Annales UMCS Informatica AI XII, 3 (2012) 11–24
DOI: 10.2478/v10065-012-0025-z

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI
http://www.annales.umcs.lublin.pl/

Parameterized Hash Functions

Tomasz Bilski[1*], Krzysztof Bucholc[1†], Anna Grocholewska-Czurylo[1‡], Janusz Stoklosa[1§]

[1] *Institute of Control and Information Engineering, Poznań University of Technology pl. Marii Skłodowskiej Curie 5, 60-965 Poznań, Poland*

Abstract – In this paper we describe a family of highly parameterized hash functions. This parameterization results in great flexibility between performance and security of the algorithm. The three basic functions, HaF-256, HaF-512 and HaF-1024 constitute this hash function family. Lengths of message digests are 256, 512 and 1024 bits respectively. The paper discusses the details of functions structure. The method used to generate function S-box is also described in detail.

1 Introduction

Hash functions are used to generate a short form of an original message of any size. This short form is called a hash of a message or a message digest and is used in many cryptographic applications including message integrity verification and message authentication, in which case a keyed hash function is used.

Hash function h operates on a message m of an arbitrary length. The result is a hash value $h(m)$ which has a fixed size.

A lot of recent cryptographic research has been devoted to methods of generating new hash functions which resulted for example in 64 proposals being submitted to the NIST SHA-3 competition for a new hash function in 2008 [**1**].

The objective while designing the HaF family of hash function was obviously the highest security while maintaining the best possible performance, however, at the same time the function should allow a flexible balance between security and performance which was achieved through parameterization.

*tomasz.bilski@put.poznan.pl
†krzysztof.bucholc@put.poznan.pl
‡anna.grocholewska-czurylo@put.poznan.pl
§janusz.stoklosa@put.poznan.pl

The organization of this paper is the following: In Section 2 we describe the family of the HaF hash functions in general. In Section 3 we concentrate on the details of S-box generation along with our reasoning for designing and choosing this particular method. Reference implementation is briefly described in Section 4. Finally Section 5 contains the concluding remarks.

2 Parameterized family HaF of hash functions

2.1 Design Principles

The following assumptions were taken into account during the design process:

- family should be parameterized;
- message digest length should be selectable;
- flexibility between performance and security should be guaranteed;
- iteration structure and compression function should be resistant to known attacks;
- its iteration mode should be HAIFA (it provides resistance to long message second preimage attacks, and handles hashing with a salt) **[2, 3]**.

2.2 Description of HaF

The HaF family is formed of the three hash functions: HaF-256, HaF-512 and HaF-1024, producing hash values (message digests) with the length equal to 256, 512 and 1024 bits, respectively. The general model of HaF family is presented in (Fig. 1). After formatting the original message m we have the message M. We divide M into blocks $M_0, M_1, \ldots, M_{k-1}$, $k \in \{1, 2, \ldots\}$, and each block M_i is processed with the salt s by the iterative compression function φ **[2]**. The output H_k is the final result of the function.

2.2.1 Notation

In the paper we use the following notation:
$a \odot b$ – multiplication mod $(2^n + 1)$ of n-bit non-zero integers a and b;
A_r – working variable, $r = 0, 1, \ldots, 15$;
F_j – step function, $j = 0, 1, \ldots, 15$;
$GF(2)$ – Galois field of characteristic 2;
$length$ – bitstring representing the length of the original message m, $|length| = 128$;
$lsb_q(v)$ – q least significant bits of the string v;
IV – initial value;
m – original message, $|m| < 2^{128}$;
M – formatted message;
n – length of the working variable A_r (16 or 32 or 64 bits);
s – salt, $|s| = 16n$;
$|v|$ – length in bits of a string v;

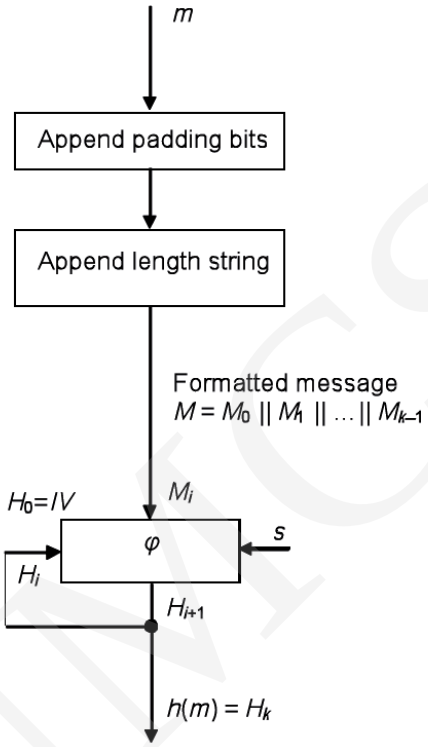

Fig. 1. General model for HaF.

$v \ll t$ – t-bit left rotation of a string v, $|v| = 16n$;
$v \oplus w$ – bitwise XOR of strings v and w, $|v| = |w|$;
$v \boxplus w$ – addition mod 2^n of integers represented (in base 2) by strings v and w;
$p_1(x) \otimes p_2(x)$ – multiplication of polynomials p_1 and p_2 modulo an irreducible polynomial $R(x)$;
x^q – bitstring of the length q; x^0 means the empty string;
φ – compression function;
$\|$ – concatenation of bitstrings.

2.2.2 Message Padding

The original message m has to be formatted before hash value computation begins. The length of formatted message should be a multiple of $16n$ bits. The message m is formatted by appending to it a single 1-bit and as few 0-bits as necessary to obtain a string whose bit-length increased by 128 bits is a multiple of $16n$. Finally, we must additionally append the original message length. As a result, we obtain the formatted message $M = M_0 \| M_1 \| \ldots \| M_{k-1}$ for some positive integer k, where M_i is a block of M.

Therefore, $M = m\|10^t\|$ *length*, where t is the smallest nonnegative integer necessary to format m, and $|M| = 16nk$.

2.2.3 Compression Function

In the proposed schema the compression function is defined as follows: $\varphi : \{0,1\}^\mu \times \{0,1\}^\eta \times \{0,1\}^\sigma \to \{0,1\}^\rho$. The integers μ, η and σ are the lengths of block M_i, chaining variable H_i, and salt s, respectively, where $|M_i| = |H_i| = |s| = 16n$ and $i = 0, 1, \ldots, k\check{}1$. The integer ρ is the length of the resulting hash value $h(m) = H_k$, $|h(m)| = 16n$.

The block M_i is processed in two rounds. The length of the block equals $16n$ bits, where n is a parameter depending on the hash value we want to obtain. For HaF-256, HaF-512 and HaF-1024 the parameter n equals 16, 32 and 64 bits, respectively. The parameter n indicates, in fact, the length of the working variable A_r used in the step function.

The method of one block processing is presented in Fig. 2. M_i, H_i and s are the inputs for φ. Before processing in round $\#l$, $l = 1$ or 2, the block M_i is modified. In the round $\#1$ four least significant bits of $N_i = M_i \oplus s$ indicate the number of bits the string N_i is rotated to the left: $N_i^* = N_i \ll lsb_4(N_i)$. Before processing in the round $\#2$, the blocks are permuted: $N_i = H_i^*$ and $H_i = N_i^*$. After two rounds, the value H_i^* of chaining variable is split into 16 subblocks $A_0, A_1, \ldots, A_{15}$ of equal lengths. Each of them is modified by adding (mod 2^n) the respective input subblock of H_i which is the input to the round $\#1$. Next, all subblocks $A_0, A_1, \ldots, A_{15}$ are concatenated giving $H_{i+1} = A_0\|A_1\| \ldots \|A_{15}$.

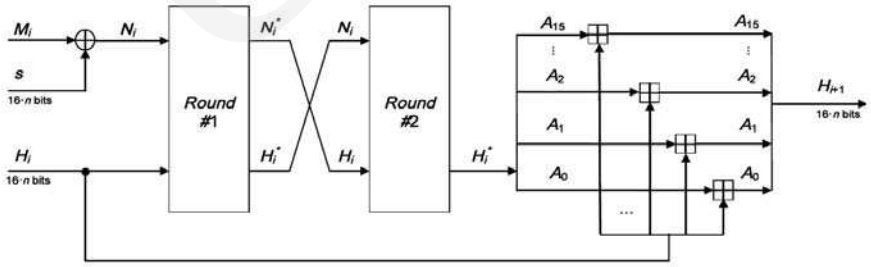

Fig. 2. Method of one block processing.

2.2.4 Round Function

The round function (Fig. 3) has two inputs N_i, H_i and two outputs N_i^*, H_i^*.

The input block N_i is rotated by the number of bits corresponding to $lsb_4(N_i)$ and added (mod 2 of respective bits) to H_i. Next the block $H_i \oplus (N_i \ll lsb_4(Ni))$ is divided into 16 subblocks of equal length: $A_0, A_1, \ldots, A_{15}$. They are processed

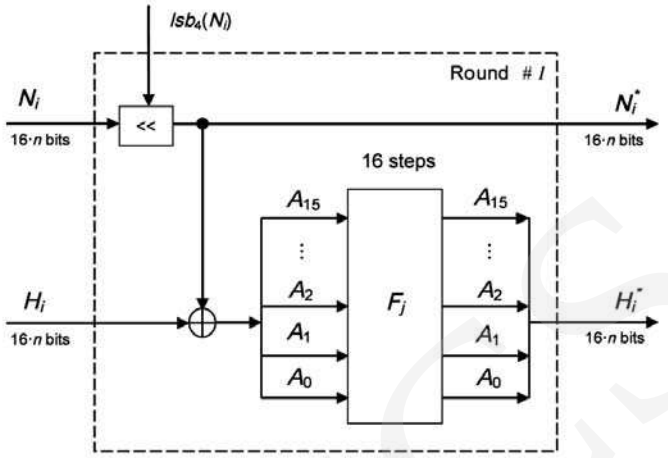

Fig. 3. Round function.

by a step function. After processing they are concatenated giving H_i^*. The output $N_i^* = N_i \lll lsb_4(N_i)$.

2.2.5 Step Function

The essential part of the round is the step function F_j (Fig. 4). In each round the step function is executed 16 times, for $j = 0, 1, \ldots, 15$.

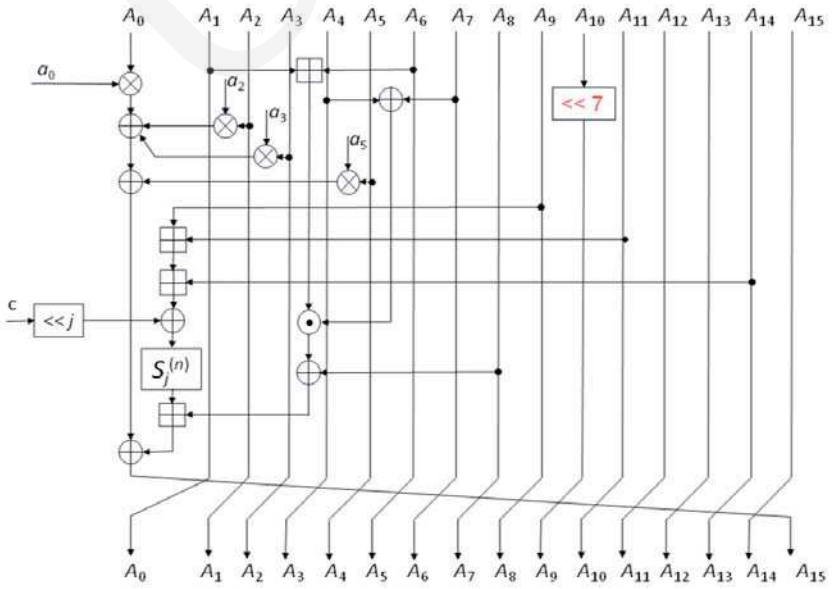

Fig. 4. Step function F_j.

Let $GF[x]_n$ be a set of polynomials over $GF(2)$ of the degree smaller than n. If $w(x) \in GF[x]_n$ then $w(x) = w_{n\breve{}1}x^{n\breve{}1} \oplus w_{n\breve{}2}x^{n\breve{}2} \oplus \ldots \oplus w_2x^2 \oplus w_1x \oplus w_0$ or simply $w(x) = w_{n\breve{}1}w_{n\breve{}2}\ldots w_2w_1w_0$, where $w_r \in GF(2)$ for $r \in \{0, 1, \ldots, n\breve{}1\}$. Let $u(x)$, $v(x)$, $w(x) \in GF[x]_n$. We define two operations on polynomials, addition ($\oplus$) and multiplication ($\otimes$): $u(x) = v(x) \oplus w(x) \leftrightarrow u_t = v_t \oplus w_t$, $t = 1, 2, \ldots, n$, and $u(x) = v(x) \otimes w(x) = v(x) \cdot w(x) \bmod R(x)$, where $R(x)$ is a reduction polynomial of degree n. In the construction of the step function, the multiplication of polynomials is performed four times: $a_0 \otimes A_0$, $a_2 \otimes A_2$, $a_3 \otimes A_3$, and $a_5 \otimes A_5$. The polynomials a_0, a_2, a_3 and a_5, presented in the hexadecimal form, are given in Table 1.

Table 1. Polynomials used in the step function.

n	R(x)	Hexadecimal representation
16	$x^{16}\oplus x^{11}\oplus x^{10}\oplus x^5\oplus 1$	10C21
32	$x^{32}\oplus x^7\oplus x^6\oplus x^2\oplus 1$	1000000C5
64	$x^{64}\oplus x^4\oplus x^3\oplus x\oplus 1$	100000000000001B

The reduction polynomials must be irreducible; they are presented in Table 2.

Table 2. Reduction polynomials used in the step function.

n	a_0	a_2	a_3	a_5
16	89CB	D949	0001	0001
32	AC2D B263	0000 0110	0000 0001	0000 0001
64	EDC0 28B9 A461 A403	0000 2500 0000 0001	0000 0000 0000 0001	0000 0000 0000 0001

After performing multiplications of polynomials, a few additions modulo 2 ($\oplus$) and additions modulo 2^n ($\boxplus$) are done (Fig. 4). In each step the masking constant $c = 3236B539391FD066$ (in the hexadecimal representation) is used. The particular value of c depends on n and j, and is indicated by a window of the length n sliding (cyclically, if necessary) from left to right on bits of c. For example, if $n = 16$ and $j = 0$ then $c = 3236$; if $n = 32$ and $j = 31$ then $c = 391FD066$; if $n = 64$ and $j = 5$ then $c = 6D6A72723FA0CD9$ (cyclic rotation of c to the left by 5 bits).

In each step a substitution $S_j^{(n)}$ depending (as the masking constant c) on n and j is used. It consists of four S-boxes S_0, S_1, S_2 and S_3, each of

the dimension 16×16, working in such a way that for $n = 16$, $S_j^{(16)} = S_{(j)mod4}$; for $n = 32$, $S_j^{(32)} = S_{(j)mod4} \| S_{(j+1)mod4}$; and for $n = 64$, $S_j^{(64)} = S_{(j)mod4} \| S_{(j+1)mod4} \| S_{(j+2)mod4} \| S_{(j+3)mod4}$.

The multiplication modulo $2^n + 1$ of n-bit integers with the zero block corresponding to 2^n is denoted by $\odot$ [**4**].

Table 3. Initial values of chaining variable.

n	$H_0 = h_0 \| \| h_1 \| \| h_2 \| \| ... \| \| h_{15}$
16	34D906D3E3E5298EAC26F9FD2AC5AD23 DB84B0576C82CCA52517CF6B88B0A90C
32	34D906D3E3E5298EAC26F9FD2AC5AD23 DB84B0576C82CCA52517CF6B88B0A90C 0BC69C6F64D4B2664579E064AE220A5A 3DA7C5451DA429EF2AE8BF289D0F01E5
64	34D906D3E3E5298EAC26F9FD2AC5AD23 DB84B0576C82CCA52517CF6B88B0A90C 0BC69C6F64D4B2664579E064AE220A5A 3DA7C5451DA429EF2AE8BF289D0F01E5 8C6595B7B088D0C74BB82BF3CFDE5AA1 AB808B7E7425BC9EFA101925CBB0D528 3FA76FCBDF7B50D776DE280C8E2EE8B1 69D154F43B096994FDF52B5F148CC134

The initial values $H_0 = h_0 \| h_1 \| h_2 \| \ldots \| h_{15}$ of chaining variable (depending on n) are given in Table 3 (H_0 for $n = 64$ is obtained as the hexadecimal form of consecutive 512 decimal places after the decimal point of π broken up into groups of 32). Before processing they must be assigned to $A_0 \| A_1 \| A_2 \| \ldots \| A_{15}$ in such a way that $h_r = A_r$, $r = 0, 1, \ldots, 15$.

2.3 Security Considerations

The round function composed of 16 steps can be represented in the equivalent form as a linear shift register (FSR) over $GF(2^n)$ generating maximum length sequences, additionally equipped with nonlinear feedback, and clocked 16 times [**3**]. The corresponding approach dealing with the use of feedback shift registers (over $GF(2)$) in the construction of hash functions has been presented in [**5**]. The function defined by the nonlinear circuit is a nonlinear $8n$-argument function, $n = 16$ or 32 or 64. For the function with such a number of arguments (128, 256 and 512, respectively), it is difficult, from the computational point of view, to perform the best affine approximation attack [**6**]. The time needed for the attack is equal to that of the birthday attack, i.e. $O(2^{8n})$.

The sequence produced by the nonlinear circuit is resistant to correlation attack [**6**].

3 S-boxes

3.1 Involutional S

Let F_2 be the Galois field $GF(2)$ and F_2^n be the n-dimensional vector space over F_2. A substitution operation or an $n \times n$ S-box (or S-box of the size $n \times n$) is a mapping:

$$S : F_2^n \to F_2^n \tag{1}$$

where n is a fixed positive integer, $n \geq 2$. An n-argument Boolean function is a mapping:

$$f : F_2^n \to F_2. \tag{2}$$

An S-box S can be decomposed into the sequence $S = (f_1, f_2, \ldots, f_n)$ of the Boolean functions such that $S(x_1, x_2, \ldots, x_n) = (f_1(x_1, x_2, \ldots, x_n), f_2(x_1, x_2, \ldots, x_n), \ldots, f_n(x_1, x_2, \ldots, x_n))$. We say that the functions $f_1, f_2, \ldots, f_n$ are component functions of S.

In the case of HaF's S-box $n = 16$. HaF's S-box therefore is a function that takes 16 input bits and outputs also 16 bits – it is a 16×16 S-box. Additionally, it is generated in such a way that it is its own inverse, i.e., $S^{-1} = S$.

HaF's S-box has been generated using the multiplicative inverse procedure similar to AES [**7**] with randomly chosen primitive polynomial defining the Galois field. Nonlinearity of this S-box is 32510 and its nonlinear degree is 15. Sixteen Boolean functions that constitute this S-box have nonlinearities equal to 32510 or 32512. The degree of each function is equal to 15.

The 16×16 S-box can be stored as a table of 65536 word values. The index for this table is an input of the S-box function, i.e., $x_1, x_2, \ldots, x_{16}$. The values stored are S-box outputs (16 bits: $f_1(x_1, x_2, \ldots, x_{16}), f_2(x_1, x_2, \ldots, x_{16}), \ldots, f_{16}(x_1, x_2, \ldots, x_{16})$). To simplify the description of S-box generation let us consider a smaller S-box of the size 8×8. For presentation convenience such S-box can be displayed as a 2-dimensional table (Table 4). The input represented as a two digit hexadecimal number HL is divided – the low order digit (L) is on the horizontal axis and the high order digit (H) is on the vertical axis. For example, to see what is the S-box output at input 6F take 6 on the vertical axis and F on the horizontal axis. The S-box output is DA.

Cryptographically a strong S-box should possess some properties that are universally agreed upon among researchers. Such S-box should be balanced, highly nonlinear, have the lowest maximum value in its XOR profile (difference distribution table), have complex algebraic description (especially it should be of high degree). The above criteria are dictated by linear and differential cryptanalyses and algebraic attacks.

It is a well-known fact that S-boxes generated using finite field inversion mapping fulfill these criteria to a very high extent. However, they are susceptible to (theoretical) algebraic attacks. To resist algebraic attacks, multiplicative inverse mapping used to construct an S-box is composed of an additional invertible affine transformation. This affine transformation does not affect the nonlinearity of the S-box, its XOR profile nor

Table 4. Sample 8 × 8 S-box S.

H \ L	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	9E	BC	C3	82	A2	7E	41	5A	51	36	3F	AC	E3	68	2D	2A
1	EB	9B	1B	35	DC	1E	56	A5	B2	74	34	12	D5	64	15	DD
2	B6	4B	8E	FB	CE	E9	D9	A1	6E	DB	0F	2C	2B	0E	91	F1
3	59	D7	3A	F4	1A	13	09	50	A9	63	32	F5	C9	CC	AD	0A
4	5B	06	E6	F7	47	BF	BE	44	67	7B	B7	21	AF	53	93	FF
5	37	08	AE	4D	C4	D1	16	A4	D6	30	07	40	8B	9D	BB	8C
6	EF	81	A8	39	1D	D4	7A	48	0D	E2	CA	B0	C7	DE	28	DA
7	97	D2	F2	84	19	B3	B9	87	A7	E4	66	49	95	99	05	A3
8	EE	61	03	C2	73	F3	B8	77	E0	F8	9C	5C	5F	BA	22	FA
9	F0	2E	FE	4E	98	7C	D3	70	94	7D	EA	11	8A	5D	00	EC
A	D8	27	04	7F	57	17	E5	78	62	38	AB	AA	0B	3E	52	4C
B	6B	CB	18	75	C0	FD	20	4A	86	76	8D	5E	01	ED	46	45
C	B4	FC	83	02	54	D0	DF	6C	CD	3C	6A	B1	3D	C8	24	E8
D	C5	55	71	96	65	1C	58	31	A0	26	6F	29	14	1F	6D	C6
E	88	F9	69	0C	79	A6	42	F6	CF	25	9A	10	9F	BD	80	60
F	90	2F	72	85	33	3B	E7	43	89	E1	8F	23	C1	B5	92	4F

its algebraic degree. The best known example of such an S-box is the S-box of AES. It has been publicly known and it does not affect its security.

The algorithm used for generating the S-box for the purpose of HaF function presented in this paper uses a similar method of generating S-boxes. Additionally, it takes into account the results of some recent studies **[8, 9]** and incorporates changes in the S-box generating procedure to make it even more secure.

3.2 Generating Inverse Mapping

HaF S-box is based on the so called inverse mapping $x \to x^{-1}$, where x^{-1} denotes the multiplicative inverse in a finite field $GF(2^n)$:

$$S(x) = \begin{cases} 0 & \text{for } x = 0 \\ x^{-1} & \text{for } x \neq 0. \end{cases} \tag{3}$$

As mentioned earlier, inversion mapping can be used to generate cryptographically strong S-boxes.

For any prime integer p and any integer n ($n = 1, 2, \ldots$), there is a unique field with p^n elements, denoted $GF(p^n)$. In cryptography p almost always takes the value of 2. To generate an inverse mapping in $GF(2^n)$ we need an irreducible polynomial that defines a Galois field and another polynomial that would be the so called generator (see below). A polynomial is said to be irreducible if it cannot be factored into nontrivial polynomials over the same field. The n-bit elements of the Galois field are treated as polynomials with coefficients in F_2. For example, in the case of AES, where S-box is of the size 8×8 we operate mostly on bytes represented as $b_7b_6b_5b_4b_3b_2b_1b_0$ which corresponds to the following polynomial:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 \tag{4}$$

where $b_i \in \{0, 1\}$.

An irreducible polynomial mentioned above is used to calculate a multiplication in $GF(2^n)$. When two polynomials are multiplied, the resulting product is a polynomial of degree at most $2(n\breve{}1)$ – too much to fit into the n-bit data word that represents polynomials in $GF(2^n)$, so the intermediate product of this multiplication is divided by the irreducible polynomial and the remainder of this division is the result of the multiplication. For $GF(2^n)$ an irreducible polynomial should be of degree n. For example, in AES (with $GF(2^8)$) an irreducible polynomial selected for construction of the S-box is 11B (in the hexadecimal notation).

A generator in the Galois field is a polynomial whose successive powers take on every element except zero. Which polynomials are generators in a particular Galois field depends on the irreducible polynomial selected. So say the polynomial 03 is a generator in $GF(2^8)$ with the irreducible polynomial 11B (as in AES), but it is not a generator in $GF(2^8)$ with the irreducible polynomial 1BD, for which the generator is for example 07.

For $n = 8$ the nonlinearity of this mapping treated as an S-box is 112. For $n = 16$ it is 32512. In a general case, the nonlinearity of such a mapping is $2^{n-1} - 2^{n/2}$.

However, such an S-box would always have 0 and 1 as the first two entries. This is because for $x = 0$, $x^{-1} = 0$ and for $x = 1$, $x^{-1} = 1$. These would be undesirable fixed points of an S-box. We remove them in the next step.

3.3 Affine Transformation

To avoid algebraic attacks (given multiplicative inversion simple algebraic form) every element of the table of multiplicative inverses is changed using an affine transformation. Such transformation has to be a full permutation, so every element is changed and all possible elements are represented as the result of a change, so that no two different bytes are changed to the same byte. After applying this transformation, the table is still a bijective mapping which is inversible and that is a prerequisite for most applications of S-boxes. In the case of AES cipher, this affine transformation is given by the following equation:

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \tag{5}$$

where c is an 8-bit constant (in the case of AES, it equals 63 in the hexadecimal notation). i is the bit position. This transformation can be also represented as the matrix multiplication:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{6}$$

The algorithm used for generating S-box S of HaF function in this paper uses the same transformation, however, adopted for the 16×16 S-box size and with the constant part of this transformation (namely c_i) taken at random so that the resulting S-box does not have fixed points (such that $S(x) = x$). Particularly, the two fixed points mentioned in the previous paragraph (0 and 1) are removed by this transformation.

3.4 Removing Cycles

One of the requirements for the HaF S-box is the absence of cycles. A cycle is such a sequence of S-box values $S_0, S_1, \ldots, S_{k-1}$ where $S_{(i+1) \bmod k} = S(S_i)$. HaF S-box should have only one such cycle containing all the values of the S-box (a cycle for which $k = 2^n$).

The affine transformation described in the previous paragraph changes the number of cycles in an S-box, without changing its nonlinear properties. Note that the fixed points are also short cycles where $k = 1$.

The cycles are removed in a procedure with two steps. The first step is actually the aforementioned affine transformation. It is applied repeatedly with a random value of c until the S-box with only 2 cycles is found. This might not always be possible. In such a case, a new S-box has to be generated with another randomly chosen primitive polynomial using the inverse mapping as described earlier.

When the 2-cycle S-box is found, we move on to the next step, which is performed together with removing the affine equivalence.

3.5 Removing Affine Equivalence

According to **[8, 9]**, S-boxes based on the multiplicative inverse in a finite field have such a peculiar property that all component functions of the S-box are from the same affine equivalence class (all the output functions of the S-box can be mapped onto one another using the affine transformations). The HaF's S-box has been processed to remove this linear redundancy, so that all Boolean functions are now from different affine equivalence classes, while still maintaining exceptionally high nonlinearity of the inverse mapping. The proposed S-box has the maximum XOR difference distribution table value of 6, which is extremely good.

Removing this linear redundancy in the 2-cycle S-box is carried out in such a way that at the same time it will reduce the number of cycles to only 1. It is done by

choosing randomly two S-box entries x and y, each belonging to another cycle, and rearranging S-box entries in such a way, that both cycles are joined into one.

After such change a test for linear redundancy is performed. If the affine equivalence is still present (between any component functions), the change is reversed and different S-box entries are randomly selected and tested – this procedure is carried out until S-box without the linear redundancy is found. If such an S-box cannot be found, we need to generate another S-box with inverse mapping.

Many properties of the Boolean functions covered by various cryptographic criteria (such as algebraic degree and nonlinearity) remain unchanged by affine transformations. The absolute values of Walsh transform as well as the autocorrelation function are only rearranged by the affine transformations. The frequency distribution of the absolute values in these transforms is invariant under such affine transformations. To prove that two functions are from different equivalence classes, it is therefore sufficient to show that their respective Walsh transform or autocorrelation function frequency distribution is different.

4 Reference implementations

The HaF family is formed of the three hash functions: HaF-256, HaF-512 and HaF-1024, producing hash values (message digests) with the length equal to 256, 512 and 1024 bits. Each function has been implemented in C language and Microsoft Visual Studio 2008 environment. The HaF test suite gives the following results (for the purpose of this paper we present only two tests for each representative of the family):

```
HaF-256 ("") =                  3423E39F 153A6A6F 02D556C9 065FF867
                                2082B544 8F4C7D67 BEAD2E4A A37F82D3

HaF-256 ("message digest") =    A854BE6D 18BEA283 310663FA 3C986850
                                353140BB 9ED51CFC 25080BB2 CE9310EF

HaF-512 ("") =                  D573B56A 3C3C2B15 FB0F2785 3599703A
                                03FDEDCF E6ECAEA1 A0BBAE6D 938D04D2
                                E499F705 1B853660 4BFD5945 F46932BA
                                D393EAE6 83EB64EE 85FB8942 DB473263

HaF-512 ("message digest") =    D00FA833 6B0E294A 4EE21624 24C5C474
                                706D9FCF 2249351C A3B76C50 9221F507
                                F421864E 81AFCE7B 567635B9 85600C3E
                                2023593E 90A918AF 56D1204D 5C0C5BCA

HaF-1024 ("") =                 7C8D7124 0F60CA46 B723DD90 D525BE23
                                1637ABEF A10C7D39 802F3C16 EFAF8C6C
                                2C931E33 C51B1044 721B35B3 1EB82EF4
                                413DDD2D 705EC909 B4ADD3BA 0EBEBEDC
                                201DF434 E9C8AB7E 57298D2F 0B6375B7
                                54DCC776 B728E71C 60009EBE B40DBAE8
                                D9F5F7DE F42454C0 D17D736C 1B7BC060
                                52308D97 1B12E541 47EC59EE 42D32934

HaF-1024 ("message digest") =   4D1D0179 6DC658B1 E96CBDE0 E4868DDC
                                B610CCE6 E320BEF4 2B0C95BE 442B0959
                                B786FB80 D15E2260 5B68BF46 4E0D20CD
                                FF2310AD 52BE0277 3C8009F1 CBB5DAA2
                                E65FD145 43AB0390 140FB0CE 348EF677
                                077B1571 5DB44D7E F331779B F3C0F370
                                966A2487 1325FA47 64ECE3EA 2BD46958
                                BA0113B1 041F83C0 35468802 6318CE7D
```

HaF can be easily implemented for 32, and 64-bit processors. Here we present tentative evaluation of HaF performance. The results were obtained for reference (non-optimized) implementations in ANSI C for HaF-256, HaF-512, and HaF-1024. We compiled our programs with Intel C++ Compiler Professional 11.1 for Windows.

Both 32-bit and 64-bit codes were generated. Then the programs were executed on a PC computer with the 2.2 GHz Athlon-64 processor. We measured processing time for 20MB text file. The results are presented in Tables 5 and 6 respectively.

Table 5. HaF family performance – 32-bit code.

Function	MB/s	Cycles/byte
HaF-256	2.86	769.2
HaF-512	3.63	606.7
HaF-1024	0.84	2611.1

Table 6. HaF family performance – 64-bit code.

Function	MB/s	Cycles/byte
HaF-256	3.12	704.2
HaF-512	3.69	595.9
HaF-1024	2.09	1050.9

As we can see in Tables 5 and 6, the best performance is achieved for HaF-512. For HaF-1024, 64-bit code performs much better than 32-bit code (speed up to 150%). The measured processing speed is relatively slow. But we expect substantially better performance for optimized implementations.

5 Conclusions

Most cryptographic hash functions designers focus on high processing speed. Therefore relatively simple algorithms are preferred. Implementations of these algorithms may be vulnerable to fault attack and side channel attack.

In the HaF hash functions the family processing scheme is more elaborated and we use relatively big 16×16 S-boxes. It leads to more complex implementation.

We expect it to give greater robustness against fault attack and side channel attack.

The processing speed is relatively small. But we expect that optimised implementation will perform substantially better. Especially, multithreaded implementation exploiting parallelism of the algorithm.

Acknowledgement

This work was supported by the Polish Ministry of Science and Higher Education as the 2010–2013 research project and partially by the grant DS-PB/45-085/12.

References

[1] Regenscheid A., Perlner R., Cjen Chang S., Kelsey J., Nandi M., Paul S., Status Report on the First Round of the SHA-3 Cryptographic Hash Algorithm Competition, Technical Report 7620 NIST (2009); http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/documents/sha3_NISTIR7620.pdf

[2] Biham E., Dunkelman O., A framework for iterative hash functions - HAIFA, NIST 2nd Hash Function Workshop, Santa Barbara (2006); also: Cryptology ePrint Archive: Report 2007/278, http://eprint.iacr.org/2007/278.

[3] Bilski T., Bucholc K., Grocholewska-Czuryło A., Stokłosa J., HaF – A new family of hash functions, Proceedings of the 2nd International Conference on Pervasive Embedded Computing and Communication Systems, PECCS 2012, Rome, Italy, 24–26 February, 2012, SciTePress (2012): 188.

[4] Lai X., Massey J. L., A proposal for a new block encryption standard, Damgøard I. B. (ed.), Advances in Cryptology – EUROCRYPT '90. LNCS 473, Springer, Berlin (1991): 389.

[5] Janicka-Lipska I., Stokłosa J., Boolean feedback functions for full-length nonlinear shift registers, Journal of Telecommunications and Information Technology 5 (2004,): 28.

[6] Rueppel R. A., Analysis and Design of Stream Ciphers, Springer, Berlin (1986).

[7] Daemen J., Rijmen V., AES Proposal: Rijndael, AES'99 (1999); http://csrc.nist.gov/CryptoToolkit/aes/ rijndael/1999

[8] Fuller J., Millan W., On Linear Redundancy in the AES S-Box (2002); http://eprint.iacr.org/2002/111.

[9] Fuller J., Millan W., On Linear Redundancy in S-Boxes, FSE 2003, LNCS 2887, Springer(2003): 74.