HaF
A New Family of Hash Functions

Tomasz Bilski, Krzysztof Bucholc, Anna Grocholewska-Czurylo, Janusz Stoklosa
Institute of Control and Information Engineering, Poznań University of Technology
pl. Marii Skłodowskiej Curie 5, Poznan, Poland

Keywords: Security, Privacy, Trust, Hash Functions, S-box design.

Abstract: Paper presents a family of parameterized hash functions allowing for flexibility between security and performance. The family consists of three basic hash functions: HaF-256, HaF-512 and HaF-1024 with message digests equal to 256, 512 and 1024 bits, respectively. Details of functions' structure are presented. Method for obtaining function's S-box is described along with the rationale behind it. Security considerations are discussed.

1 INTRODUCTION
In many cryptographic applications it is necessary to generate a shortened form of a much longer message. The shortened form called digest of the message or hash value, is produced by means of a hash function. A hash function \( h \) operates on an arbitrary-length message \( m \) and returns a hash value \( h(m) \) of a fixed length. Cryptographic hash functions have many information security applications. We use hash function to verify message integrity. Keyed hash function is used for message authentication.

Recently we can see substantial effort in designing of new cryptographic hash functions. For example, as many as 64 proposals were submitted to NIST SHA-3 competition, for new hash function, in October 2008 (Regenscheid, 2009).

Our objective is to ensure that the security of HaF is high and its performance is significantly satisfactory.

The paper is organized as follows: Section 2 presents general overview of the family of algorithms. Method for obtaining function's S-box, along with the rationale behind it, is described in Section 3. In Section 4 we discuss security considerations. Section 5 is devoted to reference implementation and the algorithm performance. Concluding remarks are presented in Section 6.

2 PARAMETERIZED FAMILY HaF OF HASH FUNCTIONS

2.1 Design Principles
The following assumptions were taken into account during design process:
- the 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) (Biham, 2006).

2.2 Description of HaF
The HaF family is formed of 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 for HaF is based on Merkle-Damgård paradigm proposed by Biham and Dunkelman (Menezes, 1997); (Biham, 2006) (Figure 2.1).

After formatting the original message \( m \) we have the message \( M \). We divide \( M \) into blocks \( M_k, M_{k+1}, \ldots, M_{k-1}, k \in \{1,2,\ldots\} \), and each block \( M_k \) is processed with the salt \( s \) by the iterative compression function...
\( \varphi \) (Biham, 2006). The output \( H_k \) is the final result of the function.

\[
\begin{align*}
&\text{Append padding bits} \\
&\text{Append length string} \\
&\text{Formatted message } M = M_0 \parallel M_1 \parallel \ldots \parallel M_{k-1} \\
&H_{k+1} = h(m) = H_k
\end{align*}
\]

Figure 2.1: General model for HaF.

### 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;
- \( \text{length} \) – bitstring representing the length of the original message \( m \), \(|\text{length}| = 128 \);
- \( \text{lsb}(v) = q \) 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 \( v \);
- \( v \triangleleft t \) – \( t \)-bit left rotation of a string \( v \);
- \( v \oplus w \) – bitwise XOR of strings \( v \) and \( w \);
- \( v \| w \) – addition mod \( 2^n \) of integers represented (in base 2) by strings \( v \) and \( w \);
- \( p_1(x) \odot p_2(x) \) – 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;
- \( \varphi \) – 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 original message length. As a result we obtain the formatted message \( M = M_0 \parallel M_1 \parallel \ldots \parallel M_{k-1} \) for some positive integer \( k \), where \( M_i \) is a block of \( M \). Therefore, \( M = m \parallel 10^t \parallel \text{length} \), 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: \( \varphi : \{0,1\}^\mu \times \{0,1\}^\eta \times \{0,1\}^\sigma \rightarrow \{0,1\}^\rho \). The integers \( \mu, \eta, \sigma \) are lengths of block \( M_i \), chaining variable \( H_i \), and salt \( s \), respectively, where \(|M_i| = |H_i| = |s| = 16n \) and \( i = 0, 1, \ldots, k-1 \). The integer \( \rho \) is the length of the resulting hash value \( h(m) = H_k \), \(|h(m)| = 16n \).

Figure 2.2: Method of one block processing.
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 $\text{HaF-256}$, $\text{HaF-512}$ and $\text{HaF-1024}$ 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 depicted in Figure 2.2. $M_i$, $H_i$ and $s$ are inputs for $\varpi$. Before processing in round $l$, $l = 1$ or 2, the block $M_i$ is modified. In the round $l=1$ 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: $N_i^* = N_i \ll \text{lsb}_4(N_i)$. Before processing in the round $l=2$ 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 $l=1$. Next, all subblocks $A_0, A_1, \ldots, A_{15}$ are concatenated giving $H_{i+1} = A_0 || A_1 || \ldots || A_{15}$.

2.2.4 Round Function

The round function (Fig. 2.3) 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 $\text{lsb}_4(N_i)$ and added (mod 2 of respective bits) to $H_i$. Next the block $H_i \oplus (N_i\ll \text{lsb}_4(N_i))$ is divided into 16 subblocks of equal length: $A_0, A_1, \ldots, A_{15}$. They are processed by a step function. After processing they are concatenated giving $H_i^*$. The output $N_i^* = N_i \ll \text{lsb}_4(N_i)$.

2.2.5 Step Function

The essential part of the round is the step function $F_j$ (Fig. 2.4). In each round the step function is executed 16 times, for $j=0, 1, \ldots, 15$.

Let $\text{GF}[x]_n$ be a set of polynomials over $\text{GF}(2)$ of the degree smaller than $n$. If $w(x) \in \text{GF}[x]_n$, then $w(x) = w_{n-1}x^{n-1} \oplus w_{n-2}x^{n-2} \oplus \ldots \oplus w_2x^2 \oplus w_1x \oplus w_0$ or $w(x) = w_0 + w_1x + \ldots + w_{n-1}x^{n-1}$.
simply \(w(x) = w_nw_{n-1}...w_1w_0\), where \(w_i \in \text{GF}(2)\) for \(r \in \{0,1,...,n-1\}\). Let \(u(x), v(x), w(x) \in \text{GF}(2^n)\). We define two operations on polynomials, addition (\(\oplus\)) and multiplication (\(\otimes\)); \(u(x) = v(x) \oplus w(x) \iff u_i = v_i \oplus w_i\) for \(i = 1,2,...,n\), and \(u(x) = v(x) \otimes w(x) = v(x) \cdot w(x) \mod R(x)\), 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 hexadecimal form, are given in Table 2.1.

<table>
<thead>
<tr>
<th>(n)</th>
<th>(\text{R}(x))</th>
<th>Hexadecimal representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>(x^{16}+x^4+x^2+x+1)</td>
<td>10021</td>
</tr>
<tr>
<td>32</td>
<td>(x^{32}+x^{26}+x^{25}+x^{24}+x^{19}+x^{18}+x^{14}+x^{12}+x^7+x^5+x^4+x+1)</td>
<td>10000000C5</td>
</tr>
<tr>
<td>64</td>
<td>(x^{64}+x^{48}+x^{44}+x^{40}+x^{36}+x^{32}+x^{28}+x^{24}+x^{20}+x^{16}+x^{12}+x^{8}+x^{4}+1)</td>
<td>10000000000000001B</td>
</tr>
</tbody>
</table>

The reduction polynomials must be irreducible; they are presented in Table 2.2.

<table>
<thead>
<tr>
<th>(n)</th>
<th>(a_0)</th>
<th>(a_2)</th>
<th>(a_3)</th>
<th>(a_5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>89CB</td>
<td>0001</td>
<td>0001</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>AC2D</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>64</td>
<td>EDC0</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

The equivalent form of polynomials is used. Before processing they must be assigned to \(A_0||A_2||A_3||A_5\) in such a way that \(h_i = A_i\), for \(r = 0,1,...,15\).

The multiplication modulo \(2^n + 1\) of \(n\)-bit integers with the zero block corresponding to \(2^n\) is denoted by \(\oplus\) (Lai, 1991).

<table>
<thead>
<tr>
<th>(n)</th>
<th>(H_0 = h_0)</th>
<th>(h_1)</th>
<th>...</th>
<th>(h_{15})</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>34D906D3E3B5298EAC26F9DF2AC5A0D3</td>
<td>D84B0576CB2CC52517C6F8880A90C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>34D906D3E3B5298EAC26F9DF2AC5A0D3</td>
<td>D84B0576CB2CC52517C6F8880A90C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>34D906D3E3B5298EAC26F9DF2AC5A0D3</td>
<td>D84B0576CB2CC52517C6F8880A90C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The initial values \(H_0 = h_0\) are given in Table 2.3 (\(H_0\) for \(n = 64\) is obtained as the hexadecimal form of consecutive 512 decimal places after the decimal point of \(\pi\) broken up into groups of 32). Before processing they must be assigned to \(A_0||A_2||A_3||A_5\) in such a way that \(h_i = A_i\), for \(r = 0,1,...,15\).

### 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 \(\text{GF}(2^n)\) generating maximum length sequences, additionally equipped with nonlinear feedback NL, and clocked 16 times (Fig. 2.5). The corresponding approach dealing with the use of feedback shift registers (over \(\text{GF}(2^n)\)) in the construction of hash functions has been presented in (Janicka-Lipska, 2004; Stoklosa, 1995).

![Figure 2.5: Equivalent form of round function.](image-url)
Maximum 16-stages linear feedback shift register defined over GF(2^n) generates the sequence of period length \( T = 2^{16n} - 1 \) \((n = 16 \text{ or } 32 \text{ or } 64)\). This period length is considerably decreased by the nonlinear circuit (NL in Fig. 2.5). The processing of every consecutive block \( M_i \) of the formatted message modifies initial content of the register and consequently changes the period (meant as a sequence of states) of the FSR. The same effect can be observed when adding \( H_i \) to the result of processing the input by two rounds to obtain \( H_{i+1} \), (Fig. 2.2). This implies that collisions exist but finding them is difficult.

In order to achieve randomized hashing we use the construction (see Fig. 2.2) in which the random salt value \( s \) is added (mod 2) to each block \( M_i \) (Biham, 2006).

The function defined by the nonlinear circuit is a nonlinear \( 8n \)-argument function, \( n = 16 \text{ or } 32 \text{ or } 64 \). 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 (Rueppel, 1986). Time needed for the attack is equal to time of the birthday attack, i.e. \( O(2^{3n}) \).

The sequence produced by the nonlinear circuit is immune to correlation attack (Rueppel, 1986).

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:

\[
S : F_2^n \rightarrow F_2^n
\]

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

\[
f : F_2^n \rightarrow F_2
\]

An S-box \( S \) can be decomposed into the sequence \( S = (f_1, f_2, \ldots, f_n) \) of 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 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 16x16 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 [Daemen 1999] 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 16x16 S-box can be stored as a table of 65536 word values. Index for this table is an input to the S-box function, i.e., \( x_1, x_2, \ldots, x_{16} \). 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’s consider a smaller S-box of size \( 8 \times 8 \). For presentation convenience such S-box can be displayed as a 2-dimensional table (Table 3.1). The input represented as a two digit hexadecimal number \( HL \) is divided into the low order digit \( L \) on the horizontal axis and the high order digit \( H \) 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.

<table>
<thead>
<tr>
<th>L</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>B</td>
<td>E9</td>
<td>9B</td>
<td>1B</td>
<td>35</td>
<td>DC</td>
<td>1E</td>
<td>56</td>
<td>A5</td>
<td>B2</td>
<td>74</td>
<td>34</td>
<td>12</td>
<td>D5</td>
<td>64</td>
<td>15</td>
<td>6D</td>
</tr>
<tr>
<td>C</td>
<td>6B</td>
<td>4B</td>
<td>8E</td>
<td>FB</td>
<td>CB</td>
<td>E9</td>
<td>D9</td>
<td>A1</td>
<td>D5</td>
<td>68</td>
<td>0F</td>
<td>EF</td>
<td>43</td>
<td>89</td>
<td>E1</td>
<td>8F</td>
</tr>
<tr>
<td>D</td>
<td>59</td>
<td>C7</td>
<td>3A</td>
<td>F4</td>
<td>1A</td>
<td>13</td>
<td>09</td>
<td>50</td>
<td>A9</td>
<td>63</td>
<td>32</td>
<td>DF</td>
<td>C9</td>
<td>CC</td>
<td>AD</td>
<td>0A</td>
</tr>
<tr>
<td>E</td>
<td>5B</td>
<td>06</td>
<td>B6</td>
<td>P7</td>
<td>47</td>
<td>BF</td>
<td>BE</td>
<td>44</td>
<td>67</td>
<td>7B</td>
<td>B7</td>
<td>21</td>
<td>AF</td>
<td>53</td>
<td>93</td>
<td>FF</td>
</tr>
<tr>
<td>F</td>
<td>57</td>
<td>0B</td>
<td>AE</td>
<td>4D</td>
<td>C4</td>
<td>D1</td>
<td>16</td>
<td>A4</td>
<td>D6</td>
<td>30</td>
<td>07</td>
<td>40</td>
<td>8B</td>
<td>90</td>
<td>BB</td>
<td>8C</td>
</tr>
<tr>
<td>0</td>
<td>EF</td>
<td>81</td>
<td>A8</td>
<td>39</td>
<td>1D</td>
<td>D4</td>
<td>7A</td>
<td>48</td>
<td>OD</td>
<td>E2</td>
<td>CA</td>
<td>B0</td>
<td>C7</td>
<td>DE</td>
<td>28</td>
<td>DA</td>
</tr>
<tr>
<td>1</td>
<td>97</td>
<td>D2</td>
<td>F2</td>
<td>84</td>
<td>19</td>
<td>B3</td>
<td>89</td>
<td>A7</td>
<td>E4</td>
<td>66</td>
<td>49</td>
<td>95</td>
<td>99</td>
<td>05</td>
<td>A3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>E4</td>
<td>61</td>
<td>03</td>
<td>C2</td>
<td>73</td>
<td>F3</td>
<td>B8</td>
<td>77</td>
<td>E0</td>
<td>F8</td>
<td>9C</td>
<td>5C</td>
<td>5F</td>
<td>BA</td>
<td>22</td>
<td>FA</td>
</tr>
<tr>
<td>3</td>
<td>F0</td>
<td>2E</td>
<td>FE</td>
<td>9B</td>
<td>7C</td>
<td>D3</td>
<td>70</td>
<td>94</td>
<td>7D</td>
<td>E0</td>
<td>EA</td>
<td>11</td>
<td>8A</td>
<td>5D</td>
<td>00</td>
<td>EC</td>
</tr>
<tr>
<td>4</td>
<td>DA</td>
<td>27</td>
<td>04</td>
<td>7F</td>
<td>57</td>
<td>17</td>
<td>B5</td>
<td>78</td>
<td>62</td>
<td>38</td>
<td>AB</td>
<td>AA</td>
<td>0B</td>
<td>3E</td>
<td>52</td>
<td>4C</td>
</tr>
<tr>
<td>5</td>
<td>4B</td>
<td>CB</td>
<td>18</td>
<td>75</td>
<td>CO</td>
<td>FD</td>
<td>29</td>
<td>4A</td>
<td>86</td>
<td>7C</td>
<td>BD</td>
<td>SE</td>
<td>01</td>
<td>ED</td>
<td>46</td>
<td>45</td>
</tr>
<tr>
<td>6</td>
<td>B4</td>
<td>FC</td>
<td>83</td>
<td>02</td>
<td>54</td>
<td>DF</td>
<td>6C</td>
<td>CD</td>
<td>3C</td>
<td>6A</td>
<td>B1</td>
<td>3D</td>
<td>8C</td>
<td>24</td>
<td>B8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>DC</td>
<td>55</td>
<td>71</td>
<td>96</td>
<td>65</td>
<td>1C</td>
<td>5B</td>
<td>31</td>
<td>A0</td>
<td>26</td>
<td>6F</td>
<td>29</td>
<td>14</td>
<td>1F</td>
<td>6D</td>
<td>C6</td>
</tr>
<tr>
<td>8</td>
<td>EE</td>
<td>1D</td>
<td>03</td>
<td>C2</td>
<td>73</td>
<td>F3</td>
<td>B8</td>
<td>77</td>
<td>E0</td>
<td>F8</td>
<td>9C</td>
<td>5C</td>
<td>5F</td>
<td>BA</td>
<td>22</td>
<td>FA</td>
</tr>
<tr>
<td>9</td>
<td>0F</td>
<td>2E</td>
<td>FE</td>
<td>9B</td>
<td>7C</td>
<td>D3</td>
<td>70</td>
<td>94</td>
<td>7D</td>
<td>E0</td>
<td>EA</td>
<td>11</td>
<td>8A</td>
<td>5D</td>
<td>00</td>
<td>EC</td>
</tr>
<tr>
<td>10</td>
<td>D8</td>
<td>27</td>
<td>04</td>
<td>7F</td>
<td>57</td>
<td>17</td>
<td>B5</td>
<td>78</td>
<td>62</td>
<td>38</td>
<td>AB</td>
<td>AA</td>
<td>0B</td>
<td>3E</td>
<td>52</td>
<td>4C</td>
</tr>
<tr>
<td>11</td>
<td>4B</td>
<td>CB</td>
<td>18</td>
<td>75</td>
<td>CO</td>
<td>FD</td>
<td>29</td>
<td>4A</td>
<td>86</td>
<td>7C</td>
<td>BD</td>
<td>SE</td>
<td>01</td>
<td>ED</td>
<td>46</td>
<td>45</td>
</tr>
<tr>
<td>12</td>
<td>B4</td>
<td>FC</td>
<td>83</td>
<td>02</td>
<td>54</td>
<td>DF</td>
<td>6C</td>
<td>CD</td>
<td>3C</td>
<td>6A</td>
<td>B1</td>
<td>3D</td>
<td>8C</td>
<td>24</td>
<td>B8</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>DC</td>
<td>55</td>
<td>71</td>
<td>96</td>
<td>65</td>
<td>1C</td>
<td>5B</td>
<td>31</td>
<td>A0</td>
<td>26</td>
<td>6F</td>
<td>29</td>
<td>14</td>
<td>1F</td>
<td>6D</td>
<td>C6</td>
</tr>
<tr>
<td>14</td>
<td>EE</td>
<td>1D</td>
<td>03</td>
<td>C2</td>
<td>73</td>
<td>F3</td>
<td>B8</td>
<td>77</td>
<td>E0</td>
<td>F8</td>
<td>9C</td>
<td>5C</td>
<td>5F</td>
<td>BA</td>
<td>22</td>
<td>FA</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>2E</td>
<td>FE</td>
<td>9B</td>
<td>7C</td>
<td>D3</td>
<td>70</td>
<td>94</td>
<td>7D</td>
<td>E0</td>
<td>EA</td>
<td>11</td>
<td>8A</td>
<td>5D</td>
<td>00</td>
<td>EC</td>
</tr>
<tr>
<td>16</td>
<td>H</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
</tbody>
</table>

Cryptographically strong S-box should possess some properties that are universally agreed upon among researchers. Such S-box should be balanced, highly nonlinear, have 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 cryptanalysis 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...
with an additional invertible affine transformation. This affine transformation does not affect the nonlinearity of the S-box, its XOR profile nor 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 similar method of generating S-boxes. Additionally it takes into account results of some recent studies (Fuller, 2002; Fuller, 2003) 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 so called inverse mapping $x \rightarrow x^{-1}$, where $x^{-1}$ denotes the multiplicative inverse in a finite field $\text{GF}(2^n)$:

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

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

For any prime integer $p$ and any integer $n$ ($n = 1, 2, ..., p$), there is a unique field with $p^n$ elements, denoted $\text{GF}(p^n)$. In cryptography $p$ almost always takes the value of 2. To generate an inverse mapping in $\text{GF}(2^n)$ we need an irreducible polynomial that defines a Galois field and another polynomial that would be a 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 $\text{F}_2$. For example, in case of AES, where S-box is of size 8x8 we operate mostly on bytes represented as $b_0b_1b_2b_3b_4b_5b_6b_7$ which corresponds to the following polynomial:

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

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

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

A generator in 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 polynomial 03 is a generator in $\text{GF}(2^8)$ with irreducible polynomial 11B (as in AES), but it is not a generator in $\text{GF}(2^9)$ with 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 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 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's 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 irreversible and that is a prerequisite for most applications of S-boxes. In case of AES cipher this affine transformation is given by the following equation:

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

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

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

193
The algorithm used for generating S-box $S$ of HaF function in this paper uses the same transformation, however adopted for $16 \times 16$ S-box size and with the constant part of this transformation (namely $c_i$) taken at random so that 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 HaF S-box is the absence of cycles. Cycle is such a sequence of S-box values $S_0, S_1, \ldots, S_{k-1}$ where $S_{i+1} \mod k = S(S_i)$. 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 previous paragraph changes number of cycles in an S-box, without changing its nonlinear properties. Note that fixed points are also short cycles where $k = 1$.

Cycles are removed in a procedure with two steps: 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 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 (Fuller, 2002; Fuller, 2003), S-boxes based on 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 affine transformations). 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 2-cycle S-box is carried out in such a way that it will at the same time 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 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 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 Boolean functions covered by various cryptographic criteria (such as algebraic degree and nonlinearity) remain unchanged by affine transformations. Absolute values of Walsh transform as well as autocorrelation function are only rearranged by 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 REFERENCCE IMPLEMENTATIONS OF HaF-256

HaF-256 algorithm was implemented using C++ language and Microsoft Visual Studio 2008 environment. Two reference implementations were separately developed and tested using of reference data. The results produced by the implementations were compared with each other in order to verify algorithm implementation accuracy.

To evaluate performance a 20 MB text file was processed and the time was measured. Several options were considered. For Windows (64-bit Windows 7) two compilers were used: native Visual Studio C++ compiler and Intel C++ compiler. The code was generated for 32-bit and 64-bit platforms.

<table>
<thead>
<tr>
<th>System</th>
<th>Platform</th>
<th>Compiler</th>
<th>Performance [MB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 7</td>
<td>32-bit</td>
<td>VS2008</td>
<td>1.29</td>
</tr>
<tr>
<td>Windows 7</td>
<td>64-bit</td>
<td>VS2008</td>
<td>1.60</td>
</tr>
<tr>
<td>Windows 7</td>
<td>32-bit</td>
<td>Intel</td>
<td>2.99</td>
</tr>
<tr>
<td>Windows 7</td>
<td>64-bit</td>
<td>Intel</td>
<td>3.13</td>
</tr>
<tr>
<td>Linux</td>
<td>32-bit</td>
<td>GCC</td>
<td>0.74</td>
</tr>
<tr>
<td>Linux</td>
<td>32-bit</td>
<td>GCC -O</td>
<td>1.17</td>
</tr>
<tr>
<td>Linux</td>
<td>32-bit</td>
<td>GCC -O2</td>
<td>1.36</td>
</tr>
<tr>
<td>Linux</td>
<td>32-bit</td>
<td>GCC -O3</td>
<td>2.33</td>
</tr>
</tbody>
</table>
Linux (Fedora 9) GCC compiler was used without and with optimization. The code was generated for 32-bit platform only. PC machine with 2.2 GHz Athlon-64 processor was used as a testing platform. The results are presented in Table 4.1.

As shown in Table 4.1, the best result was obtained using Intel compiler. For 64-bit platform the performance was about 4.7% better than for 32-bit platform. Bigger improvement may be expected for HaF-1024 implemented using 64-bit variables. For the sake of comparison we measured performance of one of the NIST SHA 3 competition finalists – BLAKE hash function [Aumasson, 2011], using the same computer. BLAKE algorithm is very simple and does not use S-boxes. Results are presented in Table 4.2. As we can see, BLAKE significantly outperforms HaF. But in some applications it does not matter. For example, it takes 0.03 s to compute hash for a 100 kB message using HaF-256, whereas 0.0005 s is required for BLAKE-256.

<table>
<thead>
<tr>
<th>Compiler</th>
<th>32-bit version</th>
<th>64-bit version</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS2008</td>
<td>175 MB/sec</td>
<td>256 MB/s</td>
</tr>
<tr>
<td>Intel C++ Compiler</td>
<td>204 MB/sec</td>
<td>240 MB/s</td>
</tr>
</tbody>
</table>

Table 4.2: BLAKE-256 performance.

5 CONCLUDING REMARKS

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

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

We currently experiment with fault attacks on HaF implementation, so it should be possible to verify what are the advantages of this approach.

ACKNOWLEDGEMENTS

This work was supported by the Polish Ministry of Science and Higher Education as a 2010–2013 research project.

REFERENCES


