-bit one-way hash function $hash(\cdot)$ has *ideal security* if both: (a) given a hash output, producing each of a preimage and a 2nd-preimage requires approximately 2^n operations; and (b) producing a collision requires approximately $2^{n/2}$ operations.

Following [3] and [4] denote by *MD-strengthening* appending an additional block at the end of of the input string containing its length.

Based on [3], [4], [5], [6], [7], [8], in a number of models, it is possible to relate the security of $hash(\cdot)$ to the security of h and g according to the following result:

Theorem 1. (cf. [8]) Let $hash(\cdot)$ be an iterated hash function with MD-strengthening. Then preimage and collision attacks on $hash(\cdot)$ (where an attacker can choose IV freely) have roughly the same complexity as the corresponding attacks on h and g.

Theorem 1 gives a lower bound on the security of $hash(\cdot)$.

According to [10] and [8] the iterated hash functions based on the Davis-Mayer compression function given by the following

$$h(M_i, H_{i-1}) = E_{M_i}(H_{i-1}) \oplus H_{i-1} \ , \tag{2}$$

where $E_K(\cdot)$ is a block cipher controlled by the key K, are believed to be as secure as the underlying cipher $E_K(\cdot)$ is.

As a direct extension of the results of security related to cipher block chaining [9], and the assumption 1 from [8] (which is a standard one in cryptography today), we assume the following.

Assumption 1. Let the compression function h be the Davis-Meyer function (2) and the employed cryptographic transformation is a secure one. Then finding collisions for h requires about $2^{n/2}$ encryption (of an n-bit block), and finding a preimage for h requires about 2^n encryption.

The above discussions imply that the main problem in the design of a secure one-way hash function can be reduced to the design of a secure compression function and a good output function.

3 Cellular Automata

A one-dimensional binary cellular automaton (CA) consists of a linearly connected array of L cells, each of which takes the value 0 or 1, and a Boolean function $f(\mathbf{x})$ with q variables. The value of the cell x_i is updated in parallel (synchronously) using this function in discrete time steps as $x_i^{'} = f(\mathbf{x})$ for $i = 1, 2, ..., L$. The boundary conditions are usually handled by taking the index value modulo L. The parameter q is usually an odd integer, i.e., $q = 2r + 1$, where r is often named the radius of the function $f(\mathbf{x})$; the new value of the ith cell is calculated using the value of the ith cell and the values of r neighboring cells to the right and left of the ith cell.

Since there are L cells, each of which takes the values of 0 or 1, there are 2^L possible state vectors. Let $\mathbf{S}_k$ denote the state vector at the time step k. Starting from an initial state vector $\mathbf{S}_0$, the cellular automaton moves to the states $\mathbf{S}_1$, $\mathbf{S}_2$, $\mathbf{S}_3$ etc., at time steps $k = 1, 2, 3, ...$ etc. The state vector $\mathbf{S}_k$ takes values from the set of L-bit binary vectors as k advances, and the state machine will eventually cycle, i.e., it will reach a state $\mathbf{S}_{k+P}$ which was visited earlier $\mathbf{S}_k = \mathbf{S}_{k+P}$. The period P is a function of the initial state, the updating function, and the number of cells.

For a CA with $q = 3$, the evolution of of the ith cell in each discrete time step t (clock cycle) can be represented as a function of the present state of the $(i-1)$th, (i)th, and $(i+1)$th cells as

$$x_i(t+1) = f\{x_{i-1}(t), x_i(t), x_{i+1}(t)\} \ . \tag{3}$$

f is also called the combinatorial logic associated with the CA. Each combinatorial logic represents an updating rule for evolving to the next state.

If the next state function of a cell is expressed in the form of a truth table, then the decimal equivalent of the output column in the truth table is conventionally called a CA rule number. A nonlinear rule, called Rule 30, proposed and considered by Wolfram in [18], realizes updating according to the following:

$$x_i(t+1) = x_{i-1}(t) \;\; XOR \;\; [x_i(t) \;\; OR \;\; x_{i+1}(t)] \;\; . \tag{4}$$

Of particular interest are two linear rules in GF(2). These are known as Rule 90 and Rule 150 respectively. Rule 90 specifies an evolution (updating) from current to the next state according the following combinatorial logic:

$$x_i(t+1) = x_{i-1}(t) \oplus x_{i+1}(t) \tag{5}$$

where $\oplus$ denotes XOR operation. Note that when Rule 90 is applied the next state of the ith cell depends on the present state of its left and right neighbors. Similarly, the combinational logic for Rule 150 is given by

$$x_i(t+1) = x_{i-1}(t) \oplus x_i(t) \oplus x_{i+1}(t) \;\; , \tag{6}$$

that is, the next state of the ith cell depends on the present states of its left and right neighbors and also on its own present state.

If in a CA the same rule applies to all cells, then the CA is called a uniform CA; otherwise it is called a hybrid CA. There can be various boundary conditions; namely, null (where extreme cells are connected to logic "0"), periodic (extreme cells are adjacent), etc.

3.1 Additive Cellular Automata

A very important class of CA are linear CA in GF(2) or additive CA. If the next-state generating logic employs only XOR or $XNOR$ operations, then the CA is said to be an *additive* CA. Linear CA is a special form of *Linear Finite State Machines* (LFSM's). Every LFSM is uniquely represented by a transition matrix over GF(2), and every transition matrix has a characteristic polynomial.

For an L-cell one-dimensional additive CA with XOR operations only, it has been shown in [19] that the CA can be characterized by a linear operator denoted by $\mathbf{T}$ which is an $L \times L$ Boolean matrix and whose ith row specifies the neighborhood dependency of the ith cell. The next state of CA is generated by applying this linear operator on the present CA state represented as a column vector. The operation is the normal matrix multiplication, but the addition involved is modulo-2 sum. If $\mathbf{x}(t)$ is a column vector representing the state of the automaton at the tth instant of time, then the next state of the automaton is given by:

$$\mathbf{x}(t+1) = \mathbf{T} \times \mathbf{x}(t) \;\; . \tag{7}$$

If the characteristic polynomial of a CA is primitive, then it is referred to as a maximal length CA. Such an L cell CA generates all $2^L - 1$ states in successive cycles excluding the all zero state.

Since, for a fixed order L, there are 2^{L^2} transition matrices (and hence 2^{L^2} LFSM's) but only 2^L degree L polynomials, we have the following situation: There is a one-to-one correspondence between L-cell LFSM and $L \times L$ matrices, and at the same time a many-to-one correspondence between the transition matrices and the polynomials of degree L.

The characteristic polynomial of an LFSM is never difficult to obtain, as it can be calculated by evaluating a determinant. On the other hand, finding a particular type of LFSM (such as a CA) with a specific characteristic polynomial is a problem solved in [20] where a method is presented for obtaining a CA that has a given characteristic polynomial. The same method can also be used to solve the problem as to whether a CA exists for each irreducible polynomial.

A systematic treatment of the additive CA theory and applications is presented in a recent book [22], as well as in [21].

3.2 Programmable Cellular Automata

Positional representations of Rule 90 and Rule 150 show that their neighborhood dependence differ in only one position, viz., on the cell itself. Therefore, by allowing a single control line per cell, one can apply both Rule 90 and Rule 150 on the same cell at different time steps. Thereby, an L cell CA structure can be used for implementing 2^L CA configurations. Realizing different CA configurations (cell updating rules) on the same structure can be achieved using a control logic to control the appropriate switches and a control program, stored in ROM, can be employed to activate the control. The 1(0) state of the ith bit of a ROM word closes (opens) the switch that controls the ith cell. Such a structure is referred as to as a *programmable cellular automaton* (PCA).

Accordingly, allowing one control input per cell that configures the updating rule, we can apply to that cell, either Rule 90 or Rule 150. The n-bits control word for an n cells PCA has 0(1) on the ith cell if Rule 90(150) is applied to the ith cell.

4 Cryptographic Applications of Cellular Automata

Cellular automata have been considered as a building block for the design of both block and stream ciphers, as well as for design of certain hash functions. The first cryptographic application of a cellular automaton was given in [18]. A block and a stream cipher based on cellular automata were proposed in [23]. Two PCA based key stream generators, called PCA with ROM (Read Only Memory) and Two Stage PCA respectively, together with results on theirs security analysis, were proposed in [23]. Some cryptographic CA / PCA applications are summarized in [22].

Also, additional cryptanalysis of certain CA /PCA based key stream generators have been published. A method for reconstructing of a CA initial state based on the sequence of bits generated by a central CA cell is given in [24]. In [26] the inversion algorithm which computes the predecessor of a given state

vector, assuming a nonlinear CA configuration rule, is proposed. Cryptographic security examination of the Two Stage PCA and the PCA with ROM have been reported in [27] and [28], respectively, assuming ciphertext only attacks. Some vulnerabilities of these schemes on certain cryptanalytic attacks were demonstrated, and it is shown that the effective secret key size is significantly smaller than its formal length. The same weaknesses are pointed out in [25] assuming known plaintext attacks.

Recently, an improved key stream generator based on programmable cellular automata was proposed and analyzed in [29].

4.1 Hash Functions Based on Cellular Automata

The first proposal of the CA application for one-way hash function design has been reported in [4].

The vulnerability of the scheme from [4] is presented in [16] together with a proposal for new CA based hash function called *Cellhash*. *Callhash* assumes preparation of a message so that it is a concatenation of N 32-bit words M_i, $i = 0, 1, ..., N-1$, and application of the following procedure:

$H^0 = IV$,
$H^j = F_c(H^{j-1}, M_{j-1} M_{jmodN} ... M_{j+\delta modN})$, $j = 1, 2, ..., N$,
H^N is the hash result,
where $F_c(H, A)$ is a function with argument H a bitstring of length 257, A is a bitstring of length 256, and IV is the all-zero bitstring of length 257; it returns a bitstring of length 257. $F_c(H, A)$ consists of five steps with the following properties [16]:
Step 1 is a nonlinear cellular automation operation where each bitvalue is updated according to the bitvalues in its neighborhood applying a nonlinear updating rule considered in [18]. The nonlinearity of the updating rule has to guarantee the needed confusion.
Step 2 consists merely of complementing 1 bit to eliminate circular symmetry in case bitstring A consists of only $0's$.
Step 3 is a linear CA operation that has to increases the diffusion.
Step 4 realizes the actual messagebits injection in H to be diffused and confused in subsequent rounds.
Step 5 is a bit permutation where bits are placed away from their previous neighbors.

In [17] a one-way function based on two-dimensional CA is proposed and analyzed. The transition function of the two-dimensional CA is the composition of two state transition functions of a one-dimensional CA. The first state transition function is computed by regarding the two-dimensional CA as a one-dimensional one in column order, and the second state transition function is computed by regarding it as a one-dimensional in a row order. For each message block the two-dimensional CA runs certain number of cycles, and the hash result is the CA state after the last cycle.

5 A Novel Cellular Automaton Based Hash Function

In this section, a novel hash function is proposed. The proposed function follows the general model for iterated hash functions (see relation (1)), and employs the Davies-Meyer principle, which according to (2) assumes that the compression function h is defined by the following:

$$h(M_i, H_{i-1}) = F_{M_i}(H_{i-1}) \oplus H_{i-1} \ , \tag{8}$$

where $F_{M_i}(H_{i-1})$ is a function which maps H_{i-1} according to M_i which consists of $2n$ bits. These would guarantee the approved basis for design and imply secure hash function construction assuming that the compression function and the output function are secure. The novel construction of the compression function h and the output function g is based on cellular automata and recently published results which imply the security of the novel h and g functions.

The proposed hash function provides:

- very fast hashing,
- application of cellular automaton theory for the security examination,
- the preimage and collision resistance due to the employed principles and building blocks.

The novel compression function h^*, the output function g^*, and the whole hash function $hash^*$ are defined by the next three parts of this section.

5.1 Compression Function $h^*(\cdot)$

We assume the following notations:
- ℓ is an integer such that n/ℓ is also an integer (for example $\ell = 8$);
- $\varphi_k(\cdot)$, $k = 1, 2, ..., K$, are functions each of which nonlinearly maps two ℓ-dimensional binary vectors into an ℓ-dimensional binary vector according to the certain ℓ Boolean functions, assuming that the criteria from [15] are satisfied;
- $CA(\cdot)$ is an operator of mapping a current CA state into the next state, assuming CA with primitive characteristic polynomial;
- $PCA_X(\cdot)$ is an operator of mapping a current PCA state into the next state assuming that the applied configuration rule is controlled by a binary vector X according to the following:

(-) if the ith bit of X is 0 then the next state of ith PCA cell is defined by Rule 90,

(-) if the ith bit of X is 1 then the next state of ith PCA cell is defined by Rule 150.

Let M_i be split into $\frac{2n}{\ell}$ successive nonoverlapping equal length blocks of ℓ-bits, $M_{i,1}, M_{i,2}, ..., M_{i,\frac{2n}{\ell}}$, and let H_{i-1} be split into $\frac{n}{\ell}$ blocks of ℓ bits each, $H_{i-1,1}, H_{i-1,2}, ..., H_{i-1,\frac{n}{\ell}}$.

The two vectors X_i and Z_i defined below are certain arguments of the novel compression function.

The novel compression function consists of the following four sets of operations.

- *First Nonlinear Processing*
  X_i is an n-dimensional binary vector obtained by concatenating and interleaving the values V_k of the functions $\varphi_{kmodK}(\cdot)$,

$$V_k = \varphi_{kmodK}(\varphi_{kmodK}(M_{i,k}, H_{i-1,k}),\ M_{i,\frac{n}{\ell}+k}\ ) ,$$
$$k = 1, 2, ..., \frac{n}{\ell} , \tag{9}$$

  according to the following:
  the jth bit of V_k is equal to the $((k-1)\ell + j)$th bit of X_i.

  Y_i is an n-dimensional binary vector obtained by concatenating and interleaving the values W_k of the functions $\varphi_{kmodK}(\cdot)$,

$$W_k = \varphi_{kmodK}(\ V_k,\ V_{\frac{n}{\ell}+1-k}\ ) ,$$
$$k = 1, 2, ..., \frac{n}{2\ell} , \tag{10}$$
$$W_k = \varphi_{kmodK}(\ V_k,\ V_{k-\frac{n}{2\ell}}\ ) ,$$
$$k = \frac{n}{2\ell} + 1, \frac{n}{2\ell} + 2, ..., \frac{n}{\ell} , \tag{11}$$

  (where V_k is defined by (9)), according to the following:
  the jth bit of W_k is equal to the $((k-1)\ell + j)$th bit of Y_i.

- *CA Processing*
  $Y_i^{'}$ is an n-dimensional binary vector obtained by the following:

$$Y_i^{'} = CA(Y_i) \ . \tag{12}$$

- *Second Nonlinear Processing*
  Let $Y_i^{'}$ be split into $\frac{n}{\ell}$ blocks of ℓ bits each, $Y_{i,1}^{'}$, $Y_{i,2}^{'}$, ... , $Y_{i,\frac{n}{\ell}}^{'}$.
  Z_i is an n-dimensional binary vector obtained by concatenating and interleaving the values $Y_k^{''}$ of the functions $\varphi_{kmodK}(\cdot)$,

$$Y_k^{''} = \varphi_{kmodK}(\ Y_k^{'},\ Y_{\frac{n}{\ell}+1-k}^{'}\ ) ,$$
$$k = 1, 2, ..., \frac{n}{2\ell} , \tag{13}$$
$$Y_k^{''} = \varphi_{kmodK}(\ Y_k^{'},\ Y_{k-\frac{n}{2\ell}}^{'}\ ) ,$$
$$k = \frac{n}{2\ell} + 1, \frac{n}{2\ell} + 2, ..., \frac{n}{\ell} , \tag{14}$$

  (where $Y_k^{'}$ is defined by (12)), according to the following:
  the jth bit of $Y_k^{''}$ is equal to the $((k-1)\ell + j)$th bit of Z_i.

– *PCA Processing*
The compression function $h^*(\cdot)$ is then defined by the following

$$h^*(M_i, H_{i-1}) = PCA_{X_i}(Z_i) \oplus H_{i-1} \ , \tag{15}$$

where $\oplus$ denotes bit-by-bit $mod2$ addition.

Note that in defining $h^*(\cdot)$, we have assumed that CA and PCA embodied in it run only one transition cycle. Theoretical analysis to be presented below indicates that this arrangement suffices in resisting currently known attacks. In practice, however, one may choose to allow the CA and PCA to run a number of transitions before reaching a state which will be used as an output. Such a variant would provide a higher level of security.

5.2 Output Function $g^*(\cdot)$

The output function $g^*(\cdot)$ is a variant of the cellular automaton based key stream generator proposed and analyzed in [29]. The output function uses the input argument H_m as a secret key and based on it generates n output bits.

The main parts of the key stream generator which realizes the output function g^* are the following: an n-cell PCA, a ROM which contains the configuration rules for the PCA, an n-length binary buffer, and an n-dimensional varying permutation.

Assume that $\eta < n$ maximal length CA's are chosen out of all possible maximal length CA's with Rule 90 and Rule 150. These rules are noted as $\{R_0, R_1, ..., R_\eta\}$. The rule configuration control word corresponding to a rule R_i is stored in a ROM word. The output function operates as following:

– Initially the PCA is configured with the rule $R_{0+\Delta_0}$, where Δ_0 is $mod\ \eta$ value of H_m decimal representation, and loaded with the output H_m of the compression function from the last iteration. With this configuration the PCA runs one clock cycle. Then it is reconfigured with next rule (i.e., R_i) and runs another cycle. The rule configuration of PCA changes after every run, i.e., in the next run, a rule is $R_{(i+1+\Delta) mod\ \eta}$, where Δ is decimal equivalent of the previous PCA state.

– After each clock cycle, the content of a middle cell of the PCA is taken as an output and stored in the n-length binary buffer.

– After n clock cycles, the buffer content is permuted according the varying permutation controlled by the current PCA state.

5.3 Hash Function $hash^*(\cdot)$ Algorithm

Accordingly, we propose the following fast hash function.

1. INPUT. A bitstring of the message M, and the n-bits initial value IV.

2. PREPROCESSING.
   - MD-strengthening and padding using the approach proposed in [15].

   - Splitting the processed message into m blocks of $2n$-bits each: $M = (M_1, M_2, ..., M_m)$.

3. ITERATIVE PROCESSING.
   Assuming that $H_0 = IV$, for each $i = 1, 2, ..., m$, do the following:
   - calculate the compression function $h^*(\cdot)$ value: $H_i = h^*(M_i, H_{i-1})$,

   where $h^*(\cdot)$ is defined in the Section 5.1.

4. If H_m is the all zero vector recalculate H_m according to the following: $H_m = h^*(M_m, H_0)$, and proceed to the next step.

5. OUTPUT FUNCTION. Calculate $g^*(H_m)$, where $g^*(\cdot)$ is defined in the Section 5.2.

6. OUTPUT. n-bits message digest: $hash^*(M) = g^*(H_m)$.

6 Analysis of the Proposed Hash Function

In this section the security and complexity analysis of the hash function proposed in the previous section are given.

6.1 Security Analysis

Note that according to the Theorem 1, a lower bound on security of the proposed hash function is determined by the characteristics of its compression and output functions. Accordingly, the security will be considered through the security of the proposed functions $g^*(\cdot)$ and $h^*(\cdot)$. Security of both the functions will be examined on the preimage / 2nd preimage and collisions attacks.

The facts and discussions which are given in this section imply that the proposed hash function has ideal security, i.e., given a hash output, producing each of a preimage or 2nd preimage requires approximately 2^n operations and producing a collision requires approximately $2^{n/2}$ operations. Also, due to the structure of the compression function $h^*(\cdot)$, the Assumption 1 implies that we can expect that the proposed hash function is an ideal one.

Security of Compression Function $h^*(\cdot)$

Processing of each message block M_i, $i = 1, 2, ..., m$, by the compression function $h^*(M_i, H_{i-1})$ consists of the following:

- nonlinear mapping of M_i and H_{i-1} into two n-dimensional binary vectors: the CA current state Y_i and the configuration rule vector X_i for the PCA;
- CA mapping of its current state into the next one - an n-bits vector $CA(Y_i)$;
- nonlinear mapping of $CA(Y_i)$ into the vector Z_i.
- PCA mapping of its current state equal to the the vector Z_i into the next one - an n-bits vector $PCA_{X_i}(Z_i)$ assuming that the PCA configuration rule is controlled by the binary vector X according to the following: the next state transition rule for the ith PCA cell is 90 or 150 if the ith bit of X_i is 0 or 1, respectively;
- bit-by-bit $mod2$ addition of the n-bits vectors $PCA_{X_i}(Z_i)$ and H_{i-1} yielding the new intermediate result H_i.

Accordingly, the following facts imply the security of the compression function:

1. The CA has primitive characteristic polynomial so that any nonzero state is mapped into a nonzero state which belongs to the sequence of all possible different $2^n - 1$ nonzero n-dimensional vectors in such manner that the expected Hamming distance between the current state and the next one is $n/2$. The pattern generated by maximal length CA's meet the cryptographic criteria (and the quality of randomness of the patterns generated by CA's is significantly better than that of linear feedback shift register based structures), [23].
2. High nonlinearity of the compression function due to the employed Boolean functions and PCA (with unknown configuration rule, [23]).
3. So far published algorithms for reconstruction of a CA/PCA state employing certain CA/PCA outputs, are the following:
   (a) algorithm from [24] based on noiseless sequence of bits generated by certain CA cell assuming, in general, a nonlinear configuration rule;
   (b) algorithm from [26] based on error-free next CA state assuming a nonlinear configuration rule;
   (c) algorithm from [28] based on the sequence of noisy CA (PCA) states assuming an additive configuration rule;
   (d) algorithm from [29] based on the noisy sequence of bits sampled from CA (PCA) states assuming an additive configuration rule.

   It can be directly shown that all these methods for reconstruction of certain CA (PCA) state can not work in the case of $h^*(\cdot)$.
4. The compression function is a cryptographic transformation.

Facts 1-4 imply that $h^*(\cdot)$ could be considered as a cryptographically secure one-way function, so that according to the Assumption 1 the following hold:

- finding preimage for given $h^*(\cdot)$ output requires about 2^n operations (i.e. testing of 2^n hypothesis);

- finding collision for $h^*(\cdot)$ requires about $2^{n/2}$ operations (testing of $2^{n/2}$ hypothesis).

Security of Output Function $g^*(\cdot)$

Recall that the output function $g^*(\cdot)$ is realized by a variant of the key stream generator proposed and analyzed in [29].

Cryptographic security examination of this generator shows that this generator is resistant on all attacks known so far, assuming that the length of employed PCA is greater than 120, [29].

Accordingly, we can accept that the output function $g^*(\cdot)$ is the secure one, and that finding the input argument of $g^*(\cdot)$ (preimage or 2nd preimage), i.e., the value H_m for given hash value $hash^*(M)$ has complexity 2^n assuming that $n > 120$.

Due to the same reasons, i.e., because $g^*(\cdot)$ is realized by the cryptographically secure key stream generator, we can accept that no better attack than the Yuval's birthday attack, [2], can be expected for finding the collisions for the output function. The previous implies that finding a collision for $g^*(\cdot)$ requires testing about $2^{n/2}$ hypothesis, i.e. employing about $2^{n/2}$ operations.

6.2 Complexity Analysis

As the first, note that the set of functions $\varphi_k(\cdot)$ (see the Section 5.1) can be efficiently realized by the truth tables in ROM.

Based on the structure of the compression function $h^*(\cdot)$ it can be directly shown that processing of each $2n$-bits message block employs no more than $n + 3n + 3n = 7n \ mod 2$ additions (recalling that updating of each CA cell employ 2 or 3 $mod 2$ additions), and approximately no more than n reading from ROM.

Similarly, it can be directly shown that the processing cost in the output function $g^*(\cdot)$ (for its n-bits input) is approximately equal to $3n^2 \ mod 2$ additions $+ \ n \ mod \ \eta$ additions + realization of the permutation.

Accordingly, the overall complexity of processing (hashing) a message consisting of m blocks and each $2n$-bits long, can be estimated as approximately equal to performing $m(8n) + 3n^2 \ mod 2$ additions (including the ROM reading costs, $mod \ \eta$ additions, and realization of the permutation). So, the proposed hash function employs the number of operations approximately equal to $4 + \frac{3n}{2m}$ $mod 2$ additions for hashing each message bit.

Also, it can be directly shown that, according to the previous result, the proposed hash function is significantly faster than all other dedicated hash functions published so far, assuming hashing of a binary string, i.e. the situations where, due to certain reasons, a word is equal to a bit which appear in hardware implementations. Finally, note that hardware implementation could be realized using 4k ROM (assuming $\ell = 4$ and $K = 3$), two VLSI PCA chips and moderate complexity control logic.

6.3 Comparison with Published CA Based Hash Functions

The novel proposal will be compared with the proposals from [16] and [17], only, because of the reported vulnerabilities of the scheme from [4] (see [16]).

The novel scheme employs a secure and fast PCA based key-stream generator, [29], as the output function. On the other hand, the schemes from [16] and [17] do not employ the output function block.

The compression function from [16] employs a nonlinear CA and a linear CA, and the scheme from [17] could be considered as one which employs two nonlinear CA. But, the employed nonlinear CA's belong to a class of nonlinear CA for which a procedure for inversion of the CA iterations is very recently published in [26].

The compression function, $h^*(\cdot)$, of the hash function proposed in this paper is one of the Davis-Mayer type (which is recognized as a recommended one), and it employs cascade of the nonlinear function and PCA which yields strengthening of security in comparison with employed nonlinear CA in [16] and [17]. Also, it employs an efficient nonlinear and iteration dependent injection of the message blocks into the $h^*(\cdot)$.

Note that employment of the cascade (nonlinear mapping + PCA) for a transformation of the cascade input could be considered as approximately equivalent with processing of the nonlinear CA input through a number of cycles, and that the multiple CA steps are more complex for a factor approximately equal to number of the CA clocks. So, the scheme [17] is significantly more complex than the novel scheme.

Also, it can be directly shown that complexity measured by the average number of elementary operations per bit for a message hashing by the scheme from [16], due to the employed approach of injecting the message bits into the compression function (each message bit is processed 8 times), is nearly eight times greater than the complexity of the here reported scheme.

Accordingly, the proposed hash function preserves all good characteristics of the schemes from [16] and [17], and yields improvement of the security and reduction of the complexity.

7 Conclusions

The paper addresses the problem of designing a fast one-way hash function for bits oriented applications, and it points out a new application of programmable cellular automata.

A novel hash function is proposed and its security and complexity are analyzed. The proposed hash function employs the approved model of iterative hash function with novel compression and output functions.

The proposed compression function is one of the Davis-Meyer type based on cryptographic transformation employing cellular automata, and the output function is a key stream generator, also based on cellular automata. The employment of cellular automata ensures the efficiency of the proposed hash function.

The security of the proposed hash function was analyzed through the security of the compression and output functions. The analysis, based on the so far published results, implies that the proposed hash function has ideal security, i.e., given a hash n-bits output, producing each of a preimage or 2nd preimage requires testing of approximately 2^n hypothesis, and producing of a collision requires testing of approximately $2^{n/2}$ hypothesis, assuming $n > 120$.

Assuming a message of m blocks, each with $2n$ bits, the proposed hash function employs number of operations approximately equal to $4 + \frac{3n}{2m}\ mod 2$ additions, for hashing each message bit. Accordingly it can be directly shown that the proposed hash function is significantly faster than all other dedicated hash functions published so far, assuming the bits oriented hashing.

Note that using the linear feedback shift register instead CA / PCA in the proposed hash function yields insecure hash function.

Finally, note that an extension of the proposed hash function for the word oriented applications instead of the here considered bit oriented could be also considered. Future research will be directed toward employment of the CA/PCA over the finite field GF(q) which could be more appropriate for standard word oriented applications.

References

1. A.J. Menezes, P.C. van Oorschot and S.A. Vanstone, *Handbook of Applied Cryptography*. Boca Roton: CRC Press, 1997.
2. G. Yuval, "How to swindle Rabin", *Cryptologia* vol. 3, pp. 187-190, 1979.
3. R. Merkle, "One way hash functions and DES", Advances in cryptology - CRYPTO 89, *Lecture Notes in Computer Science*, vol. 435, pp. 428-446, 1990.
4. I.B. Damgard, "A design principle for hash functions", Advances in Cryptology - CRYPTO 89, *Lecture Notes in Computer Science*, vol. 435, pp. 416-427, 1990.
5. X. Lai, "On the design and security of block ciphers", ETH Series in Information Processing, Vol. 1, J.L. Massey, Ed., Hartung-Gorre Verlag, Konstanz, 1992.
6. M. Naor and M. Yung, "Universal one-way hash functions and their cryptographic applications", Proc. 21st ACM Symp. on the Theory of Computing, ACM, pp. 387-394, 1989.
7. Y. Zheng, T. Matsumoto and H. Imai, "Structural properties of one-way hash functions", Advances in cryptology - CRYPTO 90, *Lecture Notes in Computer Science*, vol. 537, pp. 303-313, 1991.
8. L. Knudsen and B. Preneel, "Fast and secure hashing based on codes", Advances in cryptology - CRYPTO 97, *Lecture Notes in Computer Science*, vol. 1294, pp. 485-498, 1997.
9. M. Bellare, J. Kilian and P. Rogaway, "The security of cipher block chaining", Advances in cryptology - CRYPTO 94, *Lecture Notes in Computer Science*, vol. 839, pp. 341-358, 1994.
10. B. Preneel, R. Govaerts and J. Vandewalle, "Hash functions based on block ciphers: a synthetic approach", Advances in cryptology - CRYPTO 93, *Lecture Notes in Computer Science*, vol. 773, pp. 368-378, 1994.
11. R.L. Rivest, "The MD4 message-digest algorithm", Advances in cryptology - CRYPTO 90, *Lecture Notes in Computer Science*, vol. 537, pp. 303-311, 1991.

12. RFC 1321,"The MD5 message-digest algorithm", Internet request for comments 1321, R.L. Rivest, April 1992.
13. FIPS 180-1, "Secure hash standard", Federal Information Processing Standards Publication 180-1, U.S. Department of Commerce / NIST, 1995.
14. Integrity Primitives for Secure Information Systems: Final Report of RACE Integrity Primitives Evaluation RIPE-RACE 1040. *Lecture Notes in Computer Science*, vol. 1007, 1995.
15. Y. Zheng, J. Pieprzyk and J. Sebery, "HAVAL - a one-way hashing algorithm with variable length of output", Advances in cryptology - AUSCRYPT 92, *Lecture Notes in Computer Science*, vol. 718, pp. 83-104, 1993.
16. J. Daemen, R. Govaerts and J. Vandewalle, "A framework for the design of one-way hash functions including cryptanalysis of Damgard's one-way function based on cellular automaton", Advances in cryptology - ASIACRYPT '91, *Lecture Notes in Computer Science*, vol. 739, 1993.
17. 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)*, Matsuyama, Japan, Dec. 1997, Proc. vol. 1, pp. 213-216.
18. S. Wolfram, "Cryptography with Cellular Automata", Advances in cryptology - CRYPTO 85, *Lecture Notes in Computer Science*, vol. 218, pp. 429-432, 1985.
19. A.K. Das, A. Ganguly,A. Dasgupta, S.Bhawmik, and P. Pal Chaudhuri, "Efficient characterization of cellular automata", *IEE Proc. Pt. E*, vol. 137, pp. 81-87, Jan. 1990.
20. K. Catteell and J.C. Muzio, "Synthesis of one-dimensional linear hybrid cellular automata", *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 325-335, March 1996.
21. S. Wolfram, *Celular Automata and Complexity.* Reading MA: Addison-Wesley, 1994.
22. P.P. Chaudhuri, D.R. Chaudhuri, S. Nandi and S. Chattopadhyay, *Additive Cellular Automata: Theory and Applications.* New York: IEEE Press, 1997.
23. S. Nandi, B.K. Kar and P. Pal Chaudhuri, "Theory and applications of cellular automata in cryptography", *IEEE Trans. Comput.*, vol. 43, pp.1346-1357, 1994.
24. W. Meier and O. Staffelbach, "Analysis of pseudo random sequences generated by cellular automata", Advances in Cryptology - EUROCRYPT 91, *Lecture Notes in Computer Science*, vol. 547, pp. 186-189, 1992.
25. S.R. Blackburn, S. Murphy and K.G. Peterson, "Comments on "Theory and Applications of Cellular Automata in Cryptography"", *IEEE Trans. Comput.*, vol. 46, pp. 637-638, May 1997.
26. C.K. Koc and A.M. Apohan, "Inversion of cellular automata iterations", *IEE Proc. - Comput. Digit. Tech.*, vol. 144, pp. 279-284, 1997.
27. M. Mihaljević, "Security examination of certain cellular automata based key stream generator", *ISITA 96 - 1996 IEEE Int. Symp. Inform. Theory and Appl.*, Canada, Victoria, B.C., Sept. 1996, Proc. pp. 246-249.
28. M. Mihaljević, "Security examination of a cellular automata based pseudorandom bit generator using an algebraic replica approach", Applied Algebra, Algorithms and Error Correcting Codes - AAECC 12, *Lecture Notes in Computer Science*, vol. 1255, pp. 250-262, 1997.
29. M. Mihaljević, "An improved key stream generator based on the programmable cellular automata", Information and Communication Security - ICICS '97, *Lecture Notes in Computer Science*, vol. 1334, pp. 181-191, 1997.

This article was processed using the LaTeX macro package with LLNCS style