computing@computingonline.net
www.computingonline.net

ISSN 1727-6209
International Journal of Computing

THE GENERALIZED CONSTRUCTION OF PSEUDONONDETERMINISTIC HASHING

Volodymyr A. Luzhetsky, Yuriy V. Baryshev

Information protection department of Vinnytsia National Technical University,
Khmelnytske shosse, 95, Vinnytsia, Ukraine,
yuriy.baryshev@gmail.com

Abstract: *This article is devoted to the development of hash constructions, which are based on the pseudonondeterministic hash conception. The conception allows to design hash functions with improved infeasibility to cryptanalysis. Both proposed constructions and known ones are generalized as pseudonondeterministic constructions.*

Keywords: *hashing, multipipe, the pseudonondeterminancy,the automaton, the cryptography.*

1. INTRODUCTION

The requirement of the cryptographic algorithms publicity is brought forward by the cryptographic tools market. It is caused by the customer's desire of being able to insure these tools efficiency before they buy them. At once it is known that the private cryptographic algorithm is more difficult to break than the public one [1]. If algorithm infeasibility is proven theoretically, then this publicity doesn't cause problems, but algorithms of this type seldom gain much popularity and spreading within practical implementations, because they are based on operations, that are unnatural for the computing machinery. The publicity of cryptographic algorithms, infeasibility of which isn't theoretically proven, let talk about their practical infeasibility in the case these algorithms weren't broken by known methods. Still this kind of infeasibility couldn't be guaranteed in the future, when new methods of breaking appear. Thereafter the cryptanalytic's knowledge of round transformation, which infeasibility is proven only practically, makes it easier to study and break the hash function. That's why the development of hash functions, which would provide both the publicity of algorithms, those implement it, and round transformation nondeterminancy to the intruder during certain round cryptanalysis is topical.

The goal of the research is improving of hash infeasibility by public hashing approaches development, which would provide round transformation nondeterminancy to a cryptanalytic, and their implementation by the generalized hash construction.

The following tasks are to be solved to achieve the goal:

- known hash approaches analysis;
- the development of the new hash conception, which is based on the nondeterministic automaton;
- the development of the generalized hash construction which is based on the pseudonondeterministic hash conception.

2. ANALYSIS OF MODERN HASH APPROACHES

The majority of modern hash approaches reaches by their roots the Merkle-Damgaard construction [2], which became the classical one. In correspondence with the construction hashing is to be meant process of gaining fixed length hash digit according to the arbitrary length message, which is spitted up to the l data blocks of the same length. The initialization vector h_0 usually is used as a key. Thus the hash iteration has the following view [2]:

$$h_i = f\left(h_{i-1}, m_i\right), \qquad (1)$$

where h_i – the intermediate hash digit, that is obtained after the i-th data block processing;

m_i – the i-th data block ($i = \overline{1, l}$);

$f(\cdot)$ – the reduction function, which provides fixed length of the output value.

The hash digit of the whole message would be h_l [2]. A lot of other hash constructions are known, which differ from the Merkle-Damgaard

construction, but all of them stipulate fixed sequence of actions, that are to be performed for hash digit computing. It allows cryptanalytics to analyze algorithms and find weaknesses in hashing, which appeared at the constructions level.

Several works were performed to avoid these weaknesses within hashing [3-5]. The work [3] is devoted to the hash construction development, that contains two nonstandard hash parameters and could be formalized in this way:

$$h_i = f\left(h_{i-1}, m_i, \#bits, c\right), \quad (2)$$

where $\#bits$ – the quantity of data hashed so far;
c – the certain constant – the cryptographic salt.

According to the work [3] the constant c in the hash construction (2) is new for an each hash process. Thus the construction has certain parameter, which is to be chosen at random. That's why it is more difficult for the cryptanalytic to prepare attack beforehand, when the parameter's value is unknown. Nevertheless, the hash algorithm, because of being known, could be an object of cryptanalytic's research and known methods of breaking or their modifications could be implemented. Moreover, the approach reduces hashing rapidity comparatively with the construction (1), because the additional data is to be processed.

The work [4] contains the brief description of two hash functions Dynamic SHA and Dynamic SHA-2, the peculiarity of which is using of the data driven shift. Also cryptanalysis of these functions is performed at the work [4] and the attack, that allows to break them, is proposed. The attack could succeed through fixed structure of hash algorithms and their publicity.

The work [5] is devoted to the hashing, based on data driven primitives: permutations and substitutions. These primitives have two types of inputs: one – for a data to be hashed, other – for driven data, which value determines the type of the substitution or the permutation is to be performed under the data from the former input at each round. The driven hash primitives, which manipulate input data of a small width (2-3 bits of input/output data and 1-2 bits of driven data), are considered at the work [5]. These primitives are cascaded in order to implement more complicated elements. The data permutation is used for correspondence maintaining between different bits of a data block and for arbitrary dependence gaining between each input data bit and output bits. Thereby authors of the work [5] use several reduction functions, each of them is chosen at the certain round depending on the driven vector value:

$$h_i = f_{v_i}\left(h_{i-1}, m_i\right), \quad (3)$$

where v_i – the driven vector, which value is computed using the following function:

$$v_i = g\left(v_{i-1}, k, m_i\right), \quad (4)$$

where k – the key data.

The peculiarity of the hash construction (3) is composed of forming driven vector using both the key and input data, that allows authors of the work [4] to achieve nonlinearity of functions based on these primitives usage. Notwithstanding the large amount of developed substitution-permutation networks and the generalized methodology of their developing, the work [4] has a weakness – it also could be an object of the cryptanalysis, because the driven vector is formed with the participation of the input data. In the other hand there wouldn't be much correlation between the input and output data of the reduction function without the participation and this is rather unwanted for cryptographic hash functions [1, 2].

3. THE CONCEPTION OF PSEUDONONDETERMINISTIC HASHING

The conception, which is based on the pseudonondeterministic approach [6], was proposed to avoid weaknesses of modern approaches of round transformation hiding from cryptanalytics, which was described supra. The hash algorithm is considered as an actions sequence, that is performed by the automaton, states of which are intermediate hash values $\{h_i\}$, and input signals are data blocks $\{m_i\}$, those are to be hashed. Thereafter the set of all possible data blocks is the alphabet of the automaton, and all possible sets of data blocks – rows of the alphabet.

An automaton is deterministic one if it proceeds the one and only one state from any current state for any input symbol of its alphabet [7]. A deterministic automaton is described by the set $\{\mathbf{S}, \mathbf{A}, \delta, s_0, \mathbf{D}\}$, where $\mathbf{S}$ – the set of the automaton's states; $\mathbf{A}$ – the alphabet of input symbols; δ – the function, that implements the mapping $\mathbf{S} \times A \to \mathbf{S}$, s_0 – the initial state of the automaton ($s_0 \in \mathbf{S}$); $\mathbf{D}$ – the subset of the set $\mathbf{S}$, which is called set of final states [8]. Thus an automaton, that performs deterministic hashing could be described as the set $\{\mathbf{H}, \mathbf{M}, f(\cdot), h_0, h_l\}$, where $\mathbf{H}$ – the set of all intermediate hash values; $\mathbf{M}$ – the set of all data blocks values; h_l – the final

hash state, which is obtained after finishing of the last *l*th data block of the input message processing, $h_l \in \mathbf{H}$.

It is obvious, that the infeasibility of modern key hash algorithms is based on the point, that an intruder doesn't know the initial state h_0 of the deterministic automaton, gaining which the intruder breaks the hash function. The intruder deals with the deterministic automaton, while trying to break hash algorithms, those don't use a key, the main difficulty in the case concerns finding a path of the same length as the original message l $\{h'_1, h'_2, ..., h'_l\}$, (where $h'_i = f(h'_{i-1}, m'_i)$, and $h'_0 = h_0$), which has the same final state $h'_l = h_l$. The situation, that hash breaking task adds up to this one is caused by the determinancy of the automaton – each instant automaton's state (h_i, m_i) causes unambiguous automaton proceeding into the next state h_{i+1}. This feature of hashing makes effective the set of attacks, which use precomputation and are considered in particular at the work [3].

A nondeterministic automaton is the one, the proceeding rule of which isn't obligatory described by a function. The automaton could proceed several different states, when it stands at the certain state s_i and the same input symbol is processed [7]. Let ε be an empty message (zero-length), then a nondeterministic automaton is described by the set $\{\mathbf{S}, \mathbf{A}, \delta', s_0, \mathbf{D}\}$, which is analogical to the one of a deterministic automaton, where δ' is mapping $\mathbf{S} \times (A \cup \{\varepsilon\}) \to \mathbf{S}$ [8]. It is obviously, that deterministic automaton is the special case of the nondeterministic one.

The task of hash breaking could be more difficult if hashing process would be described by the nondeterministic automaton model. In the case it is more difficult to an intruder to find collisions, because he doesn't have the information about automaton proceeding to the next state, while he knows the instant state (h_i, m_i). It is clear that the nondeterministic automaton couldn't be used for the hash digit computation in the pure state (as is), because the same message would have several hash digits in this case, that contradicts the definition and the purpose of the hashing process. That's why the methods of pseudonondeterministic hashing are proposed. The hashing process is described for the intruder by the model similar to the nondeterministic automaton model according to these methods. So the automaton, which implements pseudonondeterministic hashing is describe by the set $\{\mathbf{H}, \mathbf{M}, \mathbf{F}, h_0, h_l\}$, where $\mathbf{F}$ – the set of functions $\{f_{v_i}(\cdot)\}$, which could be performed by the automaton at the each round depending on the driven vector's value v_i. The vector should be hidden from the intruder in the case of keyed hashing, and it should significantly depend on the input data in the case of unkeyed hashing.

4. PSEUDONONDETERMINISTIC HASH CONSTRUCTIONS

The construction (3) could be similar to the Merkle-Damgaard construction for pseudonondeterministic hashing, because it anticipates several reduction functions which are chosen according to the driven vector's value. However it is needed to exclude driven vector depending on its previous value and the input data from the formula (4). Instead, it is proposed to generate driven vector using intermediate hash values to keep the dependence on the input data. So the vector forming function has the following view:

$$v_i = g(h_{i-1}). \tag{5}$$

As both the reduction function and the vector forming function at the construction, which is described by formulas (3) and (5), consider output receiving depending on the intermediate hash value received at the previous iteration, that's why it is impossible for the hashing process based on the construction (3), (5) to achieve pure pseudonondeterminancy, because the same round transformation would be always performed for the certain intermediate hash value. It is proposed to avoid the undesirable dependence by vector forming and the hash value computation depending on different intermediate hash values. In the simplest case the hash construction is the following:

$$\begin{cases} h_i = f(h_{i-1}, m_i); \\ v_i = g(h_{i-2}). \end{cases} \tag{6}$$

The construction (6) is generalized in the following way:

$$\begin{cases} h_i = f_{v_i}(\mathbf{H}_i^*, m_i); \\ v_i = g(\mathbf{H}_i^{**}), \end{cases} \tag{7}$$

where $\mathbf{H}_i^*$, $\mathbf{H}_i^{**} \subset \{h_0, h_1, ..., h_{i-1}\}$ and $\mathbf{H}_i^* \cap \mathbf{H}_i^{**} = \varnothing$.

The construction (7) is inconvenient from the practical point of view, because in the case of keyed

hashing the key should have the length exceeded the output hash value's one. That's why authors find it more perspective to implement pseudonondeterministic hashing for the multipiped case of hashing. The instance of the proposed approach implementation for the double-pipe hashing is the following:

$$\begin{cases} h_i^{(1)} = f_{v_i^{(1)}}\left(h_{i-1}^{(1)}, m_i\right); \\ h_i^{(2)} = f_{v_i^{(2)}}\left(h_{i-1}^{(2)}, m_i\right); \\ v_i^{(1)} = g^{(1)}\left(h_{i-1}^{(2)}\right); \\ v_i^{(2)} = g^{(2)}\left(h_{i-1}^{(1)}\right), \end{cases} \quad (8)$$

where $h_i^{(j)}$ – the intermediate hash value, which is received at the *j*th pipe at the *i*th iteration.

The generalization of the construction (8) for the arbitrary number of pipes q $(q \geq 2)$ is the following:

$$\begin{cases} h_i^{(1)} = f_{v_i^{(1)}}\left(\mathbf{H}_i^{*(1)}, m_i\right); \\ h_i^{(2)} = f_{v_i^{(2)}}\left(\mathbf{H}_i^{*(2)}, m_i\right); \\ \ldots \\ h_i^{(q)} = f_{v_i^{(q)}}\left(\mathbf{H}_i^{*(q)}, m_i\right); \\ v_i^{(1)} = g^{(1)}\left(\mathbf{H}_i^{**(1)}\right); \\ v_i^{(2)} = g^{(2)}\left(\mathbf{H}_i^{**(2)}\right); \\ \ldots \\ v_i^{(q)} = g^{(q)}\left(\mathbf{H}_i^{**(q)}\right), \end{cases} \quad (9)$$

where $\mathbf{H}_i^{*(j)}$, $\mathbf{H}_i^{**(j)} \subset \left\{h_0^{(1)}, h_1^{(1)}, \ldots., h_{i-1}^{(1)}, h_0^{(2)}, h_1^{(2)}, \ldots., h_{i-1}^{(q)}\right\}$ and $\mathbf{H}_i^{*(j)} \cap \mathbf{H}_i^{**(j)} = \varnothing$, $j = \overline{1, q}$.

The construction (9), as most known hash constructions [2, 3, 5, 9-12], considers consequent data block processing – one block is processed at the one iteration. The approach simplifies the hash methods implementation, but simultaneously it facilitates to the cryptanalytic the task of the breaking automaton design. That's why it is proposed to use several data blocks at the each iteration. For instance, the following construction is proposed to "bond" the data block to their positions at the message, that would complicate the part of the message replacement task for the cryptanalytic:

$$\begin{cases} h_i^{(1)} = f_{v_i^{(1)}}\left(\mathbf{H}_i^{*(1)}, m_i, m_{i-rand(m_i)}\right); \\ h_i^{(2)} = f_{v_i^{(2)}}\left(\mathbf{H}_i^{*(2)}, m_i, m_{i-rand(m_i)}\right); \\ \ldots \\ h_i^{(q)} = f_{v_i^{(q)}}\left(\mathbf{H}_i^{*(q)}, m_i, m_{i-rand(m_i)}\right); \\ v_i^{(1)} = g^{(1)}\left(\mathbf{H}_i^{**(1)}\right); \\ v_i^{(2)} = g^{(2)}\left(\mathbf{H}_i^{**(2)}\right); \\ \ldots \\ v_i^{(q)} = g^{(q)}\left(\mathbf{H}_i^{**(q)}\right), \end{cases} \quad (10)$$

where $rand(\cdot)$ – the function of random number sequence generating.

Using of the construction (10) considers processing of two data blocks at the each iteration, but in the general case all data blocks could be processed in the following way:

$$\begin{cases} h_i^{(1)} = f_{v_i^{(1)}}\left(\mathbf{H}_i^{*(1)}, m_1, m_2, \ldots, m_l\right); \\ h_i^{(2)} = f_{v_i^{(2)}}\left(\mathbf{H}_i^{*(2)}, m_1, m_2, \ldots, m_l\right); \\ \ldots \\ h_i^{(q)} = f_{v_i^{(q)}}\left(\mathbf{H}_i^{*(q)}, m_1, m_2, \ldots, m_l\right); \\ v_i^{(1)} = g^{(1)}\left(\mathbf{H}_i^{**(1)}\right); \\ v_i^{(2)} = g^{(2)}\left(\mathbf{H}_i^{**(2)}\right); \\ \ldots \\ v_i^{(q)} = g^{(q)}\left(\mathbf{H}_i^{**(q)}\right). \end{cases} \quad (11)$$

The cryptographic salt (a random number) could be used as the argument of the reduction function $f(\cdot)$ in addition to data blocks and intermediate hash values, which are used at the construction (11), as it was proposed at the work [3]. However it was proposed to use the same salt value at the each iteration at the work [3], so the cryptographic salt is the static one. The static cryptographic salt is generalized by the dynamic one, so it is proposed to input new pseudorandom number at the each iteration [13, 14]:

$$\begin{cases} h_i^{(1)} = f_{v_i^{(1)}}\left(\mathbf{H}_i^{*(1)}, m_1, m_2, \dots, m_l, r_i\right); \\ h_i^{(2)} = f_{v_i^{(2)}}\left(\mathbf{H}_i^{*(2)}, m_1, m_2, \dots, m_l, r_i\right); \\ \dots \\ h_i^{(q)} = f_{v_i^{(q)}}\left(\mathbf{H}_i^{*(q)}, m_1, m_2, \dots, m_l, r_i\right); \\ v_i^{(1)} = g^{(1)}\left(\mathbf{H}_i^{**(1)}\right); \\ v_i^{(2)} = g^{(2)}\left(\mathbf{H}_i^{**(2)}\right); \\ \dots \\ v_i^{(q)} = g^{(q)}\left(\mathbf{H}_i^{**(q)}\right); \\ r_i = rand\left(r_{i-1}\right), \end{cases} \quad (12)$$

where r_i – the dynamic cryptographic salt and r_0 is determined before beginning of the hash process in the same way as the static salt is determined.

The dynamic cryptographic salt is the same for each pipe at the construction (12), but it could be different for the each pipe in the general case:

$$\begin{cases} h_i^{(1)} = f_{v_i^{(1)}}\left(\mathbf{H}_i^{*(1)}, m_1, m_2, \dots, m_l, r_i^{(1)}\right); \\ h_i^{(2)} = f_{v_i^{(2)}}\left(\mathbf{H}_i^{*(2)}, m_1, m_2, \dots, m_l, r_i^{(2)}\right); \\ \dots \\ h_i^{(q)} = f_{v_i^{(q)}}\left(\mathbf{H}_i^{*(q)}, m_1, m_2, \dots, m_l, r_i^{(q)}\right); \\ v_i^{(1)} = g^{(1)}\left(\mathbf{H}_i^{**(1)}\right); \\ v_i^{(2)} = g^{(2)}\left(\mathbf{H}_i^{**(2)}\right); \\ \dots \\ v_i^{(q)} = g^{(q)}\left(\mathbf{H}_i^{**(q)}\right); \\ r_i^{(1)} = rand^{(1)}\left(r_{i-1}^{(1)}\right); \\ r_i^{(2)} = rand^{(2)}\left(r_{i-1}^{(2)}\right); \\ \dots \\ r_i^{(q)} = rand^{(q)}\left(r_{i-1}^{(q)}\right). \end{cases} \quad (13)$$

The dynamic cryptographic salt could be used several times at the each iteration similar to the data blocks at the construction (11). Consequently the construction (13) is generalized by the following construction:

$$\begin{cases} h_i^{(1)} = f_{v_i^{(1)}}\left(\mathbf{H}_i^{*(1)}, m_1, m_2, \dots, m_l, \mathbf{R}^{(1)}\right); \\ h_i^{(2)} = f_{v_i^{(2)}}\left(\mathbf{H}_i^{*(2)}, m_1, m_2, \dots, m_l, \mathbf{R}^{(2)}\right); \\ \dots \\ h_i^{(q)} = f_{v_i^{(q)}}\left(\mathbf{H}_i^{*(q)}, m_1, m_2, \dots, m_l, \mathbf{R}^{(q)}\right); \\ v_i^{(1)} = g^{(1)}\left(\mathbf{H}_i^{**(1)}\right); \\ v_i^{(2)} = g^{(2)}\left(\mathbf{H}_i^{**(2)}\right); \\ \dots \\ v_i^{(q)} = g^{(q)}\left(\mathbf{H}_i^{**(q)}\right), \end{cases} \quad (14)$$

where $\mathbf{R}^{(j)}$ – the set of pseudorandom numbers, which is used at the *j*th hashing pipe.

Let us use the following parametric description of hashing, which is based on the construction (14) $PNDH_q\left(k; d; z; \gamma; \phi\right)$ (PseudoNonDeterministic Hashing), where the following denotations are used:

q – the quantity of hashing pipes;

k – the quantity of intermediate hash values, which are arguments of the reduction function $f^{(j)}(\cdot)$ at the *j*th $\left(j = \overline{1, q}\right)$ pipe;

d – the quantity of data blocks, which are processed for the intermediate hash values computation at the *j*th pipe $\left(d = \overline{1, l}\right)$;

z – the quantity of pseudorandom numbers, which are used at the each iteration at the one pipe;

γ – the correlation between the intermediate hash value width and the output hash digit width;

ϕ – the quantity of intermediate hash values, which are used to form the driven vector at the *j*th pipe.

The construction (14) was received by the sequential generalization of hash constructions, that's why it generalized all constructions considered supra. Parametric descriptions of the known hash constructions is presented at the table 1 to prove, that the construction (14) is generalized for other constructions and deterministic hashing is particular case of the pseudonondeterministic one.

Table 1. Parametric descriptions of known hash constructions

The hash construction	The parametric description
Merkle-Damgaard [2]	$PNDH_1(1; 1; 0; 1; 0)$
Cascading [2]	$PNDH_q(1; 1; 0; 1; 0)$
HAIFA [3]	$PNDH_1(1; 1; 1; 1; 0)$
Hirose [10]	$PNDH_2(2; 1; 1; 2; 0)$
Double-pipe [11]	$PNDH_2(2; 1; 0; 2; 0)$
Wilde pipe [11]	$PNDH_1(1; 1; 0; 2; 0)$
3C [9]	$PNDH_2(2/1; 0/1; 0; 2; 0)$
Sponge [12]	$PNDH_1(1; 1; 0; \gamma; 0)$

Parameters k and d at the table 1 for the construction 3C are described using slash because of different arguments of pipes reduction functions. For the first pipe they are the data block and the intermediate hash value from the pipe, for the second pipe – intermediate hash values from both pipes.

5. AN EXAMPLE OF PSEUDONONDETERMINISTIC HASH FUNCTION IMPLEMENTATION

Several hash functions were developed to implement hash constructions described above [15]. These hash functions are based on well-known non-linear functions of three arguments x_1, x_2, x_3 (in particular first pair of them is used at the hashing standard SHA-2 [16]):

$$y = (x_1 \wedge x_2) \oplus (\neg x_1 \wedge x_3), \quad (15)$$

$$y = (x_1 \wedge x_2) \oplus (x_1 \wedge x_3) \oplus (x_2 \wedge x_3), \quad (16)$$

$$y = (x_1 \vee x_2) \oplus (\neg x_1 \vee x_3), \quad (17)$$

$$y = (x_1 \vee x_2) \oplus (x_1 \vee x_3) \oplus (x_2 \vee x_3). \quad (18)$$

Each iteration of hashing is supposed to use one of functions (15-18) according to the driven vector value. Also it is proposed to use circular shift operation for each argument before it would be used by one of these non-linear functions.

For instance the hash function was developed, which implements hashing $PNDH_8(5;1;0;1;3)$ with output hash digest length of 256 bits. At the start of each iteration the driven vector value is computed according to the function:

$$v_i^{(j)} = h_{i-1}^{((j+1) \bmod q)} \oplus h_{i-1}^{((j+4) \bmod q)} \oplus h_{i-1}^{((j+7) \bmod q)}. \quad (19)$$

Then the driven vector $v_i^{(j)}$ is divided into seven parts: one part of 2 bits length c and six parts of 5 bits length $s_1, s_2, ..., s_6$. The reduction function is chosen according to the c value:

- "11" – the function (15);
- "10" – the function (16);
- "01" – the function (17);
- "00" – the function (18).

At each iteration reduction function arguments m_i, $h_{i-1}^{(j)}$, $h_{i-1}^{((j+2) \bmod q)}$, $h_{i-1}^{((j+3) \bmod q)}$, $h_{i-1}^{((j+5) \bmod q)}$, $h_{i-1}^{((j+6) \bmod q)}$ are circular shifted rightwards to $s_1, s_2, s_3, s_4, s_5, s_6$ bits respectively. For example if the driven vector value is 0x01234567 it would be divided in binary form into codes: 00 00000 10010 00110 10001 01011 00111. The following reduction function $f_i^{(j)}(\cdot)$ is to be performed at this iteration:

$$\begin{aligned} h_i^{(j)} = {} & \left(m_i \wedge \left|h_{i-1}^{(j)}\right|_{>>>18}\right) \oplus \\ & \oplus \left(\neg m_i \wedge \left|h_{i-1}^{((j+2) \bmod q)}\right|_{>>>6}\right) \oplus \\ & \oplus \left(\left|h_{i-1}^{((j+3) \bmod q)}\right|_{>>>17} \wedge \left|h_{i-1}^{((j+5) \bmod q)}\right|_{>>>11}\right) \oplus \\ & \oplus \left(\left|\neg h_{i-1}^{((j+3) \bmod q)}\right|_{>>>17} \wedge \left|h_{i-1}^{((j+6) \bmod q)}\right|_{>>>7}\right). \end{aligned} \quad (20)$$

The hash function was tested by Known answer tests, which were used by NIST (USA) for SHA-3 competitors [15, 17]. For instance, the results of extremely long message testing are the following:

Repeat = 163840
Text = abcdefghbcdefghicdefghijdefghijkefghijklfgh ijklmghijklmnhijklmno
MD = 611D589256CD92FCE44225A445B6DC1 729F953851F4DAE07D204EDC8FAF40AE9

The short message test results are the following:

Len = 0
Msg = 00
MD = C4377DE5D712E89EFF4AB66FB2D635FE3 43A6B9869C06295468D483199D4AFC8

Len = 1
Msg = 00
MD = D89E7E9FD569456A989333B4B3A3B3AB E882E038E3C3F4FE690808596F16ACDE

Len = 2
Msg = C0
MD = 66B816FF6E5EC2D52A78B31BE896D980 76EBB323A55F878139F333B8ACDF88A2

Len = 3
Msg = C0
MD = 54D4B72538B7A5D72736EE8C53A18141 7E343B34A9B134FC605E5D607CF12447

…

Results of testing showed that the hash function doesn't degenerate while extremely long or short messages are processed.

6. CONCLUSIONS

The performed analysis of known hash approaches showed, that they could be described by the deterministic automaton model, moreover it is the very feature used by cryptanalytics for many attacks performing. That's why the conception of pseudonondeterministic hashing is proposed by

authors. The conception is generalization of the deterministic hashing one. While hash algorithms are developed according to the conception, an intruder is unavailable to perform cryptanalysis of round transformations, because he doesn't know operations, which are performed to process the certain data block. Moreover the approach is proposed, which allows to hide the information about the data block being processed at the certain iteration from an intruder. The set of hash constructions is proposed to implement the pseudonondeterministic hash conception. These constructions provide hash infeasibility improving towards known attacks.

7. REFERENCES

[1] S. Burnett, S. Paine, *RSA Security's Official Guide to Cryptography*, Binom-press, Moscow, 2002, p. 384. (in Russian)

[2] B. Preneel, *Analysis and Design of Cryptographic Hash Functions,* Katholieke Universiteit Leuven, 1993, p. 323. http://homes.esat.kuleuven.be/~preneel/phd_preneel_feb1993.pdf

[3] E. Biham, O. Dunkelman, A framework for iterative hash functions, 2007, p. 9. http://csrc.nist.gov/groups/ST/hash/documents/DUNKELMAN_NIST3.pdf

[4] J-P. Aumasson, O. Dunkelman, S. Indesteege and B. Preneel, *Cryptanalysis of Dynamic SHA(2)*, COmputer Security and Industrial Cryptography publications, 2009, p. 18. https://www.cosic.esat.kuleuven.be/publications/article-1277.pdf

[5] N. A. Moldovyan, A. A. Moldovyan, M. A. Eremeev, *Cryptography: from Primitives to the Algoritms Synhesis,* BHV-Petersburgh, St. Petersburgh, 2004, p. 448. (in Russian)

[6] V. A. Luzhetsky, Y. V. Baryshev, The pseudo-nondeterministic hashing conception, *Systems of Control, Navigation and Communications*, (3) (2010), pp. 94-98. (in Ukrainian)

[7] J. A. Anderson, *Discrete Mathematics with Combinatorics*, Williams Publishing House, Moscow, 2004, p. 960. (in Russian)

[8] A. V. Aho, J. E. Hopcroft, J. D. Ullman, *The Design and Analysis of Computer Algoritms,* Mir, Moscow, 1979, p. 536. (in Russian)

[9] P. Gauravaram, *Cryptographic Hash Functions: Cryptanalysis, Design and Applications,* Thesis submitted in accordance with the regulations for Degree of Doctor of Philosophy, 2009, p. 298, http://eprints.qut.edu.au/16372/1/Praveen_Gauravaram_Thesis.pdf

[10] S. Hirose, *Some Plausible Constructions of Double-Block-Length Hash Functions*, 2006, p. 13. www.iacr.org/archive/fse2006/40470213/40470213.pdf

[11] S. Lucks, Design principles for iterated hash functions, *Cryptology ePrint Archive*, 2004, p. 22. http://eprint.iacr.org/2004/253.pdf

[12] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, *Sponge Functions*, 2007, p. 22, http://sponge.noekeon.org/SpongeFunctions.pdf

[13] Y. V. Baryshev, Methods of multipipe hash function infeasibility improving against generic attacks, *Computer Science and Engineering – 2010*, Lviv Polytechnic Publishing, Lviv, 2010, pp. 338-339. (in Ukrainian)

[14] Y. V. Baryshev. Pseudonondeterministic hashing mathematical model and cryptographic primitives for its implementation, *Information technologies and computer engineering – 2010,* VNTU, Vinnytsia, pp. 268-269. (in Ukrainian)

[15] Y. V. Baryshev, Methods and software means of multipipe driven hashing, *Methods and tools of coding, protection and compression of information-2011,* VNTU, Vinnytsia, pp. 100-101. (in Ukrainian)

[16] Secure Hash Standard: Federal Information Processing Publication Standard Publication 180-3. – Gaithersburg, 2008, p. 27, http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf

[17] Test files and Source Code for Conducting KAT and MCT, NIST, http://csrc.nist.gov/groups/ST/hash/sha-3/documents/KAT1.zip

---

Volodymyr A. Luzhetsky, *Dr. of Science, professor, head of the information protection department of Vinnytsia National Technical University.*

Scientific interests: the cryptography, data compression methods, Fibonacci codes.

Yuriy V. Baryshev*, phD, member of the information protection department of Vinnytsia National Technical University.*

Scientific interests: multipipe hashing and generic attacks, the evaluation of information protection.