

# UPDATED FAMILY OF PARAMETERIZED HASH ALGORITHMS

Przemysław Rodwald  
Military Communication Institute  
22a Warszawska Str., 05-131 Zegrze, Poland  
p.rodwald@wil.waw.pl

Janusz Stokłosa  
Poznań University of Technology  
5 Skłodowskiej Sq., 60-965 Poznań, Poland  
janusz.stoklosa@put.poznan.pl

## ABSTRACT

*Family of Parameterized Hash Algorithms – PHAL is a proposal of a new dedicated hash algorithm designed as an answer to weaknesses of MD/SHA hash function family. Recently proposed attacks on well-known and widely used hash functions motivate a design of new hash functions. In this paper new approach is presented, where few elements of hash function are parameterized. This approach makes the hash algorithm more secure and more flexible. In this paper a slightly modified version of PHAL family [12] is presented.*

## 1. INTRODUCTION

The last few years brought a great progress in cryptanalysis of hash function. Especially the results of Wang *et al.* have changed the view on security of well-known hash functions. Most hashes of MD/SHA family have succumbed to the Chinese attacks [14]-[17], many others have been broken (including “proven secure” ones). On November 2, 2007 NIST announced an open competition for a new SHA-3 function [11].

Most cryptographic hash functions of any significance, *i.e.* MD4, MD5, SHA, RIPEMD and several others, share the iteration mechanism known as Merkle-Damgård [4][9]. It does guarantee collision-resistance property provided the compression function has the same property. For years it was widely believed that MD-type construction maintains the preimage resistance and the second preimage resistance as well. Unfortunately these beliefs were questioned. Recent attacks showed several undesirable properties and vulnerability to some attacks [3], [5], [7]. It was one of the reasons for NIST to open the

new iterative designs. In 2006 Biham and Dunkleman [2] proposed a new iterative schema HAsH Iterative FrAmework (HAIFA). The main ideas of this schema are the introduction into the compression functions a number of bits that have been hashed so far, and a *salt* value.

For PHAL hash algorithm a similar approach was chosen. Additionally, the number of rounds was added as a parameter. The design goals of this hash algorithm are determined as follows:

- Hash algorithm must provide message digests of 224, 256, 384 and 512 bits and shall support a maximum message length of at least  $2^{64}-1$  bits [11].
- Its iteration structure should be resistant against known attack against the MD-type structure.
- Its compression function should be resistant against known attack.
- Its structure should be parameterized, to reach flexibility between performance and security.

The rest of the paper is organized as follows. In section 2 a complete specification of PHAL family is presented. Then explanation of design strategy is discussed (section 3). Finally, one instance of PHAL family is proposed (section 4) with performance comparison, security study and results of statistical tests.

## 2. DESCRIPTION OF PHAL FAMILY

In this section, Parameterized Hash Algorithm is described. The following notations are used in the sequel:

- $w$  : length of a word (32 or 64 bits),
- $m$  : length of the message block (512 or 1024 bits),
- $n$  : length of the chaining variable (256 or 512 bits),
- $d$  : length of the digest (224, 256, 384 or 512 bits),
- $s$  : length of the *salt* (128 or 256 bits),

- $X \boxplus Y$  : addition mod  $2^w$  of vectors  $X$  and  $Y$ ,
- $X \boxminus Y$  : subtraction mod  $2^w$  of vectors  $X$  and  $Y$ ,
- $X \oplus Y$  : bitwise XOR of vectors  $X$  and  $Y$ ,
- $X \lll/s \ggg$  :  $s$ -bit left/right rotation for a  $w$ -bit vector  $X$ ,
- $X \ll/s \gg$  :  $s$ -bit left/right shift for a  $w$ -bit vector  $X$ .

### Message Padding

The message ( $M$ ) has to be padded before hash computation begins. The length of padded message should be a multiple of  $m$  bits. The message is padded by appending a zero or a greater number of bits “0” until the length of the message is congruent to  $(m-80)$  mod  $m$ . Finally, we append 10-bit digest length  $d$  and 6-bit length value *rounds* which defined number of rounds and at the end appends original message length (mod  $2^{64}$ ). The message  $M$  is then divided into  $k$   $m$ -bit blocks  $M_0, M_1, \dots, M_{k-1}$  (Figure 1).



Figure 1. PHAL – message padding

### Iteration Schema

The hash function  $h: \{0,1\}^* \rightarrow \{0,1\}^d$  uses the compression function

$$\varphi: \{0,1\}^n \times \{0,1\}^m \times \{0,1\}^c \times \{0,1\}^s \times \{0,1\}^{rs} \rightarrow \{0,1\}^d,$$

where:  $c$  is the size of counter, *i.e.* the number of bits hashed so far taken modulo  $2^{64}$  (64 bits),  $rs$  denotes the size of number of *rounds* (6 bits).

The process of hashing looks as follows:

$$CV_0 = IV;$$

$$H_{i+1} = \varphi(CV_i, M_i, counter, salt, rounds); 0 \leq i \leq k-1;$$

$$h(M) = CV_k;$$

where  $\varphi$  is the compression function of  $h$ ,  $H_i$  is the chaining variable and  $IV$  denotes the initial value. Number of *rounds* and *salt* are values defined by the user, where *salt* should be random or pseudorandom value. Number of rounds should not be smaller than a certain

value defined after security and statistical analysis of each instance.

### Message Modification

Each  $m$ -bit message block  $M_i$  is divided into sixteen  $w$ -bit words  $m_i[0], \dots, m_i[15]$ . Before each round  $r$ , except the first one  $2 \leq r \leq round$ , words are modified three times using the following schema. Before the first round substitution  $w_i^1 := m_i$  must be performed. Before the second round substitution  $w_i^2 := w_i^1$  must be done.

$$\begin{aligned} w_i^x[0] &:= w_i^x[0] & \boxplus w_i^x[15] \oplus MMConst_1 \\ w_i^x[1] &:= w_i^x[1] & \boxplus w_i^x[0] \\ w_i^x[2] &:= w_i^x[2] & \oplus w_i^x[1] \oplus salt[0] \\ w_i^x[3] &:= w_i^x[3] & \boxplus w_i^x[2] \oplus (-w_i^x[1] \lll MMLR_1) \\ w_i^x[4] &:= w_i^x[4] & \boxplus w_i^x[3] \\ w_i^x[5] &:= w_i^x[5] & \oplus w_i^x[4] \\ w_i^x[6] &:= w_i^x[6] & \boxplus w_i^x[5] \oplus salt[1] \\ w_i^x[7] &:= w_i^x[7] & \boxplus w_i^x[6] \oplus (-w_i^x[5] \ggg MMRR_1) \\ w_i^x[8] &:= w_i^x[8] & \boxplus w_i^x[7] \oplus MMConst_2 \\ w_i^x[9] &:= w_i^x[9] & \boxplus w_i^x[8] \\ w_i^x[10] &:= w_i^x[10] & \oplus w_i^x[9] \oplus salt[2] \\ w_i^x[11] &:= w_i^x[11] & \boxplus w_i^x[10] \oplus (-w_i^x[9] \lll MMLR_2) \\ w_i^x[12] &:= w_i^x[12] & \boxplus w_i^x[11] \\ w_i^x[13] &:= w_i^x[13] & \oplus w_i^x[12] \\ w_i^x[14] &:= w_i^x[14] & \boxplus w_i^x[13] \oplus salt[3] \\ w_i^x[15] &:= w_i^x[15] & \boxplus w_i^x[14] \oplus (-w_i^x[13] \ggg MMRR_2) \end{aligned}$$

$MMConst_1$  and  $MMConst_2$  are two  $w$ -bit message modification constants.  $MMLR_1$  and  $MMLR_2$  are two message modification left rotation values.  $MMRR_1$  and  $MMRR_2$  are two message modification right rotation values.

As a result a modified message  $w_i^r$  is obtained. This message is then used in the round function and is used as an input for message modification for the next round. Both branches  $BRANCH_b$ , for  $0 \leq b \leq 1$ , use modified message words with different order  $\sigma_b$ . Each branch uses each message word twice (Table 1).

Table 1. Message word ordering

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_0(t)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_1(t)$	15	11	4	12	2	14	13	9	3	7	0	5	10	6	8	1
t	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$\sigma_0(t)$	14	10	9	13	11	15	12	8	6	2	1	4	7	3	5	0
$\sigma_1(t)$	5	8	7	6	1	0	3	10	13	12	15	14	9	4	11	2

### Constants

PHAL uses sixteen  $w$ -bit constants  $\Omega[0], \dots, \Omega[15]$ . Some of them are not predefined, but they are set with parameters *salt* and *counter*:

$$\Omega[0] = salt[0], \Omega[4] = salt[1] \oplus counter,$$

$\Omega[8] = \text{salt}[2]$ ,  $\Omega[12] = \text{salt}[3] \oplus \text{counter}$ .

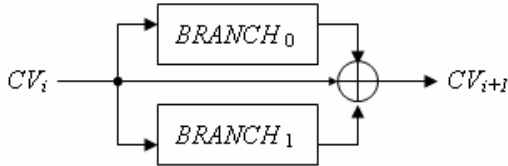
These constants are used in each branch with different order  $\rho_b$ , for  $0 \leq b \leq 1$  (Table 2).

**Table 2.** Constants word ordering

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\rho_0(t)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\rho_1(t)$	13	8	12	9	15	14	10	11	6	0	2	4	1	5	7	3

### Compression Function

The outline of the compression function of PHAL family is presented below (Figure 2). Compression function consists of two parallel branches  $BRANCH_0$  and  $BRANCH_1$ . Let  $CV_i$  be the chaining variable of the compression function.



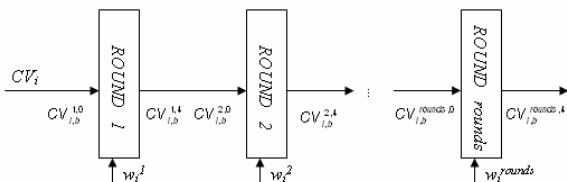
**Figure 2.** PHAL - compression function

The following statement is true:

$$CV_{i+1} = CV_i \oplus BRANCH_0(CV_i) \oplus BRANCH_1(CV_i).$$

### Branch Function

Each branch  $BRANCH_b$ , for  $0 \leq b \leq 1$ , consists of  $rounds$  round functions, each called  $ROUND$ , where  $rounds$  is a parameter defined by the user (Figure 3).



**Figure 3.** PHAL - branch function

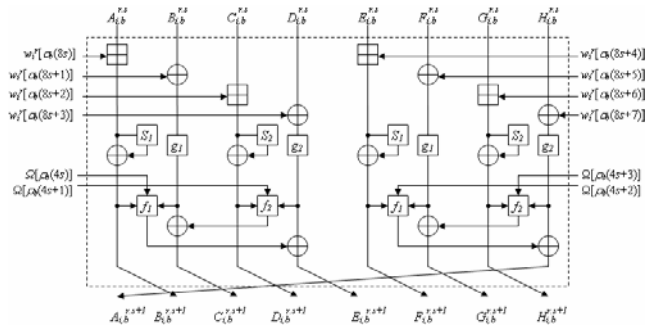
Let  $CV_{i,b}^{r,s} = (A_{i,b}^{r,s}, B_{i,b}^{r,s}, \dots, H_{i,b}^{r,s})$  be the chaining variable inside branch function. For it:

- $CV_{i+1} = CV_{i,0}^{rounds,4} \oplus CV_{i,1}^{rounds,4} \oplus CV_i$ ,
- $CV_{i,b}^{1,0} = CV_i$ , for  $0 \leq b \leq 1$ ,
- $CV_{i,b}^{r+1,0} = CV_{i,b}^{r,4}$  for  $1 \leq r \leq round-1$ ,  $0 \leq b \leq 1$ .

### Round Function

Each round consists of four steps ( $STEP_{i,b}^{r,s}$ , for  $0 \leq s \leq 3$ ). Step function (Figure 4) is computed as follows:

$$\begin{aligned} B_{i,b}^{r,s+1} &= S_1(A_{i,b}^{r,s} \oplus w_i^r[\sigma_b(8s)]) \oplus (A_{i,b}^{r,s} \oplus w_i^r[\sigma_b(8s)]), \\ C_{i,b}^{r,s+1} &= g_1(B_{i,b}^{r,s} \oplus w_i^r[\sigma_b(8s+1)]) \oplus f_2(D_{i,b}^{r,s+1}, \Omega[\alpha_b(4s+1)], E_{i,b}^{r,s+1}), \\ D_{i,b}^{r,s+1} &= S_2(C_{i,b}^{r,s} \oplus w_i^r[\sigma_b(8s+2)]) \oplus (C_{i,b}^{r,s} \oplus w_i^r[\sigma_b(8s+2)]), \\ E_{i,b}^{r,s+1} &= g_2(D_{i,b}^{r,s} \oplus w_i^r[\sigma_b(8s+3)]) \oplus f_1(B_{i,b}^{r,s+1}, \Omega[\alpha_b(4s)], C_{i,b}^{r,s+1}), \\ F_{i,b}^{r,s+1} &= S_1(E_{i,b}^{r,s} \oplus w_i^r[\sigma_b(8s+4)]) \oplus (E_{i,b}^{r,s} \oplus w_i^r[\sigma_b(8s+4)]), \\ G_{i,b}^{r,s+1} &= g_1(F_{i,b}^{r,s} \oplus w_i^r[\sigma_b(8s+5)]) \oplus f_2(H_{i,b}^{r,s+1}, \Omega[\alpha_b(4s+3)], A_{i,b}^{r,s+1}), \\ H_{i,b}^{r,s+1} &= S_2(G_{i,b}^{r,s} \oplus w_i^r[\sigma_b(8s+6)]) \oplus (G_{i,b}^{r,s} \oplus w_i^r[\sigma_b(8s+6)]), \\ A_{i,b}^{r,s+1} &= g_2(H_{i,b}^{r,s} \oplus w_i^r[\sigma_b(8s+7)]) \oplus f_1(F_{i,b}^{r,s+1}, \Omega[\alpha_b(4s+2)], G_{i,b}^{r,s+1}). \end{aligned}$$



**Figure 4.** PHAL - step function

In the step function the following functions are used:

- $g_1(x) = x \oplus (x \lll GIL1) \oplus (x \lll GIL2)$ ,
- $g_2(x) = x \oplus (x \lll G2L1) \oplus (x \lll G2L2)$ ,
- $f_1(x,y,z) = xy \oplus (-x)z$ ,
- $f_2(x,y,z) = xy \oplus xz \oplus yz$ ,
- $S_1(x) = SBox1(x \gg 2(w/4)) \oplus SBox0(x \gg 3(w/4))$

where  $S_2(x) = SBox1(x) \oplus SBox0(x \gg (w/4))$ , where  $GIL1$  and  $GIL2$  are two left rotation values for function  $g_1$ ,  $G2L1$  and  $G2L2$  are two left rotation values for function  $g_2$ ,  $SBox1$  and  $SBox0$  are two S-boxes of dimension  $(w/4) \times w$ .

## 3. DESIGN STRATEGY

### Message Padding

As a message padding method, the algorithm without padding single bit 1 was chosen. As was shown by Johnson [6], adding this single bit "1" has no influence on security. For messages with the size equal exactly to  $(m-80) \bmod m$  bits, the performance cost for a hash function decreases.

### Iteration Schema

The number of bits hashed so far ( $counter$ ) was added to increase the resistance of hash function to fixed-points attacks. The random value  $salt$  increases resistance of hash function to attacks, which use precomputation table generated in advance (message - hash value).

Number of rounds (*rounds*) was added to make this function more flexible. There is a trade-off between performance and security. Small number of rounds should be used in systems where performance is most important. When security is the most important factor, a greater number of rounds should be used. More security factors connected with parameters *salt* and *counter* can be found in HAIFA design analysis [2].

### Initial Value and Constants

In many hash functions, as an initial value or constants, the first thirty two bits (sixty four, respectively) of the fractional parts of the square or cube roots of the first prime numbers is taken. In PHAL as an initial value and constants a balanced vector was chosen. Hamming weight of each word is equal to 16 (32), and more generally  $w/2$ . Hamming weight of each bit position is equal to 4 in initial value and equal to 6 in constants.

### Message Modification

Most hash functions could be classified in one of two groups, according to message word input methods. In the first group, words of the original message are permuted, before every round (MD4, MD5, RIPEMD, FORK-256). In the second group input message words are computed using a message expansion function (SHA, TIGER). Functions from the first group may not be secure against Wang *et al.* attacks [8]. In PHAL family message is modified using block-cipher key schedule philosophy. Similar approach is in the TIGER hash function [1], and generally in block ciphers. Values of left and right rotation, inside message modification algorithm, were chosen after investigation of all possible odd values (in PHAL-256). They were chosen in such a way, to reach strict avalanche criterion. Two constants  $MMConst_1$  and  $MMConst_2$  were used to balance the sparse messages.

### Modified Message Word Ordering

In addition to the fact that an original message is modified before computation, each branch has its own message order. This was done as an answer to Wang *at al.*'s attacks against RIPEMD family [14]. RIPEMD-160, due to different message-ordering in branches,

is still not broken by their attacks. If an attacker will construct an intended differential characteristics for one branch, the different word order in the second branch will cause unintended differential patterns. The order of message words was chosen with respect to balancedness in upper, lower, left and right part.

### S-boxes

Two S-boxes were generated with the following parameters: high nonlinearity, balancedness and good XOR profile.

### One-argument Functions – g

Functions  $g_1$  and  $g_2$  output one word with one input word. For PHAL instances, where the length of digest is equal to 224 or 256, all possible functions  $g(x) = x \oplus (x \ll n) \oplus (x \ll m)$ , for  $n, m \in \{1, \dots, 31\}$  and all ( $2^{32}$ ) possible values of input vector were investigated. Values of shift rotations were chosen from sets satisfying the following conditions:

1. if  $HW(x) = 1$ , then  $HW(g(x)) \geq 2$ ,
2. if  $HW(x) = 2$ , then  $HW(g(x)) \geq 4$ ,
3. if  $HW(x) = 3$ , then  $HW(g(x)) \geq 3$ ,
4.  $n$  and  $m$  are not divisors of 32,
5.  $4 < n < 28$ ,  $4 < m < 28$ ,
6.  $|n - m| > 8$ ,
7. if  $n$  is even, then  $m$  is odd,
8. if  $m$  is even, then  $n$  is odd,

where  $HW$  means the Hamming weight. By the above conditions, functions  $g_1$  and  $g_2$  were chosen.

### Chaining Value

In the most popular hash functions many words of chaining variable are not modified in a single step. They are just copied. Additionally, output words of Boolean functions are used to update only one chaining variable. The situation in PHAL family is different. Each word of chaining variable is modified in a single step at least twice: once with the help of the message word and once with the help of the function:  $g$  or  $f$  or  $S$ .

## 4. PHAL-256

### Definition

One of the instances of PHAL family will be introduced in this section [13]. It is the case (called PHAL-256) in which the length of the digest is equal 256. PHAL-256 has following parameters:

- $w = 32$  (length of a word),
- $m = 512$  (length of the message block),
- $n = 256$  (length of the chaining variable),
- $d = 256$  (length of the digest),
- $s = 128$  (length of the *salt*).

$IV_A=0 \times 6A09E667$ ,  $IV_B=0 \times BB67AE85$ ,  $IV_C=0 \times 3C6EF372$ ,  
 $IV_D=0 \times A54FF53A$ ,  $IV_E=0 \times 510E527F$ ,  $IV_F=0 \times 9B05688C$ ,  
 $IV_G=0 \times 1F83D9AB$ ,  $IV_H=0 \times 5BE0CD19$ .

$MMConst_1 = 0 \times 5FA2C0D3$ ,  $MMConst_2 = 0 \times B1487E96$ .  
 $MMLR_1 = 1$ ,  $MMLR_2 = 13$ ,  $MMRR_1 = 5$ ,  $MMRR_2 = 3$ .  
 $G1L1 = 5$ ,  $G1L2 = 18$ ,  $G2L1 = 13$ ,  $G2L2 = 27$ .

$\Omega[1] = 0 \times 698D3AD4$ ,  $\Omega[2] = 0 \times A62E8B66$ ,  
 $\Omega[3] = 0 \times 557255A9$ ,  $\Omega[5] = 0 \times 9AD1E41B$ ,  
 $\Omega[6] = 0 \times 2D3CF0C3$ ,  $\Omega[7] = 0 \times 3AC6359C$ ,  
 $\Omega[9] = 0 \times C5638B36$ ,  $\Omega[10] = 0 \times D2994E69$ ,  
 $\Omega[11] = 0 \times 5393E178$ ,  $\Omega[13] = 0 \times B82E1D6C$ ,  
 $\Omega[14] = 0 \times 0F556A93$ ,  $\Omega[15] = 0 \times E4E89687$ .

Two S-boxes: *SBox1* and *SBox0* were presented in Appendix.

The same values, with one exception, are defined for PHAL-224 instance, where length of the digest is equal to 224. 256-bit long digest is simply cut down to first 224 bits.

### Efficiency

Total number of operations of PHAL-256 and SHA-256 is presented in Table 3.

**Table 3.** Number of operations

	PHAL-256			SHA-256
	1 round	2 rounds	3 rounds	
Message expansion or modification	⊞ : 0 ⊕ : 0 ⋈ : 0	⊞ : 36 ⊕ : 54 ⋈ : 24	⊞ : 72 ⊕ : 108 ⋈ : 48	⊞ : 144 ⊕ : 384 ⋈ : 480
Compression function	⊞ : 32 ⊕ : 304 ⋈ : 128 ∧ : 80	⊞ : 64 ⊕ : 608 ⋈ : 256 ∧ : 160	⊞ : 96 ⊕ : 912 ⋈ : 384 ∧ : 240	⊞ : 448 ⊕ : 832 ⋈ : 768 ∧ : 320
Output transformation	⊕ : 16	⊕ : 16	⊕ : 16	⊞ : 8

To make the comparison more accurate, the following simplification was assumed:  $\lll$  is the same as:  $\ll$ ,  $\oplus$ ,  $\ll$ ;  $\ggg$  is the same as  $\gg$ ;  $\boxplus$  is the same as  $\boxplus$ .

The efficiency of PHAL-256 and SHA-256 was tested and compared (Table 4) in the following environment: Intel Core Solo 1.2 GHz, 1GB RAM, Microsoft Windows XP

Professional. The comparison was done without optimization.

**Table 4.** Efficiency comparison

Hash function	Average Speed [Mbps]
MD5	355
PHAL-256 (3 rounds)	88
SHA-256	84
PHAL-256 (4 rounds)	66

### Cryptanalysis

One of the approaches to find a collision is using the message modification technique. The attacker could expect the following event for finding collision:

$$CV_{i,0}^{rounds,4} \oplus CV_{i,1}^{rounds,4} = 0$$

The attacker inserts the message difference in the original message. Even if the attacker finds inner collision in one of the branches, finding inner collision for the another branch will be extremely difficult because of difference message word ordering in both branches.

### Statistical tests

The hash function PHAL-256 was tested using NIST statistical tests suite for random and pseudorandom number generators for cryptographic applications [10]. Compression function was tested with variety of configuration: different number of rounds and salt value. Additionally, the avalanche effect was investigated. Various one-bit differences in random messages and in sparse messages was considered, where sparse message is a message with small (equal to 1, 2 or 3) or high (equal to 509, 510 or 511) Hamming weight. Only one-round compression function did not pass the tests.

### Minimal number of rounds

Despite the fact, that no weaknesses of PHAL-256 construction were found and it looks resistant against existing attack, it is suggested to use PHAL-256 algorithm with at least three rounds. Three rounds seem to be optimal trade-off between security and performance.

## 5. CONCLUSION

In this paper a new dedicated hash function family PHAL was proposed. It is designed to be not only secure but also flexible. The main features are as follows:

- Number of rounds as a parameter was added to make this function flexible. The performed tests show that *rounds* must be greater than 1, but authors suggest number of round greater then 2.
- The number of bits hashed so far (*counter*) and random value (*salt*) were added to increase resistance of hash function to attacks against MD-type iteration structure.
- Instead of message expansion or message ordering, message modification technique with different message ordering for branches was used.
- Two branches are used in parallel. This means that PHAL family can be efficiently implemented in hardware and it is difficult to analyze both branches simultaneously.
- PHAL family looks resistant against existing attacks, in particular against Wang *at al.*'s attacks.

## 6. APPENDIX

```
SBox0[256] = {
0xD819C303, 0x0AE9164D, 0x62E2AEA4, 0x069CF4B4,
0xE8B9FD08, 0x286B8B91, 0xDDE5F864, 0x85C7905E,
0xF5A8CE09, 0x1E7DC728, 0x8C97760E, 0xA90BE091,
0x3AAC4505, 0xE4B3D1A9, 0xF926550A, 0x4A589A5C,
0x15C8697B, 0xF6913C4E, 0x714F8763, 0x6CB68738,
0xD9574A3D, 0x40FACB85, 0x68BE493A, 0x3ED0ED19,
0xCE3D1B26, 0xE83B0FCD, 0x2B16A6C6, 0x71D5F349,
0x67C15F3E, 0xBCE75483, 0x229EC8E6, 0x138959FA,
0xCFC54C76, 0x4AB54BBD, 0x87B585CD, 0x8376722E,
0xF139E915, 0xDF12E3E2, 0xD518E2F1, 0xD71B2113,
0xF813FC72, 0x3D489F91, 0xC49B9C5A, 0x2D2BA3DA,
0x3A5B319C, 0x02FF701A, 0xF254754D, 0xC642D6EC,
0xC91B52D9, 0x40DBCD45, 0x6927F8B0, 0x522A9F48,
0x3B84AF13, 0xE15ECC33, 0xA773DC17, 0x19B06FDA,
0xE4AC8EC6, 0x74D46B62, 0x936D3469, 0xDFA0D741,
0x784688DB, 0x36C257C7, 0xD49626D6, 0xAFC8D937,
0xBDE19637, 0x9C320759, 0xC2E063A4, 0x66A51F83,
0xA57D4B8E, 0x64313DE9, 0xDCD5384F, 0x26469391,
0x7499582E, 0x37B02D35, 0x3E6BBD28, 0xE5A11B4E,
0x19049B8C, 0x89ED569C, 0x5A8B1CFA, 0x64206EDF,
0x4ECC6A27, 0x696AFA46, 0xD6852B2C, 0x6727DC5A,
0x3B6F69D0, 0xA71D2EA4, 0x132DF8BA, 0xE334D152,
0x1F3A15EC, 0xCD2679A4, 0x9649CE91, 0x39D5065D,
0xA84ECCFC, 0x336E547A, 0xCB9CE0ED, 0x75459ECA,
0x48B5544A, 0x0C9741B2, 0x397523DB, 0xC626FC18,
0x1A7DA555, 0x18BE9889, 0x4C4C7335, 0x9F1DDE28,
0xE1E12EBA, 0x5AEBCA13, 0x7C6CC2EC, 0xB00F3A33,
0x0C1EB0EF, 0xB2F9AAE2, 0x6B4027CD, 0xD9B2666,
0x71B79C85, 0xCFC91952, 0xD52EB3AA, 0x78D2E16B,
0x65383678, 0x8E6CB4A3, 0x64FCFC72, 0x29D71E64,
0x9550F0E3, 0x1F66E694, 0x0039EB2A, 0x068CF453,
0xE27404E3, 0x741ADAD3, 0x0F0B1727, 0xAFB49E0E,
0x84669357, 0x86DCE157, 0x08F6E784, 0xA77955C5,
0xA59E843F, 0x22FEA857, 0x708E7B09, 0xC4D43979,
0x58252BB6, 0x2BA45D52, 0x0A2FC9C8, 0x645A2196,
0x744F601D, 0x2DE373AC, 0xF3B8A6B1, 0x0E58626F,
```

```
0x528AB4F1, 0x11CAFBA2, 0x1ED32823, 0x8EFA5569,
0x466C2C85, 0x23F45A94, 0x80B1AA98, 0xD5F407CE,
0xC0E73CD6, 0xA056DE0C, 0x96129FC1, 0xDF0E15A9,
0x2FD92E80, 0xB6015EAE, 0x2705AEC5, 0x4B46961D,
0xE420FB8B, 0xCB96D216, 0x42F13FD8, 0xE7419AE0,
0x61B49EEC, 0x207560AD, 0x54CA9D12, 0xB242A5B5,
0x3E1903F4, 0xB514EF12, 0xCBEF007C, 0x9F994D47,
0x9002DCB5, 0x1128EF6C, 0xFD482CEA, 0x8976094D,
0xD9A368FA, 0xE837424E, 0xA8EB0D84, 0xF19044A6,
0x7B11972D, 0xA680B50B, 0xF187433B, 0xF5D2198D,
0xB901A70F, 0x099E5158, 0xAB51E2A, 0xDE1AAC2A,
0xD4B07574, 0x66AB22BD, 0xD48FEA6C, 0x38332732,
0x2E60B99C, 0xAF5928EE, 0x530BD43B, 0x38BF419D,
0x0EA7CD56, 0x8159F1D1, 0x8E69C4D5, 0x9AAAC549,
0x266B626E, 0xE5164BB2, 0x472CF571, 0x078AD216,
0xA85A454B, 0xAF4A1B63, 0xABE491B6, 0xDDDF6113,
0xF271485E, 0x71CEC251, 0x46E67B92, 0x2A90F1F4,
0x13ABB197, 0x3BED014B, 0x14DD24E5, 0xF3E7C43A,
0xC61FC9E5, 0xF835339A, 0xF4A208AD, 0x49F32FC3,
0x1F39029D, 0x4676A3C8, 0x31F879B0, 0xD165B8B2,
0x9CEB14C8, 0x724A26F6, 0x03CE4E33, 0x59CEA369,
0x770B3D1B, 0x5A07E4B9, 0x1233F329, 0x25BB3270,
0xD8743CE3, 0xA8C8456F, 0x84E7A1B5, 0xFDA42A751,
0x193CA239, 0x5ED1CA2B, 0x93838395, 0xFD11B23E,
0x9F80595D, 0x1ADA69A6, 0xEA183D61, 0xA097C17B,
0xDB52F1B1, 0xB74C3AC6, 0x4FF71DA1, 0x1B46B5CC,
0x271A46D7, 0x9DE12251, 0xC30A1C9E, 0xD8E970E7,
0xB97D06B3, 0xDB8C9056, 0xF0E52664, 0x43309FE4}
```

```
SBox1[256] = {
0xE539CCD8, 0x143A7AD9, 0xD270223F, 0x1EED03C2,
0x5A3B8538, 0x21AC3463, 0xF97416C3, 0x197C7ECC,
0x48439E3C, 0x0901FF44, 0x10D58915, 0xAB4A1870,
0xC2F8ED11, 0xE58B61CB, 0xBAE30519, 0x4930DE37,
0xABA31539, 0xBCF4D941, 0x6299B63A, 0x09D549EC,
0x5C59A0A9, 0xF68E32C6, 0xFB6E1609, 0x6AEA581D,
0x4ACD8A4F, 0xCF38FF48, 0x27871379, 0x581F0CF2,
0xF36936B5, 0x68A758BA, 0xB789945A, 0x15C46377,
0x681A595F, 0x55FC823A, 0x54909B71, 0x621F975A,
0xA056D7CA, 0xDA0BF664, 0x58E38B3B, 0x4A2F2696,
0x37D2852D, 0x25493DEC, 0xBD19D30E, 0x90F74B4E,
0x0E3D60F8, 0x870EAA9C, 0x8967C531, 0x0E9779E5,
0xCA7138D5, 0x3052CF95, 0xF68243EA, 0xA386D693,
0x176D4C66, 0x7DC24E73, 0xF12D942E, 0xEAD0F32D,
0xD5086F90, 0x468ECC3E, 0x3D9A8A93, 0x58D18716,
0xEA5A55A8, 0xF428B8C7, 0xF9008F8B, 0x68B2EA9B,
0x73E67AE0, 0x3F8C26F8, 0xB8583FA1, 0x8AE59856,
0xDDA47836, 0x7D72DAB1, 0x49E8E996, 0xF7D2781A,
0x20AC5B71, 0x3B4F6223, 0x9B7099D4, 0xD339C486,
0xFB35A1AA, 0x98EF8507, 0x278CCF88, 0xA3C9928B,
0x06376C9E, 0x29FE16A7, 0xDD8605A4, 0x25B00F3A,
0x777A5C26, 0xD5C4C3B6, 0xE6D728CD, 0xFA3C1B68,
0x09D9761D, 0x07DC5E57, 0x548AADCE, 0xC5520FB2,
0x33206723, 0xC033A2D3, 0xC885AA66, 0x1CECDE2E,
0xE6C19AE1, 0xAB9D6E11, 0x541993CE, 0xCF0C689D,
0xDDF35D02, 0x007235F8, 0x955461DA, 0xF34ABF06,
0xE99A23C2, 0xB82E9B73, 0xDE8A64E9, 0xF937B725,
0xD7717931, 0x981F1825, 0x65DCAE8B, 0xB321E791,
0x1DA734B4, 0x709EF069, 0xB98C517, 0xB7D8D41E,
0xE472CA23, 0x958BD24C, 0x01BC6E5E, 0xC3533F8A,
0xA0B3F0E9, 0x25615758, 0xA6B1E36A, 0xC407F8C4,
0xED1A594C, 0x760994DB, 0xB658E45E, 0x34E6F40E,
0x377489D5, 0x2F21EF24, 0x0B9EA3A9, 0x0C32CD5E,
0x6AB71536, 0x42C2E4B5, 0x17C5FC4B, 0x8727571E,
0x0E81BCAF, 0xF8684FAB, 0x2E806A77, 0x22D929E9,
0xC04D9717, 0x95FD90BA, 0x64BD7051, 0x21F6ACC6,
0x4ACD57B1, 0x81E89BC3, 0xDF9644CF, 0xAA3C33E2,
0xA09F74F3, 0x09C429FD, 0xE92E9C03, 0x248F1D78,
0x914B2172, 0xD26A7994, 0x27545ECD, 0xEE06823C,
0x842F4C9D, 0x984EE131, 0x66D4D5BA, 0xD1CEC9D7,
0x3B9B7E0A, 0x596D322D, 0x3273D9D6, 0x6B3CADE4,
0x57369D2A, 0xBEDF8EA2, 0x8E9825D1, 0xD8338473,
0x3AB76564, 0x670BB314, 0xF029EAC, 0xC40B6DCC,
0x6FEA2347, 0xDF922F48, 0xD14F1B75, 0xB4259D93,
0xAED7180D, 0x3AD468ED, 0x6DAE526C, 0x1B825CB7,
0x2E474BD4, 0x8512BD49, 0xCBAD0AE0, 0x566E3053,
0x4521167E, 0xAC749574, 0xCE343D33, 0x735F5626,
0xD6763639, 0x2F2BE5E5, 0xA99BD545, 0x779E3388,
0x5E16C01A, 0x7B780CEA, 0x9E4141CA, 0x63C48939,
0x12D713C7, 0xA963F598, 0x5AA65CC3, 0xB676AA03,
0x199B861D, 0x71F96C32, 0xCACAB549, 0x3A5D502A4,
0x0E7F2453, 0x16669B8D, 0x062D453E, 0x34E05F85,
0x1EA6D1E9, 0xEB1C2BDA, 0x0F7608AE, 0x7BA68ACB,
0x54437B85, 0xD5712563, 0x0B23A5B8, 0x4BA74AF2,
```

0xADB07357, 0xE5702774, 0xBBA605A5, 0xCC7C76C1,  
0x1C91CAF4, 0xCF25E4A7, 0xE43724B9, 0x2C38BA61,  
0x08FB420B, 0x58F321F8, 0xDDA10CD6, 0x31E0A9DC,  
0xD52CE815, 0x9F59B2C1, 0xACD131CD, 0x41E9EA84,  
0x9B047C74, 0x1593EAD4, 0x6C5DE0B4, 0x43343A78,  
0x69D9C571, 0x3B3CC20E, 0x860BDB30, 0xF66CD818,  
0xA6FAEB13, 0x85E6A8EE, 0xD649AB2E, 0xF123A1D4,  
0x20A3326E, 0xDB809E8B, 0x44CB0769, 0x80CD131F,  
0xCF6976F1, 0x30CF516D, 0xAAC0ABE1, 0x1046D283,  
0x2EC8C522, 0x5875370E, 0x189CE36D, 0xD477D0C6}

## BIBLIOGRAPHY

- [1] R. Anderson, E. Biham, *Tiger: A Fast New Hash Function*. Fast Software Encryption - FSE 1996, LNCS 1039, Springer-Verlag, 1996
- [2] E. Biham, O. Dunkelman, *A Framework for Iterative Hash Functions - HAIFA*. The Second Cryptographic Hash Workshop, Santa Barbara, USA, 2006
- [3] A. Joux, *Multicollisions in Iterated Hash Functions*. Advances in Cryptology - CRYPTO 2004, LNCS 3152, Springer-Verlag, 2004
- [4] I. Damgård, *A design principle for hash functions*. Advances in Cryptology - CRYPTO 1989, LNCS 435, Springer-Verlag, 1989
- [5] R. D. Dean, *Formal Aspects of Mobile Code Security*. Ph.D. dissertation, Princeton University, 1999
- [6] D. Johnson, *Improving Hash Function Padding*. The First Cryptographic Hash Workshop, Gaithersburg, USA, 2005
- [7] J. Kelsey, B. Schneier, *Second Preimages on  $n$ -Bit Hash Functions for Much Less than  $2^n$* . Advances in Cryptology - EUROCRYPT 2005, LNCS 3494, Springer-Verlag, 2005
- [8] J. Lee, D. Chang, H. Kim, E. Lee, D. Hong, J. Sung, S. Hong, S. Lee, *A New 256-bit Hash Function DHA-256: Enhancing the Security of SHA-256*. The First Cryptographic Hash Workshop, Gaithersburg, USA, 2005
- [9] R. Merkle, *One way hash function and DES*. Advances in Cryptology - CRYPTO 1989, LNCS 435, Springer-Verlag, 1989
- [10] NIST, *A statistical test suite for random and pseudorandom number generators for cryptographic applications*. NIST Special Publication 800-22, <http://csrc.nist.gov/groups/ST/toolkit/rng>
- [11] NIST, *Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family*. 2007, <http://www.nist.gov/hash-competition>
- [12] P. Rodwald, J. Stokłosa, *Family of Parameterized Hash Algorithms*. Proceedings of the International Conference on Emerging

Security Information, Systems and Technologies, IEEE Computer Society Press, Cap Esterel, France, 2008

- [13] P. Rodwald, J. Stokłosa, *PHAL-256 - Parameterized Hash Algorithm*. Proceedings of the Fourth International Conference on Information Assurance and Security, IEEE Computer Society Press, Naples, Italy, 2008
- [14] X. Wang, X.J. Lai, D. Feng, H. Chen, X. Yu, *Cryptanalysis of the Hash Function MD4 and RIPEMD*. Advances in Cryptology - EUROCRYPT 2005, LNCS 3494, Springer-Verlag, 2005
- [15] X. Wang, H. Yu, *How to Break MD5 and Other Hash Functions*. Advances in Cryptology - EUROCRYPT 2005, LNCS 3494, Springer-Verlag, 2005
- [16] X. Wang, H. Yu, Y.L. Yin, *Efficient Collision Search Attacks on SHA-0*. Advances in Cryptology - CRYPTO 2005, LNCS 3621, Springer-Verlag, 2005
- [17] X. Wang, Y.L. Yin, H. Yu, *Finding Collisions in the Full SHA-1*. Advances in Cryptology - CRYPTO 2005, LNCS 3621, Springer-Verlag, 2005

## BIOGRAPHY

Przemysław Rodwald was born in Lębork, Poland, in 1977. He was graduated from the Cybernetics Faculty of the Military University of Technology in Warsaw in 2001. Since 2002 he has been working at the Military Communication Institute. He is currently working in the NATO Internet Protocol Security Task Force group. His main objects of interest: hash functions and cryptanalysis.

Janusz Stokłosa is a professor at Poznan University of Technology, Poznan, Poland. Cryptology is the main object of his interest.