Hash Function *Luffa*

Specification Ver. 1.0.1

Christophe De Cannière  
ESAT-COSIC, Katholieke Universiteit Leuven

Hisayoshi Sato, Dai Watanabe  
Systems Development Laboratory, Hitachi, Ltd.

15 January 2009
<table>
<thead>
<tr>
<th>E</th>
<th>Implementations of Message Injection Function $\mathcal{MI}$</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-1</td>
<td>$w = 3$</td>
<td>24</td>
</tr>
<tr>
<td>E-2</td>
<td>$w = 4$</td>
<td>25</td>
</tr>
<tr>
<td>E-3</td>
<td>$w = 5$</td>
<td>26</td>
</tr>
</tbody>
</table>
1 Introduction

This document specifies a family of cryptographic hash function algorithms \textit{Luffa}. The input and the output lengths of the algorithms are summarized in Table 1.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Message length (bits)</th>
<th>Hash length (bits)</th>
<th>Security (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{Luffa}-224</td>
<td>$&lt; 2^{64}$</td>
<td>224</td>
<td>112</td>
</tr>
<tr>
<td>\textit{Luffa}-256</td>
<td>$&lt; 2^{64}$</td>
<td>256</td>
<td>128</td>
</tr>
<tr>
<td>\textit{Luffa}-384</td>
<td>$&lt; 2^{128}$</td>
<td>384</td>
<td>192</td>
</tr>
<tr>
<td>\textit{Luffa}-512</td>
<td>$&lt; 2^{128}$</td>
<td>512</td>
<td>256</td>
</tr>
</tbody>
</table>

Firstly, the notations used in the document is defined in Section 2. The hash function \textit{Luffa} consists of the chaining and the mixing function used in each round of the chaining. The chaining and the underlying mixing function are described in Section 3 and 4 respectively. An optional usage of the hash function \textit{Luffa} are given in Section 5. In addition, some useful informations to implement the hash function such as the test vectors are given in Appendices.
2 Preliminary

In this section, the basic terms and notations to describe the specification of Luffa are defined.

2.1 Notations

2.1.1 Parameters

- \( L \): The message length in bits
- \( L' \): The padded message length in bits
- \( N \): The number of message block (of 256 bits)
- \( w \): The number of sub-permutations (described in 3.2)
- \( n_h \): The hash length
- \( n_b \): The block length (Fixed to 256 bits in this document)
- \( V_j \): The starting variables
- \( H_j^{(i)} \): The variable which specifies the intermediate values of the state at \( i \)-th round, \( j \)-th block
- \( M_j^{(i)} \): The message block at the \( i \)-th round
- \( i \): A subscript which specifies the round
- \( j \): A subscript which specifies the sub-permutation
- \( k \): A subscript which specifies the word
- \( l \): A subscript which specifies the bit position in a word
- \( r \): A subscript which specifies the step
- \( MI \): The message injection function
- \( P \): The permutation of \( n_b w \) bits
- \( Q_j \): The permutation dealing with \( j \)-th block of \( n_b \) bits
- \( OF \): The output function
- \( b_{j,k,l} \): The variable which specifies the \( k \)-th word, \( l \)-th bit of the input of the \( j \)-th block permutation \( Q_j \)
- \( a_{j,k,l}^{(i,r)} \): The variable which specifies the \( k \)-th word, \( l \)-th bit of the input of \( i \)-th round, \( j \)-th block, \( r \)-th step function
- \( x_{j,k,l}^{(i,r)} \): The variable which specifies the \( k \)-th word, \( l \)-th bit of the output of SubCrumb at \( i \)-th round, \( j \)-th block, \( r \)-th step
- \( y_{j,k,l}^{(i,r)} \): The variable which specifies the \( k \)-th word, \( l \)-th bit of the output of MixWord at \( i \)-th round, \( j \)-th block, \( r \)-th step
$c_{j,k,l}^{(r)}$: The variable which specifies the $k$-th word, $l$-th bit of the constant used in $j$-th block, $r$-th step function

### 2.1.2 Symbols

In this paper, the following symbols are used to identify the operations.

- $\oplus$: Bitwise XOR operation
- $\land$: Bitwise AND operation
- $\|$: Concatenation of two bit strings
- $\gg n$: Rotation $n$ bits to the right (A 32-bit register is expected)
- $\ll n$: Rotation $n$ bits to the left (A 32-bit register is expected)
- $0x$: Hexadecimal prefix

On the other hand, some pseudo codes are given in the paper. They are written in C language manner and 32-bit registers are expected. In order to remove any ambiguity, we also list up the operation used in the pseudo codes as follows:

- $\wedge$: XOR operation
- $\lVert$: OR operation
- $\gg n$: Shift $n$ bits to the right
- $\ll n$: Shift $n$ bits to the left

### 2.2 Data Structure

The basic data size is a 32-bit and it is called a *word*. A 4 bytes data is stored to a word in the big endian manner. In other words, the given 4 bytes data $x_0, \ldots, x_3$ is stored into a word $a$ as follows:

$$a = [\text{MSB}] x_0 \| x_1 \| x_2 \| x_3 [\text{LSB}],$$

where [MSB] (and [LSB]) means the most (and least) significant byte of the word.

In the specification of *Luffa*, a 256-bit data block is stored in 8 32-bit registers. In order to remove any ambiguity, we also define the ordering of a
32 bytes data in 8 words. A 32 bytes data \( X = x_0, x_1, \ldots, x_{31} \) is stored to 8 32-bit registers \( a_0, \ldots, a_7 \) in the following manner:

\[
X = [\text{MSW}] \ a_0||a_1||\cdots||a_7 \ [\text{LSW}], \\
a_k = [\text{MSB}] \ x_{4k}||x_{4k+1}||x_{4k+2}||x_{4k+3} \ [\text{LSB}], \quad 0 \leq k < 8,
\]

where [MSW] (and [LSW]) means the most (and least) significant word.

A bit position in a word sequence is denoted by subscripts. Let \( a_0, \ldots, a_n \) be a word sequence. Then the \( l \)-th bit (from the least significant bit) of the \( k \)-th word is denoted by \( a_{k,l} \), where the least significant bit is the 0-th bit. In other words, the bit information of \( a_k \) is given by

\[
a_k = [\text{msb}] \ a_{k,31}||a_{k,30}||\cdots||a_{k,1}||a_{k,0} \ [\text{lsb}],
\]

where [msb] and [lsb] mean the most and the least significant bit of the word respectively.

### 2.3 Iterations

The message processing of Luffa is a chaining of a mixing function of a fixed length input and a fixed length output. We call the mixing function as a \textit{round function}. The outline of the mixing function is defined in Section 3. A term \textit{round} means the procedure to apply the round function.

The building block of the round function is a family of non-linear permutations defined in Section 4. It consists of iterations of a sub-function called a \textit{step function}. A term \textit{step} means the procedure to apply the step function.

In order to clarify the round, the super-script with a parenthesis is used. I.e., the input to the \( i \)-th round function is denoted by \( X^{(i-1)} \). The corresponding output of the round function is denoted by \( X^{(i)} = \text{Round}(X^{(i-1)}) \). In the same manner, the input to the \( r \)-th step function of the \( i \)-th round is denoted by \( X^{(i-1,r-1)} \). The corresponding output of the step function is denoted by \( X^{(i-1,r)} = \text{Step}(X^{(i-1,r-1)}) \). The round can be abbreviated if it is not necessary in the context.

The intermediate state of Luffa consists of 8\( w \) words, where \( w \geq 3 \) is a positive integer (See Table 2 for the choice of \( w \)). An 8 words data is called a \textit{block}. The \( l \)-th bit of the input of \( i \)-th round, \( r \)-th step, \( j \)-th block, \( k \)-th word is denoted by \( a_{j,k,l}^{(i-1,r-1)} \).
Figure 1: A generic construction of a hash function based on a permutation

3 Chaining

The chaining of Luffa is a variant of a sponge function [1][2]. Figure 1 shows the basic structure of the chaining. The chaining of a hash function consists of the intermediate mixing $C'$ (called a round function) and the finalization $C''$. In addition to above two functions, the message padding is defined in this section. The starting variables $V_0, V_1, \ldots, V_{w-1}$ used in the chaining are given in Appendix A.

3.1 Message Padding

Suppose that the length of the message $M$ is $l$ bits. First of all, the bit string 100...0 is appended to the end of the message. The number of zeros $k$ should be the smallest non-negative integer which satisfies the equation $l + 1 + k \equiv 0 \mod 256$. Therefore the length of the padded message should be a multiple of 256 bits.

3.2 Round Function

The round function is a composition of a message injection function $MI$ and a permutation $P$ of $w \cdot n_b$ bits input. The permutation is divided into plural sub-permutation $Q_j$ of $n_b$ bits input (See Figure 2). Let the input of the $i$-th
round be \((H_0^{(i-1)}, \ldots, H_{w-1}^{(i-1)})\), then the output of the \(i\)-th round is given by

\[
H_j^{(i)} = Q_j(X_j), \quad 0 \leq j < w,
\]

\[
X_0 \| \cdots \| X_{w-1} = MI(H_0^{(i-1)}, \ldots, H_{w-1}^{(i-1)}, M^{(i)}),
\]

where \(H_j^{(0)} = V_j\).

In the specification of \(Luffa\), the input length of the sub-permutation \(Q_j\) is fixed to \(n_b = 256\) bits, and the number of the sub-permutations \(w\) is defined in Table 2.

<table>
<thead>
<tr>
<th>Hash length (n_b)</th>
<th>Number of permutations (w)</th>
</tr>
</thead>
<tbody>
<tr>
<td>224</td>
<td>3</td>
</tr>
<tr>
<td>256</td>
<td>3</td>
</tr>
<tr>
<td>384</td>
<td>4</td>
</tr>
<tr>
<td>512</td>
<td>5</td>
</tr>
</tbody>
</table>

The message injection functions can be represented by the matrix over a ring \(GF(2^8)^{32}\). The definition polynomial of the field is given by \(\phi(x) = x^8 + x^4 + x^3 + x + 1\). The map from an 8 words value \((a_0, \ldots, a_7)\) to an element of the ring is defined by \((\sum_{0 \leq k < 8} a_k x^k)_{0 \leq i < 32}\). Note that the least significant word \(a_7\) is the coefficient of the heading term \(x^7\) in the polynomial representation.

In order to remove any ambiguity, we also define the multiplication by \(0x02\) (equivalent to \(x\) in the polynomial representation) as the following pseudo code:

```pseudo
tmp = a[7];
a[7] = a[6];
a[6] = a[5];
a[5] = a[4];
a[2] = a[1];
a[1] = a[0] ^ tmp;
a[0] = tmp;
```

Copyright ©2008 Hitachi, Ltd. All rights reserved.
In the following, the matrices representing the massage injection functions $MI$ for $w = 3, 4, 5$ are defined. How to implementing $MI$ only with XORings and multiplications by $0x02$ is shown in Appendix E.

Figure 2: The round function ($w = 3$)

### 3.2.1 Message Injection Function for $w = 3$

The matrix representation of the massage injection function $MI$ for $w = 3$ is defined by

$$
\begin{pmatrix}
X_0 \\
X_1 \\
X_2
\end{pmatrix} =
\begin{pmatrix}
3 & 2 & 2 & 1 \\
2 & 3 & 2 & 2 \\
2 & 2 & 3 & 4
\end{pmatrix}
\begin{pmatrix}
H_0^{(i-1)} \\
H_1^{(i-1)} \\
H_2^{(i-1)} \\
M^{(i)}
\end{pmatrix},
$$

where numerics $0x01$, $0x02$, $0x03$, $0x04$ correspond to polynomials $1$, $x$, $x+1$, $x^2$ respectively. The prefix $0x$ is omitted in order to reduce the redundancy.
3.2.2 Message Injection Function for $w = 4$

The matrix representation of the massage injection function $MI$ for $w = 4$ is defined by

$$
\begin{pmatrix}
X_0 \\
X_1 \\
X_2 \\
X_3
\end{pmatrix} =
\begin{pmatrix}
4 & 6 & 6 & 7 & 1 \\
7 & 4 & 6 & 6 & 2 \\
6 & 7 & 4 & 6 & 4 \\
6 & 6 & 7 & 4 & 8
\end{pmatrix}
\begin{pmatrix}
H_0^{(i-1)} \\
H_1^{(i-1)} \\
H_2^{(i-1)} \\
H_3^{(i-1)} \\
M^{(i)}
\end{pmatrix}.
$$

3.2.3 Message Injection Function for $w = 5$

The matrix representation of the massage injection function $MI$ for $w = 5$ is defined by

$$
\begin{pmatrix}
X_0 \\
X_1 \\
X_2 \\
X_3 \\
X_4
\end{pmatrix} =
\begin{pmatrix}
0F & 08 & 0A & 0A & 08 & 01 \\
08 & 0F & 08 & 0A & 0A & 02 \\
0A & 08 & 0F & 08 & 0A & 04 \\
0A & 0A & 08 & 0F & 08 & 08 \\
08 & 0A & 0A & 08 & 0F & 10
\end{pmatrix}
\begin{pmatrix}
H_0^{(i-1)} \\
H_1^{(i-1)} \\
H_2^{(i-1)} \\
H_3^{(i-1)} \\
M^{(i)}
\end{pmatrix}.
$$

3.3 Finalization

The finalization consists of iterations of an output function $OF$ and a round function with a fixed message $0x00\ldots0$. If the number of (padded) message blocks is more than one, a blank round with a fixed message block $0x00\ldots0$ is applied at the beginning of the finalization.

The output function $OF$ XORs all block values and outputs the resultant 256-bit value. Let the output at the $i$-th iteration be $Z_i$, then the output function is defined by

$$
Z_i = \bigoplus_{j=0}^{w-1} H_j^{(N+i')},
$$

where $i' = i$ if $N = 1$ and $i' = i + 1$ otherwise.

The detailed output words are defined in Table 3. In fact, Luffa-224 just truncates the last 1 word of the output of Luffa-256.
Figure 3: The finalization function

Table 3: The hash values

<table>
<thead>
<tr>
<th>Hash length $n_h$</th>
<th>Hash value $H$</th>
</tr>
</thead>
<tbody>
<tr>
<td>224</td>
<td>$Z_{0,0} | \cdots | Z_{0,6}$</td>
</tr>
<tr>
<td>256</td>
<td>$Z_{0,0} | \cdots | Z_{0,7}$</td>
</tr>
<tr>
<td>384</td>
<td>$Z_{0,0} | \cdots | Z_{0,7} | Z_{1,0} | \cdots | Z_{1,3}$</td>
</tr>
<tr>
<td>512</td>
<td>$Z_{0,0} | \cdots | Z_{0,7} | Z_{1,0} | \cdots | Z_{1,7}$</td>
</tr>
</tbody>
</table>

4 Non-Linear Permutation

In this section, the detailed specification of the permutation $Q_j$. Some subscripts such as $i, j, r$ will be omitted in this section if it is clear in the context. For example, $a^{(i,r)}_{j,k,l}$ is denoted by $a_{k,l}$.

4.1 Outline

The *Luffa* hash function uses a non-linear permutation $Q_j$ whose input and output length is 256 bits. The permutation $Q_j$ is defined as a composition of an input tweak and iterations of a step function $\text{Step}$. The number of iterations of a step function is 8 and the tweak is applied only once per a
At the beginning of the step function process, the 256 bits data stored in 8 32-bit registers is denoted by $a_k^{(r)}$ for $0 \leq k < 8$. The data before applying the permutation $Q_j$ is denoted by $b_k$ and the data after the tweak is denoted by $a_k^{(0)}$. The step function consists of the following three functions: SubCrumb, MixWord, AddConstant. The pseudo code for $Q_j$ is given by

```c
Permute(a[8], j) { //Permutation Q_j
    Tweak(a);
    for (r = 0; r < 8; r++) {
        SubCrumb(a[0], a[1], a[2], a[3]);
        SubCrumb(a[4], a[5], a[6], a[7]);
        for (k = 0; k < 4; k++)
            MixWord(a[k], a[k+4]);
        AddConstant(a, j, r);
    }
}
```

Each function is described below in turn and the tweaks are described in Section 4.5.
4.2 SubCrumb

SubCrumb substitutes \( l \)-th bits of \( a_0, a_1, a_2, a_3 \) (or \( a_4, a_5, a_6, a_7 \)) by an Sbox \( S \) defined by

\[
S[16] = \{7, 13, 11, 10, 12, 4, 8, 3, 5, 15, 6, 0, 9, 1, 2, 14\}.
\]

Let the output of SubCrumb be \( x_0, x_1, x_2, x_3 \) (or \( x_4, x_5, x_6, x_7 \)). Then the substitution by SubCrumb is given by

\[
x_{3,l}||x_{2,l}||x_{1,l}||x_{0,l} = S[a_{3,l}||a_{2,l}||a_{1,l}||a_{0,l}]), \quad 0 \leq l < 32,
x_{7,l}||x_{6,l}||x_{5,l}||x_{4,l} = S[a_{7,l}||a_{6,l}||a_{5,l}||a_{4,l}]), \quad 0 \leq l < 32.
\]

![Figure 5: The input and output bits of the Sbox](image)

Appendix D shows the optimal instruction set for Intel® Core™2 Duo processors.

4.3 MixWord

MixWord is a linear permutation of two words. Figure 6 shows the outline of MixWord. Let the output words be \( y_k \) and \( y_{k+4} \) where \( 0 \leq k < 4 \). Then MixWord given by the following equations:

\[
y_{k+4} = x_{k+4} \oplus x_k,
y_k = x_k \ll \sigma_1,
y_k = y_k \oplus y_{k+4},
\]

\(^1\)Intel® is a registered trademark and Core™ is a trademark of Intel Corporation in the U.S. and other countries.

Copyright ©2008 Hitachi, Ltd. All rights reserved.
The parameters $\sigma_i$ are given by $\sigma_1 = 2, \sigma_2 = 14, \sigma_3 = 10, \sigma_4 = 1$.

### 4.4 AddConstant

AddConstant is given by

$$a_{j,k}^{(r)} = y_{j,k}^{(r-1)} \oplus c_{j,k}^{(r-1)}, \quad k = 0, 4.$$  

Note that the step constant $c_{j,k}^{(r-1)}$ is not equal to $c_{j',k}^{(r-1)}$ if $j \neq j'$. The step constants are generated sequentially from fixed initial values $c_{j,L}^{(0)}$ and $c_{j,R}^{(0)}$. The initial values are given in Appendix B. The constant generation function
generates two 32-bit constants $c_{j,0}^{(r-1)}$ and $c_{j,4}^{(r-1)}$ in the following manner:

\[
\begin{align*}
& t_L || t_R = c_{j,L}^{(r-1)} || c_{j,R}^{(r-1)}, \\
& t_L || t_R = f_L(t_L || t_R), \\
& c_{j,0}^{(r-1)} = t_L, \\
& t_L || t_R = f_L(t_R || t_L), \\
& c_{j,4}^{(r-1)} = t_L, \\
& c_{j,L}^{(r)} || c_{j,R}^{(r)} = t_R || t_L,
\end{align*}
\]

where the function $f_L$ is an LFSR of Galois configuration with defined by the polynomial $g$ given by

\[
g(x) = x^{64} + x^{63} + x^{62} + x^{58} + x^{55} + x^{54} + x^{52} + x^{50} + x^{49} + x^{46} + x^{43} + x^{40} + x^{38} + x^{37} + x^{35} + x^{34} + x^{30} + x^{28} + x^{26} + x^{24} + x^{23} + x^{22} + x^{18} + x^{17} + x^{12} + x^{11} + x^{10} + x^7 + x^3 + x^2 + 1.
\]

In order to remove any ambiguity, we also define a step of the constant generator as the following pseudo code:

```c
    c = tl >> 31;
    tl = (tl << 1) | (tr >> 31);
```

Copyright ©2008 Hitachi, Ltd. All rights reserved.
4.5 Tweaks

For each permutation $Q_j$, the least significant four words of a 256-bit input are rotated $j$ bits to the left in 32-bit registers. Let the $j$-th block, $k$-th word input be $b_{j,k}$, and the tweaked word (namely the input to the first step function) be $a_{j,k}^{(0)}$, then the tweak is defined by

$$a_{j,k,l}^{(0)} = b_{j,k,l}, \quad 0 \leq k < 4,$$
$$a_{j,k,l}^{(0)} = b_{j,k,((l-j) \mod 32)}, \quad 4 \leq k < 8.$$

5 Optional Usage

Dispite the size of the outputs being specified in Section 3.3, the design of Luffa allows to generate bit strings of arbitrary length by iterating the output function $OF$ and the round function $Round$. This feature is useful for some applications. On the other hand, it should be pointed out that a longer output with a small $w$ does not improve the security level.

References


A Starting Variables

The values are taken from Appendix C.1.

\[ V_{0,0} = 0x6d251e69, V_{0,1} = 0x44b051e0, V_{0,2} = 0x4eaa6fb4, V_{0,3} = 0xdbf78465, \\
V_{0,4} = 0x6e292011, V_{0,5} = 0x90152df4, V_{0,6} = 0xee058139, V_{0,7} = 0xdef610bb, \]

\[ V_{1,0} = 0xc3b44b95, V_{1,1} = 0xd9d2f256, V_{1,2} = 0x70ee9a0, V_{1,3} = 0xde099fa3, \\
V_{1,4} = 0x5d9b0557, V_{1,5} = 0x8fc944b3, V_{1,6} = 0xcf1ccf0e, V_{1,7} = 0x746cd581, \]

\[ V_{2,0} = 0xf7efc89d, V_{2,1} = 0xe58075d5, V_{2,2} = 0x04016ce5, V_{2,3} = 0xad659c05, \\
V_{2,4} = 0x0306194f, V_{2,5} = 0x24aa230a, V_{2,6} = 0x8b264ae7, \]

\[ V_{3,0} = 0x858075d5, V_{3,1} = 0x36d79cce, V_{3,2} = 0xe571f7d7, V_{3,3} = 0x204b1f67, \\
V_{3,4} = 0x35870c6a, V_{3,5} = 0x57e9e923, V_{3,6} = 0xb07224cc, V_{3,7} = 0x03e86cea. \]

B Constants

B–1 Initial Values

The initial values of the constant generator for \( Q_j \) are taken from Appendix C.2.

\[ c_{0,L}^{(0)} = 0x181cca53, \quad c_{0,R}^{(0)} = 0x380cde06, \]
\[ c_{1,L}^{(0)} = 0x5b6f0876, \quad c_{1,R}^{(0)} = 0xf16f8594, \]
\[ c_{2,L}^{(0)} = 0x7e106ce9, \quad c_{2,R}^{(0)} = 0x38979cb0, \]
\[ c_{3,L}^{(0)} = 0xbb62f364, \quad c_{3,R}^{(0)} = 0x92e93c29, \]
\[ c_{4,L}^{(0)} = 0x9a025047, \quad c_{4,R}^{(0)} = 0xcff2a940. \]
\[ \begin{align*}
&c_{0,0}^{(0)} = 0x303994a6, \quad c_{0,4}^{(0)} = 0xe0337818 \\
&c_{0,0}^{(1)} = 0xc0e65299, \quad c_{0,4}^{(1)} = 0x441ba90d \\
&c_{0,0}^{(2)} = 0x6cc33a12, \quad c_{0,4}^{(2)} = 0x7f34d442 \\
&c_{0,0}^{(3)} = 0xdc56983e, \quad c_{0,4}^{(3)} = 0x9389217f \\
&c_{0,0}^{(4)} = 0x1e00108f, \quad c_{0,4}^{(4)} = 0xe5a8bce6 \\
&c_{0,0}^{(5)} = 0x7800423d, \quad c_{0,4}^{(5)} = 0x5274baf4 \\
&c_{0,0}^{(6)} = 0x8f5b7882, \quad c_{0,4}^{(6)} = 0x26889ba7 \\
&c_{0,0}^{(7)} = 0x96e1db12, \quad c_{0,4}^{(7)} = 0x9a226e9d \\
&c_{1,0}^{(0)} = 0xb6de10ed, \quad c_{1,4}^{(0)} = 0x01685f3d \\
&c_{1,0}^{(1)} = 0x70f47aae, \quad c_{1,4}^{(1)} = 0x05a17cf4 \\
&c_{1,0}^{(2)} = 0x0707a3d4, \quad c_{1,4}^{(2)} = 0xbd09caca \\
&c_{1,0}^{(3)} = 0x1c1e8f51, \quad c_{1,4}^{(3)} = 0xf4272b28 \\
&c_{1,0}^{(4)} = 0x707a3d45, \quad c_{1,4}^{(4)} = 0x144ae5cc \\
&c_{1,0}^{(5)} = 0xaeb28562, \quad c_{1,4}^{(5)} = 0xfaa7ae2b \\
&c_{1,0}^{(6)} = 0xbaca1589, \quad c_{1,4}^{(6)} = 0x2e48f1c1 \\
&c_{1,0}^{(7)} = 0x40a46f3e, \quad c_{1,4}^{(7)} = 0xb923c704 \\
&c_{2,0}^{(0)} = 0xfc20d9d2, \quad c_{2,4}^{(0)} = 0xe0337818 \\
&c_{2,0}^{(1)} = 0x34552e25, \quad c_{2,4}^{(1)} = 0xe623bb72 \\
&c_{2,0}^{(2)} = 0x7ad8818f, \quad c_{2,4}^{(2)} = 0x5c58a4a4 \\
&c_{2,0}^{(3)} = 0x8438764a, \quad c_{2,4}^{(3)} = 0x1e38e2e7 \\
&c_{2,0}^{(4)} = 0xbb6de032, \quad c_{2,4}^{(4)} = 0x78e38b9d \\
&c_{2,0}^{(5)} = 0xedb780c8, \quad c_{2,4}^{(5)} = 0x27586719 \\
&c_{2,0}^{(6)} = 0xd9847356, \quad c_{2,4}^{(6)} = 0x36eda57f \\
&c_{2,0}^{(7)} = 0xa2c78434, \quad c_{2,4}^{(7)} = 0x703aace7
\end{align*} \]
B–3  $w = 4$

\[
\begin{align*}
  c_{3,0}^{(0)} &= 0xb213afa5, & c_{3,4}^{(0)} &= 0xe028c9bf \\
  c_{3,0}^{(1)} &= 0xc84ebe95, & c_{3,4}^{(1)} &= 0x44756f91 \\
  c_{3,0}^{(2)} &= 0x4e608a22, & c_{3,4}^{(2)} &= 0x7e8fcede \\
  c_{3,0}^{(3)} &= 0x56d858fe, & c_{3,4}^{(3)} &= 0x956548be \\
  c_{3,0}^{(4)} &= 0x343b138f, & c_{3,4}^{(4)} &= 0xfe191be2 \\
  c_{3,0}^{(5)} &= 0xd0ec4e3d, & c_{3,4}^{(5)} &= 0xc3c226e5 \\
  c_{3,0}^{(6)} &= 0x2ceb4882, & c_{3,4}^{(6)} &= 0x5944a28e \\
  c_{3,0}^{(7)} &= 0xb3ad2208, & c_{3,4}^{(7)} &= 0xa1c4355
\end{align*}
\]

B–4  $w = 5$

\[
\begin{align*}
  c_{4,0}^{(0)} &= 0xf0d2e9e3, & c_{4,4}^{(0)} &= 0x5090d577 \\
  c_{4,0}^{(1)} &= 0xac11d7fa, & c_{4,4}^{(1)} &= 0x2d1925ab \\
  c_{4,0}^{(2)} &= 0x1bcbb6f2, & c_{4,4}^{(2)} &= 0xb46496ac \\
  c_{4,0}^{(3)} &= 0x6f2d9bc9, & c_{4,4}^{(3)} &= 0xd1925ab0 \\
  c_{4,0}^{(4)} &= 0x78602649, & c_{4,4}^{(4)} &= 0x29131ab6 \\
  c_{4,0}^{(5)} &= 0x8edae952, & c_{4,4}^{(5)} &= 0xf0c053c3 \\
  c_{4,0}^{(6)} &= 0x3b6ba548, & c_{4,4}^{(6)} &= 0x3f014f0c \\
  c_{4,0}^{(7)} &= 0xedae9520, & c_{4,4}^{(7)} &= 0xfc053c31
\end{align*}
\]
C  Test Vectors

Let the message \( M \) be the 24 bits ASCII string “abc”. Then the resultant message digest of each algorithm is as follows.

C–1  \textit{Luffa}-224

The message digest of the message “abc” is

\[
\begin{align*}
Z_{0,0} &= \text{0xf1d566a4}, \quad Z_{0,1} = \text{0xb469a38e}, \\
Z_{0,2} &= \text{0xa31717db}, \quad Z_{0,3} = \text{0xb35d1bb9}, \\
Z_{0,4} &= \text{0xac184ec2}, \quad Z_{0,5} = \text{0xc08ee58c}, \\
Z_{0,6} &= \text{0x31bfcabc6}.
\end{align*}
\]

C–2  \textit{Luffa}-256

The message digest of the message “abc” is

\[
\begin{align*}
Z_{0,0} &= \text{0xf1d566a4}, \quad Z_{0,1} = \text{0xb469a38e}, \\
Z_{0,2} &= \text{0xa31717db}, \quad Z_{0,3} = \text{0xb35d1bb9}, \\
Z_{0,4} &= \text{0xac184ec2}, \quad Z_{0,5} = \text{0xc08ee58c}, \\
Z_{0,6} &= \text{0x31bfcabc6}, \quad Z_{0,7} = \text{0x41645526}.
\end{align*}
\]

C–3  \textit{Luffa}-384

The message digest of the message “abc” is

\[
\begin{align*}
Z_{0,0} &= \text{0xb13b97f6}, \quad Z_{0,1} = \text{0x739ad0d5}, \\
Z_{0,2} &= \text{0x75972c1c}, \quad Z_{0,3} = \text{0x81a242f7}, \\
Z_{0,4} &= \text{0x47ac1029}, \quad Z_{0,5} = \text{0xf19a87f3}, \\
Z_{0,6} &= \text{0x5e1ce165}, \quad Z_{0,7} = \text{0x68b4e730}, \\
Z_{1,0} &= \text{0x54a962fa}, \quad Z_{1,1} = \text{0xde288e43}, \\
Z_{1,2} &= \text{0x452395cf}, \quad Z_{1,3} = \text{0x05737ff9}.
\end{align*}
\]
C–4  **Luffa-512**

The message digest of the message “abc” is

\[
\begin{align*}
Z_{0,0} &= 0xc4c1faae4, \quad Z_{0,1} = 0xbda064ee, \\
Z_{0,2} &= 0xc9c50b695, \quad Z_{0,3} = 0xc2eb95c3e, \\
Z_{0,4} &= 0xc016c684, \quad Z_{0,5} = 0xb09e498c, \\
Z_{0,6} &= 0xc2514eb93, \quad Z_{0,7} = 0xc78377fe9, \\
Z_{1,0} &= 0xcfe2d61e, \quad Z_{1,1} = 0xc17bc3953, \\
Z_{1,2} &= 0xc6682d1c, \quad Z_{1,3} = 0xcbb8ce685, \\
Z_{1,4} &= 0xe5f4602c8, \quad Z_{1,5} = 0xc5b2ed11b, \\
Z_{1,6} &= 0xc5c3e453, \quad Z_{1,7} = 0xc314b1feb.
\end{align*}
\]

D  **Implementations of SubCrumb**

D–1  **For Intel® Core2 Processors**

The instructions are given by Table 4. At the first, the four words data

<table>
<thead>
<tr>
<th>cycle</th>
<th>MOV r4 r0</th>
<th>XOR r2 r1</th>
<th>AND r0 r1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>r0 r0</td>
<td>r2 r2</td>
<td>r1 r1</td>
</tr>
<tr>
<td>2</td>
<td>XOR r0 r2</td>
<td>NOT r1</td>
<td>OR r2 r4</td>
</tr>
<tr>
<td>3</td>
<td>XOR r2 r3</td>
<td>XOR r4 r0</td>
<td>AND r3 r0</td>
</tr>
<tr>
<td>4</td>
<td>XOR r3 r1</td>
<td>NOT r4</td>
<td>OR r1 r2</td>
</tr>
<tr>
<td>5</td>
<td>XOR r4 r1</td>
<td>XOR r0 r3</td>
<td>AND r1 r2</td>
</tr>
<tr>
<td>6</td>
<td>XOR r1 r3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(a_0, a_1, a_2, a_3\) are loaded to the registers \(r0, r1, r2, r3\) respectively. Then the resultant registers \(r0, r1, r3, r4\) provides the outputs of Sbox, namely, \(x_0 = r0, x_1 = r1, x_2 = r3, x_3 = r4\).
E Implementations of Message Injection Function \( MI \)

The message injection function \( MI \) defined in Section 3.2 can be implemented only with XORings and multiplications by a fixed constant 0x02.

**E–1 \( w = 3 \)**

The matrix representation can be transformed as follows:

\[
\begin{pmatrix}
3 & 2 & 2 & 1 \\
2 & 3 & 2 & 2 \\
2 & 2 & 3 & 4
\end{pmatrix} =
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0
\end{pmatrix} \oplus
\begin{pmatrix}
2 & 2 & 2 & 0 \\
2 & 2 & 2 & 0 \\
2 & 2 & 2 & 0
\end{pmatrix} \oplus
\begin{pmatrix}
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 2 \\
0 & 0 & 0 & 4
\end{pmatrix}.
\]

In other words, the message injection function \( MI \) for \( w = 3 \) can be also defined by the following equation:

\[
X_j = H_j^{(i-1)} \oplus \left( 0x02 \cdot \bigoplus_{j'}^{2} H_{j'}^{(i-1)} \right) \oplus 0x02^j \cdot M^{(i)}, \quad 0 \leq j < 3,
\]

Figure 8 shows an implementation image of \( MI \) for \( w = 3 \).
E–2  $w = 4$

The message injection function $MI$ for $w = 4$ can be also defined by the following equations for $0 \leq j < 4$:

$$\eta_j = H_j^{(i-1)} \oplus \left( 0x02 \cdot \bigoplus_{j'=0}^{3} H_{j'}^{(i-1)} \right),$$

$$X_j = 0x02 \cdot \eta_j \oplus \eta_{j-1} \mod 4 \oplus 0x02^j \cdot M^{(i)}.$$

Figure 9 shows an implementation image of $MI$ for $w = 4$.

Figure 9: The message injection function ($w = 4$)
The message injection function $MI$ for $w = 5$ can be also defined by the following equations for $0 \leq j < 5$:

$$
\eta_j = H^{(i-1)}_j \oplus \left( 0x02 \cdot \bigoplus_{j'=0}^{4} H^{(i-1)}_{j'} \right),
$$

$$
\xi_j = 0x02 \cdot \eta_j \oplus \eta_{j + 1 \mod 5},
$$

$$
X_j = 0x02 \cdot \xi_j \oplus \xi_{j - 1 \mod 5} \oplus 0x02^j \cdot M^{(i)}.
$$

Figure 10 shows an implementation image of $MI$ for $w = 5$.