Hash Function \textit{Luffa}

Specification

Christophe De Cannière
ESAT-COSIC, Katholieke Universiteit Leuven

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

31 October 2008
## Contents

1 Introduction 4

2 Preliminary 5
   2.1 Notations 5
      2.1.1 Parameters 5
      2.1.2 Symbols 6
   2.2 Data Structure 6
   2.3 Iterations 7

3 Chaining 8
   3.1 Message Padding 8
   3.2 Round Function 8
      3.2.1 Message Injection Function for $w = 3$ 10
      3.2.2 Message Injection Function for $w = 4$ 11
      3.2.3 Message Injection Function for $w = 5$ 11
   3.3 Finalization 11

4 Non-Linear Permutation 12
   4.1 Outline 12
   4.2 SubCrumb 14
   4.3 MixWord 14
   4.4 AddConstant 15
   4.5 Tweaks 17

5 Optional Usage 17

A Starting Variables 19

B Constants 19
   B-1 Initial Values 19
   B-2 $w = 3$ 20
   B-3 $w = 4$ 21
   B-4 $w = 5$ 21

C Test Vectors 22
   C-1 Luffa-224 22
   C-2 Luffa-256 22
   C-3 Luffa-384 22
   C-4 Luffa-512 23

D Implementations of SubCrumb 23
   D-1 For Intel® 686 Processors 23

Copyright ©2008 Hitachi, Ltd. All rights reserved.
<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>
Luffa Specification

1 Introduction

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

Table 1: Input and output lengths

<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>Luffa-224</td>
<td>$&lt; 2^{64}$</td>
<td>224</td>
<td>112</td>
</tr>
<tr>
<td>Luffa-256</td>
<td>$&lt; 2^{64}$</td>
<td>256</td>
<td>128</td>
</tr>
<tr>
<td>Luffa-384</td>
<td>$&lt; 2^{128}$</td>
<td>384</td>
<td>192</td>
</tr>
<tr>
<td>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 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 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.
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 $n_h w$ bits
- $Q_j$: The permutation dealing with $j$-th block of $n_h$ bits
- $OF$: The output function
- $b_{j,k,l}^{(i)}$: 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
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
- $|$: 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 $[\text{MSB}]$ (and $[\text{LSB}]$) 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 *round function*. The outline of the mixing function is defined in Section 3. A term *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 *step function*. A term *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 $8w$ words, where $w \geq 3$ is a positive integer (See Table 2 for the choice of $w$). An 8 words data is called a *block*. The $l$-th bit of the input of $i$-th round, $r$-th step, $j$-th block, $k$-th word is denoted by $a_{j,k,i-1,r}^{(i-1,r-1)}$.
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 $100\ldots0$ 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
Luffa Specification  NIST SHA3 Proposal

round be $\left( H_{0}^{(i-1)}, \ldots, H_{w-1}^{(i-1)} \right)$, then the output of the $i$-th round is given by

$$H^{(i)}_j = Q_j(X_j), \quad 0 \leq j < w,$$

$$X_0 || \cdots || X_{w-1} = MI(H_{0}^{(i-1)}, \ldots, H_{w-1}^{(i-1)}; M^{(i)}),$$

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 $\text{GF}(2^8)^{32}$. 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 $(\sum_{0\leq k<8} a_{k,l} x^k)_{0\leq l<32}$. 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:

```c
  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.

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

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.

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
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.

Copyright ©2008 Hitachi, Ltd. All rights reserved.
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, \( a_{j,k,d}^{(i,r)} \) 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
permutation.

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],[2],a[3]);
        SubCrumb(a[4],a[5],[6],a[7]);
        for (k = 0; k < 4; k++)
            MixColumn(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

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

Figure 5: The input and output bits of the Sbox

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:

\[
\begin{align*}
 y_{k+4} &= x_{k+4} \oplus x_k, \\
 y_k &= x_k \ll \sigma_1, \\
 y_k &= y_k \oplus y_{k+4}.
\end{align*}
\]

\(^1\)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
Figure 7: Constant generator

generates two 32-bit constants $c_{j,0}^{(r-1)}$ and $c_{j,4}^{(r-1)}$ in the following manner:

$$
\begin{align*}
\text{tl} || \text{tr} & = c_{j,L}^{(r-1)} || c_{j,R}^{(r-1)}, \\
\text{tl} || \text{tr} & = f_L(\text{tl} || \text{tr}), \\
c_{j,0}^{(r-1)} & = \text{tl}, \\
\text{tl} || \text{tr} & = f_L(\text{tr} || \text{tl}), \\
c_{j,4}^{(r-1)} & = \text{tl}, \\
c_{j,L}^{(r)} || c_{j,R}^{(r)} & = \text{tr} || \text{tl},
\end{align*}
$$

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.
tr = tr << 1;
if (c == 1){ tl ^= 0xc4d6496c; tr ^= 0x55c61c8d; }
SWAP(tl, tr);
step_const[j][r][k] = tr; /* k=0,4 */

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

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

5 Optional Usage

Despite 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


Luffa Specification


A Starting Variables

The values are taken from Appendix C.1.

\[ V_{0,0} = 0x6d251e69, V_{0,1} = 0x44b051e0, V_{0,2} = 0x4eaa6fb4, V_{0,3} = 0xdbf78465, \]
\[ V_{0,4} = 0x6e292011, V_{0,5} = 0x90152df4, V_{0,6} = 0xee058139, V_{0,7} = 0xdef610bb, \]
\[ V_{1,0} = 0xc3b44b95, V_{1,1} = 0xd9d2f256, V_{1,2} = 0x70eee9a0, V_{1,3} = 0xde099fa3, \]
\[ V_{1,4} = 0x5d9b0557, V_{1,5} = 0x8fc944b3, V_{1,6} = 0xcf1ccf0e, V_{1,7} = 0x746cd581, \]
\[ V_{2,0} = 0xf7efc89d, V_{2,1} = 0x5dba5781, V_{2,2} = 0x04016ce5, V_{2,3} = 0xad659c05, \]
\[ V_{2,4} = 0x306194f, V_{2,5} = 0x666d1836, V_{2,6} = 0x24aa230a, V_{2,7} = 0x8b264ae7, \]
\[ V_{3,0} = 0x858075d5, V_{3,1} = 0x36d79cce, V_{3,2} = 0xe57ff7d7, V_{3,3} = 0x204bf67, \]
\[ V_{3,4} = 0x35870c6a, V_{3,5} = 0x57e9e923, V_{3,6} = 0x14bcb808, V_{3,7} = 0x7cde72ce, \]
\[ V_{4,0} = 0x6c68e9be, V_{4,1} = 0x5ec41e22, V_{4,2} = 0xc825b7c7, V_{4,3} = 0xaffb4363, \]
\[ V_{4,4} = 0xf5df3999, V_{4,5} = 0x0fc688f1, V_{4,6} = 0xb07224cc, V_{4,7} = 0x03e86cea. \]

B Constants

B–1 Initial Values

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

\[ c_{0,L}^{(0)} = 0x181cca53, \quad c_{0,R}^{(0)} = 0x380cde06, \]
\[ c_{1,L}^{(0)} = 0x5b6f0876, \quad c_{1,R}^{(0)} = 0xf16f8594, \]
\[ c_{2,L}^{(0)} = 0x7e106ce9, \quad c_{2,R}^{(0)} = 0x38979cb0, \]
\[ c_{3,L}^{(0)} = 0xbb62f364, \quad c_{3,R}^{(0)} = 0x92e93c29, \]
\[ c_{4,L}^{(0)} = 0x9a025047, \quad c_{4,R}^{(0)} = 0xcff2a940. \]
B–2  \( w = 3 \)

\[
\begin{align*}
&c^{(0)}_{0,0} = 0x303994a6, \quad c^{(0)}_{0,4} = 0xe0337818 \\
&c^{(1)}_{0,0} = 0xc0e65299, \quad c^{(1)}_{0,4} = 0x441ba90d \\
&c^{(2)}_{0,0} = 0x6cc33a12, \quad c^{(2)}_{0,4} = 0x7f34d442 \\
&c^{(3)}_{0,0} = 0xdc56983e, \quad c^{(3)}_{0,4} = 0x9389217f \\
&c^{(4)}_{0,0} = 0x1e00108f, \quad c^{(4)}_{0,4} = 0xe5a8bce6 \\
&c^{(5)}_{0,0} = 0x7800423d, \quad c^{(5)}_{0,4} = 0x5274baf4 \\
&c^{(6)}_{0,0} = 0x8f5b7882, \quad c^{(6)}_{0,4} = 0x26889ba7 \\
&c^{(7)}_{0,0} = 0x96e1db12, \quad c^{(7)}_{0,4} = 0x9a226e9d \\
&c^{(0)}_{1,0} = 0xb6de10ed, \quad c^{(0)}_{1,4} = 0x01685f3d \\
&c^{(1)}_{1,0} = 0x70f47aae, \quad c^{(1)}_{1,4} = 0x05a17cf4 \\
&c^{(2)}_{1,0} = 0x0707a3d4, \quad c^{(2)}_{1,4} = 0xbd09caca \\
&c^{(3)}_{1,0} = 0x1c1e8f51, \quad c^{(3)}_{1,4} = 0xf4272b28 \\
&c^{(4)}_{1,0} = 0x707a3d45, \quad c^{(4)}_{1,4} = 0x144ae5cc \\
&c^{(5)}_{1,0} = 0xaeb28562, \quad c^{(5)}_{1,4} = 0xfaa7ae2b \\
&c^{(6)}_{1,0} = 0xbaca1589, \quad c^{(6)}_{1,4} = 0x2e48f1c1 \\
&c^{(7)}_{1,0} = 0x40a46f3e, \quad c^{(7)}_{1,4} = 0xb923c704 \\
&c^{(0)}_{2,0} = 0xfc20d9d2, \quad c^{(0)}_{2,4} = 0xe25e72c1 \\
&c^{(1)}_{2,0} = 0x34552e25, \quad c^{(1)}_{2,4} = 0xe623bb72 \\
&c^{(2)}_{2,0} = 0x7ad8818f, \quad c^{(2)}_{2,4} = 0x5c58a4a4 \\
&c^{(3)}_{2,0} = 0x8438764a, \quad c^{(3)}_{2,4} = 0x1e38e2e7 \\
&c^{(4)}_{2,0} = 0xbb6de032, \quad c^{(4)}_{2,4} = 0x78e38b9d \\
&c^{(5)}_{2,0} = 0xedb780c8, \quad c^{(5)}_{2,4} = 0x27586719 \\
&c^{(6)}_{2,0} = 0xd9847356, \quad c^{(6)}_{2,4} = 0x36eda57f \\
&c^{(7)}_{2,0} = 0xa2c78434, \quad c^{(7)}_{2,4} = 0x703aae7
\end{align*}
\]
B–3  \( w = 4 \)

\[
\begin{align*}
\ell_{3,0}^{(0)} &= 0xb213afa5, & \ell_{3,4}^{(0)} &= 0xe028c9bf \\
\ell_{3,0}^{(1)} &= 0xc84ebe95, & \ell_{3,4}^{(1)} &= 0x44756f91 \\
\ell_{3,0}^{(2)} &= 0x4e608a22, & \ell_{3,4}^{(2)} &= 0x7e8fcede \\
\ell_{3,0}^{(3)} &= 0x56d85f8e, & \ell_{3,4}^{(3)} &= 0x95654fe8 \\
\ell_{3,0}^{(4)} &= 0x343b138f, & \ell_{3,4}^{(4)} &= 0xfee19be2 \\
\ell_{3,0}^{(5)} &= 0xd0ec4e3d, & \ell_{3,4}^{(5)} &= 0x3cb22a45 \\
\ell_{3,0}^{(6)} &= 0x2ceb48a2, & \ell_{3,4}^{(6)} &= 0x5944a8be \\
\ell_{3,0}^{(7)} &= 0xb3ad2208, & \ell_{3,4}^{(7)} &= 0xa1c4c355
\end{align*}
\]

B–4  \( w = 5 \)

\[
\begin{align*}
\ell_{4,0}^{(0)} &= 0x3f0d2e9e3, & \ell_{4,4}^{(0)} &= 0x5090d577 \\
\ell_{4,0}^{(1)} &= 0xac1d7fa, & \ell_{4,4}^{(1)} &= 0x2d1925ab \\
\ell_{4,0}^{(2)} &= 0x1bcb66f2, & \ell_{4,4}^{(2)} &= 0x46496ac \\
\ell_{4,0}^{(3)} &= 0x6f2d9bc9, & \ell_{4,4}^{(3)} &= 0xd1925ab0 \\
\ell_{4,0}^{(4)} &= 0x78602649, & \ell_{4,4}^{(4)} &= 0x29131ab6 \\
\ell_{4,0}^{(5)} &= 0x8edae952, & \ell_{4,4}^{(5)} &= 0xf0c0f3c3 \\
\ell_{4,0}^{(6)} &= 0x3b6ba548, & \ell_{4,4}^{(6)} &= 0x3f01f0f0c \\
\ell_{4,0}^{(7)} &= 0xedae9520, & \ell_{4,4}^{(7)} &= 0xfc053c31
\end{align*}
\]
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  Luffa-224

The message digest of the message “abc” is

$$Z_{0,0} = 0xf1d566a4, \quad Z_{0,1} = 0xb469a38e,$$
$$Z_{0,2} = 0xa31717db, \quad Z_{0,3} = 0xb35d1bb9,$$
$$Z_{0,4} = 0xac184ec2, \quad Z_{0,5} = 0xc08ee58c,$$
$$Z_{0,6} = 0x31bfcbc6.$$

C–2  Luffa-256

The message digest of the message “abc” is

$$Z_{0,0} = 0xf1d566a4, \quad Z_{0,1} = 0xb469a38e,$$
$$Z_{0,2} = 0xa31717db, \quad Z_{0,3} = 0xb35d1bb9,$$
$$Z_{0,4} = 0xac184ec2, \quad Z_{0,5} = 0xc08ee58c,$$
$$Z_{0,6} = 0x31bfcbc6, \quad Z_{0,7} = 0x41645526.$$

C–3  Luffa-384

The message digest of the message “abc” is

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

The message digest of the message “abc” is

\[
\begin{align*}
Z_{0,0} &= 0x4c1faae4, & Z_{0,1} &= 0xbda064ee, \\
Z_{0,2} &= 0x9c50b695, & Z_{0,3} &= 0x2eb95c3e, \\
Z_{0,4} &= 0x1026c684, & Z_{0,5} &= 0x0b9e498c, \\
Z_{0,6} &= 0x2514eb93, & Z_{0,7} &= 0x78377fe9, \\
Z_{1,0} &= 0xef2d6d1e, & Z_{1,1} &= 0x17bc39f3, \\
Z_{1,2} &= 0x46982d1c, & Z_{1,3} &= 0xbb8ce685, \\
Z_{1,4} &= 0x5f4602c8, & Z_{1,5} &= 0xbf2ed11b, \\
Z_{1,6} &= 0xfcd3e453, & Z_{1,7} &= 0x314b1feb.
\end{align*}
\]

D  **Implementations of SubCrumb**

D–1  **For Intel® 686 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></td>
<td></td>
<td></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, \(x_0 = r0, x_1 = r1, x_2 = r3, x_3 = r4\).
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:

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

Figure 8 shows an implementation image of $MI$ for $w = 3$.

![Figure 8: The message injection function ($w = 3$)](image)
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 (\( w = 4 \))
E–3  \( w = 5 \)

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

\[
\eta_j = H_j^{(i-1)} \oplus \left( 0x02 \cdot \bigoplus_{j'=0}^{4} H_{j'}^{(i-1)} \right),
\]

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

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

Figure 10 shows an implementation image of \( MI \) for \( w = 5 \).

Figure 10: The message injection function \((w = 5)\)