AURORA-384/512 version 2: Revised Cryptographic Hash Algorithms in AURORA Family

Submitters:

Sony Corporation[1] and Nagoya University[2]

Algorithm Designers:

Tetsu Iwata[2], Kyoji Shibutani[1], Taizo Shirai[1], Shiho Moriai[1], Toru Akishita[1]

May 25, 2009

Contents

Chapter 1

Introduction

AURORA is a hash function family which is a first round candidate of SHA-3 competition [7]. AURORA consists of four hash algorithms including AURORA-384 and AURORA-512 which supports 384-bit and 512-bit hash values, respectively. Due to detailed analysis by external researchers after the submission to SHA-3 competition, unexpected security property of AURORA-384 and AURORA-512 was reported.[1] In this document, we describe the revised algorithms called AURORA-384 version 2 and AURORA-512 version 2, which are designed to get rid of the undesirable security problem of previous version by slightly modifying the specification. Although the updated versions are not formal inputs to the competition at this moment, we consider that providing the possibility of minor changes is useful for consideration of the ongoing competition. We also provide the preliminary implementation results, and confirmed that the updated version can be implemented efficiently on a variety of platforms.

Outline of Changes Before presenting the revised specification, the changes in AURORA-384/512 version 2 are briefly described in this paragraph. The main point for the changes of the specification of AURORA-384/512 is to replace the underlying double-block length (DBL) construction. Since AURORA-384 and AURORA-512 employ a common DBL scheme, hereafter we focus on explanations for the AURORA-512 case.

The domain extension scheme of AURORA-512 is basically the strengthened MD transform with the finalization function. Additionally a mixing function is inserted after hashing every 8 blocks of the message. The message block is 512 bits and the chaining value is also 512 bits. The compression functions are constructed according to the underlying Double-Mix Merkle-Damgård (DMMD) transform which is a sort of double block length (DBL) construction based on blockciphers.

Conceptually, the DMMD transform has two separate chaining values in parallel, and has the mixing function after every eight blocks. However some attacks which exploit the weakness of the DMMD transform [9, 8, 3, 10] have been reported so far, and the status of AURORA-512 graded wounded category that is defined by NIST. Therefore, we decided to change the DMMD transform to a different one for the updated version of the algorithm.

AURORA-512 version 2 uses another DBL scheme proposed by Hirose [5] in place of the DMMD. The Hirose's DBL construction uses two blockciphers to make a compression function, and the generated compression function can be used in a strengthened Merkle-Damgård (sMD). Similar to the DMMD, it is advantageous that the key scheduling part of underlying blockciphers can be shared. According to the replacement of the DBL scheme, specification of the related parts including the domain extension and component functions are adjusted to it.

Table 1.1 compares the original and the updated versions of AURORA-512. Each domain extension scheme is determined by the corresponding underlying DBL scheme. Both of the compression functions treat a 1024-bit input value and a 512-bit output value. The compression

[1] AURORA-224/256 are also included in the AURORA hash family, but there have not been reported any analytic result by external researchers. Thus we do not modify the specification of AURORA-224/256.

Table 1.1: Outline of AURORA-512 and AURORA-512 version 2

Algorithm	AURORA-512	AURORA-512 version 2
Domain extension	sMD + finalization + mixing	sMD + finalization
DBL construction	**DMMD**	**Hirose's DBL**
Compression Function	1024-bit input / 512-bit output	1024-bit input / 512-bit output
CF's components	2 *MSM* and 2 *CPM*	3 *MSM* and 2 CPM^{512}
MF's components	2 *MSM* and 2 *CPM*	*none*

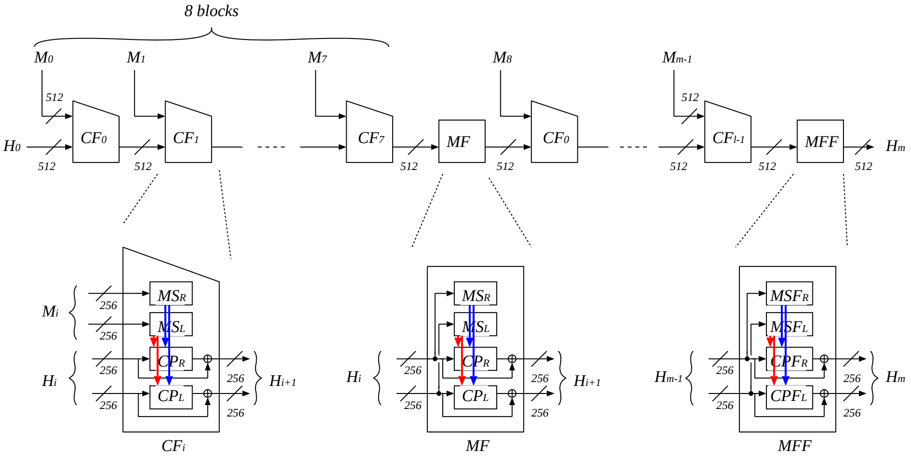


Figure 1.1: AURORA-512: Domain extension and compression function.

function of the previous version is constructed from two message scheduling modules (*MSM*s) and two chaining value processing module (*CPM*s). On the other hand, the compression function of the new version is constructed from three *MSM*s and two CPM^{512}s. One *MSM* is added, because Hirose's DBL construction requires a half of the chaining value to be input to the message scheduling part. The number of rounds of CPM^{512} is increased from *CPM* to absorb data from the additional *MSM*. Moreover, the new version does not need the mixing functions (*MF*) any more.

Fig 1.1 and 1.2 show the domain extension and the compression functions in each version of AURORA-512, and they illustrate the outline of the changes explained above. Detailed description of the specification are described later in this document.

Impact of Changes The above changes have a influence on two aspects of the hash function: security and performance. As for the security, the new version resolves the security concern of the previous version with regard to the 2nd preimage attacks and the collision attacks, because all reported analyses are only applicable for the DMMD transform based hash algorithms. As for the performance, the impact on efficiency loss is limited. In software implementation on the NIST reference platform (64-bit), AURORA-512 version 2 achieves 37.8 cycles/byte. Also, AURORA shows good performance across a variety of platforms, because it uses platform-independent operations. Also in hardware implementation, AURORA enables a variety of implementations, from high-speed to area-restricted implementations. Using a 0.13μm CMOS ASIC library, AURORA-512 version 2 can be implemented with only 12.4 Kgates in an area-optimized implementation. In a speed-optimized implementation, AURORA-512 version 2 achieves the highest throughput of 6.9 Gbps.

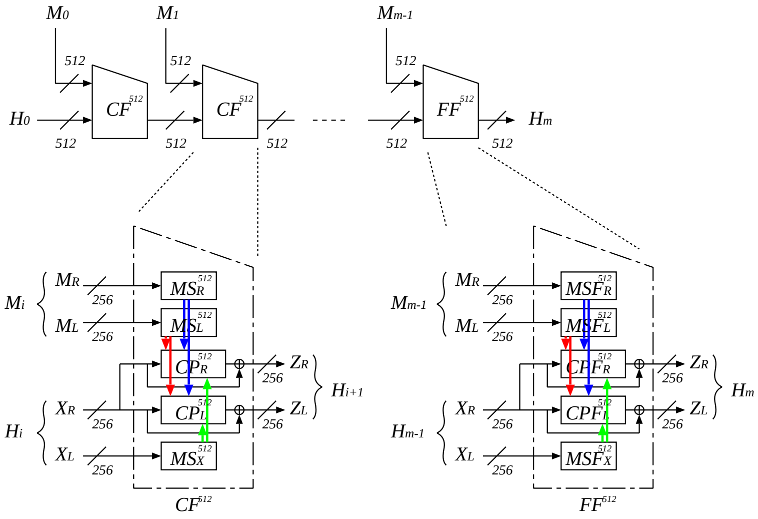


Figure 1.2: AURORA-512 v2: Domain extension and compression function.

Organization of the document. This document is organized as follows: Chapter 2 describes the specification of the AURORA-384/512 version 2. Chapter 3 provides the design rationale and the changes in AURORA-384/512 version 2. Chapter 4 shows preliminary efficient implementation results of AURORA-384/512 version 2.

Chapter 2

Specification of AURORA-384/512 version 2

2.1 Notation

We first describe notation, conventions and symbols used throughout this document.

- We use the prefix `0x` to denote hexadecimal numbers.
- A bit string x with the suffix, $x_{(n)}$, indicates that x is n bits. This suffix is omitted if there is no ambiguity.
- For bit strings x and y, $x \,\|\, y$ or (x, y) is their concatenation.
- For bit strings x and y, $x \leftarrow y$ means that the bit string x is updated by the bit string y. For an nl-bit x, we write $(x_{0\,(n)}, x_{1\,(n)}, \ldots, x_{l-1\,(n)}) \leftarrow x_{(nl)}$ to mean that x is divided into $(x_0, x_1, \ldots, x_l)$, where $(x_{0\,(n)} \,\|\, x_{1\,(n)} \,\|\, \cdots \,\|\, x_{l-1\,(n)}) = x_{(nl)}$.
- For a bit string $x_{(n)}$ and an integer l, $x \lll_n l$ is the l-bit left cyclic shift of x, and $x \ggg_n l$ is the l-bit right cyclic shift of x.
- For bit strings $x_0, x_1, \ldots, x_{n-1}$, $\{x_j\}_{0 \le j < n}$ is a shorthand for $(x_0, x_1, \ldots, x_{n-1})$.
- For an integer l, 0^l is the l times repetition of zero bits and 1^l is the l times repetition of one bits.
- For a bit string x, $\overline{x}$ is the bit-wise complement of x.
- For an element of GF(2^n) represented as a polynomial $x_{n-1}\alpha^{n-1} + x_{n-2}\alpha^{n-2} + \ldots + x_1\alpha + x_0$ where α is a root of an irreducible polynomial, $x_{n-1}||x_{n-2}||\ldots||x_1||x_0$ denotes the bit representation of the polynomial.

Following variables and symbols are commonly used in AURORA family.

M	The input message.
M_i	The i-th block of the message (after the padding).
m	The length of M in blocks (after the padding).
H_i	The i-th chaining value.
MSM	The Message Scheduling Module.
BD	The Byte Diffusion function.
$PROTL$	The Partial ROTating Left function.
$PROTR$	The Partial ROTating Right function.
Pad	The Padding function.
Len_n	The Length of the input message in blocks encoded into n bits.
TF_n	The Truncation Function that outputs n bits.
F_0, F_1, F_2, and F_3	The F-Functions.
$\mathcal{M}_0$, $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$	The matrices used in the F-functions.
S	The S-box.

Following symbols are used for AURORA-384/512 version 2.

CPM^{512}	The Chaining value Processing Module.
DR^{512}	The Data Rotating function.
$PROTX$	The eXtra Partial ROTating function.
CF^{512}	The Compression Functions for AURORA-384/512 version 2.
FF^{512}	The Finalization Function for AURORA-384/512 version 2.
MS_L^{512}, MS_R^{512} and MS_X^{512}	The Message Scheduling functions for CF^{512}.
MSF_L^{512}, MSF_R^{512} and MSF_X^{512}	The Message Scheduling functions for Finalization.
CP_L^{512} and CP_R^{512}	The Chaining value Processing function for CF^{512}.
CPF_L^{512} and CPF_R^{512}	The Chaining value Processing function for Finalization.
$CONM_{L,j}^{512}$, $CONM_{R,j}^{512}$ and $CONM_{X,j}^{512}$	The CONstants for MS_L^{512}, MS_R^{512}, MS_X^{512}, MSF_L^{512}, MSF_R^{512} and MSF_X^{512}.
$CONC_{L,j}^{512}$ and $CONC_{R,j}^{512}$	The CONstant for CP_L^{512}, CP_R^{512}, CPF_L^{512} and CPF_R^{512}.

2.2 Building Blocks

In this section, specifications of the essential building blocks for constructing AURORA-384/512 version 2 algorithms are described.

2.2.1 Message Scheduling Module: *MSM*

The message scheduling module, *MSM*, takes the following two inputs;

- a bit string $X_{(256)}$, and
- a set of bit strings $\{Y_{j\,(32)}\}_{0\le j<32}$.

The output is a set of bit strings $\{Z_{j\,(32)}\}_{0\le j<72}$.

MSM internally uses a byte diffusion function $BD : (\{0,1\}^{32})^8 \to (\{0,1\}^{32})^8$, which is a permutation over $(\{0,1\}^{32})^8$ and is defined in Sec. 2.2.3. *MSM* is parameterized by two functions F and F', where

$$\begin{cases} F : \{0,1\}^{32} \to \{0,1\}^{32}, \\ F' : \{0,1\}^{32} \to \{0,1\}^{32}. \end{cases} \tag{2.1}$$

We write $MSM[F,F']$ when we emphasize that it is parameterized by functions F and F'. We now describe the specification of $MSM[F,F']$.

Step 1. Let $(X_{0\,(32)}, X_{1\,(32)}, \ldots, X_{7\,(32)}) \leftarrow X_{(256)}$.

Step 2. Let $(X_1, X_3, X_5, X_7) \leftarrow (X_1, X_3, X_5, X_7) \oplus (Y_0, Y_1, Y_2, Y_3)$.

Step 3. Let $(Z_0, Z_1, \ldots, Z_7) \leftarrow (X_0, X_1, \ldots, X_7)$.

Step 4. (7 round iterations) The following operations are iterated for $i = 1$ to 7.

$$\begin{cases} (X_0, X_1, \ldots, X_7) \leftarrow BD(X_0, X_1, \ldots, X_7) \\ (X_0, X_2, X_4, X_6) \leftarrow (F(X_0), F'(X_2), F(X_4), F'(X_6)) \\ (X_1, X_3, X_5, X_7) \leftarrow (X_1, X_3, X_5, X_7) \oplus (Y_{4i}, Y_{4i+1}, Y_{4i+2}, Y_{4i+3}) \\ (X_1, X_3, X_5, X_7) \leftarrow (X_1, X_3, X_5, X_7) \oplus (X_0, X_2, X_4, X_6) \\ (Z_{8i}, Z_{8i+1}, \ldots, Z_{8i+7}) \leftarrow (X_0, X_1, \ldots, X_7) \end{cases}$$

Step 5. (8-th round) Then the following operations are executed.

$$\begin{cases} (X_0, X_1, \ldots, X_7) \leftarrow BD(X_0, X_1, \ldots, X_7) \\ (X_0, X_2, X_4, X_6) \leftarrow (F(X_0), F'(X_2), F(X_4), F'(X_6)) \\ (X_1, X_3, X_5, X_7) \leftarrow (X_1, X_3, X_5, X_7) \oplus (X_0, X_2, X_4, X_6) \\ (Z_{64}, Z_{65}, \ldots, Z_{71}) \leftarrow (X_0, X_1, \ldots, X_7) \end{cases}$$

Step 6. Finally, the output is $\{Z_{j\,(32)}\}_{0\le j<72}$.

See Fig. 2.1 for an illustration and Fig. 2.8 for a pseudocode.

2.2.2 Chaining Value Processing Module: *CPM*^{512}

The chaining value processing module, CPM^{512}, takes the following three inputs;

- a bit string $X_{(256)}$,
- a set of bit strings $\{Y_{j\,(32)}\}_{0\le j<216}$, and
- a set of bit strings $\{W_{j\,(32)}\}_{0\le j<104}$.

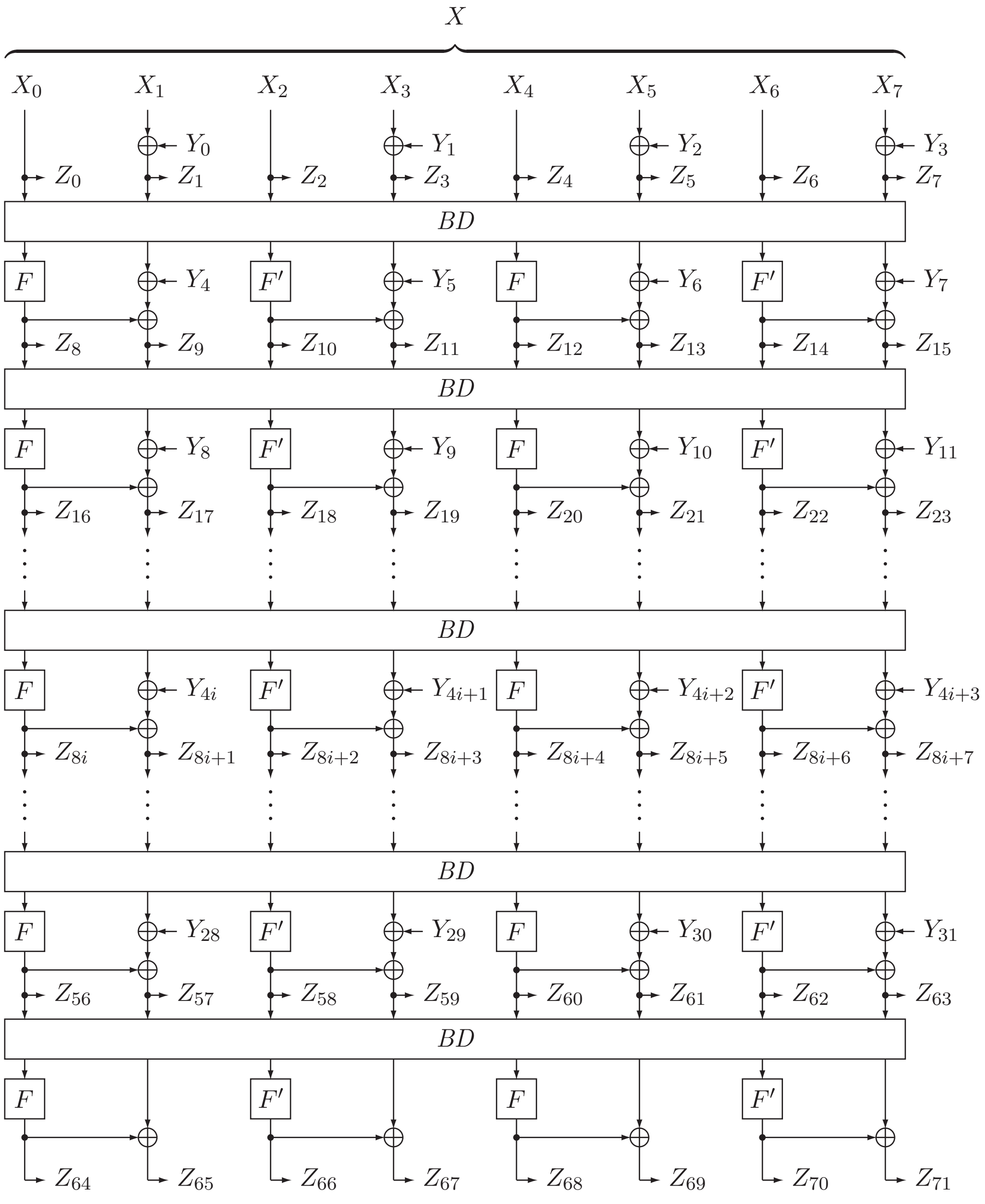


Figure 2.1: $\{Z_{j\,(32)}\}_{0\le j<72} \leftarrow MSM[F,F'](X_{(256)}, \{Y_{j\,(32)}\}_{0\le j<32})$.

The output is a bit string $Z_{(256)}$.

CPM^{512} internally uses a byte diffusion function BD, which is also used in MSM, and is defined in Sec. 2.2.3. As with MSM, CPM^{512} is parameterized by two functions F and F' over $\{0,1\}^{32}$, and we write $\mathit{CPM}^{512}[F,F']$ when we use functions F and F'.

We now describe the specification of $\mathit{CPM}^{512}[F,F']$.

Step 1. Let $(X_{0\,(32)}, X_{1\,(32)}, \ldots, X_{7\,(32)}) \leftarrow X_{(256)}$.

Step 2. Let $(X_1, X_3, X_5, X_7) \leftarrow (X_1, X_3, X_5, X_7) \oplus (W_0, W_1, W_2, W_3)$.

Step 3. Let $(X_0, X_1, \ldots, X_7) \leftarrow (X_0, X_1, \ldots, X_7) \oplus (Y_0, Y_1, \ldots, Y_7)$.

Step 4. (25 round iterations) The following operations are iterated for $i = 1$ to 25.

$$\left\{\begin{array}{l}(X_0, X_1, \ldots, X_7) \leftarrow \mathit{BD}(X_0, X_1, \ldots, X_7)\\ (X_0, X_2, X_4, X_6) \leftarrow (F(X_0), F'(X_2), F(X_4), F'(X_6))\\ (X_1, X_3, X_5, X_7) \leftarrow (X_1, X_3, X_5, X_7) \oplus (W_{4i}, W_{4i+1}, W_{4i+2}, W_{4i+3})\\ (X_1, X_3, X_5, X_7) \leftarrow (X_1, X_3, X_5, X_7) \oplus (X_0, X_2, X_4, X_6)\\ (X_0, X_1, \ldots, X_7) \leftarrow (X_0, X_1, \ldots, X_7) \oplus (Y_{8i}, Y_{8i+1}, \ldots, Y_{8i+7})\end{array}\right.$$

Step 5. (26-th round) Then the following operations are executed.

$$\left\{\begin{array}{l}(X_0, X_1, \ldots, X_7) \leftarrow \mathit{BD}(X_0, X_1, \ldots, X_7)\\ (X_0, X_2, X_4, X_6) \leftarrow (F(X_0), F'(X_2), F(X_4), F'(X_6))\\ (X_1, X_3, X_5, X_7) \leftarrow (X_1, X_3, X_5, X_7) \oplus (X_0, X_2, X_4, X_6)\\ (X_0, X_1, \ldots, X_7) \leftarrow (X_0, X_1, \ldots, X_7) \oplus (Y_{208}, Y_{209}, \ldots, Y_{215})\end{array}\right.$$

Step 6. Finally, the output is $Z_{(256)} \leftarrow (X_{0\,(32)} \,\|\, X_{1\,(32)} \,\|\, \cdots \,\|\, X_{7\,(32)})$.

See Fig. 2.2 for an illustration and Fig. 2.9 for a pseudocode.

2.2.3 Byte Diffusion Function: *BD*

The byte diffusion function, BD, takes a bit string $(X_{0\,(32)}, X_{1\,(32)}, \ldots, X_{7\,(32)})$ as the input, and outputs the updated bit string $(X_{0\,(32)}, X_{1\,(32)}, \ldots, X_{7\,(32)})$.

It works as follows.

Step 1. For $i = 0, 1, \ldots, 7$, $X_{i\,(32)}$ is divided into a 4-byte sequence as

$$(x_{4i\,(8)}, x_{4i+1\,(8)}, x_{4i+2\,(8)}, x_{4i+3\,(8)}) \leftarrow X_{i\,(32)},$$

and $(X_{0\,(32)}, X_{1\,(32)}, \ldots, X_{7\,(32)})$ is now regarded as a sequence of bytes;

$$(x_{0\,(8)}, x_{1\,(8)}, \ldots, x_{31\,(8)}) = (X_{0\,(32)}, X_{1\,(32)}, \ldots, X_{7\,(32)}).$$

Step 2. Next we permute $(x_0, x_1, \ldots, x_{31})$ according to the permutation π defined in Fig. 2.3, where the i-th byte x_i is moved to the $\pi(i)$-th byte. In other words, let $x'_{\pi(i)} = x_i$ for $i = 0, 1, \ldots, 31$. Then $(x'_0, x'_1, \ldots, x'_{31})$ is the result of the permutation. For example, $x'_0 = x_4$, $x'_1 = x_{29}$, and so on.

Step 3. For $i = 0, 1, \ldots, 7$, the 4-byte sequence $(x'_{4i\,(8)}, x'_{4i+1\,(8)}, x'_{4i+2\,(8)}, x'_{4i+3\,(8)})$ is concatenated to form the updated $X_{i\,(32)} = (x'_{4i\,(8)} \,\|\, x'_{4i+1\,(8)} \,\|\, x'_{4i+2\,(8)} \,\|\, x'_{4i+3\,(8)})$, and the output is $(X_{0\,(32)}, X_{1\,(32)}, \ldots, X_{7\,(32)})$.

See Fig. 2.4 for an illustration and Fig. 2.10 for a pseudocode.

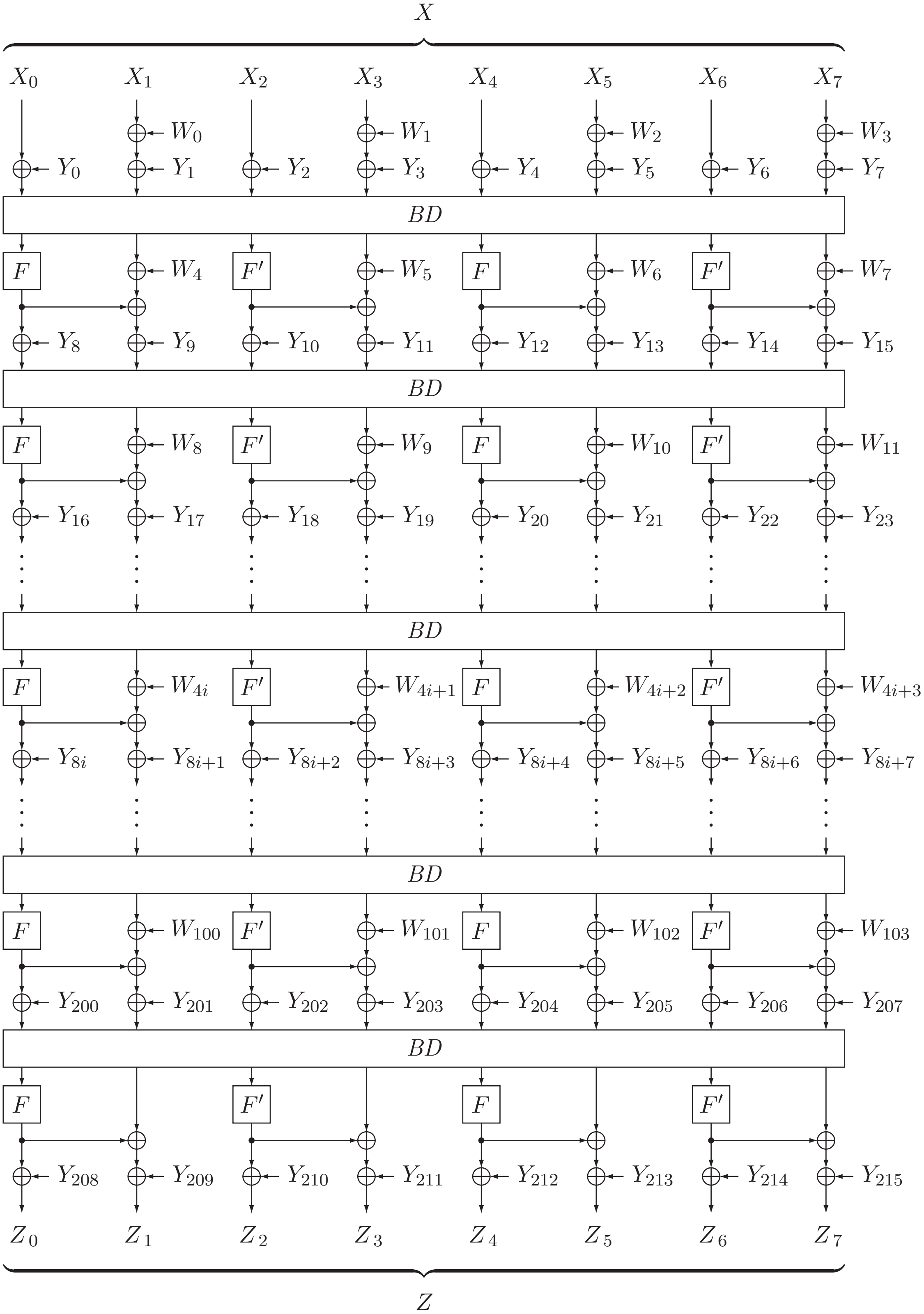


Figure 2.2: $Z_{(256)} \leftarrow CPM^{512}[F, F'](X_{(256)}, \{Y_{j\,(32)}\}_{0 \le j < 216}, \{W_{j\,(32)}\}_{0 \le j < 104})$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(i)$	28	29	30	31	0	9	18	27	4	5	6	7	8	17	26	3

i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$\pi(i)$	12	13	14	15	16	25	2	11	20	21	22	23	24	1	10	19

Figure 2.3: Definition of the permutation $\pi(\cdot) : \{0, 1, \ldots, 31\} \rightarrow \{0, 1, \ldots, 31\}$.

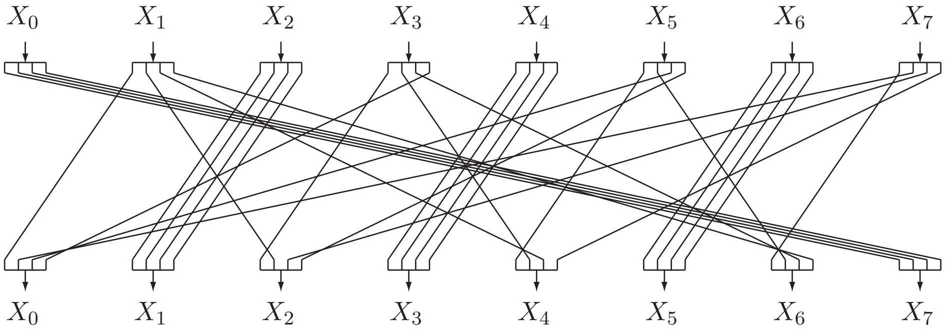


Figure 2.4: $(X_{0\,(32)}, X_{1\,(32)}, \ldots, X_{7\,(32)}) \leftarrow BD(X_{0\,(32)}, X_{1\,(32)}, \ldots, X_{7\,(32)})$.

2.2.4 F-Functions: F_0, F_1, F_2, and F_3

We use four F-functions, F_0, F_1, F_2, and F_3, where they take 32-bit input X as input and produce 32-bit output Y. Each function is used as an instantiation of a parameter functions F or F' in *MSM* and *CPM*.

Before defining these F-functions, we first define the S-box $S : \{0, 1\}^8 \rightarrow \{0, 1\}^8$, and four 4×4 matrices, $\mathcal{M}_0$, $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$.

- The S-box $S : x_{(8)} \rightarrow y_{(8)}$ is defined as follows.

$$y = \begin{cases} g(f(x)^{-1}) & \text{if } f(x) \neq 0 \\ g(0) & \text{if } f(x) = 0 \end{cases} .$$

The inverse function is performed in $\mathrm{GF}((2^4)^2)$ defined by an irreducible polynomial $z^2 + z + \{1001\}$ for which the underlying $\mathrm{GF}(2^4)$ is defined by an irreducible polynomial $z'^4 + z' + 1$. Moreover, $f : x_{(8)} \rightarrow y_{(8)}$ and $g : x_{(8)} \rightarrow y_{(8)}$ are affine transformations over GF(2), which are defined as

$$f : \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} , \tag{2.2}$$

Table 2.1: S

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.a	.b	.c	.d	.e	.f
0.	d9	dc	d3	69	bd	00	4d	eb	02	24	57	c2	b8	5d	b7	6d
1.	f5	40	37	4e	19	d8	64	62	9d	34	0f	7c	ec	ce	94	04
2.	d1	8a	74	fb	e7	87	12	23	b5	5c	1a	bb	42	49	18	85
3.	11	46	0d	71	67	8f	c6	50	58	fd	4b	a4	cd	8e	99	1f
4.	ad	63	c9	6b	f7	28	9f	65	2f	5f	61	73	3d	8b	0e	1b
5.	33	e0	ac	26	a1	e3	f3	82	83	75	44	90	13	af	f0	07
6.	96	21	f8	3f	a2	98	9a	a3	91	4c	7f	92	97	ea	01	1c
7.	1e	2d	89	39	e6	9c	0a	54	0c	51	6c	43	ae	db	53	59
8.	a6	f4	06	da	e2	78	1d	29	30	e1	35	fc	ed	bc	47	d5
9.	c0	ab	cc	a8	80	2b	09	b0	93	d4	c5	b3	d0	df	a9	aa
a.	7a	36	2a	d6	b2	fa	e8	b1	a0	68	5a	81	48	08	17	c7
b.	fe	76	bf	c4	f2	3e	4a	0b	10	14	f1	ef	a7	27	e5	c8
c.	de	9b	8d	3c	56	d7	8c	60	6a	79	ee	a5	31	2e	77	41
d.	ff	95	dd	25	3b	55	ca	52	9e	2c	15	4f	e4	16	70	7d
e.	72	3a	7b	84	f6	32	86	03	b4	38	6f	b9	c1	45	88	e9
f.	ba	b6	6e	5e	be	7e	20	f9	22	66	05	d2	cb	c3	cf	5b

and

$$g: \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \tag{2.3}$$

where $(x_{0(1)}||x_{1(1)}||x_{2(1)}||x_{3(1)}||x_{4(1)}||x_{5(1)}||x_{6(1)}||x_{7(1)}) \leftarrow x_{(8)}$ and $(y_{0(1)}||y_{1(1)}||y_{2(1)}||y_{3(1)}$ $||y_{4(1)}||y_{5(1)}||y_{6(1)}||y_{7(1)}) \leftarrow y_{(8)}$. Table 2.1 shows the output values of S.

- The four matrices are defined as follows.

$$\mathcal{M}_0 = \begin{pmatrix} \mathtt{0x01} & \mathtt{0x02} & \mathtt{0x02} & \mathtt{0x03} \\ \mathtt{0x03} & \mathtt{0x01} & \mathtt{0x02} & \mathtt{0x02} \\ \mathtt{0x02} & \mathtt{0x03} & \mathtt{0x01} & \mathtt{0x02} \\ \mathtt{0x02} & \mathtt{0x02} & \mathtt{0x03} & \mathtt{0x01} \end{pmatrix}, \tag{2.4}$$

$$\mathcal{M}_1 = \begin{pmatrix} \mathtt{0x01} & \mathtt{0x06} & \mathtt{0x08} & \mathtt{0x02} \\ \mathtt{0x02} & \mathtt{0x01} & \mathtt{0x06} & \mathtt{0x08} \\ \mathtt{0x08} & \mathtt{0x02} & \mathtt{0x01} & \mathtt{0x06} \\ \mathtt{0x06} & \mathtt{0x08} & \mathtt{0x02} & \mathtt{0x01} \end{pmatrix}, \tag{2.5}$$

$$\mathcal{M}_2 = \begin{pmatrix} \mathtt{0x03} & \mathtt{0x01} & \mathtt{0x02} & \mathtt{0x02} \\ \mathtt{0x02} & \mathtt{0x03} & \mathtt{0x01} & \mathtt{0x02} \\ \mathtt{0x02} & \mathtt{0x02} & \mathtt{0x03} & \mathtt{0x01} \\ \mathtt{0x01} & \mathtt{0x02} & \mathtt{0x02} & \mathtt{0x03} \end{pmatrix}, \tag{2.6}$$

$$\mathcal{M}_3 = \begin{pmatrix} \mathtt{0x06} & \mathtt{0x08} & \mathtt{0x02} & \mathtt{0x01} \\ \mathtt{0x01} & \mathtt{0x06} & \mathtt{0x08} & \mathtt{0x02} \\ \mathtt{0x02} & \mathtt{0x01} & \mathtt{0x06} & \mathtt{0x08} \\ \mathtt{0x08} & \mathtt{0x02} & \mathtt{0x01} & \mathtt{0x06} \end{pmatrix}. \tag{2.7}$$

Multiplications are operated over GF(2^8) defined by an irreducible polynomial $z^8+z^4+z^3+z^2+1$.

Now we describe F-functions.

Step 1. Let $(x_{0\,(8)}, x_{1\,(8)}, x_{2\,(8)}, x_{3\,(8)}) \leftarrow X_{(32)}$.

Step 2. Let $(x_0, x_1, x_2, x_3) \leftarrow (S(x_0), S(x_1), S(x_2), S(x_3))$.

Step 3. For $i \in \{0,1,2,3\}$, the output of F_i is $Y_{(32)} = (y_{0\,(8)} \,\|\, y_{1\,(8)} \,\|\, y_{2\,(8)} \,\|\, y_{3\,(8)})$, where

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \mathcal{M}_i \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

2.2.5 Data Rotating Function: DR^{512}

The data rotating function, DR^{512}, takes the following three inputs;

- a set of bit strings $\{X_{j\,(32)}\}_{0\le j<72}$,
- a set of bit strings $\{Y_{j\,(32)}\}_{0\le j<72}$, and
- a set of bit strings $\{W_{j\,(32)}\}_{0\le j<72}$.

The output is a set of bit strings $\{Z_{j\,(32)}\}_{0\le j<216}$.

DR^{512} uses the following three functions;

$$\begin{cases} PROTL : (\{0,1\}^{32})^8 \to (\{0,1\}^{32})^8, \\ PROTR : (\{0,1\}^{32})^8 \to (\{0,1\}^{32})^8, \\ PROTX : (\{0,1\}^{32})^8 \to (\{0,1\}^{32})^8. \end{cases}$$

which we define as

$$PROTL(X_{0\,(32)}, X_{1\,(32)}, \ldots, X_{7\,(32)}) = (X'_{0\,(32)}, X'_{1\,(32)}, \ldots, X'_{7\,(32)}), \tag{2.8}$$

where $X'_i = X_i$ for $i = 0,2,4,5,6,7$, and $(X'_1 \,\|\, X'_3) = (X_1 \,\|\, X_3) \lll_{64} 1$.

Similarly, we define

$$PROTR(X_{0\,(32)}, X_{1\,(32)}, \ldots, X_{7\,(32)}) = (X'_{0\,(32)}, X'_{1\,(32)}, \ldots, X'_{7\,(32)}), \tag{2.9}$$

where $X'_i = X_i$ for $i = 0,2,4,5,6,7$, and $(X'_1 \,\|\, X'_3) = (X_1 \,\|\, X_3) \ggg_{64} 1$.

Finally, we define

$$PROTX(X_{0\,(32)}, X_{1\,(32)}, \ldots, X_{7\,(32)}) = (X'_{0\,(32)}, X'_{1\,(32)}, \ldots, X'_{7\,(32)}), \tag{2.10}$$

where $X'_i = X_i$ for $i = 0,2,4,5,6,7$, and $(X'_1 \,\|\, X'_3) = (X_1 \,\|\, X_3) \lll_{64} 2$.

In other words, they rotate the two words by one bit or two bits, where these words are concatenated and regarded as one 64 bit string.

Now DR^{512} works as follows.

Step 1. For inputs $\{X_{j\,(32)}\}_{0\le j<72}$, $\{Y_{j\,(32)}\}_{0\le j<72}$ and $\{W_{j\,(32)}\}_{0\le j<72}$, we define $\{Z_{j\,(32)}\}_{0\le j<216}$ by iterating the following operations for $i = 0$ to 8.

$$\begin{cases} (Z_{24i}, Z_{24i+1}, \ldots, Z_{24i+7}) \leftarrow PROTL(X_{8i}, X_{8i+1}, \ldots, X_{8i+7}) \\ (Z_{24i+8}, Z_{24i+9}, \ldots, Z_{24i+15}) \leftarrow PROTR(Y_{8i}, Y_{8i+1}, \ldots, Y_{8i+7}) \\ (Z_{24i+16}, Z_{24i+17}, \ldots, Z_{24i+23}) \leftarrow PROTX(W_{8i}, W_{8i+1}, \ldots, W_{8i+7}) \end{cases}$$

Step 2. The output is $\{Z_{j\,(32)}\}_{0\le j<216}$ defined in the above operations.

See Fig. 2.5 for an illustration and Fig. 2.11 for a pseudocode.

$$
\begin{array}{ccccc|ccr|cccccc}
\cline{6-8}
X_0 & X_1 & \cdots & X_7 & \rightarrow & (Z_1 \,\|\, Z_3) & \leftarrow & (X_1 \,\|\, X_3) \lll_{64} 1 & \rightarrow & Z_0 & Z_1 & \cdots & Z_7 \\
Y_0 & Y_1 & \cdots & Y_7 & \rightarrow & (Z_9 \,\|\, Z_{11}) & \leftarrow & (Y_1 \,\|\, Y_3) \ggg_{64} 1 & \rightarrow & Z_8 & Z_9 & \cdots & Z_{15} \\
W_0 & W_1 & \cdots & W_7 & \rightarrow & (Z_{17} \,\|\, Z_{19}) & \leftarrow & (W_1 \,\|\, W_3) \lll_{64} 2 & \rightarrow & Z_{16} & Z_{17} & \cdots & Z_{23} \\
X_8 & X_9 & \cdots & X_{15} & \rightarrow & (Z_{25} \,\|\, Z_{27}) & \leftarrow & (X_9 \,\|\, X_{11}) \lll_{64} 1 & \rightarrow & Z_{24} & Z_{25} & \cdots & Z_{31} \\
Y_8 & Y_9 & \cdots & Y_{15} & \rightarrow & (Z_{33} \,\|\, Z_{35}) & \leftarrow & (Y_9 \,\|\, Y_{11}) \ggg_{64} 1 & \rightarrow & Z_{32} & Z_{33} & \cdots & Z_{39} \\
W_8 & W_9 & \cdots & W_{15} & \rightarrow & (Z_{41} \,\|\, Z_{43}) & \leftarrow & (W_9 \,\|\, W_{11}) \lll_{64} 2 & \rightarrow & Z_{40} & Z_{41} & \cdots & Z_{47} \\
\vdots & \vdots & & \vdots & & & \vdots & & & \vdots & \vdots & & \vdots \\
X_{64} & X_{65} & \cdots & X_{71} & \rightarrow & (Z_{193} \,\|\, Z_{195}) & \leftarrow & (X_{65} \,\|\, X_{67}) \lll_{64} 1 & \rightarrow & Z_{192} & Z_{193} & \cdots & Z_{199} \\
Y_{64} & Y_{65} & \cdots & Y_{71} & \rightarrow & (Z_{201} \,\|\, Z_{203}) & \leftarrow & (Y_{65} \,\|\, Y_{67}) \ggg_{64} 1 & \rightarrow & Z_{200} & Z_{201} & \cdots & Z_{207} \\
W_{64} & W_{65} & \cdots & W_{71} & \rightarrow & (Z_{209} \,\|\, Z_{211}) & \leftarrow & (W_{65} \,\|\, W_{67}) \lll_{64} 2 & \rightarrow & Z_{208} & Z_{209} & \cdots & Z_{215} \\
\cline{6-8}
\end{array}
$$

Figure 2.5: $\{Z_{j\,(32)}\}_{0\le j<216} \leftarrow DR^{512}(\{X_{j\,(32)}\}_{0\le j<72}, \{Y_{j\,(32)}\}_{0\le j<72}, \{W_{j\,(32)}\}_{0\le j<72})$.

2.3 Specification of AURORA-512 version 2

2.3.1 Overall Structure

AURORA-512 version 2 takes the input message of length at most $512 \times (2^{64} - 1) = 2^{73} - 512$ bits, and outputs the hash value of 512 bits. It internally uses a compression function CF^{512} and a finalization function FF^{512}, where

$$\begin{cases} CF^{512}(\cdot,\cdot) : \{0,1\}^{512} \times \{0,1\}^{512} \rightarrow \{0,1\}^{512}, \\ FF^{512}(\cdot,\cdot) : \{0,1\}^{512} \times \{0,1\}^{512} \rightarrow \{0,1\}^{512}. \end{cases}$$

The compression function CF^{512} is defined in Sec. 2.3.2 and a finalization function FF^{512} is defined in Sec. 2.3.3.

Now AURORA-512 version 2 works as follows.

Step 1. The input message M is padded with the following padding function $Pad(\cdot)$;

$$Pad(M) = M \parallel 1 \parallel 0^b \parallel Len_{64}, \tag{2.11}$$

where b is the minimum non-negative integer (possibly zero) such that $|M| + b + 65 = 512m$ for some integer m, and Len_{64} is an encoding of $\lceil |M|/512 \rceil$ in 64-bit string. That is, Len_{64} is the length of M in blocks, where a partial block counts for one block, and b is the minimal integer such that the total length of $Pad(M)$ is a multiple of 512 bits. Then $Pad(M)$ is divided into blocks $M_0, M_1, \ldots, M_{m-1}$ each of length 512 bits, i.e., we let

$$(M_{0\,(512)}, M_{1\,(512)}, \ldots, M_{m-1\,(512)}) \leftarrow Pad(M).$$

Step 2. Let $H_{0\,(512)} = 0^{512}$, and compute $H_{1\,(512)}, H_{2\,(512)}, \ldots, H_{m-1\,(512)}$ by iterating

$$H_{i+1} \leftarrow CF^{512}(H_i, M_i)$$

for $i = 0$ to $m - 2$.

Note that when $Pad(M)$ has one block (i.e., when $m = 1$ and $Pad(M) = M_0$), then Step 2 is not executed.

Step 3. Finally, let $H_m \leftarrow FF^{512}(H_{m-1}, M_{m-1})$, and the output is $H_{m\,(512)}$.

See Fig. 2.6 for an illustration and Fig. 2.12 for a pseudocode.

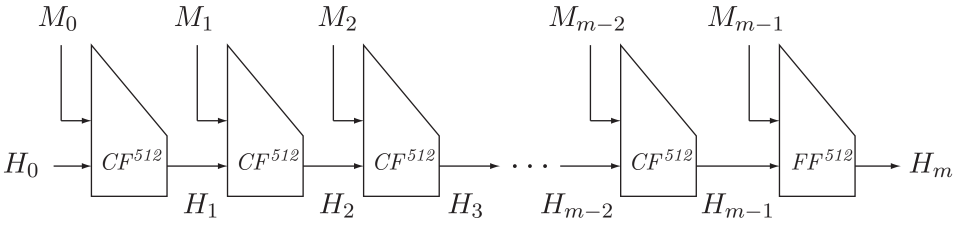


Figure 2.6: AURORA-512 version 2.

2.3.2 Compression Functions: CF^{512}

The compression function, CF^{512}, takes the chaining value H_i of 512 bits and the input message block M_i of 512 bits, and outputs the chaining value H_{i+1} of 512 bits.

It internally uses three message scheduling functions MS_L^{512}, $MS_R^{512}$and MS_X^{512}, a data rotating function DR^{512}, and a chaining value processing functions CP_L^{512} and CP_R^{512}, where

$$\begin{cases} MS_L^{512}(\cdot) : \{0,1\}^{256} \to (\{0,1\}^{32})^{72}, \\ MS_R^{512}(\cdot) : \{0,1\}^{256} \to (\{0,1\}^{32})^{72}, \\ MS_X^{512}(\cdot) : \{0,1\}^{256} \to (\{0,1\}^{32})^{72}, \\ DR^{512}(\cdot,\cdot) : (\{0,1\}^{32})^{72} \times (\{0,1\}^{32})^{72} \times (\{0,1\}^{32})^{72} \to (\{0,1\}^{32})^{216}, \\ CP_L^{512}(\cdot,\cdot) : \{0,1\}^{256} \times (\{0,1\}^{32})^{216} \to \{0,1\}^{256}, \\ CP_R^{512}(\cdot,\cdot) : \{0,1\}^{256} \times (\{0,1\}^{32})^{216} \to \{0,1\}^{256}. \end{cases}$$

These functions are described below.

Components of CF^{512}

- MS_L^{512} is an instance of MSM described in Sec. 2.2.1, and for any $X \in \{0,1\}^{256}$, it is defined as
$$MS_L^{512}(X) = MSM[F_0, F_1](X, \{CONM_{L,j}^{512}\}_{0 \le j < 32}), \tag{2.12}$$
where F_0 and F_1 are F-functions defined in Sec. 2.2.4, and $\{CONM_{L,j}^{512}\}_{0 \le j < 32}$ is the set of constants defined in Sec. 2.5.

- Similarly, for any $X \in \{0,1\}^{256}$, MS_R^{512} is defined as
$$MS_R^{512}(X) = MSM[F_2, F_3](X, \{CONM_{R,j}^{512}\}_{0 \le j < 32}), \tag{2.13}$$
where F_2 and F_3 are F-functions defined in Sec. 2.2.4, and $\{CONM_{R,j}^{512}\}_{0 \le j < 32}$ is the set of constants defined in Sec. 2.5.

- Also, for any $X \in \{0,1\}^{256}$, MS_X^{512} is defined as
$$MS_X^{512}(X) = MSM[F_0, F_3](X, \{CONM_{X,j}^{512}\}_{0 \le j < 32}), \tag{2.14}$$
where F_0 and F_3 are F-functions defined in Sec. 2.2.4, and $\{CONM_{X,j}^{512}\}_{0 \le j < 32}$ is the set of constants defined in Sec. 2.5.

- DR^{512} is the data rotating function defined in Sec. 2.2.5.

- CP_L^{512} is an instance of CPM^{512} described in Sec. 2.2.2, and for any $X \in \{0,1\}^{256}$ and $Y \in (\{0,1\}^{32})^{216}$, it is defined as
$$CP_L^{512}(X,Y) = CPM^{512}[F_1, F_0](X, Y, \{CONC_{L,j}^{512}\}_{0 \le j < 104}), \tag{2.15}$$
where F_0 and F_1 are F-functions defined in Sec. 2.2.4, and $\{CONC_{L,j}^{512}\}_{0 \le j < 104}$ is the set of constants defined in Sec. 2.5.

- CP_R^{512} is an instance of CPM^{512} described in Sec. 2.2.2, and for any $X \in \{0,1\}^{256}$ and $Y \in (\{0,1\}^{32})^{216}$, it is defined as
$$CP_R^{512}(X,Y) = CPM^{512}[F_3, F_2](X, Y, \{CONC_{R,j}^{512}\}_{0 \le j < 104}), \tag{2.16}$$
where F_2 and F_3 are F-functions defined in Sec. 2.2.4, and $\{CONC_{R,j}^{512}\}_{0 \le j < 104}$ is the set of constants defined in Sec. 2.5.

Specification of CF^{512}

Now we present the specification of CF^{512}.

Step 1. Let $(M_{L\,(256)}, M_{R\,(256)}) \leftarrow M_{i\,(512)}$, and let $(X_{L(256)}, X_{R(256)}) \leftarrow H_{i\,(512)}$.

Step 2. Let $\{T_{L,j\,(32)}\}_{0 \le j < 72} \leftarrow MS_L^{512}(M_{L\,(256)})$.

Step 3. Let $\{T_{R,j\,(32)}\}_{0\le j<72} \leftarrow MS_R^{512}(M_{R\,(256)})$.

Step 4. Let $\{T_{X,j\,(32)}\}_{0\le j<72} \leftarrow MS_X^{512}(X_{L\,(256)})$.

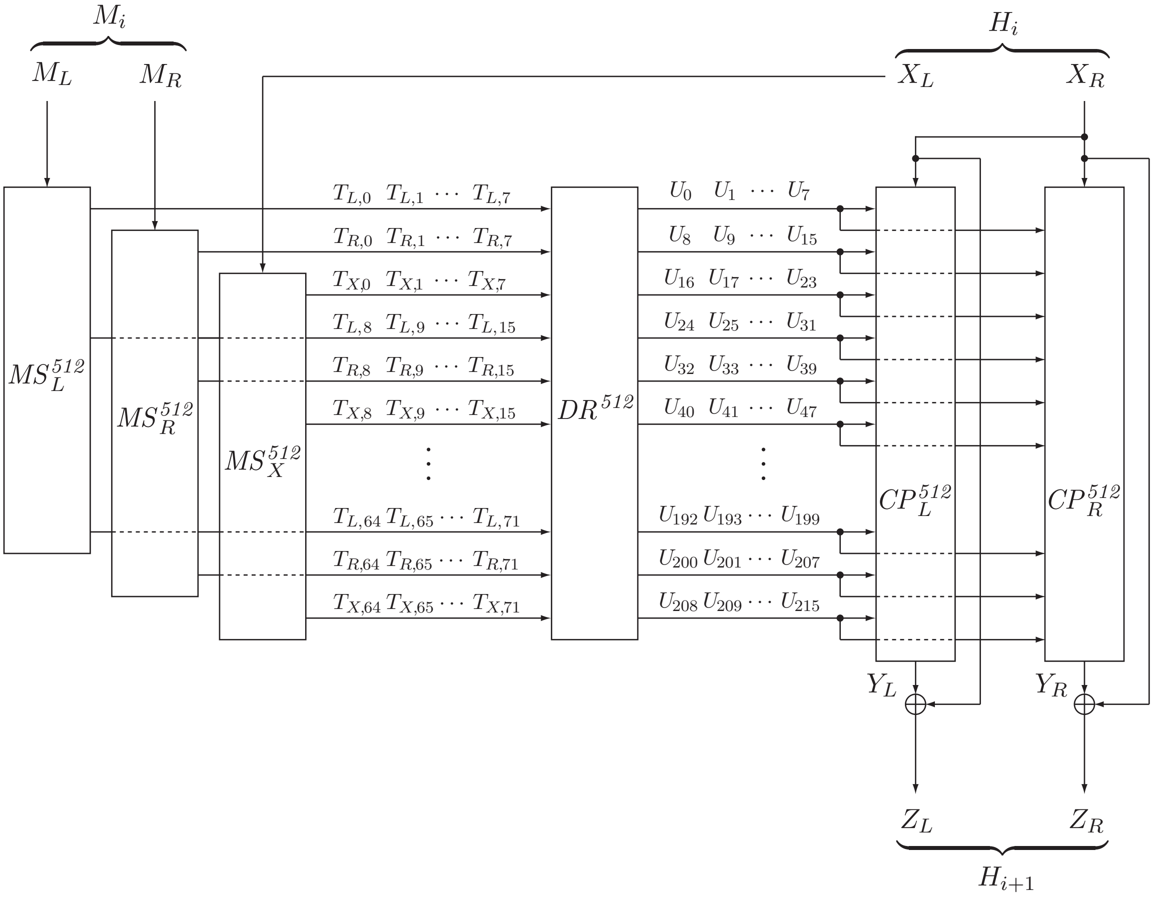
Step 5. Let $\{U_{j\,(32)}\}_{0\le j<216} \leftarrow DR^{512}(\{T_{L,j\,(32)}\}_{0\le j<72}, \{T_{R,j\,(32)}\}_{0\le j<72}, \{T_{X,j\,(32)}\}_{0\le j<72})$.

Step 6. Let $Y_{L(256)} \leftarrow CP_L^{512}(X_{R(256)}, \{U_{j\,(32)}\}_{0\le j<216})$.

Step 7. Let $Y_{R(256)} \leftarrow CP_R^{512}(X_{R(256)}, \{U_{j\,(32)}\}_{0\le j<216})$.

Step 8. Finally, the output is $H_{i+1\,(512)} \leftarrow (Y_{L(256)} \oplus X_{R(256)} || Y_{R(256)} \oplus X_{R(256)})$.

See Fig. 2.7 for an illustration and Fig. 2.13 for a pseudocode.



M_i
M_L
M_R
H_i
X_L
X_R
MS_L^{512}
MS_R^{512}
MS_X^{512}
$T_{L,0}\ T_{L,1} \cdots T_{L,7}$
$T_{R,0}\ T_{R,1} \cdots T_{R,7}$
$T_{X,0}\ T_{X,1} \cdots T_{X,7}$
$T_{L,8}\ T_{L,9} \cdots T_{L,15}$
$T_{R,8}\ T_{R,9} \cdots T_{R,15}$
$T_{X,8}\ T_{X,9} \cdots T_{X,15}$
$T_{L,64}\ T_{L,65} \cdots T_{L,71}$
$T_{R,64}\ T_{R,65} \cdots T_{R,71}$
$T_{X,64}\ T_{X,65} \cdots T_{X,71}$
DR^{512}
$U_0\ U_1 \cdots U_7$
$U_8\ U_9 \cdots U_{15}$
$U_{16}\ U_{17} \cdots U_{23}$
$U_{24}\ U_{25} \cdots U_{31}$
$U_{32}\ U_{33} \cdots U_{39}$
$U_{40}\ U_{41} \cdots U_{47}$
$U_{192}\ U_{193} \cdots U_{199}$
$U_{200}\ U_{201} \cdots U_{207}$
$U_{208}\ U_{209} \cdots U_{215}$
CP_L^{512}
CP_R^{512}
Y_L
Y_R
Z_L
Z_R
H_{i+1}

Figure 2.7: $H_{i+1\,(512)} \leftarrow CF^{512}(H_{i\,(512)}, M_{i\,(512)})$.

2.3.3 Finalization Functions: FF^{512}

The finalization function, FF^{512}, takes the chaining value H_{m-1} of 512 bits and the input message block M_{m-1} of 512 bits, and outputs the final hash value H_m of 512 bits.

FF^{512} is structurally equivalent to CF^{512}, and the only difference is the constants used in the components.

It internally uses three message scheduling functions MSF_L^{512}, $MSF_R^{512}$and MSF_X^{512}, a data rotating function DR^{512}, and a chaining value processing functions CPF_L^{512} and CPF_R^{512}, where

$$\begin{cases} MSF_L^{512}(\cdot) : \{0,1\}^{256} \to (\{0,1\}^{32})^{72}, \\ MSF_R^{512}(\cdot) : \{0,1\}^{256} \to (\{0,1\}^{32})^{72}, \\ MSF_X^{512}(\cdot) : \{0,1\}^{256} \to (\{0,1\}^{32})^{72}, \\ DR^{512}(\cdot,\cdot) : (\{0,1\}^{32})^{72} \times (\{0,1\}^{32})^{72} \times (\{0,1\}^{32})^{72} \to (\{0,1\}^{32})^{216}, \\ CPF_L^{512}(\cdot,\cdot) : \{0,1\}^{256} \times (\{0,1\}^{32})^{216} \to \{0,1\}^{256}, \\ CPF_R^{512}(\cdot,\cdot) : \{0,1\}^{256} \times (\{0,1\}^{32})^{216} \to \{0,1\}^{256}. \end{cases}$$

These functions are described below.

Components of FF^{512}

- MSF_L^{512} is an instance of MSM described in Sec. 2.2.1, and for any $X \in \{0,1\}^{256}$, it is defined as
$$MSF_L^{512}(X) = MSM[F_0, F_1](X, \{CONM_{L,j}^{512}\}_{32 \le j < 64}), \quad (2.17)$$
where F_0 and F_1 are F-functions defined in Sec. 2.2.4, and $\{CONM_{L,j}^{512}\}_{32 \le j < 64}$ is the set of constants defined in Sec. 2.5.

- Similarly, for any $X \in \{0,1\}^{256}$, MS_R^{512} is defined as
$$MSF_R^{512}(X) = MSM[F_2, F_3](X, \{CONM_{R,j}^{512}\}_{32 \le j < 64}), \quad (2.18)$$
where F_2 and F_3 are F-functions defined in Sec. 2.2.4, and $\{CONM_{R,j}^{512}\}_{32 \le j < 64}$ is the set of constants defined in Sec. 2.5.

- Also, for any $X \in \{0,1\}^{256}$, MS_X^{512} is defined as
$$MSF_X^{512}(X) = MSM[F_0, F_3](X, \{CONM_{X,j}^{512}\}_{32 \le j < 64}), \quad (2.19)$$
where F_0 and F_3 are F-functions defined in Sec. 2.2.4, and $\{CONM_{X,j}^{512}\}_{32 \le j < 64}$ is the set of constants defined in Sec. 2.5.

- DR^{512} is the data rotating function defined in Sec. 2.2.5.

- CPF_L^{512} is an instance of CPM^{512} described in Sec. 2.2.2, and for any $X \in \{0,1\}^{256}$ and $Y \in (\{0,1\}^{32})^{216}$, it is defined as
$$CPF_L^{512}(X, Y) = CPM^{512}[F_1, F_0](X, Y, \{CONC_{L,j}^{512}\}_{104 \le j < 208}), \quad (2.20)$$
where F_0 and F_1 are F-functions defined in Sec. 2.2.4, and $\{CONC_{L,j}^{512}\}_{104 \le j < 208}$ is the set of constants defined in Sec. 2.5.

- CPF_R^{512} is an instance of CPM^{512} described in Sec. 2.2.2, and for any $X \in \{0,1\}^{256}$ and $Y \in (\{0,1\}^{32})^{216}$, it is defined as
$$CPF_R^{512}(X, Y) = CPM^{512}[F_3, F_2](X, Y, \{CONC_{R,j}^{512}\}_{104 \le j < 208}), \quad (2.21)$$
where F_2 and F_3 are F-functions defined in Sec. 2.2.4, and $\{CONC_{R,j}^{512}\}_{104 \le j < 208}$ is the set of constants defined in Sec. 2.5.

Specification of FF^{512}

Now we present the specification of FF^{512}.

Step 1. Let $(M_{L\,(256)}, M_{R\,(256)}) \leftarrow M_{m-1\,(512)}$, and let $(X_{L(256)}, X_{R(256)}) \leftarrow H_{m-1\,(512)}$.

Step 2. Let $\{T_{L,j\,(32)}\}_{0 \le j < 72} \leftarrow MSF_L^{512}(M_{L\,(256)})$.

Step 3. Let $\{T_{R,j\,(32)}\}_{0\leq j<72} \leftarrow MSF_R^{512}(M_{R\,(256)})$.

Step 4. Let $\{T_{X,j\,(32)}\}_{0\leq j<72} \leftarrow MSF_X^{512}(X_{L\,(256)})$.

Step 5. Let $\{U_{j\,(32)}\}_{0\leq j<216} \leftarrow DR^{512}(\{T_{L,j\,(32)}\}_{0\leq j<72}, \{T_{R,j\,(32)}\}_{0\leq j<72}, \{T_{X,j\,(32)}\}_{0\leq j<72})$.

Step 6. Let $Y_{L(256)} \leftarrow CPF_L^{512}(X_{R(256)}, \{U_{j\,(32)}\}_{0\leq j<216})$.

Step 7. Let $Y_{R(256)} \leftarrow CPF_R^{512}(X_{R(256)}, \{U_{j\,(32)}\}_{0\leq j<216})$.

Step 8. Finally, the output is $H_{m\,(512)} \leftarrow (Y_{L(256)} \oplus X_{R(256)} || Y_{R(256)} \oplus X_{R(256)})$.

See Fig. 2.14 for a pseudocode.

2.4 Specification of AURORA-384 version 2

AURORA-384 version 2 takes the input message of length at most $512 \times (2^{64} - 1) = 2^{73} - 512$ bits, and outputs the hash value of 384 bits. It uses the same padding function Pad, the compression functions CF^{512}, the finalization function FF^{512} as AURORA-512 defined in Sec. 2.3.

The difference is that AURORA-384 uses $H_0 = 1^{512}$ as the initial value, and the output of FF^{512} is truncated to 384 bits by the truncation function TF_{384}.

The truncation function, $\mathit{TF}_{384}(\cdot) : \{0,1\}^{512} \to \{0,1\}^{384}$, first parses the input $H_{m\,(512)}$ into a sequence of bytes $H_{m\,(512)} = (m_{0\,(8)}, m_{1\,(8)}, \dots, m_{63\,(8)})$ and drops the following bytes;

$$m_6, m_7, m_{14}, m_{15}, m_{22}, m_{23}, m_{30}, m_{31}, m_{38}, m_{39}, m_{46}, m_{47}, m_{54}, m_{55}, m_{62}, m_{63},$$

to produce the 384-bit hash value $H'_{m\,(384)} = (m'_{0\,(8)}, m'_{1\,(8)}, \dots, m'_{47\,(8)})$.

That is, for the 512-bit input $H_{m\,(512)} = (m_{0\,(8)}, m_{1\,(8)}, \dots, m_{63\,(8)})$, the 384-bit output is $H'_{m\,(384)} = (m'_{0\,(8)}, m'_{1\,(8)}, \dots, m'_{47\,(8)})$, where

$$\begin{cases} m'_i = m_i & \text{for } 0 \le i \le 5 \\ m'_i = m_{i+2} & \text{for } 6 \le i \le 11 \\ m'_i = m_{i+4} & \text{for } 12 \le i \le 17 \\ m'_i = m_{i+6} & \text{for } 18 \le i \le 23 \\ m'_i = m_{i+8} & \text{for } 24 \le i \le 29 \\ m'_i = m_{i+10} & \text{for } 30 \le i \le 35 \\ m'_i = m_{i+12} & \text{for } 36 \le i \le 41 \\ m'_i = m_{i+14} & \text{for } 42 \le i \le 47 \end{cases}$$

Now we describe the specification of AURORA-384.

Step 1. The input message M is first padded with $\mathit{Pad}(\cdot)$ in (2.11), and the result of $\mathit{Pad}(M)$ is divided into blocks $M_0, M_1, \dots, M_{m-1}$ each of length 512 bits, i.e., let

$$(M_{0\,(512)}, M_{1\,(512)}, \dots, M_{m-1\,(512)}) \leftarrow \mathit{Pad}(M).$$

Step 2. Let $H_{0\,(512)} = 1^{512}$, and compute $H_{1\,(512)}, H_{2\,(512)}, \dots, H_{m\,(512)}$ by iterating

$$H_{i+1} \leftarrow \mathit{CF}^{512}(H_i, M_i)$$

for $i = 0$ to $m - 2$. Note that when $\mathit{Pad}(M)$ has one block (i.e., when $m = 1$ and $\mathit{Pad}(M) = M_0$), then Step 2 is not executed.

Step 3. Let $H_m \leftarrow \mathit{FF}^{512}(H_{m-1}, M_{m-1})$, and the output is $H'_{m\,(384)} \leftarrow \mathit{TF}_{384}(H_{m\,(512)})$.

See Fig. 2.15 for a pseudocode.

2.5 Constant Values

This section describes the generation procedures and the lists of constant values.

2.5.1 Constant Values for AURORA-384/512 version 2

Following constants are used in AURORA-384/512 version 2;

- $\{CONM^{\mathit{512}}_{L,j}\}_{0\leq j<32}$, $\{CONM^{\mathit{512}}_{R,j}\}_{0\leq j<32}$, $\{CONM^{\mathit{512}}_{X,j}\}_{0\leq j<32}$, $\{CONC^{\mathit{512}}_{L,j}\}_{0\leq j<104}$, $\{CONC^{\mathit{512}}_{R,j}\}_{0\leq j<104}$ for CF^{512}, and
- $\{CONM^{\mathit{512}}_{L,j}\}_{32\leq j<64}$, $\{CONM^{\mathit{512}}_{R,j}\}_{32\leq j<64}$, $\{CONM^{\mathit{512}}_{X,j}\}_{32\leq j<64}$, $\{CONC^{\mathit{512}}_{L,j}\}_{104\leq j<208}$, $\{CONC^{\mathit{512}}_{R,j}\}_{104\leq j<208}$ for FF^{512}.

These constants are generated with the procedure described below.

Step 1. Let IV_0^{512}, IV_1^{512}, $mask_0^{512}$, $mask_1^{512}$, $mask_2^{512}$ and $mask_3^{512}$ be the following values.

$$\begin{cases} IV_0^{512} \leftarrow (11^{1/2}-3)2^{16} = \texttt{0x510e} \\ IV_1^{512} \leftarrow (13^{1/2}-3)2^{16} = \texttt{0x9b05} \\ mask_0^{512} \leftarrow (11^{1/3}-2)2^{16} = \texttt{0x3956} \\ mask_1^{512} \leftarrow (13^{1/3}-2)2^{16} = \texttt{0x59f1} \\ mask_2^{512} \leftarrow (11^{1/5}-1)2^{16} = \texttt{0x9d8a} \\ mask_3^{512} \leftarrow (13^{1/5}-1)2^{16} = \texttt{0xab97} \end{cases}$$

Step 2. The following operations are iterated for $i = 0$ to 25.

$$\begin{cases} T_{0,i}^{512} \leftarrow IV_0^{512} \cdot \texttt{0x0002}^{i} \\ T_{1,i}^{512} \leftarrow IV_1^{512} \cdot \texttt{0x0002}^{-i} \\ CONC_{L,4i}^{512} \leftarrow (T_{0,i}^{512} \oplus mask_0^{512} \,\|\, \overline{T_{0,i}^{512}} \lll_{16} 8) \\ CONC_{L,4i+1}^{512} \leftarrow (T_{1,i}^{512} \oplus mask_1^{512} \,\|\, \overline{T_{1,i}^{512}} \lll_{16} 8) \\ CONC_{L,4i+2}^{512} \leftarrow (T_{0,i}^{512} \lll_{16} 8 \,\|\, T_{0,i}^{512} \oplus mask_2^{512}) \\ CONC_{L,4i+3}^{512} \leftarrow (T_{1,i}^{512} \lll_{16} 9 \,\|\, T_{1,i}^{512} \oplus mask_3^{512}) \end{cases}$$

Step 3. The following operation is iterated for $i = 0$ to 103.

$$CONC_{R,i}^{512} \leftarrow CONC_{L,i}^{512} \lll_{32} 3$$

Step 4. The following operations are iterated for $i = 0$ to 7.

$$\begin{cases} CONM_{L,4i}^{512} \leftarrow CONC_{L,12i}^{512} \lll_{32} 1 \\ CONM_{L,4i+1}^{512} \leftarrow CONC_{L,12i+1}^{512} \lll_{32} 1 \\ CONM_{L,4i+2}^{512} \leftarrow CONC_{L,12i+2}^{512} \lll_{32} 1 \\ CONM_{L,4i+3}^{512} \leftarrow CONC_{L,12i+3}^{512} \lll_{32} 1 \\ CONM_{R,4i}^{512} \leftarrow CONC_{L,12i+4}^{512} \ggg_{32} 1 \\ CONM_{R,4i+1}^{512} \leftarrow CONC_{L,12i+5}^{512} \ggg_{32} 1 \\ CONM_{R,4i+2}^{512} \leftarrow CONC_{L,12i+6}^{512} \ggg_{32} 1 \\ CONM_{R,4i+3}^{512} \leftarrow CONC_{L,12i+7}^{512} \ggg_{32} 1 \\ CONM_{X,4i}^{512} \leftarrow CONC_{L,12i+8}^{512} \lll_{32} 2 \\ CONM_{X,4i+1}^{512} \leftarrow CONC_{L,12i+9}^{512} \lll_{32} 2 \\ CONM_{X,4i+2}^{512} \leftarrow CONC_{L,12i+10}^{512} \lll_{32} 2 \\ CONM_{X,4i+3}^{512} \leftarrow CONC_{L,12i+11}^{512} \lll_{32} 2 \end{cases}$$

Step 5. The following operations are iterated for $i = 0$ to 25.

$$\left\{\begin{array}{l}
CONC^{512}_{L,4i+104} \leftarrow CONC^{512}_{L,4i} \\
CONC^{512}_{L,4i+105} \leftarrow CONC^{512}_{L,4i+1} \\
CONC^{512}_{L,4i+106} \leftarrow CONC^{512}_{L,4i+2} \\
CONC^{512}_{L,4i+107} \leftarrow CONC^{512}_{L,4i+3} \oplus \texttt{0x01010101} \\
CONC^{512}_{R,4i+104} \leftarrow CONC^{512}_{R,4i} \\
CONC^{512}_{R,4i+105} \leftarrow CONC^{512}_{R,4i+1} \\
CONC^{512}_{R,4i+106} \leftarrow CONC^{512}_{R,4i+2} \\
CONC^{512}_{R,4i+107} \leftarrow CONC^{512}_{R,4i+3} \oplus \texttt{0x01010101}
\end{array}\right.$$

Step 6. The following operations are iterated for $i = 0$ to 7.

$$\left\{\begin{array}{l}
CONM^{512}_{L,4i+32} \leftarrow CONM^{512}_{L,4i} \\
CONM^{512}_{L,4i+33} \leftarrow CONM^{512}_{L,4i+1} \\
CONM^{512}_{L,4i+34} \leftarrow CONM^{512}_{L,4i+2} \\
CONM^{512}_{L,4i+35} \leftarrow CONM^{512}_{L,4i+3} \\
CONM^{512}_{R,4i+32} \leftarrow CONM^{512}_{R,4i} \\
CONM^{512}_{R,4i+33} \leftarrow CONM^{512}_{R,4i+1} \\
CONM^{512}_{R,4i+34} \leftarrow CONM^{512}_{R,4i+2} \\
CONM^{512}_{R,4i+35} \leftarrow CONM^{512}_{R,4i+3} \\
CONM^{512}_{X,4i+32} \leftarrow CONM^{512}_{X,4i} \\
CONM^{512}_{X,4i+33} \leftarrow CONM^{512}_{X,4i+1} \\
CONM^{512}_{X,4i+34} \leftarrow CONM^{512}_{X,4i+2} \\
CONM^{512}_{X,4i+35} \leftarrow CONM^{512}_{X,4i+3}
\end{array}\right.$$

2.6 List of Constant Values

The following tables offer the list of the constant values for reference. These described values are all required constant values for CF^{512} of AURORA-384/512 version 2. In the following tables, the constant values are arranged from the left to the right.

Constant Values for AURORA-384/512 version 2, CF^{512}: $\{CONC^{512}_{L,j}\}_{0 \le j < 104}$							
6858f1ae	c2f4fa64	0e51cc84	0b363092	9b4ae35d	c06b6566	1ca23f96	3533320d
d55ff613	153c32b3	09ec7183	9a99e75a	4975dc8f	ab8f810d	2370eda9	fde459e9
d910b91f	20cec086	46e07dcc	7ef2d2a8	51eb4297	b1767817	bd68f537	0fd14310
e82c852e	f9aaa45f	7ad14cf0	b7400bcc	33933af5	ddc4ca7b	c50a974f	6b082fa2
2cdc75ea	cff3fd69	8a158800	052c3d95	1242ebd4	12f0feb4	142bb69e	0296e096
6f7ed7a9	a869670e	2856cba2	31e35a0f	9506af53	213d3387	50ac31da	98f1d35b
c9c76e0f	659799c3	91f06d1b	cc7897f1	7045ecb6	47c2cce1	1349d499	663cb5a4
ab70d96d	82f0fe24	26920fac	03b67096	b52b8273	e0696746	7d8c11f7	3173120f
899d344f	053d33a3	cbb02d41	98b9f75b	f0f15836	779799d1	a7c9542d	cc5c85f1
022980c4	4ec2cce8	7f3ba6f5	662ebca4	4fa80189	86707e20	fe76eb74	03bf7416
d4aa0312	e2292744	fced7076	b177104f	4a9f368c	041d13a2	c973ee43	d8bbf67b
dec46d18	770709d1	92e77a18	ec5d8561	5e43ea98	4e8a84e8	1567fa9f	f62ebcec
f77cd531	86545a20	2ace53a0	4bbf7432	0d339acb	e23b3544	6534a9ef	9577105d

Constant Values for AURORA-384/512 version 2, CF^{512}: $\{CONC^{512}_{R,j}\}_{0\leq j<104}$

42c78d73	17a7d326	728e6420	59b18490	da571aec	035b2b36	e511fcb0	a9999069
aaffb09e	a9e19598	4f638c18	d4cf3ad4	4baee47a	5c7c086d	1b876d49	ef22cf4f
c885c8fe	06760431	3703ee62	f7969543	8f5a14ba	8bb3c0bd	eb47a9bd	7e8a1880
41642977	cd5522ff	d68a6783	ba005e65	9c99d7a9	ee2653de	2854ba7e	58417d13
66e3af51	7f9feb4e	50ac4004	2961eca8	92175ea0	9787f5a0	a15db4f0	14b704b0
7bf6bd4b	434b3875	42b65d11	8f1ad079	a8357a9c	09e99c39	85618ed2	c78e9adc
4e3b707e	2cbcce1b	8f8368dc	63c4bf8e	822f65b3	3e16670a	9a4ea4c8	31e5ad23
5b86cb6d	1787f124	34907d61	1db384b0	a95c139d	034b3a37	ec608fbb	8b989079
4ce9a27c	29e99d18	5d816a0e	c5cfbadc	878ac1b7	bcbcce8b	3e4aa16d	62e42f8e
114c0620	76166742	f9dd37ab	3175e523	7d400c4a	3383f104	f3b75ba7	1dfba0b0
a5501896	11493a27	e76b83b7	8bb8827d	54f9b462	20e89d10	4b9f721e	c5dfb3de
f62368c6	b8384e8b	973bd0c4	62ec2b0f	f21f54c2	74542742	ab3fd4f8	b175e767
bbe6a98f	32a2d104	56729d01	5dfba192	699cd658	11d9aa27	29a54f7b	abb882ec

Constant Values for AURORA-384/512 version 2, CF^{512}: $\{CONM^{512}_{L,j}\}_{0\leq j<32}$

d0b1e35c	85e9f4c9	1ca39908	166c6124	92ebb91e	571f021b	46e1db52	fbc8b3d3
d0590a5d	f35548bf	f5a299e0	6e801799	2485d7a8	25e1fd68	28576d3c	052dc12c
938edc1f	cb2f3386	23e0da37	98f12fe3	6a5704e7	c0d2ce8d	fb1823ee	62e6241e
04530188	9d8599d0	fe774dea	cc5d7948	953e6d18	083a2744	92e7dc87	b177ecf7

Constant Values for AURORA-384/512 version 2, CF^{512}: $\{CONM^{512}_{R,j}\}_{0\leq j<32}$

cda571ae	6035b2b3	0e511fcb	9a999906	ec885c8f	10676043	23703ee6	3f796954
99c99d7a	eee2653d	e2854ba7	358417d1	b7bf6bd4	5434b387	142b65d1	98f1ad07
3822f65b	a3e16670	89a4ea4c	331e5ad2	c4ce9a27	829e99d1	e5d816a0	cc5cfbad
a7d400c4	43383f10	7f3b75ba	01dfba0b	6f62368c	bb8384e8	4973bd0c	f62ec2b0

Constant Values for AURORA-384/512 version 2, CF^{512}: $\{CONM^{512}_{X,j}\}_{0\leq j<32}$

557fd84f	54f0cacc	27b1c60c	6a679d6a	47ad0a5d	c5d9e05e	f5a3d4de	3f450c40
b371d7a8	3fcff5a7	28562002	14b0f654	541abd4e	84f4ce1c	42b0c769	63c74d6e
adc365b6	0bc3f892	9a483eb0	0ed9c258	c3c560db	de5e6745	9f2550b6	317217c7
52a80c4b	88a49d13	f3b5c1db	c5dc413e	790faa61	3a2a13a1	559fea7c	d8baf3b3

2.7 Pseudocodes

The pseudocodes of the specifications of the AURORA family are described in this section.

```
$MSM[F, F'](X_{(256)}, \{Y_{j\,(32)}\}_{0 \le j < 32})$
$(X_0, X_1, \ldots, X_7) \leftarrow X$
$(X_1, X_3, X_5, X_7) \leftarrow (X_1, X_3, X_5, X_7) \oplus (Y_0, Y_1, Y_2, Y_3)$
$(Z_0, Z_1, \ldots, Z_7) \leftarrow (X_0, X_1, \ldots, X_7)$
for $i \leftarrow 1$ to 7 do
    $(X_0, X_1, \ldots, X_7) \leftarrow BD(X_0, X_1, \ldots, X_7)$
    $(X_0, X_2, X_4, X_6) \leftarrow (F(X_0), F'(X_2), F(X_4), F'(X_6))$
    $(X_1, X_3, X_5, X_7) \leftarrow (X_1, X_3, X_5, X_7) \oplus (Y_{4i}, Y_{4i+1}, Y_{4i+2}, Y_{4i+3})$
    $(X_1, X_3, X_5, X_7) \leftarrow (X_1, X_3, X_5, X_7) \oplus (X_0, X_2, X_4, X_6)$
    $(Z_{8i}, Z_{8i+1}, \ldots, Z_{8i+7}) \leftarrow (X_0, X_1, \ldots, X_7)$
$(X_0, X_1, \ldots, X_7) \leftarrow BD(X_0, X_1, \ldots, X_7)$
$(X_0, X_2, X_4, X_6) \leftarrow (F(X_0), F'(X_2), F(X_4), F'(X_6))$
$(X_1, X_3, X_5, X_7) \leftarrow (X_1, X_3, X_5, X_7) \oplus (X_0, X_2, X_4, X_6)$
$(Z_{64}, Z_{65}, \ldots, Z_{71}) \leftarrow (X_0, X_1, \ldots, X_7)$
return $\{Z_{j\,(32)}\}_{0 \le j < 72}$
```

Figure 2.8: A pseudocode of $MSM : \{0,1\}^{256} \times (\{0,1\}^{32})^{32} \rightarrow (\{0,1\}^{32})^{72}$. BD is defined in Sec. 2.2.3. F and F' are functions over $\{0,1\}^{32}$.

```
$CPM^{512}[F, F'](X_{(256)}, \{Y_{j\,(32)}\}_{0 \le j < 216}, \{W_{j\,(32)}\}_{0 \le j < 104})$
$(X_0, X_1, \ldots, X_7) \leftarrow X$
$(X_1, X_3, X_5, X_7) \leftarrow (X_1, X_3, X_5, X_7) \oplus (W_0, W_1, W_2, W_3)$
$(X_0, X_1, \ldots, X_7) \leftarrow (X_0, X_1, \ldots, X_7) \oplus (Y_0, Y_1, \ldots, Y_7)$
for $i \leftarrow 1$ to 25 do
    $(X_0, X_1, \ldots, X_7) \leftarrow BD(X_0, X_1, \ldots, X_7)$
    $(X_0, X_2, X_4, X_6) \leftarrow (F(X_0), F'(X_2), F(X_4), F'(X_6))$
    $(X_1, X_3, X_5, X_7) \leftarrow (X_1, X_3, X_5, X_7) \oplus (W_{4i}, W_{4i+1}, W_{4i+2}, W_{4i+3})$
    $(X_1, X_3, X_5, X_7) \leftarrow (X_1, X_3, X_5, X_7) \oplus (X_0, X_2, X_4, X_6)$
    $(X_0, X_1, \ldots, X_7) \leftarrow (X_0, X_1, \ldots, X_7) \oplus (Y_{8i}, Y_{8i+1}, \ldots, Y_{8i+7})$
$(X_0, X_1, \ldots, X_7) \leftarrow BD(X_0, X_1, \ldots, X_7)$
$(X_0, X_2, X_4, X_6) \leftarrow (F(X_0), F'(X_2), F(X_4), F'(X_6))$
$(X_1, X_3, X_5, X_7) \leftarrow (X_1, X_3, X_5, X_7) \oplus (X_0, X_2, X_4, X_6)$
$(X_0, X_1, \ldots, X_7) \leftarrow (X_0, X_1, \ldots, X_7) \oplus (Y_{208}, Y_{209}, \ldots, Y_{215})$
$Z \leftarrow (X_0 \,\|\, X_1 \,\|\, \cdots \,\|\, X_7)$
return $Z_{(256)}$
```

Figure 2.9: A pseudocode of $CPM^{512} : \{0,1\}^{256} \times (\{0,1\}^{32})^{216} \times (\{0,1\}^{32})^{104} \rightarrow \{0,1\}^{256}$. BD is defined in Sec. 2.2.3. F and F' are functions over $\{0,1\}^{32}$.

```
$BD(X_{0\,(32)}, X_{1\,(32)}, \ldots, X_{7\,(32)})$
for $i \leftarrow 0$ to $7$ do
    $(x_{4i}, x_{4i+1}, x_{4i+2}, x_{4i+3}) \leftarrow X_i$
for $i \leftarrow 0$ to $31$ do
    $x'_{\pi(i)} \leftarrow x_i$
for $i \leftarrow 0$ to $7$ do
    $X_i \leftarrow (x'_{4i} \,\|\, x'_{4i+1} \,\|\, x'_{4i+2} \,\|\, x'_{4i+3})$
return $(X_{0\,(32)}, X_{1\,(32)}, \ldots, X_{7\,(32)})$
```

Figure 2.10: A pseudocode of $BD : (\{0,1\}^{32})^8 \to (\{0,1\}^{32})^8$. π is defined in Fig. 2.3.

```
$DR^{512}(\{X_{j\,(32)}\}_{0 \le j < 72}, \{Y_{j\,(32)}\}_{0 \le j < 72}, \{W_{j\,(32)}\}_{0 \le j < 72})$
for $i \leftarrow 0$ to $8$ do
    $(Z_{24i}, Z_{24i+1}, \ldots, Z_{24i+7}) \leftarrow PROTL(X_{8i}, X_{8i+1}, \ldots, X_{8i+7})$
    $(Z_{24i+8}, Z_{24i+9}, \ldots, Z_{24i+15}) \leftarrow PROTR(Y_{8i}, Y_{8i+1}, \ldots, Y_{8i+7})$
    $(Z_{24i+16}, Z_{24i+17}, \ldots, Z_{24i+23}) \leftarrow PROTX(W_{8i}, W_{8i+1}, \ldots, W_{8i+7})$
return $\{Z_{j\,(32)}\}_{0 \le j < 216}$
```

Figure 2.11: A pseudocode of $DR^{512} : (\{0,1\}^{32})^{72} \times (\{0,1\}^{32})^{72} \times (\{0,1\}^{32})^{72} \to (\{0,1\}^{32})^{216}$. The functions $PROTL$, $PROTR$ and $PROTX$ are defined in (2.8), (2.9) and (2.10), respectively.

```
AURORA-512v2($M$)
$(M_0, M_1, \ldots, M_{m-1}) \leftarrow Pad(M)$
$H_0 \leftarrow 0^{512}$
for $i \leftarrow 0$ to $m-2$ do
    $H_{i+1} \leftarrow CF^{512}(H_i, M_i)$
$H_m \leftarrow FF^{512}(H_{m-1}, M_{m-1})$
return $H_{m\,(512)}$
```

Figure 2.12: A pseudocode of AURORA-512 version 2. The padding function, $Pad(\cdot)$, is defined in (2.11), CF^{512} is defined in Sec. 2.3.2, FF^{512} is defined in Sec. 2.3.3.

```
$CF^{512}(H_{i\,(512)}, M_{i\,(512)})$
$(M_L, M_R) \leftarrow M_i$
$(X_L, X_R) \leftarrow H_i$
$\{T_{L,j}\}_{0 \le j < 72} \leftarrow MS_L^{512}(M_L)$
$\{T_{R,j}\}_{0 \le j < 72} \leftarrow MS_R^{512}(M_R)$
$\{T_{X,j}\}_{0 \le j < 72} \leftarrow MS_X^{512}(X_L)$
$\{U_j\}_{0 \le j \le 216} \leftarrow DR^{512}(\{T_{L,j}\}_{0 \le j < 72}, \{T_{R,j}\}_{0 \le j < 72}, \{T_{X,j}\}_{0 \le j < 72})$
$Y_L \leftarrow CP_L^{512}(X_R, \{U_j\}_{0 \le j < 216})$
$Y_R \leftarrow CP_R^{512}(X_R, \{U_j\}_{0 \le j < 216})$
$Z_L \leftarrow Y_L \oplus X_R$
$Z_R \leftarrow Y_R \oplus X_R$
$H_{i+1} \leftarrow (Z_L, Z_R)$
return $H_{i+1\,(512)}$
```

Figure 2.13: A pseudocode of $CF^{512} : \{0,1\}^{512} \times \{0,1\}^{512} \to \{0,1\}^{512}$. MS_L^{512}, MS_R^{512}, MS_X^{512}, DR^{512}, CP_L^{512}, and CP_R^{512} are defined in (2.12), (2.13), (2.14), Sec. 2.2.5, (2.15), and in (2.16), respectively.

```
$\mathit{FF}^{512}(H_{m-1\,(512)}, M_{m-1\,(512)})$
$(M_L, M_R) \leftarrow M_{m-1}$
$(X_L, X_R) \leftarrow H_{m-1}$
$\{T_{L,j}\}_{0 \le j < 72} \leftarrow \mathit{MSF}_L^{512}(M_L)$
$\{T_{R,j}\}_{0 \le j < 72} \leftarrow \mathit{MSF}_R^{512}(M_R)$
$\{T_{X,j}\}_{0 \le j < 72} \leftarrow \mathit{MSF}_X^{512}(X_L)$
$\{U_j\}_{0 \le j \le 216} \leftarrow \mathit{DR}^{512}(\{T_{L,j}\}_{0 \le j < 72}, \{T_{R,j}\}_{0 \le j < 72}, \{T_{X,j}\}_{0 \le j < 72})$
$Y_L \leftarrow \mathit{CPF}_L^{512}(X_R, \{U_j\}_{0 \le j < 216})$
$Y_R \leftarrow \mathit{CPF}_R^{512}(X_R, \{U_j\}_{0 \le j < 216})$
$Z_L \leftarrow Y_L \oplus X_R$
$Z_R \leftarrow Y_R \oplus X_R$
$H_m \leftarrow (Z_L, Z_R)$
return $H_{m\,(512)}$
```

Figure 2.14: A pseudocode of $\mathit{FF}^{512} : \{0,1\}^{512} \times \{0,1\}^{512} \rightarrow \{0,1\}^{512}$. MSF_L^{512}, MSF_R^{512}, MSF_X^{512}, DR^{512}, CPF_L^{512}, and CPF_R^{512} are defined in (2.17), (2.18), (2.19), Sec. 2.2.5, (2.20), and in (2.21), respectively.

```
AURORA-384v2($M$)
$(M_0, M_1, \ldots, M_{m-1}) \leftarrow \mathit{Pad}(M)$
$H_0 \leftarrow 1^{512}$
for $i \leftarrow 0$ to $m-2$ do
    $H_{i+1} \leftarrow \mathit{CF}^{512}(H_i, M_i)$
$H_m \leftarrow \mathit{FF}^{512}(H_{m-1}, M_{m-1})$
$H'_m \leftarrow \mathit{TF}_{384}(H_m)$
return $H'_{m\,(384)}$
```

Figure 2.15: A pseudocode of AURORA-384 version 2. Pad, CF^{512}, FF^{512} are the same as AURORA-512 and defined in Sec. 2.3.

2.8 AURORA Examples

This section describes example vectors of the AURORA hash algorithm family. Table 2.2 gives three examples for the messages M_1, M_2, and M_3 defined below for each hash function.

Let the message M_1 be the 24-bit ASCII string "**abc**", which is equivalent to the following binary string:

`01100001 01100010 01100011.`

Let the message M_2 be the 448-bit ASCII string

"**abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq**".

Let the message M_3 be the binary-coded form of the ASCII string which consists of 1,000,000 repetitions of the character "**a**".

Table 2.2: AURORA Examples

AURORA-384 version 2	
Message	Hash Value
M_1	cb8c22c8 15e3e5a3 8a1691ee f1dc1ad9 15dfea22 9f27a170 455aaaec b4a9f55a 3a372d1e 412d8853 b754ea23 c28a9e12
M_2	b1849343 f6601342 471176d7 bd671692 d3c39ca0 6f5d7a7c dccd802d 47ad5875 b6528095 d51d6be4 4bfb0b0d a5a90099
M_3	b579aa54 199a921d df7a3225 3dc82390 a40eb36d 1b649b8f 79430ef2 75b27f50 595ee272 979eef4d e108d540 b3004556

AURORA-512 version 2	
Message	Hash Value
M_1	51c0c29f d45b4bcf f7f54733 5af4424d 74817faf 1983bf5b e2afafd8 86f830bf b0a49fc2 9f65447b 5336d68c 5793d649 ad19dade 635a84c9 817681e0 1d36acae
M_2	5f2a16e9 99edf233 a9b96f52 1b6e792b bf33ea51 549bc0e7 9f5a62e4 17ceff99 ce7d9592 aae2edf8 1d46ec8e ad8181ec 6cba448e 4170b8cb f0c4ec12 eaceab6f
M_3	d00e5327 16a8dbb7 11aec003 36daeebd dccb72b9 278ad094 2f417e42 b2b09b67 80f18bfc 6d0403e0 6f32ca24 fe4b9b89 43281215 a3a6c560 46d828c5 b6fd6068

Chapter 3

Design Rationale of AURORA-384/512 version 2

This chapter describes design rationale of the AURORA-384/512 v2, then explains the algorithm changes from the previous version with their reasons.

3.1 AURORA-384/512 version 2

3.1.1 Domain Extension

AURORA-384/512 v2 adopts the strengthened Merkle-Damgård (sMD) transform with a finalization function, which is the same domain extension transform as AURORA-256 (See Fig. 3.1). The sMD transform is widely deployed and its security have been studied extensively. The sMD transform with the finalization function preserves collision-resistance (CR) and indifferentiability (PRO) of the underlying compression function [6].

Similarly to AURORA-256, the structure of the finalization function FF^{512} is the same as the structure of the compression function CF^{512} except for the constants.

3.1.2 Compression Function – Hirose's DBL construction

The AURORA-384/512 v2 compression function CF^{512} is based on one of Hirose's DBL constructions [4]. It is known that the construction shown in Fig. 3.2 composes collision-resistant hash functions when e_U and e_L are independent ideal blockciphers. As shown in Fig. 3.1, a half of the chaining value (X_R) is input to two chaining value processing functions CP_L^{512} and CP_R^{512}, which are different 256-bit permutations. The other half of the chaining value (X_L) is input to the message scheduling function MS_X^{512}. The 512-bit message block M_i is divided into M_L and M_R, and they are input to the message scheduling functions MS_L^{512} and MS_R^{512}, respectively.

Let a blockcipher with the block size n and the key size k an (n, k) blockcipher. Let e be an (n, k) blockcipher, $e : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$. Then $e(key, \cdot)$ is a permutation for every $key \in \{0,1\}^k$. AURORA-384/512 v2 compression function is constructed of $e_0(M_i||X_L, X_R)$ and $e_1(M_i||X_L, X_R)$, where e_0 and e_1 are two independent $(256, 768)$ blockciphers. Three message scheduling functions MS_L^{512}, MS_R^{512}, MS_X^{512} serve as common key scheduling of e_0 and e_1.

This construction is shown to be collision-resistant in the ideal cipher model [4].

3.1.3 Components

Message Scheduling Module and Chaining Value Processing Module The message scheduling functions of AURORA-384/512 v2 (i.e. MS_L^{512}, MS_R^{512}, and MS_X^{512}) use the same build-

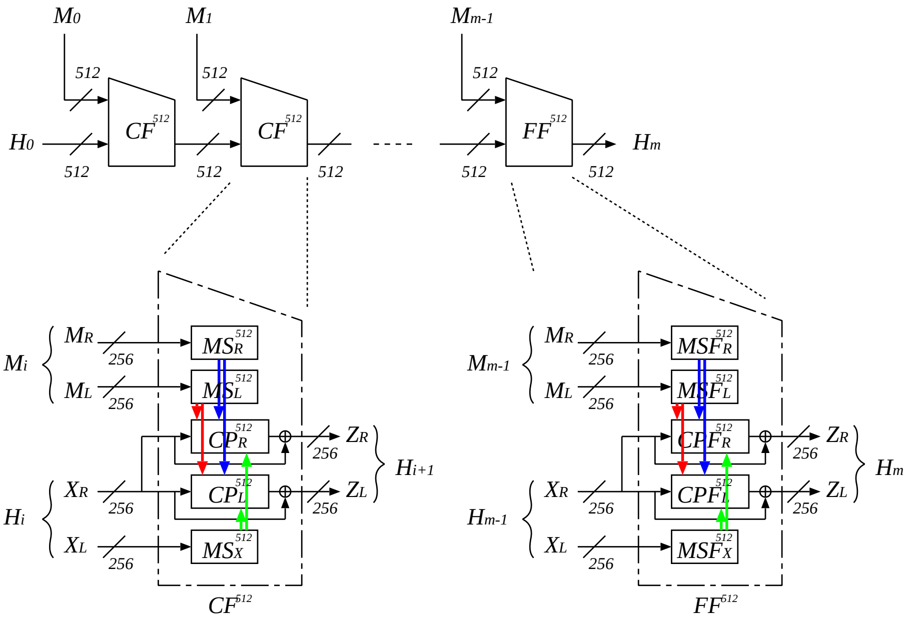


Figure 3.1: AURORA-384/512 v2: Domain extension and compression function.

ing block as those of AURORA-256: the message scheduling module (*MSM*), which is the 8-round AURORA structure.

The chaining value processing functions of AURORA-384/512 v2 (i.e. CP_L^{512} and CP_R^{512}) use a building block with more rounds than those of AURORA-256. While the chaining value processing module (*CPM*) used in AURORA-256 is the 17-round AURORA structure, CPM^{512} used in AURORA-512 v2 is the 26-round AURORA structure. The reason why we adopted the 26-round AURORA structure for CPM^{512} is as follows. First of all, the 26-round AURORA structure fits to input data from three message scheduling functions with the 8-round AURORA structure. Secondly, unexpected properties in blockciphers e_0 or e_1 should not be detected at $< 2^{512}$ work. According to the guaranteed numbers of active S-boxes in AURORA structure in Table 3.1, there are no distinguishers with differential probability higher than ${2^{-6}}^{46} = 2^{-276} < 2^{-256}$ for 14 rounds and more, since $DP_{max} = 2^{-6}$. Additional 12 ($= 26-14$)-round is considered to be enough margin. Note that e_0 and e_1 have 768-bit keys but at most 2^{512} work is allowed for attacks on 512-bit hash functions or its components.

3.2 Changes from the AURORA-384/512 version 1

This section explains the algorithm changes with their rationale.

DBL costruction AURORA-384/512 adopted the Double-Mix Merkle-Damgård (DMMD) transform. Due to the structual weakness of the DMMD transform [9, 8, 3, 10], we decided not to use the DMMD transform. However, we wanted to maintain the strategy of designing *double length* hash functions based on single length component functions, so we decided to adopt one of Hirose's DBL constructions. This strategy is advantageous in the hardware point of view and it keeps AURORA's strong point: compact, efficient and flexible hardware implementations with a wide variety of area/speed trade-offs. It also allows only minor changes from AURORA-384/512.

In the adopted Hirose's construction, key scheduling (i.e. three message scheduling functions MS_L^{512}, MS_R^{512}, MS_X^{512}) can be shared between two chaining value processing functions (i.e. CP_L^{512}

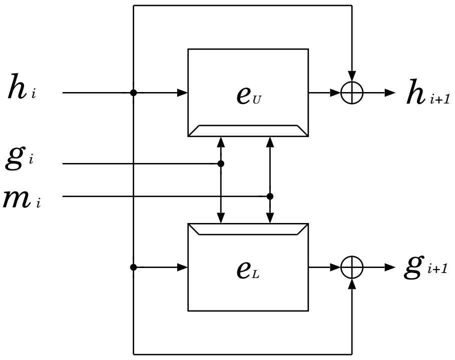


Figure 3.2: A compression function based on one of Hirose's DBL constructions

Table 3.1: Guaranteed Numbers of Active S-boxes in AURORA structure.

Round	♯ of Active S-boxes	Round	♯ of Active S-boxes
1	0	14	46
2	1	15	50
3	5	16	52
4	6	17	56
5	9	18	60
6	15	19	62
7	22	20	66
8	26	21	70
9	30	22	72
10	32	23	76
11	36	24	80
12	40	25	82
13	42	26	86

and CP_R^{512}). Moreover, no extra registers for input/output data of the compression function are required, so the Hirose's construction keeps compact hardware implementations.

MSM and CPM In order to adapt the Hirose's DBL construction to the compression function of AURORA-384/512 v2, additional message scheduling function MS_X^{512} and additional number of rounds for CP_L^{512} and CP_R^{512} are required. Due to this change, the throughput of the compression function decreases, but it does not have a big impact on performance (See Sec. 4).

Regarding security, the building blocks of AURORA-384/512 v2 (i.e. MS_L^{512}, MS_R^{512}, MS_X^{512}, CP_L^{512} and CP_R^{512}) are the same as those of AURORA-256 except the number of rounds, therefore existing analysis results are not invalidated.

The components in each building block of AURORA family are summarized in Table 3.2.

Diffusion Matrices for MS_X^{512} AURORA-384/512 v2 requires diffusion matrices for MS_X^{512} but does not require brand-new diffusion matrices. A different pair of the diffusion matrices used in AURORA suffices for this purpose. It is desirable that a pair of diffusion matrices for MS_X^{512} be chosen so that every function of *MSM* and CPM^{512} is different and that the DSM technique works not only within the single function (MS_X^{512}) but also across plural functions MS_X^{512}, CP_L^{512} and CP_R^{512}. Since ($\mathcal{M}_0$, $\mathcal{M}_1$), ($\mathcal{M}_2$, $\mathcal{M}_3$), ($\mathcal{M}_0$, $\mathcal{M}_3$), and ($\mathcal{M}_1$, $\mathcal{M}_2$) are DSM pairs, respectively [6, Sec. 3.4.2], we chose $\mathcal{M}_0$ and $\mathcal{M}_3$ as diffusion matrices for MS_X^{512}.

Table 3.2: Components in each building block of AURORA family.

AURORA-224/256	Function	MS_L	MS_R		CP	
[SBL]	Building block	MSM	MSM		CPM	
Davies-Meyer	# of rounds	8-round	8-round		17-round	
construction	F-functions	F_0,F_1	F_2,F_3		F_1,F_0	
	Matrices	$\mathcal{M}_0$,$\mathcal{M}_1$	$\mathcal{M}_2$,$\mathcal{M}_3$		$\mathcal{M}_1$,$\mathcal{M}_0$	
AURORA-384/512	Function	MS_L	MS_R		CP_L	CP_R
[DBL]	Building block	MSM	MSM		CPM	CPM
DMMD (Double-Mix	# of rounds	8-round	8-round		17-round	17-round
Merkle-Damgård)	F-functions	F_0,F_1	F_2,F_3		F_1,F_0	F_3,F_2
	Matrices	$\mathcal{M}_0$,$\mathcal{M}_1$	$\mathcal{M}_2$,$\mathcal{M}_3$		$\mathcal{M}_1$,$\mathcal{M}_0$	$\mathcal{M}_3$,$\mathcal{M}_2$
AURORA-384/512 v2	Function	MS_L^{512}	MS_R^{512}	MS_X^{512}	CP_L^{512}	CP_R^{512}
[DBL]	Building block	MSM	MSM	MSM	CPM^{512}	CPM^{512}
Hirose's	# of rounds	8-round	8-round	8-round	26-round	26-round
construction	F-functions	F_0,F_1	F_2,F_3	F_0,F_3	F_1,F_0	F_3,F_2
	Matrices	$\mathcal{M}_0$,$\mathcal{M}_1$	$\mathcal{M}_2$,$\mathcal{M}_3$	$\mathcal{M}_0$,$\mathcal{M}_3$	$\mathcal{M}_1$,$\mathcal{M}_0$	$\mathcal{M}_3$,$\mathcal{M}_2$

Note that there are no changes in the other components such as the byte diffusion BD, the S-box, and the truncation function.

Additional Constants AURORA-384/512 v2 requires the additional constant values for the new function MS_X^{512} and increased rounds of CP_L^{512}, CP_R^{512}. Constants are also required for functions MSF_X^{512}, CPF_L^{512}, and CPF_R^{512} in the finalization function. They are generated in a similar and consistent manner with the procedure described in the AURORA specification [6]. The generated sequences of constants passed the statistical test suites: the mono bit test, the poker test, and the runs test that we checked in [6].

3.2.1 Effect on Security

Collision attacks The Hirose's construction adopted for AURORA-384/512 v2 is shown to be collision-resistant in the ideal cipher model [4]. The components used in the compression function are the same as the previous version, so similar analysis approach described in [6, Sec. 4.3.1] is valid. According to preliminary analysis, we have not found effective differential paths for collision attacks.

Preimage attacks AURORA-384/512 v2 is preimage resistant if the finalization function FF^{512} is preimage resistant. According to preliminary analysis of FF^{512}, we have not found effective methods for preimage attacks.

Second preimage attacks The components used in the compression function are the same as the previous version, so similar analysis approach described in [6, Sec. 4.3.3] is valid. According to preliminary analysis, we have not found effective methods for second preimage attacks.

Regarding generic long-message second preimage attacks, Kelsey and Schneier's attack works on the strengthened Merkle-Damgård transform, so AURORA-512 v2 provides second preimage resistance of about $(512 - k)$ bits for 2^k-block messages.

Other attacks AURORA-384/512 v2 has resistance to length-extension attacks due to the strengthened Merkle-Damgård transform with the finalization function.

On the other hand, this domain extension does not help to increase the security against multicollision attacks. Therefore, finding K collision for AURORA-384/512 v2 is not much harder than finding ordinary collisions.

Slide attacks do not work on AURORA-384/512 v2, because AURORA-384/512 v2 are carefully designed to avoid self-similarity.

3.2.2 Impact on Performance

The changes which have an impact on efficiency are the change of the DBL construction and the changes in *MSM* and CPM^{512}.

Although AURORA-384/512 v2 does not require mixing functions *MF* inserted every 8 blocks, it has a new function MS_X^{512}, and CP_L^{512} and CP_R^{512} with increased number of rounds. Due to this change, the throughput of AURORA-384/512 v2 decreases, about 25% decrease in software implementations and about 25% decrease in the fastest hardware implementation.

However, this change has almost no impact on the area (gate counts) in area-optimized hardware implementation. AURORA-384/512 v2 keeps AURORA family's strong point: compact, efficient and flexible hardware implementations with a wide variety of area/speed trade-offs.

Chapter 4

Efficient Implementation of AURORA-384/512 version 2

This chapter describes our preliminary evaluation results of AURORA-384/512 version 2 in both software and hardware implementations. In software implementations, AURORA-512 version 2 achieves 63.9 cycles/byte and 37.8 cycles/byte on the NIST 32-bit reference platform and on the NIST 64-bit reference platform, respectively. In hardware, AURORA enables a variety of implementations from small-area to high-throughput implementations. In our evaluations using a $0.13\mu m$ CMOS ASIC library, the smallest area of AURORA-512 version 2 is 12.4 Kgates with throughout of 467 Mbps, and the highest throughput of AURORA-512 version 2 is 6.9 Gbps with area of 59.7 Kgates.

Detailed results of software and hardware implementations are shown in Sec. 4.1 and 4.2, respectively.

4.1 Software Implementation

This section describes the software performance results of AURORA-384/512 version 2 on 32/64-bit processors. We apply five implementation types suitable for either 32-bit or 64-bit processors: two types for 32-bit processors and three types for 64-bit processors. The details of five implementation types are shown in [6].

Tables 4.2 and 4.3 represent the evaluation results of AURORA-384 version 2 and AURORA-512 version 2 on 32/64-bit processors at the present, respectively. The platforms used for the evaluation are shown in Table 4.1. We use cycle counters included in 'cycle.h' [2]. This code provides machine dependent cycle counters. All five implementation types are evaluated with looped architecture, where the round functions of AURORA are implemented by loop functions, for both AURORA-384 and 512 version 2.

Table 4.1: 32/64-bit Processors

Platform	Processor	Clock speed [GHz]	Memory [GB]	OS	Compiler
A	Core 2 Duo	2.4	2.0	Windows Vista Ultimate (32-bit)	Visual Studio 2005 Professional Edition
B	Core 2 Duo	2.4	2.0	Windows Vista Ultimate (64-bit)	Visual Studio 2005 Professional Edition
C	Opteron	2.6	16.0	Linux kernel 2.4	gcc 3.2.3 (x64)
D	Pentium 4	2.26	1.0	Red Hat Linux 7.3	gcc 2.96

Table 4.2: AURORA-384 version 2 on 32/64-bit processors

		Hash Function [cycles/byte]					1 CF call [cycles]	code size [bytes]
message size [bytes]		1	10	100	1,000	10,000	-	-
		Platform A (Core 2 Duo (32-bit))						
Type-S1	(looped)	4,657.4	466.3	91.6	110.3	68.1	4,317.2	71,042
Type-S2	(looped)	4,353.9	441.8	86.1	65.8	63.9	4,055.1	69,182
Type-S3	(looped)	6,629.6	666.2	131.3	100.4	97.9	6,175.6	125,704
Type-S4	(looped)	5,631.8	567.0	110.9	84.8	82.7	6,111.5	101,012
Type-S5	(looped)	5,427.9	548.2	107.2	81.9	79.9	5,064.0	88,792
		Platform B (Core 2 Duo (64-bit))						
Type-S1	(looped)	2,994.4	303.8	58.3	44.8	43.6	2,764.2	72,752
Type-S2	(looped)	3,390.9	340.7	66.7	51.2	50.1	3,172.7	71,756
Type-S3	(looped)	2,740.1	274.6	52.9	39.7	38.7	2,441.3	133,132
Type-S4	(looped)	2,651.1	266.3	51.6	38.8	37.8	2,403.3	108,440
Type-S5	(looped)	2,895.6	291.1	56.4	42.8	41.7	2,649.6	96,220
		Platform C (Opteron)						
Type-S1	(looped)	4,077.7	410.8	77.2	57.9	56.3	3,568.8	25,460
Type-S2	(looped)	4,463.8	449.6	84.3	63.1	61.3	3,889.5	19,892
Type-S3	(looped)	3,177.0	320.6	59.3	43.5	42.1	2,680.1	48,740
Type-S4	(looped)	3,213.8	323.2	59.5	43.8	42.5	2,697.5	32,516
Type-S5	(looped)	3,447.1	347.6	64.6	48.0	46.6	2,964.0	24,484
		Platform D (Pentium 4)						
Type-S1	(looped)	9,861.4	1,001.3	192.1	145.2	141.4	9,497.9	26,164
Type-S2	(looped)	10,847.3	1,060.2	204.5	157.5	153.6	9,663.7	20,496
Type-S3	(looped)	16,045.5	1,621.1	310.1	237.8	232.0	14,566.9	61,300
Type-S4	(looped)	15,094.8	1,518.9	292.4	223.1	217.5	13,845.5	43,952
Type-S5	(looped)	14,622.1	1,476.1	283.6	216.3	210.9	13,642.5	34,988

Table 4.3: AURORA-512 version 2 on 32/64-bit processors

		Hash Function [cycles/byte]					1 CF call [cycles]	code size [bytes]
message size [bytes]		1	10	100	1,000	10,000	-	-
Type-S1	(looped)	4,614.0	464.9	91.4	69.8	68.1	4,317.2	71,042
Type-S2	(looped)	4,350.8	439.1	86.1	65.6	63.9	4,055.1	69,182
Type-S3	(looped)	6,490.7	655.8	130.8	100.3	97.9	6,175.6	125,704
Type-S4	(looped)	5,598.6	561.6	110.5	84.8	82.7	6,111.5	101,012
Type-S5	(looped)	5,408.8	543.3	106.7	81.8	79.9	5,064.0	88,792
		Platform B (Core 2 Duo (64-bit))						
Type-S1	(looped)	2,943.8	295.4	57.9	44.6	43.6	2,764.2	72,752
Type-S2	(looped)	3,365.0	336.2	66.3	51.2	50.1	3,172.7	71,756
Type-S3	(looped)	2,667.8	268.5	52.4	39.6	38.6	2,441.3	133,132
Type-S4	(looped)	2,607.1	261.9	50.6	38.8	37.8	2,403.3	108,440
Type-S5	(looped)	2,857.5	286.6	55.8	42.8	41.8	2,649.6	96,220
		Platform C (Opteron)						
Type-S1	(looped)	3,895.3	391.7	75.2	57.7	56.2	3,568.8	25,460
Type-S2	(looped)	4,244.2	426.8	82.0	62.8	61.3	3,889.5	19,892
Type-S3	(looped)	2,999.9	302.4	57.5	43.3	42.3	2,680.1	48,740
Type-S4	(looped)	3,008.6	303.5	57.5	43.6	42.6	2,697.5	32,516
Type-S5	(looped)	3,253.4	327.8	62.7	47.8	46.5	2,964.0	24,484
		Platform D (Pentium 4)						
Type-S1	(looped)	9,916.8	991.4	191.7	145.0	141.3	9,497.9	26,164
Type-S2	(looped)	10,715.7	1,114.7	206.9	157.9	154.4	9,663.7	20,496
Type-S3	(looped)	15,928.9	1,594.4	308.9	237.0	231.6	14,566.9	61,300
Type-S4	(looped)	14,851.3	1,498.3	289.7	222.6	217.3	13,845.5	43,952
Type-S5	(looped)	14,457.9	1,449.4	281.2	215.9	210.7	13,642.5	34,988

4.2 Hardware Implementation

This section describes the hardware performance results of AURORA. Since the implementations of AURORA-384 version 2 are basically same as AURORA-512 version 2 except the initial value and truncation of final hash value, we designed and evaluated the implementations of AURORA-512 version 2.

We designed three types of hardware implementations: Type-H1, Type-H4 and Type-H5 implementations. The details of Type-H1 and Type-H4 are shown in [6]. Type-H1 implementation processes a round of the AURORA architecture both in one of the message scheduling module MSM and in the two chaining value processing modules CPM^{512} simultaneously in one clock cycle. It requires 12 F-function circuits and takes 27 cycles for the compression function CF^{512} and the finalization function FF^{512}. Type-H4 implementation processes a round of the AURORA architecture in one of MSM or in one of CPM^{512} mutually in two clock cycles. It requires 2 F-function circuits and takes 166 cycles for CF^{512} and FF^{512}: 162 cycles for the message scheduling and chaining value processing, and 4 cycles for updating the chaining value. We also apply Type-H5 implementation of AURORA-256 described in [1] to AURORA-512 version 2. In Type-H5 implementation, a round of the AURORA architecture in one of MSM or in one of CPM^{512} is processed mutually in four clock cycles. It requires one F-function circuit and takes 332 cycles for CF^{512} and FF^{512}: 324 cycles for the message scheduling and chaining value processing, and 8 cycles for updating the chaining value.

The environment of our hardware design and evaluation is as follows.

Language	Verilog-HDL
Design library	0.13 μm CMOS ASIC library
Simulator	VCS version 2006.06
Logic synthesis	Design Compiler version 2007.03-SP3

One gate is equivalent to a 2-way NAND and speed is evaluated under the worst-case conditions.

Table 4.4 represents the evaluation results of AURORA-512 version 2. Type-H1, Type-H4, and Type-H5 implementations with S-boxes based on inversion over GF($(2^4)^2$) are evaluated. In addition, Type-H1 implementation with table-lookup S-boxes is also evaluated in order to achieve higher throughput. Control signals for all selectors and constant values are generated in a controller module which is included in each implementation. For each implementation of AURORA-512, two types of circuits are synthesized by specifying either area or speed optimization. In addition, we investigate the condition to maximize "Efficiency" that indicates "Throughput" per area, which we call efficiency optimization.

We also show, for comparison, the best known results of hardware performance of SHA-384/512 using a 0.13 μm CMOS ASIC library by Satoh et al. [11]. The performance of AURORA cannot be directly compared with them because different design libraries and different logic synthesis tools were used. However, AURORA-512 version 2 enables a variety of implementations from small-area to high-throughput implementations; the smallest area (12,389 gates) is about 46% smaller than that of SHA-384/512 (23,146 gates), and the highest throughput (6,901 Mbps) is about 2.37 times higher than that of SHA-384/512 (2,909 Mbps). The highest efficiency of AURORA-512 (146.8 Kbps/gate) is about 1.38 times higher than that of SHA-384/512 (106.6 Kbps/gate).

Table 4.4: Results on Hardware Performance of AURORA-512 version 2

	Data Path Architecture	Cycles	S-box	Area [gates]	Frequency [MHz]	Throughput [Mbps]	Efficiency [Kbps/gate]
AURORA-512 v2 ($0.13\mu m$)	Type-H1	27	GF($(2^4)^2$)	28,968	191.4	3,629	125.3
				41,985	285.6	5,416	129.0
				31,543	244.1	4,629	146.8
			Table	42,637	211.4	4,009	94.0
				59,657	363.9	6,901	115.7
				47,678	309.3	5,865	123.0
	Type-H4	166	GF($(2^4)^2$)	14,942	310.2	957	64.0
				17,628	504.8	1,557	88.3
				17,012	491.8	1,517	89.2
	Type-H5	332	GF($(2^4)^2$)	12,389	302.7	467	37.7
				13,914	509.3	785	56.5
				13,914	509.3	785	56.5
SHA-384/512 ($0.13\mu m$) [11]	-	88	-	23,146	125.0	1,455	62.8
				27,297	250.0	2,909	106.6

For each implementation, the 1st row and the 2nd row show the results of the synthesized circuits by area and speed optimization, respectively. The 3rd row also shows the results by efficiency optimization for each implementation of AURORA-512 version 2.

Bibliography

[1] T. Akishita, T. Yamamoto, and H. Abe. Compact implementation of aurora-256. On-line document, 2009.

[2] cycle.h. available at `http://www.fftw.org/cycle.h`.

[3] Niels Ferguson and Stefan Lucks. Attacks on AURORA-512 and the Double-Mix Merkle-Damgaard transform. In *IACR ePrint archive 2009/113*, 2009.

[4] Shoichi Hirose. Provably secure double-block-length hash functions in a black-box model. In Choonsik Park and Seongtaek Chee, editors, *Information Security and Cryptology - ICISC 2004, 7th International Conference, Seoul, Korea, December 2-3, 2004, Revised Selected Papers*, volume 3506 of *Lecture Notes in Computer Science*, pages 330–342. Springer, 2005.

[5] Shoichi Hirose. Some plausible constructions of double-block-length hash functions. In Matthew J. B. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*, pages 210–225. Springer, 2006.

[6] Tetsu Iwata, Kyoji Shibutani, Taizo Shirai, Shiho Moriai, and Toru Akishita. AURORA: A cryptographic hash algorithm family. In *Submission to NIST*, 2008.

[7] National Institute of Standards and Technology. Announcement request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family. Docket No.:070911510-7512-01, 2007. `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`.

[8] Yu Sasaki. A 2nd-preimage attack on AURORA-512. In *IACR ePrint archive 2009/112*, 2009.

[9] Yu Sasaki. A collision attack on AURORA-512. In *IACR ePrint archive 2009/106*, 2009.

[10] Yu Sasaki. A full key recovery attack on HMAC-AURORA-512. In *IACR ePrint archive 2009/125*, 2009.

[11] Akashi Satoh and Tadanobu Inoue. ASIC-hardware-focused comparison for hash functions MD5, RIPEMD-160, and SHS. Integration, the VLSI Journal, 40(1):3–10, 2007.