ARIRANG

Designed by CIST

Algorithm Name	:	ARIRANG
Principal Submitter	:	Jongin Lim
Tel	:	+82 2 3290 4044
Fax	:	+82 2 928 9109
Email	:	jilim@korea.ac.kr
Organization	:	Korea Univ.
Postal address	:	Center for Information Security Technologies (CIST), Korea Univ., 5-ga Anam-dong, Sungbuk-gu, Seoul, 136-075, Korea.
Auxiliary Submitter	:	Seokhie Hong
Tel	:	+82 2 3290 4894
Fax	:	+82 2 928 9109
Email	:	shhong@korea.ac.kr
Organization	:	Korea Univ.
Postal address	:	Center for Information Security Technologies (CIST), Korea Univ., 5-ga Anam-dong, Sungbuk-gu, Seoul, 136-075, Korea.
Algorithm Inventors	:	Donghoon Chang, Seokhie Hong, Changheon Kang, Jinkeon Kang, Jongsung Kim, Changhoon Lee, Jesang Lee, Jongtae Lee, Sangjin Lee, Yuseop Lee, Jongin Lim, Jaechul Sung
Algorithm Owner	:	Jongin Lim
Backup Point of Contact	:	Seokhie Hong
Submitter's Signatures	:	Jongin Lim ________________________
		Seokhie Hong ________________________

ARIRANG : SHA-3 Proposal

Designed by Center for Information Security Technologies

Contents

1 Introduction

In this document we present a dedicated ARIRANG[1] hash algorithm family, which includes 224, 256, 384 and 512-bit variants to allow substitution for the SHA-2 family. Throughout the document, the four members of the ARIRANG family are called ARIRANG-224, ARIRANG-256, ARIRANG-384 and ARIRANG-512 by their hashed bit sizes. The ARIRANG family process messages in the usual left-to-right order like Merkle-Damgård strengthening [16, 27, 25] and are suitable for any cryptographic applications such as digital signatures, HMAC [5], pair-wise key establishment schemes, deterministic random bit generators, randomized hashing [18], etc.

This document is organized as follows. In Sect. 2, we present definitions, notations, and the mathematical basis necessary for understanding the specification of ARIRANG, followed by its design rationale in Sect. 3 and the algorithm specifications in Sect. 4. This is followed by the motivations for all design choices in Sect. 5, and the treatment of the resistance against known types of attacks and the formal security proofs of the structure of ARIRANG in Sect. 6. In Sect. 7, we provide the security requirements of some cryptographic applications based on hash functions and give the security of them based on ARIRANG. Subsequently, constructions of ARIRANG based HMAC, Pseudo Random Functions (PRFs), and Randomized Hashing are treated in Sect. 8. Lastly, we deal with implementations and efficiencies of ARIRANG in Sect. 9.

2 Preliminaries

2.1 Definitions

The following terms are used in the ARIRANG family.

Bit	A binary digit having a value of 0 or 1.
Byte	An ordered collection of eight bits.
Word	An ordered collection of either 32 bits (4 bytes) or 64 bits (8 bytes), which depends on the variants of the ARIRANG family.

2.2 Algorithm Parameters and Symbols

2.2.1 Parameters

The following parameters are used in the specification of the ARIRANG family.

$a, b, c, \ldots, h$	Working variables that are the w-bit words used in the computation of the intermediate hash values H^i, where w is either 32 or 64.

[1] "ARIRANG" is arguably the most popular and best-known Korean folk song, both inside and outside Korea. ARIRANG is an ancient native Korean word with no direct modern meaning. 'Ari' means "beautiful", "lovely" and "charming". 'Rang' can mean "dear". Because of those words, arirang could be interpreted to mean "beautiful dear"; however, this modern interpretation is unlikely to match the original (ancient) meaning.

Ctr^i The i^{th} $2w$-bit counter value; the left-most w-bit word is denoted by Ctr^i_0 and the right-most w-bit word is denoted by Ctr^i_1, where w is either 32 or 64.

$F^{(256)}$ The compression function of ARIRANG-256 and ARIRANG-224.

$F^{(512)}$ The compression function of ARIRANG-512 and ARIRANG-384.

$G^{(256)}$ The function used in each step of ARIRANG-256 and ARIRANG-224.

$G^{(512)}$ The function used in each step of ARIRANG-512 and ARIRANG-384.

k Number of zeros appended to a message during the padding step.

l Length of the message, M, in bits.

M Message to be hashed.

m Number of bits in a message block.

M^i Message block i, with a size of m bits.

M^i_j The j^{th} word of the i^{th} message block M^i, where M^i_0 is the left-most word of M^i.

N Number of blocks in the padded message.

H^i The i^{th} hash value; H^0 is the initial value and H^N is the final hash value (H^N is used to determine the message digest).

H^i_j The j^{th} word of the i^{th} intermediate hash value H^i, where H^i_0 is the left-most word of H^i.

$K^{(256)}_0, \cdots, K^{(256)}_{15}$ Sixteen 32-bit constant values to be used in the message schedule of ARIRANG-256 and ARIRANG-224.

$K^{(512)}_0, \cdots, K^{(512)}_{15}$ Sixteen 64-bit constant values to be used in the message schedule of ARIRANG-512 and ARIRANG-384.

S LFSR to be used for generating constants.

s State of LFSR.

T_1, T_2 Temporary w-bit words used in the hash computation, where w is either 32 or 64.

$W_{\sigma(2t)}$, $W_{\sigma(2t+1)}$	Two w-bit message words into the t-th step function, where w is either 32 or 64.
$\sigma(\cdot)$	Permutation for the message ordering in the message schedule.

2.2.2 Symbols

The following symbols are used in the specification of the ARIRANG family, and each operates on w-bit words.

$\oplus$	Bitwise eXclusive-OR (XOR) operation.
$\bullet$	Multiplication in $GF(2^8)$.
$\otimes$	Multiplication in $GF(2^8)[x]$.
$\lll n$	Left rotation of n-bit position.
$\ggg n$	Right rotation of n-bit position.
$\|$	Concatenation.

2.3 Notation and Conventions

2.3.1 Bit String and Integers

The following terminologies related to bit strings and integers are used.

1. A hexadecimal digit is an element of the set {0,1,. . .,9,A,. . .,F}.
2. A word may be represented as a sequence of hexadecimal digits.

   Throughout this specification, each of 32-bit and 64-bit words is stored into a string in the 'big-endian' convention.
3. In the ARIRANG family, a word representation of an integer is required for counters and the message length l in the padding technique of Sect. 4.1.1.
   (a) In ARIRANG-224 and ARIRANG-256, integers between 0 and $2^{64}-1$ both inclusive are used. If Z is such an integer, then $Z = 2^{32} \cdot X + Y$, where $0 \le X < 2^{32}$ and $0 \le Y < 2^{32}$. So, Z can be represented as $X' \| Y'$, where X' and Y' are 32-bit word representations of X and Y by representing the least significant four bits of the integer as the right-most hexadecimal digit of the word representation.
   (b) In ARIRANG-384 and ARIRANG-512, integers between 0 and $2^{128}-1$ both inclusive are used. If Z is such an integer, then $Z = 2^{64} \cdot X + Y$, where $0 \le X < 2^{64}$ and $0 \le Y < 2^{64}$. So, Z can be represented as $X' \| Y'$, where X' and Y' are 64-bit word representations of X and Y.

4. For the ARIRANG family, the size of the message block depends on the algorithm.
   (a) For ARIRANG-224 and ARIRANG-256, each message block has 512 bits, which are represented as a sequence of sixteen 32-bit words.
   (b) For ARIRANG-384 and ARIRANG-512, each message block has 1024 bits, which are represented as a sequence of sixteen 64-bit words.

2.4 Mathematical Preliminaries

In this section, we introduce basic mathematical concepts needed in the following document.

2.4.1 The Field $GF(2^8)$

A finite field is an algebraic object with two operations: addition and multiplication, but these operations are different from the standard integer operations. All byte values in ARIRANG correspond to elements of a finite field $GF(2^8)$. A byte $b = b_7b_6 \cdots b_0$ is interpreted as a finite field element using the following polynomial representation:

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

Addition and multiplication in $GF(2^8)$ are exactly the same as in AES, i.e., ARIRANG adopts the irreducible reducing polynomial of degree 8, denoted by $m(x)$, used in AES to define the base finite field $GF(2^8)$:

$$m(x) = x^8 + x^4 + x^3 + x + 1,$$

or {01}{1B} in hexadecimal notation.

Addition in $GF(2^8)$. To add two elements in $GF(2^8)$, we apply the corresponding polynomial coefficients to the addition in $GF(2)$; the addition in $GF(2^8)$ is performed with the XOR operation $\oplus$.

Multiplication in $GF(2^8)$. In the polynomial representation, the multiplication in $GF(2^8)$, denoted by $\bullet$, is on the modular operation of the irreducible polynomial $m(x)$. The multiplication of the binary polynomial $b(x) = \sum_{i=0}^{7} b_ix^i$ and the polynomial x is computed as

$$b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x.$$

In order to reduce the above result down to degree i ($0 \leq i \leq 7$), a modular operation is performed by the irreducible polynomial $m(x)$. If $b_7 = 0$, a reduction is not needed. Otherwise, a reduction is achieved by XORing the polynomial $m(x)$, resulting in a polynomial of degree less than 8. In hexadecimal notation, a multiplication by x is accomplished at the byte level by a left shift and a subsequent conditional bitwise XOR with {1B}.

By repeated application of the above computation, a multiplication by higher powers of x can be implemented, and furthermore by adding intermediate results obtained from such multiplication, the multiplication by any polynomial can be calculated. For example, {6F}$\bullet${13} is calculated as follows:

$$
\begin{aligned}
\{\texttt{6F}\} \bullet \{\texttt{02}\} &= \{\texttt{DE}\}\\
\{\texttt{6F}\} \bullet \{\texttt{04}\} &= \{\texttt{A7}\}\\
\{\texttt{6F}\} \bullet \{\texttt{08}\} &= \{\texttt{55}\}\\
\{\texttt{6F}\} \bullet \{\texttt{10}\} &= \{\texttt{AA}\}
\end{aligned}
$$

thus,

$$
\begin{aligned}
\{\texttt{6F}\} \bullet \{\texttt{13}\} &= \{\texttt{6F}\} \bullet (\{\texttt{01}\} \oplus \{\texttt{02}\} \oplus \{\texttt{10}\})\\
&= \{\texttt{6F}\} \oplus \{\texttt{DE}\} \oplus \{\texttt{AA}\}\\
&= \{\texttt{1B}\}.
\end{aligned}
$$

2.4.2 Polynomials in $\mathbf{GF}(2^8)[x]$

A four-term polynomial and an eight-term polynomial in $GF(2^8)[x]$ represent a 4-byte word and a 8-byte word, respectively. The addition is performed in $GF(2)$; it corresponds to an XOR operation between the corresponding bytes in each of the words. The multiplication is achieved by two steps. In the first step, the polynomial multiplication is algebraically performed, and then in the second step, the result is reduced by a modular operation with a reducing polynomial of degree 4 or 8. For ARIRANG-256 and ARIRANG-512, this is accomplished with the reducing polynomials $x^4 + 1$ and $x^8 + 1$ both in $GF(2^8)[x]$, respectively. The polynomials $x^4 + 1$ and $x^8 + 1$ are not irreducible polynomials over $GF(2^8)$ each, hence the multiplication by a fixed polynomial is not necessarily invertible. In the ARIRANG family, two fixed polynomials are used so that the multiplications by them are both invertible; for ARIRANG-256 a fixed polynomial $a(x) = \{\texttt{03}\}x^3 + \{\texttt{01}\}x^2 + \{\texttt{01}\}x + \{\texttt{02}\}$ is used, and for ARIRANG-512 a fixed polynomial $a(x) = \{\texttt{02}\}x^7 + \{\texttt{0A}\}x^6 + \{\texttt{09}\}x^5 + \{\texttt{08}\}x^4 + \{\texttt{01}\}x^3 + \{\texttt{04}\}x^2 + \{\texttt{01}\}x + \{\texttt{01}\}$ is used. Thus, given an input word corresponding to $y(x) \in GF(2^8)[x]$, the output word corresponding to $z(x) \in GF(2^8)[x]$ after this word-level multiplication by the fixed polynomial is calculated by $z(x) = a(x) \otimes y(x)$ modulo $x^4 + 1$ or $x^8 + 1$ (this transformation is later called the MDS transformation in Sects. 4.1.6 and 4.3.6). Note that the byte-level multiplication caused by the word-level multiplication is performed by the operations defined in the previous section.

Multiplication by x in $GF(2^8)[x]$ If $a(x) = \sum_{i=0}^{3} a_i x^i \in GF(2^8)[x]$ and $b(x) = \sum_{i=0}^{7} b_i x^i \in GF(2^8)[x]$ are multiplied by the polynomial x in $GF(2^8)[x]$, then the results are respectively

$$
\begin{aligned}
&a_3x^4 + a_2x^3 + a_1x^2 + a_0x \text{ and}\\
&b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x.
\end{aligned}
$$

As in the reduction of the multiplication in $GF(2^8)$, modular operations are performed by the reducing polynomials $x^4 + 1$ and $x^8 + 1$, respectively. This computation results in

$$
\begin{aligned}
&a_2x^3 + a_1x^2 + a_0x + a_3 \text{ and}\\
&b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x + b_7.
\end{aligned}
$$

The resultant values show that the multiplication by x (or even by higher powers of x) represents a cyclic shift of the bytes inside the vector. The multiplication by x is also interpreted as a multiplication by a matrix. If $c(x) = x \otimes a(x)$ and $d(x) = x \otimes b(x)$ in $GF(2^8)[x]$, then they are calculated as follows.

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 00 & 00 & 00 & 01 \\ 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \end{bmatrix} = \begin{bmatrix} 00 & 00 & 00 & 00 & 00 & 00 & 00 & 01 \\ 01 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 01 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 01 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 01 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 01 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 01 & 00 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix}$$

3 Design Rationale

There are six criteria for the design rationale of ARIRANG.

- **Resistance against known attacks.** In 2004 and 2005, Wang *et al.*'s attacks on most well-known cryptographic hash algorithms such as MD5 and SHA-1 had a big impact on the field of hash functions. When designing the compression functions of the ARIRANG family, we gave a high priority to the resistance against known attacks including the Wang *et al.*'s attacks and other attacks such as local-collision-finding, near-collision-finding, pseudo-collision-finding, and fixed-point-finding attacks.

- **Formal security proofs of the domain extension of the ARIRANG family.** Recent researches revealed that the domain extension of almost all dedicated hash functions, Merkle-Damgård Strengthening, has weaknesses against multicollision, length-extension and second-preimage-finding attacks, etc. In this document, it will be shown that the domain extension of ARIRANG family is PI-preserving, CR-preserving and PRO-preserving (or indifferentiably secure), where PI denotes Preimage-resistant, CR denotes collision-Resistant and PRO denotes PseudoRandom Oracle. The domain extension of ARIRANG will be also shown to be secure against all known second-preimage-finding attacks. Furthermore, the security proofs of cryptographic applications such as HMAC-ARIRANG and randomized hashing with ARIRANG will be provided in this document.

- **Implementation efficiency.** Implementation efficiency is one of the most important factors together with security. A hash algorithm is used for secure communications, but needs the resources such as time, power, storage and area. If possible, these resources must be minimized when a hash algorithm is used to construct a secure communication channel in practical applications. We tried to maximize implementation efficiency of ARIRANG in various environments.

- **Design logics of a block cipher.** A block cipher is a mature field of research in cryptography; many cryptographers have researched and verified the design logics of a block cipher. The S-box and MDS-code, which are the optimal components implementing confusion and diffusion, are widely adopted in many modern block ciphers. In the design of ARIRANG, we adopted S-box and MDS-code as its basic components.

- **Compatibility with applications of the original SHA-2 family**: The SHA-2 family has been widely used for many "auxiliary" applications, including hash-based message authentication codes, pseudo random number generators, and key derivation functions (FIPS 186-2, FIPS 198, SP 800-56A, and SP 800-90). If a hash algorithm proposed as a SHA-3 candidate is remarkably different from the structure of the original SHA-2 family, it cannot flexibly be used in existing applications of the original SHA-2 family. Our design goal is to make ARIRANG compatible with applications of the original SHA-2 family.

- **Design simplicity**: The simple design of ARIRANG enables readers to understand it easily; we expect that it will take much less time and effort to implement ARIRANG and verify its security, and it will be easy to adapt for new environments.

4 Specification

In this section, we describe the specification of the ARIRANG family. The ARIRANG family includes four hash algorithms ARIRANG-224, ARIRANG-256, ARIRANG-384, and ARIRANG-512. Each algorithm has a message digest bit-size corresponding to its number. The four algorithms differ in terms of the size of the blocks and words of data that are used during hashing. Table 1 presents the basic properties of the ARIRANG family.

Table 1: Properties of the ARIRANG family.

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)
ARIRANG-224	$< 2^{64}$	512	32	224
ARIRANG-256	$< 2^{64}$	512	32	256
ARIRANG-384	$< 2^{128}$	1024	64	384
ARIRANG-512	$< 2^{128}$	1024	64	512

4.1 ARIRANG-256

The structure of ARIRANG-256 can be described in two phases: 1) preprocessing phase 2) message digest phase. In the preprocessing phase, it prepares the message blocks M^1, M^2,$\cdots$, M^N, the counter values Ctr^i, and the initial value H^0, which are loaded to the message digest phase. In the message digest phase, a compression function iteratively processes each message block to compute the hash value H^N.

```
ARIRANG256($M$)
{
    ARIRANG256_Preprocessing();

    For $i = 1$ to $N$
        ARIRANG256_CounterAddition($H^{i-1}$, $Ctr^i$);
        ARIRANG256_CompressionFunction($H^{i-1}$, $M^i$, $H^i$);
}
```

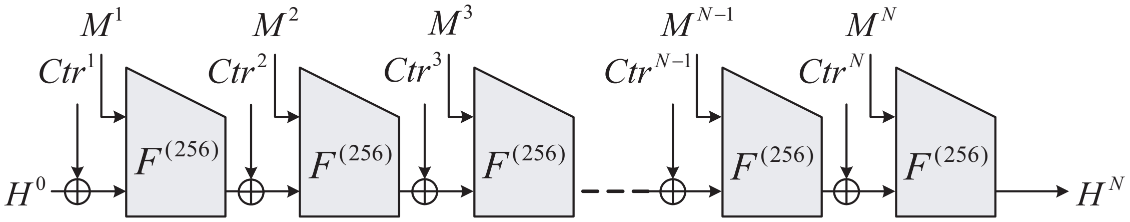

Figure 1: The domain extension of ARIRANG-256.

4.1.1 ARIRANG256_Preprocessing

The preprocessing phase consists of four steps: padding the message M, parsing the padded message into message blocks, setting the counter values Ctr^i, and setting the initial value H^0.

Padding the message. The l-bit message is padded by the following padding rule so that the length of the padded message is ensured to be a multiple of 512 bits.

1. The bit “1” is appended to the end of the message, followed by k zero bits, where k is the smallest non-negative integer such that $l + 1 + k \equiv 448 \bmod 512$.

2. The 64-bit binary representation of the integer l is again appended.

Parsing the padded message. After a message has been padded, it is parsed into N 512-bit blocks M^1, M^2,$\cdots$, M^N. Recall that message block i is denoted by the concatenation of sixteen words, that is, $M_0^i || \cdots || M_{15}^i$.

Setting the counter values (Ctr^i). The purpose of setting the counter value Ctr^i is to ensure that each compression function of ARIRANG used to produce a message digest is mutually different.

For ARIRANG-256, Ctr^i are represented as the concatenation of two 32-bit words: Ctr_0^i is the left-most 32-bit word of Ctr^i, and Ctr_1^i is the right-most 32-bit word of Ctr^i. Suppose that the number of blocks in the padded message is N. Then

$$\begin{aligned} Ctr^1 &= 0, \\ Ctr^2 &= 1, \\ Ctr^3 &= 2, \\ &\cdots \\ Ctr^{N-2} &= N-3, \\ Ctr^{N-1} &= N-2, \\ Ctr^N &= P, \end{aligned}$$

where P=`0xB7E151628AED2A6A` obtained by taking the first 64 bits of the fractional parts of a normal number e. Since possible message sizes are less than 2^{64}, N is less than $2^{55}(= 2^{64}/512)$. We choose P which is not equal to any counter number since P is greater than 2^{55}.

Setting the initial value (H^0). The initial value H^0 consists of the following eight 32-bit words in hexadecimal:

$$\begin{aligned} H_0^0 &= \texttt{0x6A09E667,} \\ H_1^0 &= \texttt{0xBB67AE85,} \\ H_2^0 &= \texttt{0x3C6EF372,} \\ H_3^0 &= \texttt{0xA54FF53A,} \\ H_4^0 &= \texttt{0x510E527F,} \\ H_5^0 &= \texttt{0x9B05688C,} \\ H_6^0 &= \texttt{0x1F83D9AB,} \\ H_7^0 &= \texttt{0x5BE0CD19.} \end{aligned}$$

These words are obtained by taking the first 32 bits of the fractional parts of the square roots for the first eight prime numbers each.

4.1.2 ARIRANG-256 Constants

The ARIRANG-256 constants $K_0^{(256)}, \cdots, K_{15}^{(256)}$ come from the following sequence of 32-bit words in hexadecimal: $K_0^{(256)}$ is set to be `0x517CC1B7` which is the first 32 bits of the fractional parts of π^{-1}. It is also used for the initial state (s_3, s_2, s_1, s_0) of 4-byte LFSR S to generate the next 15 32-bit constants. The connection polynomial of LFSR S is $X^4 + X + 1 \in GF(2^8)[X]$, and LFSR S is performed in the algorithm ConstantGeneration as follows:

```
ConstantGeneration
{
    $s_0$ = 0xB7; $s_1$ = 0xC1; $s_2$ = 0x7C; $s_3$ = 0x51;
    $K_0^{(256)} = s_3 \| s_2 \| s_1 \| s_0$;
    for $t$=1 to 15 {
        $s_{t+3} = s_t \oplus s_{t-1}$;
        $K_t^{(256)} = s_{t+3} \| s_{t+2} \| s_{t+1} \| s_t$;
}
```

Since $X^4 + X + 1$ is a primitive polynomial in $GF(2^8)[X]$, the period of S is $2^4 - 1 = 15$ and thus $K_0^{(256)} = K_{15}^{(256)}$. In hexadecimal, the sequence of the constant words $K_0^{(256)}, \cdots, K_{15}^{(256)}$ is given by

```
0x517CC1B7  0x76517CC1  0xBD76517C  0x2DBD7651
0x272DBD76  0xCB272DBD  0x90CB272D  0x0A90CB27
0xEC0A90CB  0x5BEC0A90  0x9A5BEC0A  0xE69A5BEC
0xB7E69A5B  0xC1B7E69A  0x7CC1B7E6  0x517CC1B7
```

4.1.3 ARIRANG256_CounterAddition

ARIRANG256_CounterAddition updates the $(i-1)^{th}$ intermediate hash value H^{i-1}, with the counter value Ctr^i.

```
ARIRANG256_CounterAddition($H^{i-1}$, $Ctr^i$)
{
    $H_0^{i-1} = H_0^{i-1} \oplus Ctr_0^i$;
    $H_4^{i-1} = H_4^{i-1} \oplus Ctr_1^i$;
}
```

4.1.4 ARIRANG256_CompressionFunction

After the preprocessing phase is completed, message blocks M^1, M^2,..., M^N are processed in order by the steps described algorithmically in Figure 2. ARIRANG256_CompressionFunction runs 4 rounds which contain 10 steps each, and thus 40 steps in total. It is made up of four procedures: ARIRANG256_RegisterInitialize, ARIRANG256_StepFunction, ARIRANG256_Feedforward, and ARIRANG256_MessageSchedule.

```
ARIRANG256_CompressionFunction($H^{i-1}$, $M^i$, $H^i$)
{
    ARIRANG256_MessageSchedule($M^i$, $W_t$);
    ARIRANG256_RegisterInitialize($H^{i-1}$, $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$);

    for $t = 0$ to 19
        ARIRANG256_StepFunction($W_{\sigma(2t)}$, $W_{\sigma(2t+1)}$, $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$);

    ARIRANG256_Feedforward$_1$($H^{i-1}$, $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$);

    for $t = 20$ to 39
        ARIRANG256_StepFunction($W_{\sigma(2t)}$, $W_{\sigma(2t+1)}$, $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$);

    ARIRANG256_Feedforward$_2$($H^i$, $H^{i-1}$, $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$);
}
```

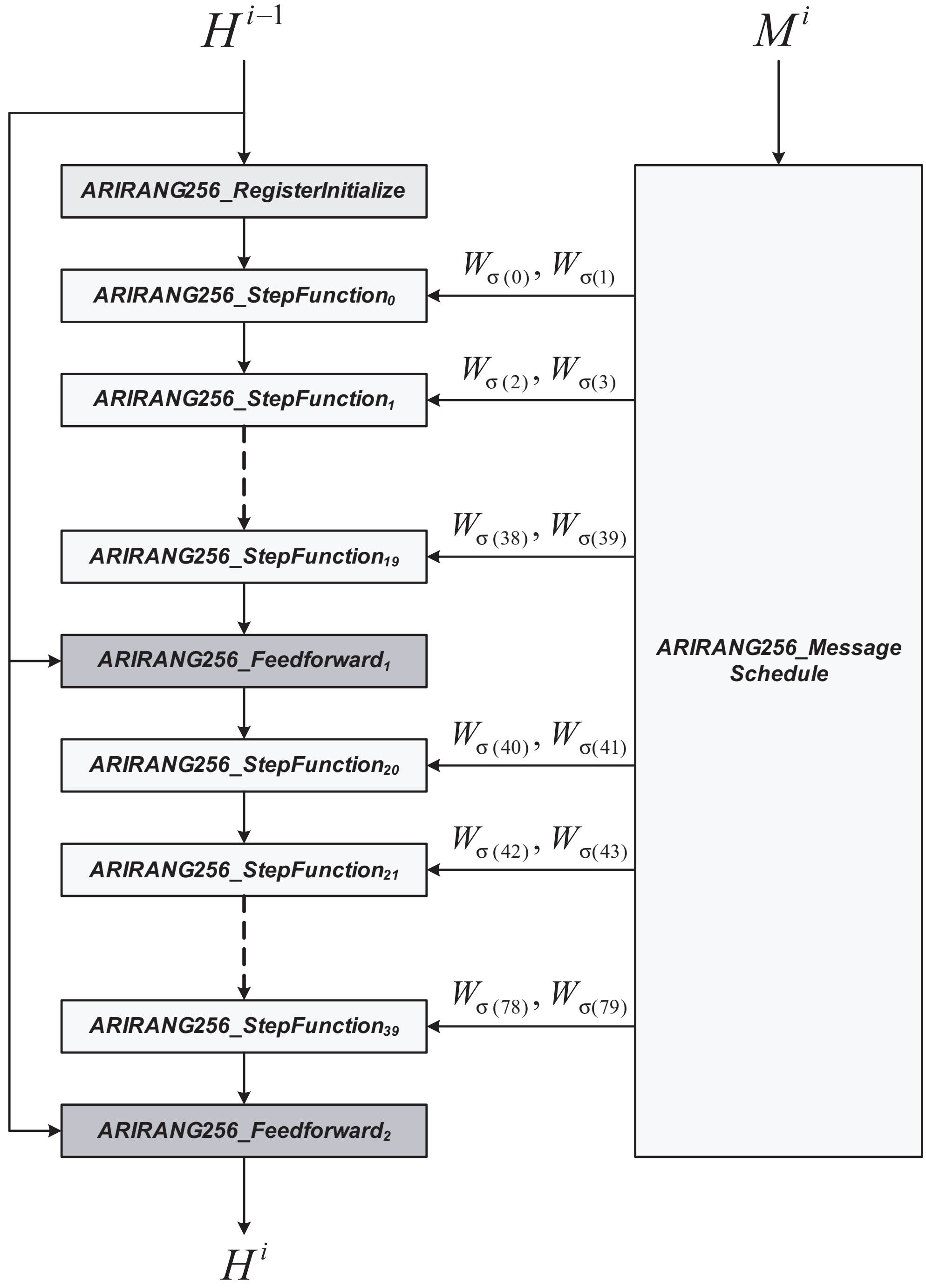
H^{i-1}
M^i
ARIRANG256_RegisterInitialize
ARIRANG256_StepFunction$_0$
$W_{\sigma(0)}, W_{\sigma(1)}$
ARIRANG256_StepFunction$_1$
$W_{\sigma(2)}, W_{\sigma(3)}$
ARIRANG256_StepFunction$_{19}$
$W_{\sigma(38)}, W_{\sigma(39)}$
ARIRANG256_Feedforward$_1$
ARIRANG256_StepFunction$_{20}$
$W_{\sigma(40)}, W_{\sigma(41)}$
ARIRANG256_StepFunction$_{21}$
$W_{\sigma(42)}, W_{\sigma(43)}$
ARIRANG256_StepFunction$_{39}$
$W_{\sigma(78)}, W_{\sigma(79)}$
ARIRANG256_Feedforward$_2$
ARIRANG256_Message Schedule
H^i

Figure 2: ARIRANG256_CompressionFunction.

ARIRANG256_RegisterInitialize. ARIRANG256_RegisterInitialize initializes the eight working variables a, b, c, d, e, f, g, and h with the $(i-1)^{th}$ intermediate hash value H^{i-1}.

```
ARIRANG256_RegisterInitialize($H^{i-1}$, $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$)
{
    $a = H_0^{i-1}$; $b = H_1^{i-1}$; $c = H_2^{i-1}$; $d = H_3^{i-1}$;
    $e = H_4^{i-1}$; $f = H_5^{i-1}$; $g = H_6^{i-1}$; $h = H_7^{i-1}$;
}
```

ARIRANG256_StepFunction. ARIRANG256_StepFunction updates the eight working variables a, b, c, d, e, f, g, and h with the function $G^{(256)}$ and the two input message words $W_{\sigma(2t)}$, $W_{\sigma(2t+1)}$. It is schematically depicted in Fig. 3; the function $G^{(256)}$ will be explained later in detail.

```
ARIRANG256_StepFunction($W_{\sigma(2t)}$, $W_{\sigma(2t+1)}$, $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$)
{
    $T_1 = G^{(256)}(a \oplus W_{\sigma(2t)})$;
    $b = a \oplus W_{\sigma(2t)}$;
    $c = b \oplus T_1$;
    $d = c \oplus (T_1 \lll 13)$;
    $e = d \oplus (T_1 \lll 23)$;
    $T_2 = G^{(256)}(e \oplus W_{\sigma(2t+1)})$;
    $f = e \oplus W_{\sigma(2t+1)}$;
    $g = f \oplus T_2$;
    $h = g \oplus (T_2 \lll 29)$;
    $a = h \oplus (T_2 \lll 7)$;
}
```

ARIRANG256_Feedforward. ARIRANG256_Feedforward$_1$ updates the eight working variables a, b, c, d, e, f, g, and h with H^{i-1} after the first 20 steps are completed.

```
ARIRANG256_Feedforward$_1$($H^{i-1}$, $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$)
{
    $a = a \oplus H_0^{i-1}$; $b = b \oplus H_1^{i-1}$; $c = c \oplus H_2^{i-1}$; $d = d \oplus H_3^{i-1}$;
    $e = e \oplus H_4^{i-1}$; $f = f \oplus H_5^{i-1}$; $g = g \oplus H_6^{i-1}$; $h = h \oplus H_7^{i-1}$;
}
```

ARIRANG256_Feedforward$_2$ computes the i^{th} intermediate hash value H^i.

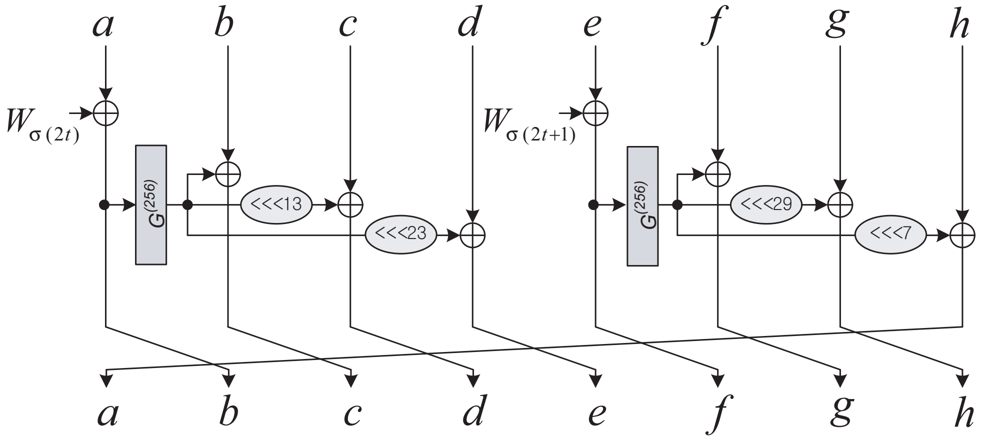

Figure 3: ARIRANG256_StepFunction.

```
ARIRANG256_Feedforward$_2$($H^i$, $H^{i-1}$, $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$)
{
        $H_0^i = a \oplus H_0^{i-1}$; $H_1^i = b \oplus H_1^{i-1}$; $H_2^i = c \oplus H_2^{i-1}$; $H_3^i = d \oplus H_3^{i-1}$;
        $H_4^i = e \oplus H_4^{i-1}$; $H_5^i = f \oplus H_5^{i-1}$; $H_6^i = g \oplus H_6^{i-1}$; $H_7^i = h \oplus H_7^{i-1}$;
}
```

ARIRANG256_MessageSchedule. ARIRANG256_MessageSchedule consists of a message word expansion and a message word ordering.

ARIRANG256_MessageSchedule only generates additional 16 extra words W_{16}, W_{17}, $\cdots$, W_{31} from the 16 input message words M_0^i, M_1^i, $\cdots$, M_{15}^i, and 16 constant values $K_0^{(256)}$, $K_1^{(256)}$, $\cdots$, $K_{15}^{(256)}$ as follows.

```
ARIRANG256_MessageSchedule($M^i$, $W_t$)
{
        For $t = 0$ to 15
                $W_t = M_t^i$;

        $W_{16} = (W_9 \oplus W_{11} \oplus W_{13} \oplus W_{15} \oplus K_0^{(256)}) \lll 5$;
        $W_{17} = (W_8 \oplus W_{10} \oplus W_{12} \oplus W_{14} \oplus K_1^{(256)}) \lll 11$;
        $W_{18} = (W_1 \oplus W_3 \oplus W_5 \oplus W_7 \oplus K_2^{(256)}) \lll 19$;
        $W_{19} = (W_0 \oplus W_2 \oplus W_4 \oplus W_6 \oplus K_3^{(256)}) \lll 31$;

        $W_{20} = (W_{14} \oplus W_4 \oplus W_{10} \oplus W_0 \oplus K_4^{(256)}) \lll 5$;
        $W_{21} = (W_{11} \oplus W_1 \oplus W_7 \oplus W_{13} \oplus K_5^{(256)}) \lll 11$;
        $W_{22} = (W_6 \oplus W_{12} \oplus W_2 \oplus W_8 \oplus K_6^{(256)}) \lll 19$;
        $W_{23} = (W_3 \oplus W_9 \oplus W_{15} \oplus W_5 \oplus K_7^{(256)}) \lll 31$;

        $W_{24} = (W_{13} \oplus W_{15} \oplus W_1 \oplus W_3 \oplus K_8^{(256)}) \lll 5$;
        $W_{25} = (W_4 \oplus W_6 \oplus W_8 \oplus W_{10} \oplus K_9^{(256)}) \lll 11$;
        $W_{26} = (W_5 \oplus W_7 \oplus W_9 \oplus W_{11} \oplus K_{10}^{(256)}) \lll 19$;
        $W_{27} = (W_{12} \oplus W_{14} \oplus W_0 \oplus W_2 \oplus K_{11}^{(256)}) \lll 31$;

        $W_{28} = (W_{10} \oplus W_0 \oplus W_6 \oplus W_{12} \oplus K_{12}^{(256)}) \lll 5$;
        $W_{29} = (W_{15} \oplus W_5 \oplus W_{11} \oplus W_1 \oplus K_{13}^{(256)}) \lll 11$;
        $W_{30} = (W_2 \oplus W_8 \oplus W_{14} \oplus W_4 \oplus K_{14}^{(256)}) \lll 19$;
        $W_{31} = (W_7 \oplus W_{13} \oplus W_3 \oplus W_9 \oplus K_{15}^{(256)}) \lll 31$;
}
```

For the ordering of the expanded message words W_t, the permutation σ of Table 2 is used.

4.1.5 Hash Value H^N of ARIRANG-256

The resulting 256-bit message digest of the message $M = (M^1, M^2, \cdots, M^N)$ is

$$H^N = H_0^N \| H_1^N \| H_2^N \| H_3^N \| H_4^N \| H_5^N \| H_6^N \| H_7^N .$$

4.1.6 The Function $G^{(256)}$

The function $G^{(256)}$ is composed of two different transformations: the SubBytes transformations and the $\mathrm{MDS}_{4\times 4}$ transformations.

Table 2: Message order.

		1round		2round		3round		4round	
s		$\sigma(s)$		$\sigma(s+20)$		$\sigma(s+40)$		$\sigma(s+60)$	
0	1	16	17	20	21	24	25	28	29
2	3	0	1	3	6	12	5	7	2
4	5	2	3	9	12	14	7	13	8
6	7	4	5	15	2	0	9	3	14
8	9	6	7	5	8	2	11	9	4
10	11	18	19	22	23	26	27	30	31
12	13	8	9	11	14	4	13	15	10
14	15	10	11	1	4	6	15	5	0
16	17	12	13	7	10	8	1	11	6
18	19	14	15	13	0	10	3	1	12

```
$G^{(256)}(X)$
{
    SubBytes($X$, $Y$);
    MDS$_{4\times4}$($Y$, $Z$);
    Return $Z$;
}
```

SubBytes transformation. The SubBytes transformation is a substitution function which satisfies a confusion property on each byte. It is defined as the SubBytes transformation of AES. The input word X is split into four bytes; each byte is run through a bijective S-box.

MDS$_{4\times4}$ transformation. The MDS$_{4\times4}$ transformation is a permutation function which mixes all bytes of the input word. It is defined as the MDS$_{4\times4}$ transformation of AES. Each byte of the input word is considered a polynomial over GF(2^8) and multiplied modulo x^4+1 with a fixed polynomial $a(x)$, given by

$$a(x) = \{\texttt{03}\}x^3 + \{\texttt{01}\}x^2 + \{\texttt{01}\}x + \{\texttt{02}\}.$$

This can be written as a matrix multiplication. Let $z(x) = a(x) \otimes y(x)$:

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

As a result of this multiplication, the four bytes are replaced by the following:

$$z_0 = (\{\texttt{02}\} \bullet y_0) \oplus (\{\texttt{03}\} \bullet y_1) \oplus y_2 \oplus y_3$$

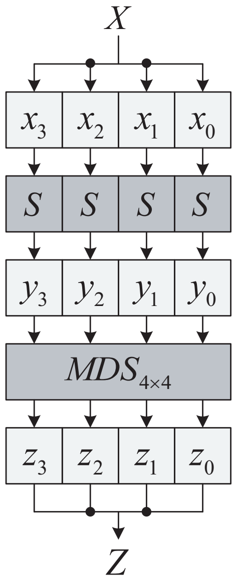

Figure 4: The function $G^{(256)}$ of ARIRANG-256.

Table 3: S-box of ARIRANG : For example, S-box(31)=C7.

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

$$
\begin{aligned}
z_1 &= y_0 \oplus (\{02\} \bullet y_1) \oplus (\{03\} \bullet y_2) \oplus y_3 \\
z_2 &= y_0 \oplus y_1 \oplus (\{02\} \bullet y_2) \oplus (\{03\} \bullet y_3) \\
z_3 &= (\{03\} \bullet y_0) \oplus y_1 \oplus y_2 \oplus (\{02\} \bullet y_3)
\end{aligned}
$$

4.2 ARIRANG-224

ARIRANG-224 is used to hash a message M with a length of l bits, where $0 \leq l < 2^{64}$. ARIRANG-224 is defined in the same manner as ARIRANG-256, with the following two exceptions:

4.2.1 Setting the Initial Value (H^0)

For ARIRANG-224, the initial value H^0 shall consist of the following eight 32-bit words, in hexadecimal:

$$
\begin{aligned}
H_0^0 &= \texttt{0xCBBB9D5D}, \\
H_1^0 &= \texttt{0x629A292A}, \\
H_2^0 &= \texttt{0x9159015A}, \\
H_3^0 &= \texttt{0x152FECD8}, \\
H_4^0 &= \texttt{0x67332667}, \\
H_5^0 &= \texttt{0x8EB44A87}, \\
H_6^0 &= \texttt{0xDB0C2E0D}, \\
H_7^0 &= \texttt{0x47B5481D}.
\end{aligned}
$$

These words are obtained by taking the first 32 bits of the fractional parts of the square roots for the ninth to the sixteenth prime numbers each.

4.2.2 Hash Value H^N of ARIRANG-224

The 224-bit message digest is obtained by truncating the final hash value H^N to its leftmost 224 bits:

$$H_0^N \| H_1^N \| H_2^N \| H_3^N \| H_4^N \| H_5^N \| H_6^N.$$

4.3 ARIRANG-512

The structure of ARIRANG-512 is similar to that of ARIRANG-256. ARIRANG-512 doubles the word size of ARIRANG-256, and thus the initial value and the constant values are changed. The structure of ARIRANG-512 is described as follows.

```
ARIRANG512($M, H^0$)
{
    ARIRANG512_Preprocessing()

    for $i = 1$ to $N$
        ARIRANG512_CounterAddition($H^{i-1}$, $Ctr^i$);
        ARIRANG512_CompressionFunction($H^{i-1}, M^i, H^i$);
}
```

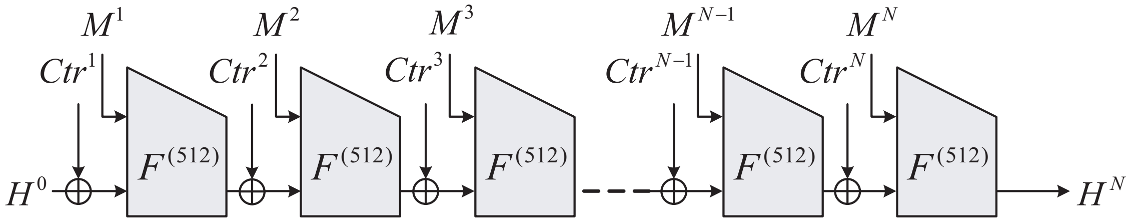

Figure 5: The domain extension of ARIRANG-512.

4.3.1 ARIRANG512_Preprocessing

The preprocessing consists of four steps: padding the message M, parsing the padded message into message blocks, setting the counter values Ctr^i, and setting the initial value H^0.

Padding the message. The l-bit message is padded by the following padding rule so that the length of the padded message is ensured to be a multiple of 1024 bits.

1. The bit "1" is appended to the end of the message, followed by k zero bits, where k is the smallest non-negative integer such that $l + 1 + k \equiv 896 \bmod 1024$.
2. The 128-bit binary representation of the integer l is again appended.

Parsing the padded message. After a message has been padded, it is parsed into N 1024-bit blocks M^1, M^2,$\cdots$, M^N. Recall that message block i is denoted by the concatenation of sixteen 64-bit words, that is, $M^i_0 || \cdots || M^i_{15}$.

Setting the counter values (Ctr^i). The purpose of setting the counter value Ctr^i is to ensure that each compression function of ARIRANG used to produce a message digest is mutually different.

For ARIRANG-512, Ctr^i are the concatenation of two 64-bit words each. Ctr^i_0 is the left-most 64-bit word of Ctr^i, and Ctr^i_1 is the right-most 64-bit word of Ctr^i. Suppose that the number of blocks in the padded message is N. Then

$$\begin{aligned} Ctr^1 &= 0, \\ Ctr^2 &= 1, \\ Ctr^3 &= 2, \\ &\cdots \\ Ctr^{N-2} &= N-3, \\ Ctr^{N-1} &= N-2, \\ Ctr^N &= P, \end{aligned}$$

P=`0xB7E151628AED2A6ABF7158809CF4F3C7` obtained by taking the first 128 bits of the fractional parts of normal number e. Since possible message sizes are less than 2^{128}, N is less than

$2^{118}(=2^{128}/1024)$. We choose P which is not equal to any counter number since P is greater than 2^{118}.

Setting the initial hash value (H^0). The initial value H^0 consists of the following eight 64-bit words in hexadecimal:

$$\begin{aligned}
H_0^0 &= \texttt{0x6A09E667F3BCC908},\\
H_1^0 &= \texttt{0xBB67AE8584CAA73B},\\
H_2^0 &= \texttt{0x3C6EF372FE94F82B},\\
H_3^0 &= \texttt{0xA54FF53A5F1D36F1},\\
H_4^0 &= \texttt{0x510E527FADE682D1},\\
H_5^0 &= \texttt{0x9B05688C2B3E6C1F},\\
H_6^0 &= \texttt{0x1F83D9ABFB41BD6B},\\
H_7^0 &= \texttt{0x5BE0CD19137E2179}.
\end{aligned}$$

These words are obtained by taking the first 64 bits of the fractional parts of the square roots for the first eight prime numbers each.

4.3.2 ARIRANG512 Constants

The ARIRANG-512 constants $K_0^{(512)}, \cdots, K_{15}^{(512)}$ come from the following sequence of 64-bit words in hex: $K_0^{(512)}$ is set to be `0x517CC1B727220A94` which is the first 64 bits of the fractional parts of π^{-1}. It is also used for the initial state (s_3, s_2, s_1, s_0) of 4-word LFSR S to generate the next 15 64-bit constants, where a word is 16 bits. The connection polynomial of LFSR S is $X^4 + X + 1 \in GF(2^{16})[X]$, and LFSR S is performed in the algorithm ConstantGeneration as follows:

```
ConstantGeneration
{
    $s_0 =$ 0x0A94; $s_1 =$ 0x2722; $s_2 =$ 0xC1B7; $s_3 =$ 0x517C;
    $K_0^{(512)} = s_3 \| s_2 \| s_1 \| s_0$;
    for $t$=1 to 15
        $s_{t+3} = s_t \oplus s_{t-1}$;
        $K_t^{(512)} = s_{t+3} \| s_{t+2} \| s_{t+1} \| s_t$;
}
```

Since $X^4 + X + 1$ is a primitive polynomial in $GF(2^{16})[X]$, the period of S is $2^4 - 1 = 15$ and thus $K_0^{(512)} = K_{15}^{(512)}$. In hex, a sequence of constants words $K_0^{(512)}, \cdots, K_{15}^{(512)}$ is given by

```
0x517CC1B727220A94  0x2DB6517CC1B72722  0xE6952DB6517CC1B7
0x90CBE6952DB6517C  0x7CCA90CBE6952DB6  0xCB237CCA90CBE695
0x765ECB237CCA90CB  0xEC01765ECB237CCA  0xB7E9EC01765ECB23
0xBD7DB7E9EC01765E  0x9A5FBD7DB7E9EC01  0x5BE89A5FBD7DB7E9
0x0A945BE89A5FBD7D  0x27220A945BE89A5F  0xC1B727220A945BE8
0x517CC1B727220A94
```

4.3.3 ARIRANG512_CounterAddition

ARIRANG512_CounterAddition updates the $(i-1)^{th}$ intermediate hash value H^{i-1} with the Counter Value Ctr^i.

```
ARIRANG512_CounterAddition($H^{i-1}$, $Ctr^i$)
{
    $H_0^{i-1} = H_0^{i-1} \oplus Ctr_0^i$;
    $H_4^{i-1} = H_4^{i-1} \oplus Ctr_1^i$;
}
```

4.3.4 ARIRANG512_CompressionFunction

After the preprocessing phase is completed, message blocks M^1, M^2,..., M^N are processed in order by the steps described algorithmically in Figure 6. ARIRANG512_CompressionFunction runs 4 rounds which contain 10 steps each, and thus 40 steps in total. It is made up of four procedures: ARIRANG512_RegisterInitialize, ARIRANG512_StepFunction, ARIRANG512_Feedforward, and ARIRANG512_MessageSchedule.

```
ARIRANG512_CompressionFunction($H^{i-1}$, $M^i$, $H^i$)
{
    ARIRANG512_MessageSchedule($M^i$, $W$);
    ARIRANG512_RegisterInitialize($H^{i-1}$, $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$);

    for $t = 0$ to 19
        ARIRANG512_StepFunction($W_{\sigma(2t)}$, $W_{\sigma(2t+1)}$, $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$);

    ARIRANG512_Feedforward$_1$($H^{i-1}$, $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$);

    for $t = 20$ to 39
        ARIRANG512_StepFunction($W_{\sigma(2t)}$, $W_{\sigma(2t+1)}$, $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$);

    ARIRANG512_Feedforward$_2$($H^i$, $H^{i-1}$, $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$);
}
```

ARIRANG512_RegisterInitialize. ARIRANG512_RegisterInitialize initializes the eight working variables a, b, c, d, e, f, g, and h with the $(i-1)^{th}$ intermediate hash value H^{i-1}.

```
ARIRANG512_RegisterInitialize($H^{i-1}$, $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$)
{
    $a = H_0^{i-1}$; $b = H_1^{i-1}$; $c = H_2^{i-1}$; $d = H_3^{i-1}$;
    $e = H_4^{i-1}$; $f = H_5^{i-1}$; $g = H_6^{i-1}$; $h = H_7^{i-1}$;
}
```

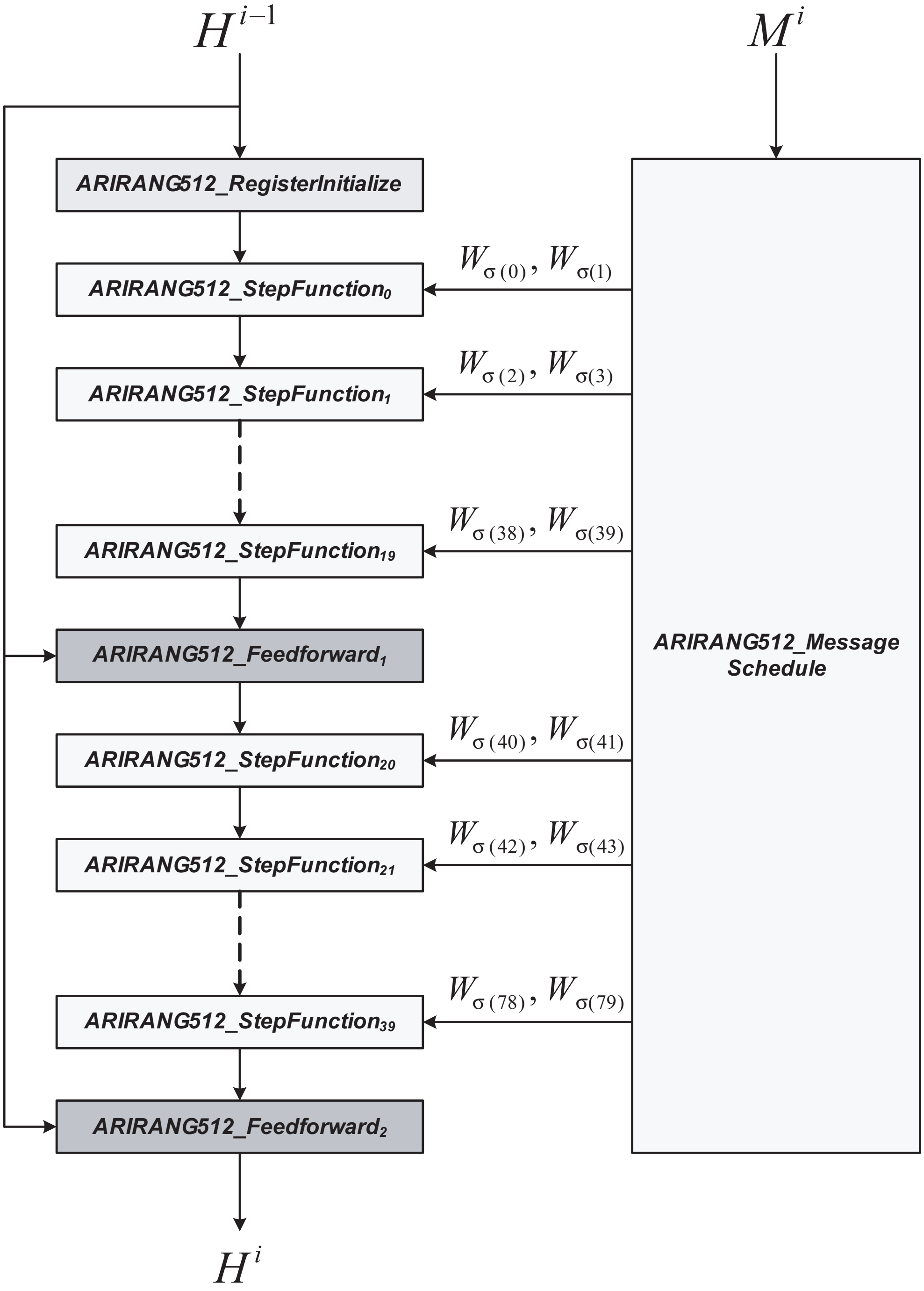
H^{i-1}
M^{i}
ARIRANG512_RegisterInitialize
ARIRANG512_StepFunction$_0$
$W_{\sigma(0)}, W_{\sigma(1)}$
ARIRANG512_StepFunction$_1$
$W_{\sigma(2)}, W_{\sigma(3)}$
ARIRANG512_StepFunction$_{19}$
$W_{\sigma(38)}, W_{\sigma(39)}$
ARIRANG512_Feedforward$_1$
ARIRANG512_Message Schedule
ARIRANG512_StepFunction$_{20}$
$W_{\sigma(40)}, W_{\sigma(41)}$
ARIRANG512_StepFunction$_{21}$
$W_{\sigma(42)}, W_{\sigma(43)}$
ARIRANG512_StepFunction$_{39}$
$W_{\sigma(78)}, W_{\sigma(79)}$
ARIRANG512_Feedforward$_2$
H^{i}

Figure 6: ARIRANG512_CompressionFunction.

ARIRANG512_StepFunction. ARIRANG512_StepFunction updates the eight working variables a, b, c, d, e, f, g, and h with the function $G^{(512)}$ and the two input message words $W_{\sigma(2t)}$, $W_{\sigma(2t+1)}$. It is schematically depicted in Fig. 7; the function $G^{(512)}$ will be explained later in detail.

```
ARIRANG512_StepFunction($W_{\sigma(2t)}$, $W_{\sigma(2t+1)}$, $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$)
{
        $T_1 = G^{(512)}(a \oplus W_{\sigma(2t)})$;
        $b = a \oplus W_{\sigma(2t)}$;
        $c = b \oplus T_1$;
        $d = c \oplus (T_1 \lll 29)$;
        $e = d \oplus (T_1 \lll 41)$;
        $T_2 = G^{(512)}(e \oplus W_{\sigma(2t+1)})$;
        $f = e \oplus W_{\sigma(2t+1)}$;
        $g = f \oplus T_2$;
        $h = g \oplus (T_2 \lll 53)$;
        $a = h \oplus (T_2 \lll 13)$;
}
```

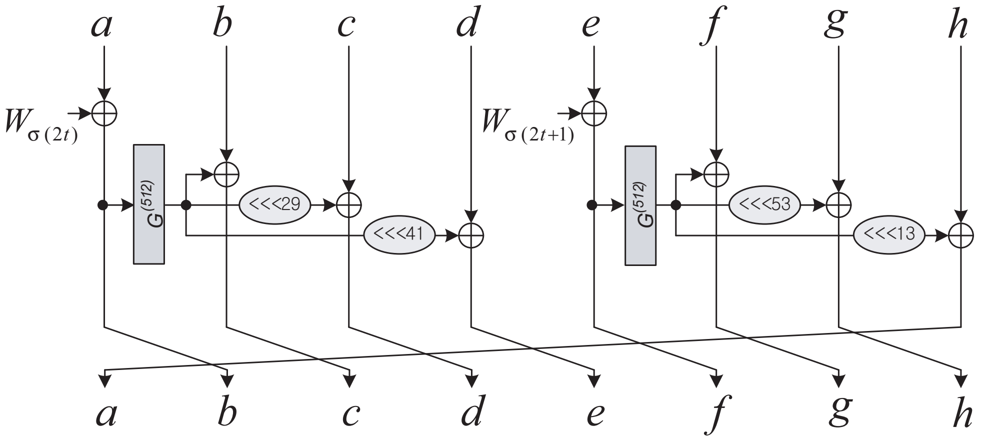

Figure 7: ARIRANG512_StepFunction.

ARIRANG512_Feedforward. ARIRANG512_Feedforward$_1$ updates the eight working variables a, b, c, d, e, f, g, and h with H^{i-1} after the first 20 steps are completed.

```
ARIRANG512_Feedforward$_1$($H^{i-1}$, $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$)
{
        $a = a \oplus H_0^{i-1}$; $b = b \oplus H_1^{i-1}$; $c = c \oplus H_2^{i-1}$; $d = d \oplus H_3^{i-1}$;
        $e = e \oplus H_4^{i-1}$; $f = f \oplus H_5^{i-1}$; $g = g \oplus H_6^{i-1}$; $h = h \oplus H_7^{i-1}$;
}
```

ARIRANG512_Feedforward$_2$ computes the i^{th} intermediate hash value H^i.

```
ARIRANG512_Feedforward$_2$($H^i$, $H^{i-1}$, $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$)
{
    $H_0^i = a \oplus H_0^{i-1}$; $H_1^i = b \oplus H_1^{i-1}$; $H_2^i = c \oplus H_2^{i-1}$; $H_3^i = d \oplus H_3^{i-1}$;
    $H_4^i = e \oplus H_4^{i-1}$; $H_5^i = f \oplus H_5^{i-1}$; $H_6^i = g \oplus H_6^{i-1}$; $H_7^i = h \oplus H_7^{i-1}$;
}
```

ARIRANG512_MessageSchedule. ARIRANG512_MessageSchedule consists of a message word expansion and a message word ordering.

ARIRANG512_MessageSchedule only generates additional 16 extra words W_{16}, W_{17}, $\cdots$, W_{31} from the 16 input message words M_0^i, M_1^i, $\cdots$, M_{15}^i, and 16 constant values $K_0^{(512)}$, $K_1^{(512)}$, $\cdots$, $K_{15}^{(512)}$ as follows.

```
ARIRANG512_MessageSchedule($M^i$, $W$)
{
    For $t = 0$ to 15
        $W_t = M_t^i$;

    $W_{16} = (W_9 \oplus W_{11} \oplus W_{13} \oplus W_{15} \oplus K_0^{(512)}) \lll 11$;
    $W_{17} = (W_8 \oplus W_{10} \oplus W_{12} \oplus W_{14} \oplus K_1^{(512)}) \lll 23$;
    $W_{18} = (W_1 \oplus W_3 \oplus W_5 \oplus W_7 \oplus K_2^{(512)}) \lll 37$;
    $W_{19} = (W_0 \oplus W_2 \oplus W_4 \oplus W_6 \oplus K_3^{(512)}) \lll 59$;

    $W_{20} = (W_{14} \oplus W_4 \oplus W_{10} \oplus W_0 \oplus K_4^{(512)}) \lll 11$;
    $W_{21} = (W_{11} \oplus W_1 \oplus W_7 \oplus W_{13} \oplus K_5^{(512)}) \lll 23$;
    $W_{22} = (W_6 \oplus W_{12} \oplus W_2 \oplus W_8 \oplus K_6^{(512)}) \lll 37$;
    $W_{23} = (W_3 \oplus W_9 \oplus W_{15} \oplus W_5 \oplus K_7^{(512)}) \lll 59$;

    $W_{24} = (W_{13} \oplus W_{15} \oplus W_1 \oplus W_3 \oplus K_8^{(512)}) \lll 11$;
    $W_{25} = (W_4 \oplus W_6 \oplus W_8 \oplus W_{10} \oplus K_9^{(512)}) \lll 23$;
    $W_{26} = (W_5 \oplus W_7 \oplus W_9 \oplus W_{11} \oplus K_{10}^{(512)}) \lll 37$;
    $W_{27} = (W_{12} \oplus W_{14} \oplus W_0 \oplus W_2 \oplus K_{11}^{(512)}) \lll 59$;

    $W_{28} = (W_{10} \oplus W_0 \oplus W_6 \oplus W_{12} \oplus K_{12}^{(512)}) \lll 11$;
    $W_{29} = (W_{15} \oplus W_5 \oplus W_{11} \oplus W_1 \oplus K_{13}^{(512)}) \lll 23$;
    $W_{30} = (W_2 \oplus W_8 \oplus W_{14} \oplus W_4 \oplus K_{14}^{(512)}) \lll 37$;
    $W_{31} = (W_7 \oplus W_{13} \oplus W_3 \oplus W_9 \oplus K_{15}^{(512)}) \lll 59$;
}
```

For the ordering of the expanded message words W_t, the permutation σ of Table 2 is used, which is the same as that of ARIRANG-256.

4.3.5 Hash Value H^N of ARIRANG-512

The resulting 512-bit message digest of the message $M = (M^1, M^2, \cdots, M^N)$ is

$$H_0^N \| H_1^N \| H_2^N \| H_3^N \| H_4^N \| H_5^N \| H_6^N \| H_7^N.$$

4.3.6 The Function $G^{(512)}$

The function $G^{(512)}$ is composed of two different transformations: the SubBytes transformations and the $\text{MDS}_{8\times 8}$ transformations.

```
$G^{(512)}(X)$
{
    SubBytes($X$, $Y$);
    MDS$_{8\times8}$($Y$, $Z$);
    Return $Z$;
}
```

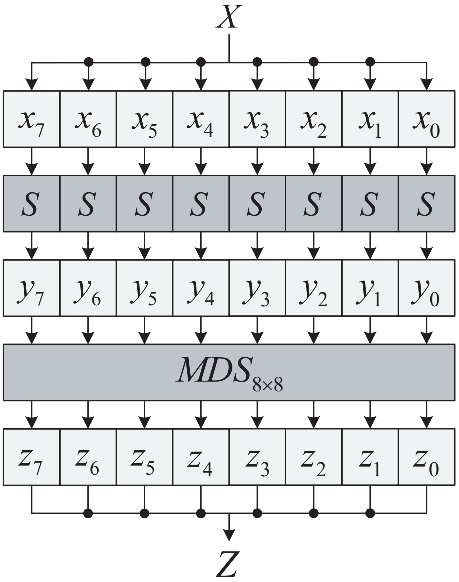

Figure 8: The function $G^{(512)}$ of ARIRANG-512.

SubBytes transformation. The SubBytes transformation is a substitution function which satisfies a confusion property on each byte. It is defined as the SubBytes transformation of AES. The input word X is split into four bytes; each byte is run through a bijective S-box (see Table 3 in Sect. 4.1.6).

MDS$_{8\times8}$ transformation. The MDS$_{8\times8}$ transformation is a permutation function which mixes all bytes of the input word. Each byte of the input word is considered a polynomial over GF(2^8) and multiplied modulo x^8+1 with a fixed polynomial $a(x)$, given by

$$a(x) = \{\texttt{02}\}x^7 + \{\texttt{0A}\}x^6 + \{\texttt{09}\}x^5 + \{\texttt{08}\}x^4 + \{\texttt{01}\}x^3 + \{\texttt{04}\}x^2 + \{\texttt{01}\}x + \{\texttt{01}\}.$$

Let $z(x) = a(x) \otimes y(x)$. This can be written as a matrix multiplication as follows:

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \end{bmatrix} = \begin{bmatrix} \texttt{01} & \texttt{02} & \texttt{0A} & \texttt{09} & \texttt{08} & \texttt{01} & \texttt{04} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{0A} & \texttt{09} & \texttt{08} & \texttt{01} & \texttt{04} \\ \texttt{04} & \texttt{01} & \texttt{01} & \texttt{02} & \texttt{0A} & \texttt{09} & \texttt{08} & \texttt{01} \\ \texttt{01} & \texttt{04} & \texttt{01} & \texttt{01} & \texttt{02} & \texttt{0A} & \texttt{09} & \texttt{08} \\ \texttt{08} & \texttt{01} & \texttt{04} & \texttt{01} & \texttt{01} & \texttt{02} & \texttt{0A} & \texttt{09} \\ \texttt{09} & \texttt{08} & \texttt{01} & \texttt{04} & \texttt{01} & \texttt{01} & \texttt{02} & \texttt{0A} \\ \texttt{0A} & \texttt{09} & \texttt{08} & \texttt{01} & \texttt{04} & \texttt{01} & \texttt{01} & \texttt{02} \\ \texttt{02} & \texttt{0A} & \texttt{09} & \texttt{08} & \texttt{01} & \texttt{04} & \texttt{01} & \texttt{01} \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix}$$

As a result of this multiplication, the eight bytes are replaced by the following:

$$\begin{aligned}
z_0 &= y_0 \oplus (\{\texttt{02}\} \bullet y_1) \oplus (\{\texttt{0A}\} \bullet y_2) \oplus (\{\texttt{09}\} \bullet y_3) \oplus (\{\texttt{08}\} \bullet y_4) \oplus y_5 \oplus (\{\texttt{04}\} \bullet y_6) \oplus y_7 \\
z_1 &= y_0 \oplus y_1 \oplus (\{\texttt{02}\} \bullet y_2) \oplus (\{\texttt{0A}\} \bullet y_3) \oplus (\{\texttt{09}\} \bullet y_4) \oplus (\{\texttt{08}\} \bullet y_5) \oplus y_6 \oplus (\{\texttt{04}\} \bullet y_7) \\
z_2 &= (\{\texttt{04}\} \bullet y_0) \oplus y_1 \oplus y_2 \oplus (\{\texttt{02}\} \bullet y_3) \oplus (\{\texttt{0A}\} \bullet y_4) \oplus (\{\texttt{09}\} \bullet y_5) \oplus (\{\texttt{08}\} \bullet y_6) \oplus y_7 \\
z_3 &= y_0 \oplus (\{\texttt{04}\} \bullet y_1) \oplus y_2 \oplus y_3 \oplus (\{\texttt{02}\} \bullet y_4) \oplus (\{\texttt{0A}\} \bullet y_5) \oplus (\{\texttt{09}\} \bullet y_6) \oplus (\{\texttt{08}\} \bullet y_7) \\
z_4 &= (\{\texttt{08}\} \bullet y_0) \oplus y_1 \oplus (\{\texttt{04}\} \bullet y_2) \oplus y_3 \oplus y_4 \oplus (\{\texttt{02}\} \bullet y_5) \oplus (\{\texttt{0A}\} \bullet y_6) \oplus (\{\texttt{09}\} \bullet y_7) \\
z_5 &= (\{\texttt{09}\} \bullet y_0) \oplus (\{\texttt{08}\} \bullet y_1) \oplus y_2 \oplus (\{\texttt{04}\} \bullet y_3) \oplus y_4 \oplus y_5 \oplus (\{\texttt{02}\} \bullet y_6) \oplus (\{\texttt{0A}\} \bullet y_7) \\
z_6 &= (\{\texttt{0A}\} \bullet y_0) \oplus (\{\texttt{09}\} \bullet y_1) \oplus (\{\texttt{08}\} \bullet y_2) \oplus y_3 \oplus (\{\texttt{04}\} \bullet y_4) \oplus y_5 \oplus y_6 \oplus (\{\texttt{02}\} \bullet y_7) \\
z_7 &= (\{\texttt{02}\} \bullet y_0) \oplus (\{\texttt{0A}\} \bullet y_1) \oplus (\{\texttt{09}\} \bullet y_2) \oplus (\{\texttt{08}\} \bullet y_3) \oplus y_4 \oplus (\{\texttt{04}\} \bullet y_5) \oplus y_6 \oplus y_7
\end{aligned}$$

4.4 ARIRANG-384

ARIRANG-384 is used to hash a message M with a length of l bits, where $0 \leq l < 2^{128}$. ARIRANG-384 is defined in the same manner as ARIRANG-512, with the following two exceptions:

4.4.1 Setting the Initial Value (H^0)

For ARIRANG-384, the initial value H^0 shall consist of the following eight 64-bit words, in hexadecimal:

$$\begin{aligned}
H_0^0 &= \texttt{0xCBBB9D5DC1059ED8}, \\
H_1^0 &= \texttt{0x629A292A367CD507}, \\
H_2^0 &= \texttt{0x9159015A3070DD17}, \\
H_3^0 &= \texttt{0x152FECD8F70E5939}, \\
H_4^0 &= \texttt{0x67332667FFC00B31}, \\
H_5^0 &= \texttt{0x8EB44A8768581511}, \\
H_6^0 &= \texttt{0xDB0C2E0D64F98FA7}, \\
H_7^0 &= \texttt{0x47B5481DBEFA4FA4}.
\end{aligned}$$

These words are obtained by taking the first 64 bits of the fractional parts of the square roots for the ninth to the sixteenth prime numbers each.

4.4.2 Hash Value H^N of ARIRANG-384

The 384-bit message digest is obtained by truncating the final hash value H^N to its leftmost 384 bits:

$$H_0^N \| H_1^N \| H_2^N \| H_3^N \| H_4^N \| H_5^N.$$

5 Motivation for Design Choices

5.1 Structure of ARIRANG

The domain extension of ARIRANG combines the counter-masking and the domain extension MDP (Merkle-Damgård Strengthening with a Permutation) which was proved to preserve multiple properties such as collision resistance, MAC, indifferentiability, and pseudorandomness. The domain extension of ARIRANG is designed by the following criteria:

- **Guaranteeing the n-bit security against known second-preimage attacks and herding attack.**

  The counter-masking gives ARIRANG the n-bit full security against the Kelsey-Schneier second-preimage attack [24], the Kelsey-Kohno herding attack [23], and the strengthened second-preimage attack recently proposed by Elena *et al.* [1].

- **Guaranteeing the $n/2$-bit security from the perspective of collision resistance, preimage resistance, indifferentiability, and pseudorandomness.**

  The domain extension of ARIRANG preserves multiple properties like the MDP domain extension [19].

- **Guaranteeing the security of HMAC-ARIRANG and randomizing hashing based on ARIRANG.**

  Security Proofs will be given for HMAC-ARIRANG and a randomized hashing with ARIRANG.

5.2 Message Schedule of ARIRANG

We adopt the message schedule which combines a message word expansion and ordering. The message schedule of ARIRANG follows the design principle of that of HAS-160 which is the Korean TTA[2]-standard hash algorithm; so far no attacks on the full-round HAS-160 have been found. The difference between the message schedule of ARIRANG and that of HAS-160 is that ARIRANG uses 16 constant words for the message word expansion and to rotate the value obtained by XORing 4 message words and a constant word.

The message schedule of ARIRANG has been chosen according to the following criteria:

- **Minimizing operations for the message schedule.**

[2] TTA : Telecommunications Technology Association

The step function of ARIRANG is heavier than that of existing hash functions such as the MD-family, and the SHA-family. As a compensation of a slightly heavy step function, ARIRANG adopts the simple message schedule to improve efficiency.

- **Infeasibility to construct intended differential characteristics.**

  Four expanded message words in each round make it difficult for an attacker to apply the technique (which is called the local-collision-pattern-based-attack) used for analyzing SHA-0 and SHA-1 [33, 32] to ARIRANG, and also to construct any intended local differential characteristics on each round. Even though some differential characteristics for each round are found, it is very unlikely to connect one characteristic to another characteristic, so that the combined differential characteristic appears in more than one round.

5.2.1 Choice of Each Component of the Message Schedule

- **The message expansion.**

  As stated above, the message word expansion of ARIRANG uses constants and a rotation operation.

  - **Choice of constants.** The message schedule of ARIRANG uses 16 constants $K_0, K_1, \cdots, K_{15}$ generated from a LFSR with a maximum period. Each constant is used to generate one expanded message word. In other words, all expanded message words are generated by using different constants, except that the constant K_0 of the first expanded message word is the same as the constant K_{15} of the last expanded message word. The constants make it difficult to repeatedly use an unintended property of some successive steps for generalizing it into the full steps of the compression function of ARIRANG.
  - **Choice of rotation values.** The message schedule of ARIRANG uses several randomly chosen numbers of rotation bits to generate the 16 expanded message words. Its role is that each bit of the XORed message words affects widely to all bit positions of expanded message words. Such an effect spreads out in the step functions of ARIRANG, so that it is difficult for an attacker to find meaningful characteristics.

- **Choice of the message ordering.**

  The most facile strategy of attackers finds some differential patterns and uses them to construct full round differential characteristics. This is because specific differential patterns used at each round may be applied to other rounds. However, it is not the case for ARIRANG; even if the order rule permuting from a round to the next round is fixed in ARIRANG, there are no same differential patterns in all rounds, and thus it would be difficult for an attacker to use local collision patterns for finding full-round differential characteristics.

5.3 The Step Function of ARIRANG

The step function of ARIRANG is totally different with those of the MD and SHA-2 families, which are based on boolean functions and addition operation ($\boxplus$). In order to complete the step function ARIRANG employs the AES round components and the structure of the FORK-256[3] step function (the FORK-256 hash function was presented at the first NIST hash workshop and FSE 2006 [21]). Unlike FORK-256, the number of steps of ARIRANG is forty. The step function of ARIRANG has been chosen with the following motivations:

5.3.1 Choice of Each Component of the Step Function

- **Choice of 32-bit and 64-bit oriented operation.**

  Since most computers today use the word-size of 32 bits, all the operations in ARIRANG-256 and ARIRANG-224 are suitable for the current environment. Recall that all the operations of ARIRANG-256 and ARIRANG-224 are 32-bit oriented. At the current state of the technology, this choice provides a good tradeoff between the ability to run the algorithm on computers which are available today (as well as on legacy systems and even 8-bit processors), and the ability to take advantage of larger word-size in future architectures.

  Recently, a next-generation processor supporting 64-bit has been published together with 64-bit operating systems. In the near future, the 64-bit processor will be the general trend. Since ARIRANG-512 and ARIRANG-384 are designed based on a 64-bit unit, they are suitable for such 64-bit machines.

- **Choice of generalized type-3 Feistel network as the structure of the step function.**

  The step function of ARIRANG updates all the working variables with input message words. The advantage of such structure, which is called generalized type-3 Feistel network, is that an input message word is used to affect several working variables simultaneously, so that it is much harder for an attacker to control values of working variables. The reason is that an attacker must take into account all the possible combinations of values for the input message words in order to make good cryptanalytic properties, which quickly leads to an unmanageable complexity.

  The working variables of ARIRANG consist of w-bit eight words. The first working variable a is used to update b, c, and d and the 5-th working variable e is used to update f, g, and h through each step. This structure is bilaterally symmetric.

  Among various network structures which are capable of handling eight working words in a block, it seems that the generalized type-3 Feistel network provides the best tradeoff between speed, strength and suitability for analysis. This structure provides much better diffusion properties with only a slightly added cost; fewer steps can be used to achieve the desired strength. Moreover, implementations of this structure can be parallelized to give higher performance efficiency.

- **Choice of the $G^{(\cdot)}$ function as the core component of the step function.**

  ARIRANG uses SubBytes and MDS transformations, which form the $G^{(\cdot)}$ function, as its core components of the step function with the generalized type-3 Feistel network. As mentioned before, the SubBytes and MDS transformations are optimal components

[3] Even if some recent papers show some weaknesses of FORK-256 due to the small number of steps, the structure of the FORK-256 step function has many merits.

of symmetric-key algorithms satisfying two properties, confusion and diffusion. The avalanche effect[4] of the step function depends on these SubBytes and MDS transformations.

- **Choice of SubBytes.**

  In cryptography, a substitution box (or S-box) is a basic component of symmetric-key algorithms. In block ciphers, it is typically used to obscure the relationship between the plaintext and the ciphertext: Shannon's property of confusion. In many cases, the S-box is carefully chosen to resist against cryptanalysis.

  As the SubBytes of ARIRANG, the S-box of AES is adopted; it was specifically designed to be resistant to linear and differential cryptanalysis. This was done by minimizing the correlation between linear transformations of input/output bits, and at the same time minimizing the difference propagation probability. In addition, in order to strengthen the S-box against algebraic attacks, the affine transformation was added. An ostensible reason for the choice of the AES S-box is that AES has already been implemented in many applications. It follows that ARIRANG can share the codes of the implemented AES S-box; this is a big advantage in cryptographic applications which need a hash algorithm and a block cipher simultaneously.

- **Choice of the MDS transformation.**

  As the MDS transformation of ARIRANG-256 and ARIRANG-224, the 4x4 MDS matrix of AES is adopted. The reason of this employment is that 1) it provides the best diffusion effect,[5] and 2) the Hamming weights of the matrix entries '01', '02' and '03' are at most 2 each, which provides advantages when implemented in hardware and in smart cards. Furthermore, the 4x4 MDS matrix of AES can also be shared where it needs a hash algorithm and a block cipher simultaneously.

  As the MDS transformation of ARIRANG-512 and ARIRANG-384, a newly developed $\mathrm{MDS}_{8\times 8}$ matrix, defined over $GF(2^8)$, is adopted. Similarly, it provides the best diffusion effect; it guarantees that the number of changed input bytes plus the number of changed output bytes is at least nine. Our design goal for this matrix is to minimize the Hamming weights of the entries for a better efficiency; the adopted circulant matrix has 3 1-entries per row, and the Hamming weights of the entries are at most 2 each.

5.4 The Feedforward Function

A good hash algorithm should satisfy the preimage-resistant property. The step function of ARIRANG itself dose not satisfy one-wayness. To provide the one-wayness of the compression function, an additional component is required, that is, the Feedforward function. By two Feedforward functions used in the compression function of ARIRANG, it is very unlikely for an attacker to find a preimage and also to find any fixed point of the compression function of ARIRANG to be used for the second-preimage attack and the herding attack.

[4] The avalanche effect is evident if, when an input is changed slightly (e.g., flipping a single bit) the output changes significantly (e.g., half the output bits flip).

[5] The $\mathrm{MDS}_{4\times 4}$ matrix guarantees that the number of changed input bytes plus the number of changed output bytes is at least five. In other words, any change in a single input byte is guaranteed to change all four output bytes, any change in any two input bytes is guaranteed to change at least three output bytes, etc.

Table 4: The 5-step left local collision pattern of ARIRANG, under the assumption that all message words in the consecutive 5 steps are independent.

Step t	$\Delta W_{(\sigma(2t))}$	Δa	Δb	Δc	Δd	$\Delta W_{(\sigma(2t+1))}$	Δe	Δf	Δg	Δh
i	α	$\cdot$	α	β	$\beta \lll 13(29)$	$\cdot$	$\beta \lll 23(41)$	$\cdot$	$\cdot$	$\cdot$
$i+1$	$\cdot$	$\cdot$	$\cdot$	α	β	$\beta \lll 23(41)$	$\beta \lll 13(29)$	$\cdot$	$\cdot$	$\cdot$
$i+2$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	α	$\beta \lll 13(29)$	β	$\cdot$	$\cdot$	$\cdot$
$i+3$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	β	α	$\cdot$	$\cdot$	$\cdot$
$i+4$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	α	$\cdot$	$\cdot$	$\cdot$	$\cdot$

6 Security Analysis

6.1 Local-Collision-Finding Attack

When an attacker inserts differences to the message words, the event that the difference of the working variables becomes zero often occurs. It is called a local collision, and a differential characteristic, which causes a local collision with a certain probability, is called a local collision pattern. In other words, a local collision is a collision within a few steps of the hash function. A local collision is not a real collision, but the notion of local collision pattern is important in cryptanalysis of hash functions because it can be repeatedly used to yield a real collision with a meaningful probability. For hash functions with a serial structure, their resistances against collision-finding attack depend mainly on how many times a local collision is repeatedly used to yield a real collision.

Security of ARIRANG against local collision-finding attack. Firstly, let us focus on one round of the compression function of ARIRANG. We can easily construct 5-step local collision pattern, under the assumption that all message words in the consecutive 5 steps are independent. Suppose a message difference first occurs in i-th step (perturbation message difference). It means that the difference of the input messages in i-th step is nonzero ($\Delta W_{\sigma(2i)} \neq 0$ or $\Delta W_{\sigma(2i+1)} \neq 0$). The difference will consecutively affect the working variables a, b, c, d, e, f, g, h in the next four steps. In order to offset these differences and reach a local collision, more message differences should be introduced in subsequent message words (correction message difference). Note that two message words are applied to each step. In the case that the left message word has a perturbation message difference, we call a local collision pattern generated from the left message word, which has a perturbation message difference, "left local collision pattern". In the case that the right message word has a perturbation message difference, we call a local collision pattern generated from the right message word, which has a perturbation message difference, "right local collision pattern". In Table 4 and 5, we illustrate two basic local collision patterns of such a local collision on ARIRANG, under the assumption that all message words in the consecutive 5 steps are independent.

However, there is no real 5-step local collision pattern on ARIRANG, because of 4 expanded message words used in the 1-st step and the 6-th step. More precisely, the following three facts block an attacker to make 5-step local collision patterns.

- **Fact 1 :** Since each expanded word is generated from four original message words, if

Table 5: The 5-step right local collision pattern of ARIRANG, under the assumption that all message words in the consecutive 5 steps are independent.

Step t	$\Delta W_{(\sigma(2t))}$	Δa	Δb	Δc	Δd	$\Delta W_{(\sigma(2t+1))}$	Δe	Δf	Δg	Δh
i	$\cdot$	$\beta \lll 7(13)$	$\cdot$	$\cdot$	$\cdot$	α	$\cdot$	α	β	$\beta \lll 29(53)$
$i+1$	$\beta \lll 7(13)$	$\beta \lll 29(53)$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	α	β
$i+2$	$\beta \lll 29(53)$	β	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	α
$i+3$	β	α	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$
$i+4$	α	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$

there is no difference among the four message words, then the expanded message word has certainly zero difference.

- **Fact 2 :** Since each expanded word is generated from four original message words, if only one of the four message words has non-zero difference, then the expanded message word has certainly non-zero difference.

- **Fact 3 :** For any nonzero difference α, $\alpha \oplus (\alpha \lll x) \oplus (\alpha \lll y)$ is nonzero.

Fact 1 and 2 restricts that there are just four 5-step local collision patterns which the difference of a message word first occurs at 2-th or 5-th step. For example, if an attacker attempts to construct 5-step left local collision pattern starting at 1-th step, the additional message words difference occurs at least one of 4 message words, $W_{\sigma(13)}$, $W_{\sigma(15)}$, $W_{\sigma(17)}$, $W_{\sigma(19)}$, because the message word $W_{\sigma(0)}$, which is used in 1-th step, is made from 4 message words, $W_{\sigma(13)}$, $W_{\sigma(15)}$, $W_{\sigma(17)}$, or $W_{\sigma(19)}$. So 5-step local collision patten can be only started at 2-th or 5-th step.

However, four 5-step local collision patterns starting at 2-th or 5-th step are also impossible, because of Fact 3. For example, 5-step left local collision pattern starting at 2-th step should satisfy the following condition.

$$\Delta W_{\sigma(5)} \oplus \Delta W_{\sigma(7)} \oplus \Delta W_{\sigma(9)} = \Delta W_{\sigma(10)} = 0$$

Suppose the output difference of left G function in 2-th step be $\alpha \neq 0$. In the case of ARIRANG-256, 5-step left local collision starting at 2-th should satisfy following equation by above condition.

$$\alpha \oplus (\alpha \lll 13) \oplus (\alpha \lll 23) = 0.$$

In above equation, for any nonzero difference α, $\alpha \oplus (\alpha \lll 13) \oplus (\alpha \lll 23)$ can not be zero because it is a bijective function. The other 5-step local collision patterns have similar properties. Thus there is no 5-step local collision pattern because of 4 expanded message words of each round.

To construct local collision pattern, 5-step local collision patterns are combined more than one pattern and then at least 7 message words must have nonzero difference. The maximum probability associated with the local collision pattern in one round is 2^{-224} and the maximum probability associated with the local collision pattern in full round is at most $(2^{-224})^2$, because the local collision pattern occur at least one in each round and we assume that an attacker can control the collision patterns of the first two rounds. We have described local collision patterns and each conditions with maximum probability in Appendix B.

Even though a local collision pattern is fortunately found without any assumption, the local collision pattern can not be applied to other rounds because of the message re-ordering. In other words, it is difficult to find a collision by using an local collision pattern. Therefore, we expect that ARIRANG is secure against the local collision-finding attack.

6.2 Collision-Finding Attack

In cryptography, a collision attack on a cryptographic hash function is a technique to find two different messages with a same output. The "birthday paradox" offers an upper bound on collision attack: if a hash function produces n bits of hash values, an attacker can find a collision (on average) by performing only $2^{n/2}$ hash operations. Thus, the goal of collision attack is to find a collision with less than $2^{n/2}$ hash operations.

A cryptographic hash algorithm is a function which maps an arbitrary-length input to a fixed-length output. The most popular method to construct a cryptographic hash function is the Merkle-Damgård construction by iterating the compression function which maps a fixed-length input to a fixed-length output. There is a statement regarding the relation between an iterated hash function H and the underlying compression function F with respect to the property of collision-resistance: if the padding procedure is a postfix-free padding such as padding the length of the input message into the last block of the message, then H is collision resistant if F is collision-resistance. It is based on the argument that a collision for H would imply a collision for F.

Security of the compression function of ARIRANG against collision finding attack. We assume that an attacker can control input message words up to 2 rounds to construct characteristics with high probabilities. Our assumption is reasonable, because the message modification normally allows an attacker to control the input message words over at most 2 rounds. Actually, for the ARIRANG family the message modification through 2 rounds is even more difficult due to the structure of the step function of ARIRANG, which is to update four working variables with an one working variable.

Now, we analyze the third and fourth rounds of the compression function under the above assumption. There is no way to modify any message words in the third and fourth rounds, because by the assumption all message words are fixed during the message modification through the first 2 rounds. It follows that the message words in the third and fourth rounds are out of attacker's control.

Based on this observation, we try to compute an upper bound of the probability of any collision producing differential characteristic by the following idea. We first set the message difference vector in which a word of the zero difference is represented by 0 and a word of a non-zero difference is represented by 1. We focus on the collision event of the last round. In the last round, there are the 16 original message words and the 4 extended message words, thus the message difference vector consists of 20 elements in total. For an easy security analysis, we assume that the original 16 message words and the 4 expanded message words are mutually independent except for Fact 1 and 2 of section 6.1.

After setting the message difference vector, we want to construct collision characteristics which can be generated from each message difference vector. The final collision event should

Table 6: Example. Process table for a collision finding attack on ARIRANG

Step t	$\Delta W_{(\sigma(2t))}$	Δa	Δb	Δc	Δd	$\Delta W_{(\sigma(2t+1))}$	Δe	Δf	Δg	Δh	Prob.
30	1	·	·	·	·	1	·	·	·	·	
31	0	·	·	·	·	0	·	·	·	·	
32	1	·	·	·	·	0	·	·	·	·	
33	0	1	1	0 *or* 1	0 *or* 1	0	1	0 *or* 1	0 *or* 1	1	$(2^{-32})^2$
34	0	0	1	0	1	0	1	1	1	1	$(2^{-32})^3$
35	0	0	0	1	0	1	1	1	0	0	2^{-32}
36	0	0	0	0	1	0	0	0	1	0	
37	0	0	0	0	0	1	1	0	0	1	2^{-32}
38	1	1	0	0	0	0	0	0	0	0	2^{-32}
39	0	0	0	0	0	0	0	0	0	0	
40	·	0	0	0	0	·	0	0	0	0	

occur in the step where the last non-zero difference of message word (marked by 1) appears. If the collision event does not occur in the step where the last non-zero difference of message word appears, it is impossible to get a collision characteristic, because the the last message word of non-zero difference influences working variables in next steps. By this observation, we could construct a collision producing characteristic step by step from the last step through the backward direction (note that this last step is in the fourth round, and leads to a collision).

In this backtracking (starting from all working variables of zero differences), we assume that for any 32-bit differences α and β, a probability that $\alpha \oplus \beta = 0$ is 2^{-32} and a probability that $\alpha \oplus \beta =$ any γ is $1 - 2^{-32}$, where α, β and γ are all nonzero differences.

One example for above process is as follows. Suppose a message difference vector used in 4-th round is "11001000000100011000", which is determined by the message re-ordering of 4-th round. First, second, 11-th, and 12-th difference bits among the above difference vector shows whether each expanded message word has a non-zero difference or not. The remained 16 bits correspond to differences of the 16 original message words.

Given the above message difference vector, we can do the backtracking for this message difference vector as shown in table 6 and each element of table 6 denotes that '1' is a non-zero difference and '0' is the all zero difference. In the given message difference vector, last nonzero bit is the 17-th bit. That the 17-th bit is 1 means that the input message word of 38-th step is the last message word of non-zero difference. Note that we only consider a collision characteristic. Thus the difference of the message word of 38-th step should be blocked and the difference of all working variable from 39-step should be zero because there is no message word of non-zero difference from 38-th step to the last step. So we know that the difference of input working variable a of 38-th step is only non-zero and differences of the others are all zero. Moreover the difference of working variable a of 38-step should equal to the input message difference of 38-th step. By our assumption, the probability that a difference of the input working variable a of 38-th step is offset by the input message difference of 38-th step is 2^{-32}. Likewise, we can backtrack collision characteristics up to 33-step as described in table 6. The number of difference which is offset by message words and working variables of non-zero differences is eight, so the probability that a message pair with the above difference vector satisfies this characteristic is $(2^{-32})^8 = 2^{-256}$. In table 6, if the difference of working variable can be zero or nonzero, it is backtracked until the differences of working variables are offset by the differences of other working variables or message words.

Based on this assumption, we investigated all the cases for 2^{20} message difference vectors except for Fact 1 and 2 of section 6.1. Note that once a difference vector of a round is chosen, difference vectors of other rounds are automatically determined by the re-ordering of message words. And we found that there is no collision producing characteristic which has a probability higher than 2^{-256} in last two rounds. Even though an attacker can control the input message words through the first 2 rounds by the message modification technique, this result supports that the compression function of ARIRANG-256 guarantees the full 128-bit security against the collision attack. With a similar analysis, it is expected that the compression function of ARIRANG-512 guarantees the full 256-bit security against the collision attack.

Security of the domain extension of ARIRANG against collision attack. ARIRANG uses different counter values in order to process each message block during the computation of the final hash value digest of a message. Although the domain extension of ARIRANG is different from the Merkle-Damgård construction, it is similarly proved that the domain extension of ARIRANG preserves collision-resistance; in our security evaluation the compression function of ARIRANG resists against the collision attack, thus we expect that the hash function ARIRANG guarantees the full $n/2$-bit security against the collision attack, where n is the output size of the compression function.

6.3 Pseudo Collision Finding Attack

We use the term "Pseudo collision" if two different initial values IV, IV' and (possibly identical) input messages M, M' are given such that $H(IV, M) = H(IV', M')$. Pseudo collisions are of much less practical importance than collisions. However, pseudo collision attack is a powerful attack where an attacker can control the initial value of a hash algorithm, because there could be some applications which require the underlying hash function to have pseudo collision resistance. In 1993, Bert den Boer and Antoon Bosselaers showed how pseudo-collisions of MD5 compression function could be found [12]. In Eurocrypt 2008, Lei, Wang *et al.* introduced a related-key attack on NMAC-MD5 using a pseudo-collision [30].

There are two kinds of pseudo-collision attacks. One is the pseudo-collision attack in forward direction. The other is the pseudo-collision attack in backward direction. In the case of forward direction, the first round is focused. More precisely, an attacker first makes collisions after the first round by controlling differences of initial values and of messages (this first-round differential characteristic has a form of all zero input/output differences at all steps except for the first a few steps). Next, an attacker finds a differential characteristic in the last three rounds with zero input difference of working variables, where the final output difference of the concatenated full-round differential characteristic should be the same as the difference of initial values. On the other hand, in the case of backward direction, the last round is focused. More precisely, an attacker first makes collisions before the last round by controlling differences of hash values and of messages (this last-round differential characteristic has a form of all zero input/output differences at all steps except for the last a few steps; note that this last-round differential characteristic holds with a probability less than 1 unlike the above first-round differential characteristic). Next, an attacker finds a differential characteristic in the first three rounds with zero output difference of working variables, where the final output difference of the concatenated full-round differential characteristic should be the same as the difference of initial values. In order to show the security of ARIRANG against these two types of pseudo-collision attackers, we assume the following.

Assumption. Given a differential characteristic of the full rounds of ARIRANG, any attacker can control message words and working variables in at most two rounds so that the message words and working variables satisfy the differential characteristic with probability 1 for the involved rounds. In the cases of MD5 and SHA-1, no known attack can control message words and working variables beyond two rounds. Since ARIRANG was more securely designed than MD5 and SHA-1 against such kind of technique generally called the message modification technique, this assumption is reasonable, which we thoroughly checked.

Pseudo-collision attack in forward direction. By the re-orderings of message schedules for ARIRANG, there are six strategies of pseudo-collision attacks in forward direction as follows.

- $\Delta W_0 \neq 0$, $\Delta W_2 \neq 0$ and $\Delta W_i = 0$ $(i \neq 0, 2)$: In this case, the last two words of initial values should have non-zero differences so that the differences of W_0 and W_2 can be offset by the differences of the two words. Since our target is a pseudo-collision, the output difference of the final step should be the same as the difference of initial values. More precisely, the last two words of the final output difference are both non-zeroes and the other words have all zero differences. Now, we want to compute an upper bound of the probability of this kind of differential characteristics. By the backtracking method introduced in Sect. 6.2, we can know that there are at least five conditions in the final four steps, which means that the probability of the differential characteristic is at most $2^{-160}(= 2^{-32\cdot 5})$ for ARIRANG-256. Therefore, ARIRANG-256 is secure against this pseudo-collision-finding strategy. In the cases of other versions of ARIRANG, the security analysis can be done in the same way.

  As in the above case, ARIRANG is secure against the following five pseudo-collision-finding strategies.

- $\Delta W_0 \neq 0$, $\Delta W_4 \neq 0$ and $\Delta W_i = 0$ $(i \neq 0, 4)$.
- $\Delta W_2 \neq 0$, $\Delta W_4 \neq 0$ and $\Delta W_i = 0$ $(i \neq 2, 4)$.
- $\Delta W_1 \neq 0$, $\Delta W_3 \neq 0$ and $\Delta W_i = 0$ $(i \neq 1, 3)$.
- $\Delta W_1 \neq 0$, $\Delta W_5 \neq 0$ and $\Delta W_i = 0$ $(i \neq 1, 5)$.
- $\Delta W_3 \neq 0$, $\Delta W_5 \neq 0$ and $\Delta W_i = 0$ $(i \neq 3, 5)$.

Pseudo-collision attack in backward direction. By the re-orderings of message schedules for ARIRANG, there are two strategies of pseudo-collision attacks in backward direction.

- $\Delta W_6 \neq 0$, $\Delta W_{12} \neq 0$ and $\Delta W_i = 0$ $(i \neq 6, 12)$: W_6 and W_{12} are used in the last two steps of the 4-th round of ARIRANG. If $\Delta W_6 \neq 0$ and $\Delta W_{12} \neq 0$, all output working variables of the last step should have non-zero differences in order for the input difference of the last two steps to be of zero difference. Since our target is a pseudo-collision, the output difference of the final step should be the same as the difference of the initial values. Since there are at least three conditions to make zero input difference and non-zero output difference through the last two steps, the probability that the final output difference is the same as the difference of initial values is at most $(2^{-32})^3 = 2^{-96}$ for ARIRANG-256. Thus, the probability that a message pair satisfies a differential characteristic for the first three rounds should be bigger than 2^{-32} to make a meaningful collision attack, which seems to be infeasible. In the cases of other versions of ARIRANG, the security analysis can be done in the same way.

- $\Delta W_1 \neq 0$, $\Delta W_{11} \neq 0$ and $\Delta W_i = 0$ $(i \neq 1, 11)$: As in the above case, ARIRANG is secure against this pseudo-collision-finding strategy.

6.4 Near-Collision-Finding Attack

In Crypto 2004 and Eurocrypt 2005, Biham *et al.* introduced a near collision attack on SHA-0 and reduced SHA-1 [10, 11]. We say that two messages M and M' cause a near-collision for a hash function H if the Hamming distance between $H(M)$ and $H(M')$ is small. Although this attack is not as strong as a collision finder, a near collision attack sometimes serves as a precursor to a full attack: some researchers have published collision attacks and key-recovery attacks against HMAC and NMAC using a near collision attack [30].

Security of ARIRANG against the near-collision-finding attack. From the perspective of the backtracking technique of ARIRANG, the near collision-finding attack can be regarded as a pseudo-collision-finding attack. That is, the security of ARIRANG against near-collision-finding attack can be guaranteed by the security of ARIRANG against the pseudo-collision-finding attack.

6.5 Fixed-Point-Finding Attack

A fixed point for a compression function F is a pair (H^{i-1}, M^i) for which $H^{i-1} = F(H^{i-1}, M^i)$. This property allows an attacker to produce second preimages or collisions by iterating a fixed point. A fixed point can be found in the compression functions based on the Davies-Meyer construction, such as the SHA family, MD4, MD5, and Tiger. For the SHA and MD4/MD5 families, an attacker selects any message M^i and then sets the previous value to be Feedforward to all zero-bit sequence. An attacker can recover an input value H^{i-1} of the compression function, because the compression function is invertible. In case of the Davies-Meyer construction, since all zero-bit sequence is the additive identity, an attacker can easily find fixed points. In Eurocrypt 2005, John Kelsey and Bruce Schneier proposed the second preimage attack on the Merkle-Damgård construction using a fixed point or a multi-collision attack.

Security of ARIRANG against the fixed point attack. The compression function of ARIRANG executes the Feedforward operation to compute the i^{th} intermediate hash value H^i and also to update the eight working variables, a, b, c, d, e, f, g, and h with H^{i-1} after the first

20 steps are completed. In other words, unlike the compression functions of SHA family, which uses only one feedfoward operation in the compression function, the compression function of ARIRANG uses the Feedforward operation two times. This characteristic of ARIRANG makes the fixed point-finding attack infeasible.

6.6 First-Preimage-Finding Attack

A preimage attack on a cryptographic hash is an attempt to find a message that has a specific hash value. Onewayness is a basic property in a cryptographic system. A preimage attack differs from a collision attack in the sense that a hash value is only given. Optimally, a preimage attack on an n-bit hash function will take an order of 2^n operations to be successful.

Security of ARIRANG against first preimage attack. The compression function of ARIRANG uses the Feedforward operation twice, which makes it very difficult to invert a given hash value. Since the domain extension of ARIRANG preserves first preimage resistance of the compression function, so we expect that ARIRANG guarantees the full n-bit security against first preimage-finding attack, where n is the output size of the compression function.

6.7 Second-Preimage-Finding Attack

In Eurocrypt 2005, Kelsey and Schneier introduced a second preimage-finding attack on the Merkle-Damgård construction Strengthening with about the birthday attack complexity. Their attack, which is based on expandable messages, works even when the compression function is a random oracle of fixed input length. In the second NIST hash workshop in 2006, Rivest proposed a new dithering method which are secure against Kelsey-Schneier attack and are more efficient than methods using counters. However, in Eurocrypt 2008, Elena *et al.* devised a new attack method to break the Rivest's dithering method by utilizing a diamond-structure used in the herding attack. Elena *et al.* also showed that when the size of the output of the compression function is n-bit, at least 2^n different values are needed to guarantee the full security against their attack method, where the full security means n-bit security.

Security of ARIRANG against known second preimage-finding attack methods. ARIRANG uses the counter-masking method so that it guarantees the full n-bit security against all known second preimage-finding attackers, where n is the output size of the compression function.

6.8 Length-Extension Attack

Merkle-Damgård Strengthening uses the specific padding pad for which $\mathsf{pad}(M) = M||10^k||\mathsf{bin}_t(|M|)$, where k is the smallest non-negative integer such that $\mathsf{pad}(x)$ is a multiple of d and $\mathsf{bin}_i(x)$ means the i-bit binary representation of x. For example, t=64 for SHA-1, 224 and 256, t=128 for SHA-384 and 512. The extension attack on Merkle-Damgård Strengthening goes as follows. A key K is chosen uniformly at random from the set of keys. An attacker makes a query M, then obtain the response z=MD($\mathsf{pad}(K||M)$). By the characteristic of the structure for Merkel-Damgård Strengthening, without any additional query, on the assumption that the length of the key K is known, an attacker can forge the MAC value z^* for the query

$\mathsf{pad}(K||M)||M'$ by computing $z^* = f(z||M'||1||\mathsf{bin}_{64}(\mathsf{pad}(K||M)||M'))$, where M' is any message, $|M'|$=d-65, d is the message block length of bits, and f is the compression function.

Security of ARIRANG against length-extension attack. In ARIRANG, the last counter is always fixed as the 128-bit fixed counter and different from all other counters. In other words, ARIRANG uses a prefix-free counter masking method so that there is little likelihood of the length-extension attack on ARIRANG with the complexity less than the birthday attack complexity.

6.9 Multicollision Attack

Since multicollision-finding attackers [22] can be used in Kelsey-Schneier's second-preimage attack [24] and Kelsey-Kohno's herding attack [23] on Merkle-Damgård Strengthening, the security against multicollision-finding attackers is significant. However, till now, all known methods that guarantee full security against multicollision-finding attackers are to make the internal output size of a hash function at least two times bigger than that of hash output. Wide-Pipe Hash and prefix-free chopMD are such examples. However, from the perspective of practical implementation, the design of a hash function with a big internal size may reduce the implementation efficiency and may require big memories and resources.

Security of ARIRANG against multicollision attack. Since the internal size of ARIRANG is the same as the hash output size of ARIRANG, Joux's multicollision-finding attack on the Merkle-Damgård Strengthening can be also applied to ARIRANG. On the other hand, multicollisions of ARIRANG does not support different types of attacks such as preimage attack and second-preimage attack because ARIRANG uses counters.

6.10 Indifferentiability

The security notion *Indifferentiability* was introduced by Maurer *et al.* in TCC 2004 [26]. Since the concept indifferentiability makes it possible to evaluate the security of domain extensions against all possible generic attackers, under the assumption that the underlying function is a random oracle or an ideal cipher, it is considered one of the significant notions of provable security. In Crypto 2005, Coron *et al.* proved that the classical MD iteration is not indifferentiable with random oracle model even if we assume that the underlying compression function is a random oracle. But they have shown indifferentiability for prefix-free MD hash functions or some other definitions of hash functions like HMAC construction, NMAC construction and chop-MD hash function.

In this section, we provide the indifferentiable security of ARIRANG. For the security proof of ARIRANG we follow the security proof technique of the Sponge construction shown in Eurocrypt 2008 [6]. Here, we just describe the indifferentiable security bound of ARIRANG. For the exact security proof, refer to the appendix A.2.

Indifferentiable Security of ARIRANG. We assume that the compression function of ARIRANG is a random oracle of fixed input length. Let N be the number of blocks of queried messages. Then, for any adversary A, the indifferentiable security of ARIRANG is bounded by $N(3N-1)/2^n$, where n is the output size of the compression function of ARIRANG. More precisely, there exists a simulator S (which simulates F) such that for any adversary A, A cannot differentiate (ARIRANG, F) from (RO, S), where F is a FIL random oracle and RO is a VIL random oracle.

6.11 Slide Attack

In 2008, Gorski, Lucks and Peyrin [17] provided a slide attack (which recovers the secret key K) on a MAC construction based on Grindahl-256 and Grindahl-512. The slide attack is not for a hash function itself, but for a MAC based on a hash function. The MAC construction they considered is $MAC(K, M) = H(K||M)$, where H is Grindahl-256 or Grindahl-512. The slide attack is useful in the case that a hash function is constructed by just iterating a same compression function without any modification.

Security of ARIRANG-based MAC against slide attack. ARIRANG uses a counter which is increased by 1 every time to process each message block of a message, which makes it difficult for an attacker to apply the slide attack to any ARIRANG-based MAC. And the formal PRF security proof of $H(K||M)$ is also given in the Appendix A.4.

6.12 Trapdoor-based Attack

A candidate hash algorithm for SHA-3 should have no trapdoor in any constants or tables used in the algorithm. This is because a trapdoor may be used to make some attack easier. The ARIRANG family have no trapdoor due to the following reasons.

- The sixteen constants in the message schedule : The role of constants prevents an attacker from repeatedly using an unintended property of some successive steps, so that the attacker can generalize it into the full steps of the compression function of ARIRANG. So, we don't care which values are used for the constants, but concern about varying constants. Therefore, we insist that the sixteen constants have no trapdoor.
- The initial value : The security of ARIRANG does not depend on the initial value, because the compression function of ARIRANG is believed to be secure against all known attacks including the pseudo-collision-finding attack, by intensive security analyses in this section.
- The value P of the last counter : For the formal security proofs of the domain extension of ARIRANG, the condition that $P \geq 2^{55} (= 2^{64}/2^{9})$ is sufficient for ARIRANG-224/256 and $P \geq 2^{118} (= 2^{128}/2^{10})$ is sufficient for ARIRANG-384/512. In other words, we have only to choose any P not equal to any counter numbers. That is, the security of ARIRANG does not depend on the specific value of P. Therefore, we insist that such a P has no trapdoor.

6.13 Randomness Test

Randomness is an essential resource for cryptography. Since constructing an ideal random number generator (RNG) which completely generates unpredictable and uncorrelated bit sequences is quite complex and takes a ridiculously high price, a pseudo-random number generator (PRNG) is frequently used in the real world instead of a RNG. A PRNG is a deterministic algorithm which produces outputs that look truly random to an outside observer, and is random enough for practical purposes. This means that nobody can computationally distinguish a PRNG from a truly random source. Specially, a hash algorithm is a representative primitive to construct a PRNG, and thus a good hash function must satisfy the randomness property.

In order for a hash function to be used as a PRNG, randomness tests should first be conducted. One of randomness tests is statistical tests; **the NIST statistical test suite** [28] is a statistical package consisting 16 tests that were developed to evaluate the randomness of binary sequences. We carried out randomness tests for the ARIRANG family by using the NIST statistical test suite for various configurations.

6.13.1 ARIRANG Data Sets

In this subsection, we present how we selected ARIRANG data sets to conduct the 16 NIST statistical tests. In our simulation, 15 different data sets were selected for each of ARIRANG-256 and ARIRANG-512: a data set consists of 2048 samples which contain $1,048,576$-bit sequences each ($1,048,576$ bits are 2^{20} bits, equivalently, approximately 2 Gbits). Consequently, 2048 (sample) × 1,048,576 (sequence) bits forms a data set, which are used for the 16 NIST statistical tests. Note that random sequences needed to construct our data sets were generated by the HAS-160 hash algorithm. Our 15 different data sets were made as follows (in our exposition below, the hash value computed after several message blocks is referred as to an initial value, and the next hash value after one more message block is referred as to a hash value).

- **Three data sets for message avalanches.** These data sets allow examining the sensitivity of ARIRANG when a message is changed.

  - **A sample of data set 1.**

    1. Set D =an empty string, and an initial value $IV = 0$.
    2. Generate a random sequence for a one-block message M, and compute the hash value h by using IV and M. Set $i = 0$.
    3. Compute $M' = M \oplus e_i$, and the hash value h' by using IV and M', where e_i is a value of the bit-length $|M|$ whose i-th bit is 1 and the rest are all zeroes.
    4. $D \longleftarrow D||(h \oplus h')$, and $i = i + 1$.
    5. If $|D| < 1,048,576$ and i is less than or equal to the one-block message bit-size, then go to step 3. If $|D| < 1,048,576$ and i is larger than the one-block message bit-size, then go to step 2. If $|D| = 1,048,576$, then output D as a sample of data set 1.

  - **A sample of data set 2.**

    1. Set D =an empty string, an initial value $IV = 0$, and $i = 0$.
    2. Generate a random sequence for a one-block message M, and compute the hash value h by using IV and M.
    3. Compute $M' = M \oplus e_i$, and the hash value h' by using IV and M'
    4. $D \longleftarrow D||(h \oplus h')$.
    5. If $|D| < 1,048,576$, then go to step 2. If $|D| = 1,048,576$, then output D as a sample of data set 2 (for the next sample, set $i = i + 1 \bmod$ one-block message bit-size).

- **A sample of data set 3.**

  1. Set D =an empty string, an initial value $IV = 0$, $i = 0$, and $j = 0$.
  2. Generate a random sequence for a one-block message M.
  3. Compute M' whose words are all the same as those of M except for the j-th word $M'_j = M_j + i$, and compute the hash value h' by using IV and M'
  4. $D \longleftarrow D||h'$.
  5. If $|D| < 1,048,576$, then go to step 3 with $i = i + 1$. If $|D| = 1,048,576$, then output D as a sample of data set 3 (for the next sample, set $j = j + 1$ mod one-block message word-size).

- **Three data sets for initial value avalanches.** These data sets allow examining the sensitivity of ARIRANG when an initial value is changed.

  - **A sample of data set 4.** In the algorithm generating a sample of data set 1, interchange the initial values and messages.
  - **A sample of data set 5.** In the algorithm generating a sample of data set 2, interchange the initial values and messages.
  - **A sample of data set 6.** In the algorithm generating a sample of data set 3, interchange the initial values and messages.

- **A data set for correlations of initial and hash values.** This data set allows examining correlations of initial and hash values for ARIRANG.

  - **A sample of data set 7.**

    1. Set D =an empty string, and generate a random sequence for a one-block message M.
    2. Generate a random sequence for an initial value IV, and compute the hash value h by using IV and M.
    3. $D \longleftarrow D||(IV \oplus h)$.
    4. If $|D| < 1,048,576$, then go to step 2. If $|D| = 1,048,576$, then output D as a sample of data set 7.

- **Four data sets for random initial or random message values.** These data sets allow examining the randomness of ARIRANG when initial values or message values are randomly selected.

  - **A sample of data set 8.**

    1. Set D =an empty string, and generate a random sequence for a one-block message M.
    2. Generate a random sequence for an initial value IV, and compute the hash value h by using IV and M.
    3. $D \longleftarrow D||h$.
    4. If $|D| < 1,048,576$, then go to step 2. If $|D| = 1,048,576$, then output D as a sample of data set 8.

  - **A sample of data set 9.** In the algorithm generating a sample of data set 8, interchange the initial values and messages.
  - **A sample of data set 10.**
    1. Set D =an empty string, and $M = 0$.
    2. Generate a random sequence for an initial value IV, and compute the hash value h by using IV and M.
    3. $D \longleftarrow D||h$.
    4. If $|D| < 1,048,576$, then go to step 2. If $|D| = 1,048,576$, then output D as a sample of data set 10 (for the next sample, set $M = M + 1$).
  - **A sample of data set 11.** In the algorithm generating a sample of data set 10, interchange the initial values and messages.

- **Four data sets for low (or high) density initial or low (or high) density message values.** These data sets allow examining the randomness of ARIRANG when initial values or message values are selected to have low or high hamming weights.

  - **A sample of data set 12.**
    1. Set D =an empty string and $i = 1$, and generate a random sequence for a one-block message M.
    2. Generate an initial value IV whose bit hamming weight is i, and compute the hash value h by using IV and M.
    3. $D \longleftarrow D||h$.
    4. If $|D| < 1,048,576$ and all IV's are not generated in step 2, then go to step 2. If $|D| < 1,048,576$ and all IV's are generated in step 2, then go to step 2 with $i = i + 1$. If $|D| = 1,048,576$, then output D as a sample of data set 12.
  - **A sample of data set 13.** In the algorithm generating a sample of data set 12, interchange the initial values and messages.
  - **A sample of data set 14.**
    1. Set D =an empty string and $i = 1$, and generate a random sequence for a one-block message M.
    2. Generate an initial value IV whose bits are all 1's except for i bit(s), and compute the hash value h by using IV and M.
    3. Compute $D = D||h$.
    4. If $|D| < 1,048,576$ and all IV's are not generated in step 2, then go to step 2. If $|D| < 1,048,576$ and all IV's are generated in step 2, then go to step 2 with $i = i + 1$. If $|D| = 1,048,576$, then output D as a sample of data set 14.
  - **A sample of data set 15.** In the algorithm generating a sample of data set 14, interchange the initial values and messages.

6.13.2 Results of NIST Statistical Tests for ARIRANG

For each of the 16 NIST statistical tests, we set the significance level to be 0.01, which implies that, ideally, no more than 20.48 out of 2,048 samples should be rejected in each data set evaluated by the statistical tests. (Note that the significance level 0.01 is applied to both data sets and each sample in them.) However, some given data sets may not follow this ideal case. Thus, we adopted one of the two approaches presented in the NIST special publication 800-22 [28]: the first approach is on the proportion of samples that pass statistical tests, and the second approach is on the distribution of P-values to check for uniformity. In the case of the first approach which we employed, the maximum number of samples that are expected to be rejected at the chosen significance level is computed by using following formula:

$$s \times \left(\alpha + 3 \times \sqrt{\frac{\alpha(1-\alpha)}{s}} \right),$$

where s is the sample size and α is the significance level. This formula is with reliability rate 0.9987. Since $s = 2,048$ and $\alpha = 0.01$ in our setting, the number of rejected samples should not exceed 33.99 for a data set. We experimentally conducted the NIST statistical tests on our data sets of ARIRANG-256 and ARIRANG-512, resulting that the number of rejected samples was less than 34 for any statistical test on any data set. Thus, the output sequences of ARIRANG is expected to be pseudorandom sequences at the significance level 0.01.

7 Construction to support HMAC, PRF, and Randomized Hashing

In this section, it is shown that the domain extension of ARIRANG provides the formal security proofs of HMAC-ARIRANG, two PRF constructions based on ARIRANG and Randomized Hashing with ARIRANG, under the assumption that the compression function of ARIRANG satisfies some security notions which will be described in this section. The security of the compression function of ARIRANG is justified by the intensive security analysis of the compression function of ARIRANG in Sect. 6.

7.1 HMAC-ARIRANG

HMAC is the MAC construction from a hash function, which was designed by Bellare, Canetti and Krawczyk in Crypto 1996 [5]. When H is a hash function, $HMAC(K, M) = H((K^* \oplus \mathsf{opad})||H((K^* \oplus \mathsf{ipad})||M))$, where ipad and opad are constants, and $K^* = K||0^t$ where t is the smallest non-negative integer such that $|K^*|$ is a block size.

In Crypto 1996, Bellare, Canetti and Krawczyk proved the security of HAMC, on the assumption that the hash function is weakly collision-resistant and the keyed compression function is RKA-pseudorandom [5]. In Crypto 2006, Bellare improved their security proof: the security of HMAC is guaranteed only with the RKA-pseudorandomness of the keyed compression function, where the hash function is based on Merkle-Damgård construction. On the other hand, the domain extension of ARIRANG is not Merkle-Damgård construction because of counter. So, the Bellare's proof of HMAC cannot be directly applied to HMAC-ARIRANG in Fig. 9.

We prove that the structure of HMAC-ARIRANG is secure if the compression function meets some security assumptions. For the details of proof, see the Appendix A.3.

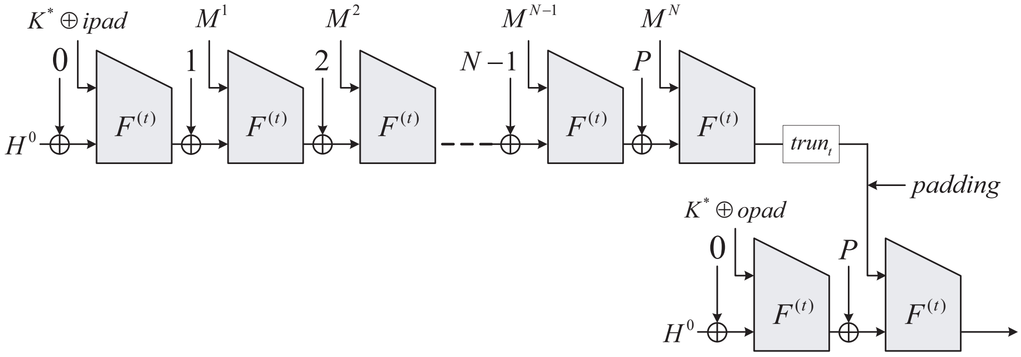

Figure 9: HMAC-ARIRANG : For $t = 224$, $trun_t$ denotes the right-most 32-bit truncation. For $t = 384$, $trun_t$ denotes the right-most 128-bit truncation.

Security of HMAC-ARIRANG. For any prf-adversary A with q queries, there exist adversaries B_A, C_A, D_A, E_{D_A} as follows, where $\mathrm{HMAC}^{\mathrm{pfCM}^0-\mathrm{MD}^f_{\mathsf{pad}}(IV,\star)}$ denotes HMAC-ARIRANG and f is the compression function of ARIRANG.

$$\mathbf{Adv}^{\mathsf{prf}}_{\mathrm{HMAC}^{\mathrm{pfCM}^0-\mathrm{MD}^f_{\mathsf{pad}}(IV,\star)}}(A) \leq \mathbf{Adv}^{\mathsf{rka\text{-}prf}}_{f(\star,K||0^{b-n}),\Phi_1}(B_A) + \mathbf{Adv}^{\mathsf{prf}}_{f(K,\star)}(C_A) + \tfrac{q(q-1)(t+t'-1)}{2}\mathbf{Adv}^{\mathsf{rka\text{-}prf}}_{f(K,\star),\Phi_2}(E_{D_A}) + \tfrac{q(q-1)}{2^{n+1}},$$

here, B_A, C_A and D_A are defined in Fig. 19, and E_{D_A} is defined in Fig. 20. $\Phi_1 = \{\phi_{\mathsf{ipad}}, \phi_{\mathsf{opad}}\}$ where $\phi_{\mathsf{ipad}}(x) = x \oplus \mathsf{ipad}$ and $\phi_{\mathsf{opad}}(x) = x \oplus \mathsf{opad}$. B_A can only make two $(IV, \phi_{\mathsf{ipad}})$ and $(IV, \phi_{\mathsf{opad}})$ queries, and C_A can make at most q queries. For any output (M, M') of D_A, $||\mathsf{pad}_1(M)||_b \leq t$ and $||\mathsf{pad}_1(M')||_b \leq t'$. $t^* = max(t, t')$, $\Phi_2 = \{\phi_1, \cdots, \phi_{t^*}, \phi_P\}$ where $\phi_i(x) = x \oplus i$. And E_{D_A} can only make at most two (M_i, ϕ) and (M'_j, ϕ') queries, where M_i and M'_j are any value of b-bit, and $\phi, \phi' \in \Phi_2$.

7.2 PRF Constructions based on ARIRANG

We propose two alternative PRF constructions. One is the case that IV of ARIRANG is replaced with a key. The other is the case that the first message block of ARIRANG is made from a key like the first message block of HMAC-ARIRANG. In Fig. 10, $K^* = K$ is the key. In Fig. 11, $K^* = K||10^t$, where t is the smallest non-negative integer such that $|K||10^t|$ is a multiple of the bit-size of a message block b. In cases of ARIRANG-224 and 256, b=512, in cases of ARIRANG-384 and 512, b=1024.

We prove that two constructions are PRFs under some assumptions of the compression function. For the details of proof, see the Appendix A.4. The security proofs of them are similar to those of [4].

Security of PRF Construction 1. For any prf-adversary A with q queries, there exist adversaries F_A and H_A as follows, where $\mathrm{pfCM}^0 - \mathrm{MD}^f_{\mathsf{pad}}(IV, K||\star)$, which is explained in the Appendix A.1, denotes the PRF Construction 1 in Fig. 10.

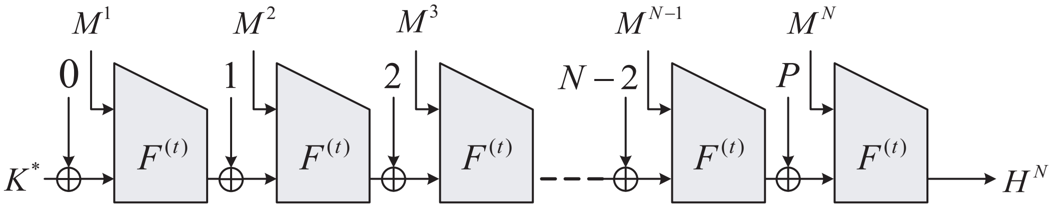

Figure 10: PRF Construction 1

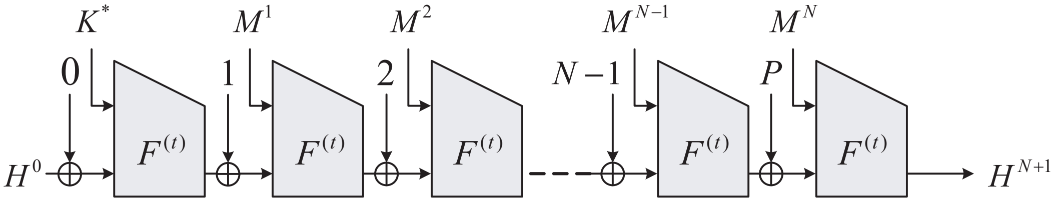

Figure 11: PRF Construction 2

$$\mathbf{Adv}^{\mathsf{prf}}_{\mathrm{pfCM}^0-\mathrm{MD}^f_{\mathsf{pad}}(IV,K||\star)}(A) \le l \cdot \mathbf{Adv}^{\mathsf{multi\text{-}rka\text{-}prf}}_{f(K,\star),\Phi_3}(H_A) + \mathbf{Adv}^{\mathsf{prf}}_{f(\star,K||0^{b-n})}(F_A),$$

where F_A and H_A are defined in Fig. 22. F_A can only make the query IV, and H_A can make at most q queries. We assume that for each query M of A, the b-bit block length of $\mathsf{pad}_1(M)$ is at most l. $\Phi_3 = \{\phi_1, \cdots, \phi_l, \phi_P\}$ where $\phi_i(X) = X \oplus i$ and P is the last counter of pfCM-MD. $\{\phi^1, \cdots, \phi^q\} \subset \{\phi_P, \phi_j\}$ for some j where (i^t, ϕ^t, X^t) is t-th query of H_A. In other words, even though H_A can make queries to any one of $\{O_1, O_2, \cdots, O_q\}$, H_A can use at most two related-key-deriving (RKD) functions ϕ's from Φ_3.

Security of PRF Construction 2. PRF Construction 2 in Fig. 11 corresponds to pfCM^0-MD$^f_{\mathsf{pad}}(IV, K||\star)$, which is explained in the Appendix A.1. Here, we only provide the security of pfCM^1-MD$^f_{\mathsf{pad}}(IV, K||\star)$, which is described in detail in the Appendix A.4. In the case of pfCM^0 $-$ MD$^f_{\mathsf{pad}}(IV, K||\star)$, its security proof can be done in the same way.

For any prf-adversary A with q queries, there exists an adversary H_A such that

$$\mathbf{Adv}^{\mathsf{prf}}_{\mathrm{pfCM}^1-\mathrm{MD}^f_{\mathsf{pad}_1}(K,\star)}(A) = l \cdot \mathbf{Adv}^{\mathsf{multi\text{-}rka\text{-}prf}}_{f(K,\star),\Phi_3}(H_A),$$

where H_A is defined as above.

7.3 Randomized Hashing based on ARIRANG

Draft NIST SP 800-106 describes a randomizing hashing for digital signatures [18]. More precisely, Draft NIST SP 800-106 defines a randomization method for randomizing messages prior to hashing. That is, the randomized method works independently from a hash function. There is only a restriction on the hash function, which should process messages in the usual left-to-right order. ARIRANG is such an example. When $\mathbf{H} = \{H_r(IV, \star\}_{r\in\mathcal{R}}$ is a hash family, the security of the randomized hashing is measured by the following game : an adversary A chooses M in advance, then a random string r is given to A, and A tries to find (r', M') such that $H_r(IV, M) = H_{r'}(IV, M')$ and $(r, M) \neq (r', M')$. The measurement of this game is formally defined by the definition of eTCR (which is described in the Appendix A.1). In this Section, we show that ARIRANG with the randomizing hashing in the Draft NIST SP 800-106 is secure if the compression function meets a security assumption. For the details of proof, see the Appendix A.6.

Security of ARIRANG with the message randomization in NIST SP 800-106. For any eTCR-adversary A, there exists a SPR†-adversary B_A such that

$$\mathbf{Adv}_{\mathbf{H}}^{\mathsf{eTCR}}(A) \leq l \cdot \mathbf{Adv}_{\mathbf{H}}^{\mathsf{eSPR}^\dagger}(B_A),$$

where $\mathbf{H} = \{\mathrm{pfCM}^0 - \mathrm{MD}^f_{\mathsf{pad}}(IV, \mathsf{mr}(r, \star))\}_{r\in\cup_{80\leq i\leq 1024}\{0,1\}^i}$, and mr is the message randomization function in NIST SP 800-16. B_A is defined in Fig. 25 in the Appendix A.6. l is defined in Fig. 25 in the Appendix A.6.

In the Appendix A.6, it is shown that the eTCR security of ARIRANG with the message randomization in NIST SP 800-106 is reduced to the eSPR†[6] security of ARIRANG with the message randomization in NIST SP 800-106. When f is the compression function of ARIRANG, eSPR† security of ARIRANG with the message randomization in NIST SP 800-106 largely depends on the security of f. See the definition of eSPR†-advantage as follows.

Definition 7.1 [eSPR†-advantage]. Given a hash family $\mathbf{H} = \{H_r(IV, \star)\}_{r\in\mathcal{R}}$, for each r we let $H_r(IV, M)[i]$ be the input value of i-th compression function during the computation of $H_r(IV, M)$, that is, $H_r(IV, M)[i] = (c, m)$, where $c \in \{0, 1\}^n$, $m \in \{0, 1\}^b$, $M \in \{0, 1\}^*$, and $H_r : \{IV\} \times \{0, 1\}^* \to \{0, 1\}^n$ is based on a compression function $f : \{0, 1\}^n \times \{0, 1\}^b \to \{0, 1\}^n$. Then, for any eSPR†-adversary A, the eSPR†-advantage of A on a hash family $\mathbf{H}$ is defined as follows,

$$\begin{aligned}\mathbf{Adv}_{\mathbf{H}}^{\mathsf{eSPR}^\dagger}(A) = \Pr[(M, \mathtt{State}) \xleftarrow{\$} A; r \xleftarrow{\$} \mathcal{R}; i \xleftarrow{\$} [1, l]; (c', m') \xleftarrow{\$} A(i, r, M, \mathtt{State})\\ : (c, m) = H_r(IV, M)[i] \text{ and } (c, m) \neq (c', m') \text{ and } f(c, m) = f(c', m')],\end{aligned}$$

where $l = \mathsf{Len}_f(H_r(IV, M))$ is the number of computations of the compression function f when computing $H_r(IV, M)$ for any r, where M is generated by the adversary A.

Relation between a Security of Compression function of ARIRANG and eSPR† Security of ARIRANG with the message randomization in NIST SP 800-106. In the definition of eSPR†-advantage, the eSPR† security of $\mathbf{H}$ is very similar to the second preimage resistance (SPR) security of f. In the case of SPR security of f, given a random input (c, m), it should be difficult for any adversary to find a different (c', m') such that $f(c, m) = f(c', m')$. Here, (c, m) has $(n+b)$-bit entropy. On the other hand, in the case of eSPR† security of $\mathbf{H}$, given an input (c, m) (which is generated from a random string M and r, where r has $|r|$-bit entropy), it should be difficult for any adversary to find a different (c', m') such that $f(c, m) = f(c', m')$. In Sect. 6, it is already shown that the compression function of ARIRANG is secure against all known collision-finding attacks and the second-preimage finding attacks. Therefore, we insist that ARIRANG with the message randomization in NIST SP 800-106 is eTCR-secure.

[6] eSPR† is similar to eSPR, which is defined in [18].

8 Security of Cryptographic Applications based on ARIRANG

8.1 Security of Digital Signatures based on ARIRANG

FIPS PUB 186-2 describes digital signature standards (DSS), where a collision resistant hash function is required to guarantee the security of DSS. ARIRANG is designed to guarantee the security against all known collision-finding attacks by intensive security analysis in the previous section and all the parameters are the same as that of SHA-2. Therefore, ARIRANG is suitable for DSS.

8.2 Security of Key Derivation based on ARIRANG

NIST special publication 800-56A describes key derivation functions (KDF) based on a hash function. Any key derivation function is used to derive secret keying material from a shared secret. Secret keying material means a symmetric key, a secret initialization vector, or a master key which is used to generate other keys. The process of KDF in the document is as follows: (See the page 49 of NIST SP 800-56A for details.)

1. $reps = \lceil keydatalen/hashlen \rceil$.
2. If $reps > (2^{32} - 1)$, then ABORT : output an error indicator and stop.
3. Initialize a 32-bit, big-endian bit string *counter* as 00000001_{16}.
4. If $counter||Z||OtherInfo$ is more than *max_hash_inputlen* bits long, then ABORT : output an error indicator and stop.
5. For $i = 1$ to *reps* by 1, do the followings:
   (a) Compute $Hash_i$=H($counter||Z||OtherInfo$).
   (b) Increment *counter* (modulo 2^{32}), treating it as an unsigned 32-bit integer.
6. Let $Hhash$ be set to $Hash_{reps}$ if $(keydatalen/hashlen)$ is an integer, otherwise, let $Hhash$ be set to the ($keydatalen$ mod $hashlen$) leftmost bits of $Hash_{reps}$.
7. Set $DerivedKeyingMaterial = Hash_1||Hash_2|| \cdots ||Hash_{reps-1}||Hhash$.

In the above process, H is a hash function, Z is a shared secret, and $OtherInfo$ is known fixed value. *Counter* is a changeable input variable. Then, the concatenation of hash outputs is used as secret keying material. In the Appendix A.5, it is shown that the pseudorandomness of KDF-ARIRANG is reduced to the RKA-pseudorandomness and pseudorandomness of the compression function of ARIRANG. When f is the compression function of a hash function, two types of (RKA-)pseudorandomness of the compression function are required for the security of KDF-ARIRANG as follows, where each security definition is described in the Appendix A.1:

- **[R1]** $\mathbf{Adv}^{\textsf{multi-rka-prf}}_{f(K,\star),\Phi_4}(A)$ is *negligible* for any multi-rka-prf-adversary A, where A can use at most two related-key-deriving (RKD) functions ϕ's from Φ_4, where $\Phi_4 = \{\phi_1, \cdots, \phi_l, \phi_P\}$, $\phi_i(X) = X \oplus i$ and P is the last counter of ARIRANG, and we assume that $\{\phi^1, \cdots, \phi^q\} \subset$

$\{\phi_P, \phi_j\}$ for some j, where (i^t, ϕ^t, X^t) is t-th query of A. In other words, even though A can make queries to any one of $\{O_1, O_2, \cdots, O_q\}$, A can use at most two related-key-deriving (RKD) functions ϕ's from Φ_4.

- **[R2]** For any prf-adversary A with at most q queries, $\mathbf{Adv}^{\mathsf{prf}}_{f(IV, \star_{32}||K||\star_{b-n-32})}(A)$ is *negligible*, where $\star_i$ denotes any i-bit string and b is the bit size of a block.

Two types of pseudorandomness of the compression function of ARIRANG. In the previous section, we insist that the compression function is resistant against all known attacks. Our security analysis also implies that the compression function of ARIRANG is secure against above pseudorandomness-breaking attackers.

8.3 Security of Hash-based Message Authentication Codes based on ARIRANG

FIPS PUB 198 describes the keyed-hash message authentication code (HMAC). In the Appendix A.3, it is shown that the pseudorandomness of HMAC-ARIRANG is reduced to the RKA-pseudorandomness and pseudorandomness of the compression function of ARIRANG. When f is the compression function of ARIRANG, three types of (RKA-)pseudorandomness of the compression function are required for the security of HMAC-ARIRANG as follows, where each security definition is described in the Appendix A.1:

- **[R1]** $\mathbf{Adv}^{\mathsf{rka\text{-}prf}}_{f(\cdot, K||0^{b-n}), \Phi_1}(A)$ is *negligible* for any rka-prf-adversary A, where A can only make two $(IV, \phi_{\mathsf{ipad}})$ and $(IV, \phi_{\mathsf{opad}})$ queries, and $\Phi_1 = \{\phi_{\mathsf{ipad}}, \phi_{\mathsf{opad}}\}$ where $\phi_{\mathsf{ipad}}(x) = x \oplus \mathsf{ipad}$ and $\phi_{\mathsf{opad}}(x) = x \oplus \mathsf{opad}$

- **[R2]** For any prf-adversary A with at most q queries, $\mathbf{Adv}^{\mathsf{prf}}_{f(K,\cdot)}(A)$ is *negligible.*

- **[R3]** $\mathbf{Adv}^{\mathsf{rka\text{-}prf}}_{f(K,\cdot), \Phi_2}(A)$ is *negligible* for any rka-prf-adversary A, where A can only make at most two (M_i, ϕ) and (M'_j, ϕ') queries, where M_i and M'_j are any value of b-bit, and $\phi, \phi' \in \Phi_2$, and $\Phi_2 = \{\phi_1, \cdots, \phi_{t^*}, \phi_P\}$ where $\phi_i(x) = x \oplus i$, and $t^* = max(t, t')$, where $||\mathsf{pad}_1(M)||_b = t$ and $||\mathsf{pad}_1(M')||_b = t'$.

Three types of pseudorandomness of the compression function of ARIRANG. In the previous section, we insist that the compression function is resistant against all known attacks. Our security analysis also implies that the compression function of ARIRANG is secure against above pseudorandomness-breaking attackers.

8.4 Security of Deterministic Random Bit Generators based on ARIRANG

NIST special publication 800-90 describes two deterministic random bit generators from a hash function. One is Hash_DRBG and the other is HMAC_DRBG. In the case of Hash_DRBG, counters are used for generating pseudorandom bits, where each counter is xored with secret internal values. So, the pseudorandomness of the output string of Hash_DRBG largely depends on counters. On the other hand, in the case of HMAC_DRBG, once a random seed is determined, the value of the key of HMAC is not changed till the seed is updated. Therefore, the pseudorandomness of the output string of HMAC_DRBG can be reduced to the pseudorandomness of HMAC.

Security of Hash_DRBG based on ARIRANG. Once a random seed is fixed, the maximum bit-length of output pseudorandom string is bounded by 2^{19}. In other words, the random seed should be changed before the output string exceeds 2^{19} bits. For example, in the case of ARIRANG-256, since ARIRANG-256 outputs a 256-bit hash value for each counter, the counter is bounded by 2^{11}. Likewise, the counter is bounded by 2^{10} in the case of ARIRANG-512. More precisely, an output pseudorandom bit string is leftmost W bits of $H(Z)||H(Z+1)||\cdots||H(Z+2^{11}-1)$, where Z is a random secret value determined by the random seed, H is ARIRANG, and W is the required length of the pseudorandom bit string. For example, when $|Z| = 256$, the computation of each $H(Z+i)$ is that of one compression function. As described in Sect. 6, we have done random tests for security of Hash_DRBG based on ARIRANG. Therefore, based on the random tests, we argue that Hash_DRBG-ARIRANG is a secure deterministic random bit generator.

Security of HMAC_DRBG based on ARIRANG. The security of HMAC_DRBG-ARIRANG depends on the pseudorandomness of HMAC-ARIRANG [20]. Since we have shown the security of HMAC-ARIRANG in Sect. 7 and 8, we argue that HMAC_DRBG-ARIRANG is a secure deterministic random bit generator.

9 Implementation and Efficiency

9.1 Software Implementation

ARIRANG is suited to be efficiently implemented on a wide range of processors. We implemented on 8-bit processors for sensor network, and on 32-bit and 64-bit processors for PCs. ARIRANG consists of SubBytes (table lookup), MDS, XOR ($\oplus$), and left rotation ($\lll$).

9.2 8-bit processors

ARIRANG was implemented on the MICAz sensor mote, with an ATmega128L 8-bit microprocessor, 7.37 MHz clock speed, 128 KByte flash ROM and 4 KByte SRAM running on TinyOS 1.1.0Jan. Using Nesc 1.1.2a, we performed simulations on ARIRANG.

To reduce the operation of MDS transformation, we used four S-boxes: $\{01\} \cdot S\text{-box}(i)$, $\{02\} \cdot S\text{-box}(i)$, $\{04\} \cdot S\text{-box}(i)$, $\{08\} \cdot S\text{-box}(i)$. In the 8-bit processor, the speed of table look-up is normally slower than a finite field multiplication.

The code size and required memory are given in Table 7. For a strict estimation, the code and memory for TinyOS and communication are excluded.

The execution times of ARIRANG using one S-box are given in Table 8 (the measurements are based on cycles per message or on cycles per byte for various message lengths and digest lengths).

The execution times of ARIRANG using four S-boxes are given in Table 9.

Our simulation results offer the fact that the implementation using one S-box is better than that using four S-boxes in terms of memory and efficiency. The algorithm set-up time is the

Table 7: Code size and required memory of ARIRANG on an 8-bit processor

Algorithm	ROM (Byte)	RAM (Byte)
ARIRANG using one S-box	34684	448
ARIRANG using four S-box	40940	1216

Table 8: Execution times of various message lengths for ARIRANG using one S-box on an 8-bit processor

Algorithm	Message (Byte)	Cycles/Msg	Cycles/Byte	Message (Byte)	Cycles/Msg	Cycles/Byte
ARIRANG-224	8	41170	5146	16	41178	2574
	32	41652	1302	64	81234	1269
	128	121252	947	1024	681924	666
ARIRANG-256	8	41242	5155	16	41254	2578
	32	41724	1304	64	81286	1270
	128	121364	948	1024	682140	666
ARIRANG-384	8	96716	12090	16	96756	6047
	32	97185	3037	64	97752	1527
	128	192934	1507	1024	865823	846
ARIRANG-512	8	97031	12129	16	96985	6062
	32	97487	3046	64	97958	1531
	128	193158	1509	1024	866383	846

Table 9: Execution times of various message lengths for ARIRANG using four S-boxes on an 8-bit processor

Algorithm	Message (Byte)	Cycles/Msg	Cycles/Byte	Message (Byte)	Cycles/Msg	Cycles/Byte
ARIRANG-224	8	43436	5430	16	43452	2716
	32	43922	1373	64	85739	1340
	128	128055	1000	1024	720518	704
ARIRANG-256	8	43508	5439	16	43524	2720
	32	43994	1375	64	85811	1341
	128	128127	1001	1024	720591	704
ARIRANG-384	8	116921	14615	16	116926	7308
	32	117407	3669	64	117907	1842
	128	233248	1822	1024	1047665	1023
ARIRANG-512	8	117196	14649	16	117262	7329
	32	117682	3678	64	118182	1847
	128	233522	1824	1024	1047940	1023

time required to generate S-box. In an 8-bit implementation, algorithm set-up takes no time because S-box was precomputed and internally stored.

9.3 32-bit processors

The SubBytes and MDS transformations can be expressed with XOR operations and table lookups as in the optimized AES code. Thus, in the case of ARIRANG-256, the SubBytes and MDS transformations are simultaneously implemented with 3 XORings and 4 table lookups whose input and output sizes are 8 bits and 32 bits, respectively. In the case of ARIRANG-512, the SubBytes and MDS transformations are simultaneously implemented with 28 XORings and 16 table lookups whose input and output sizes are also 8 bits and 32 bits, respectively. This implementation technique allows a high efficiency on processors with word size 32 bits or above.

We performed simulations on ARIRANG (along with SHA-256) with the following platform: Intel personal computer, with an Intel Core 2 Duo Processor, 2.53 GHz clock speed, 2 GB RAM, running Windows Vista Ultimate 32-bit (x86) Edition. Using the ANSI C compiler in the Microsoft Visual Studio 2005 Professional Edition, we executed 10 simulations: in each simulation various messages from 8 bytes to 10 megabytes are hashed 1,000,000 times to check the throughput of ARIRANG (our simulation results below are on the average case).

The simulation results of ARIRANG-256 and -224 are almost same, because two algorithms are same except for their initial values and the message digest sizes. This fact is also applied to ARIRANG-512 and -384. Table 10 lists our implementation results for ARIRANG and for SHA-256 on a 32-bit processor. Our results show that ARIRANG-256 and -224 are suited to implement on a 32-bit processor.

Table 10: Speed comparison between ARIRANG family and SHA-256 on a 32-bit processor

Algorithm	ARIRANG-224	ARIRANG-256	ARIRANG-384	ARIRANG-512	SHA-256
Mbps	893	899	297	296	712

As illustrated in Figure 12, the speed of ARIRANG-256 is about 25% faster than SHA-256.

The execution times of ARIRANG and SHA-256 in cycles per message and cycles per byte for various message lengths and digest lengths are given in Table 11.

To reduce memory usage, we also implemented for the following additional two versions.

- **Version using one S-box**. it uses one original S-box whose input and output sizes are 8 bits for SubBytes. The MDS transformation is computed through the field multiplication.
- **Version using seven S-boxes**. it uses seven S-boxes whose input and output sizes are 8 bits for the MDS transformation, $\{01\} \cdot S\text{-box}(i)$, $\{02\} \cdot S\text{-box}(i)$, $\{03\} \cdot S\text{-box}(i)$, $\{04\} \cdot S\text{-box}(i)$, $\{08\} \cdot S\text{-box}(i)$, $\{09\} \cdot S\text{-box}(i)$, $\{0A\} \cdot S\text{-box}(i)$.

In the case of ARIRANG-256, it recorded 22 cycles per byte in optimized version, 93 cycles per byte in one S-box version and 48 cycles per byte in seven S-box version. Figure 14 represents our implementation result of each version of ARIRANG-256. The rest of our results are given in Appendix C.

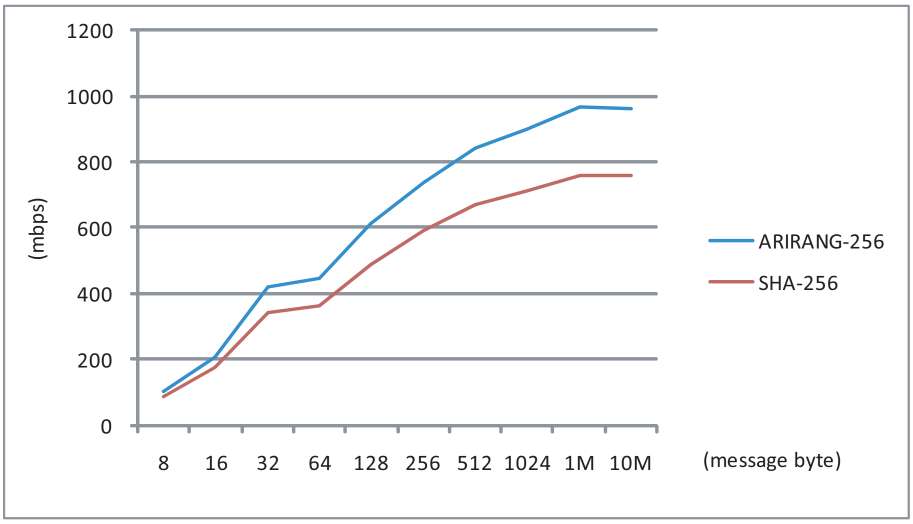

Figure 12: Performance of ARIRANG-256 and SHA-256 on a 32-bit processor

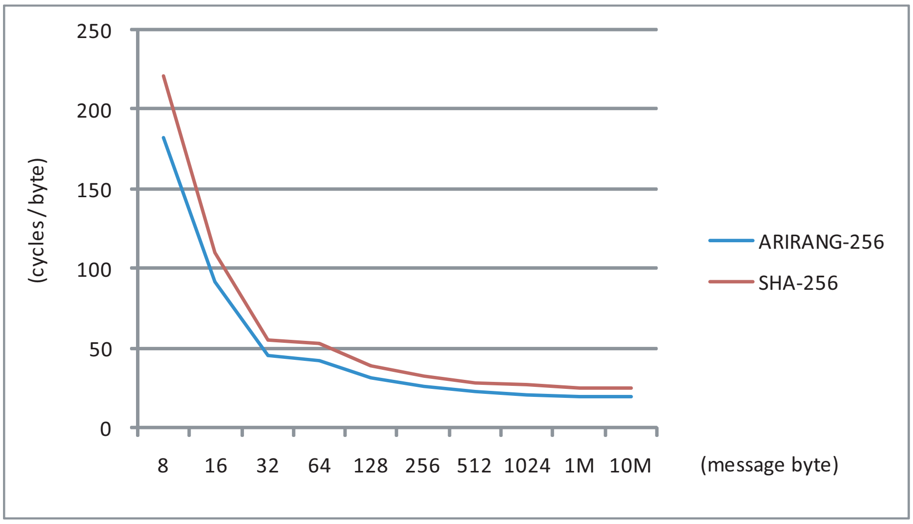

Figure 13: Cycles per byte of ARIRANG-256 and SHA-256

Table 11: Execution times of various message sizes for ARIRANG and SHA-256 on a 32-bit processor

Algorithm	Message (Byte)	Cycles/Msg	Cycles/Byte	Message (Byte)	Cycles/Msg	Cycles/Byte
ARIRANG-224	8	1462.3	182.8	16	1462.3	91.4
	32	1481.3	46.3	64	2727.3	42.6
	128	4051.8	31.7	256	6758.9	26.4
	512	11759.4	23	1024	22137.5	21.6
	1M	20981290	20	10M	210306250	20.1
ARIRANG-256	8	1462.3	182.8	16	1482.6	92.7
	32	1463.6	45.7	64	2747.6	42.9
	128	4012.6	31.3	256	6700.7	26.2
	512	11740.5	22.9	1024	21998.4	21.5
	1M	20961050	20	10M	210306250	20.1
ARIRANG-384	8	9269.9	1158.7	16	9348.4	584.3
	32	9171.3	286.6	64	9210.5	143.9
	128	16405.8	128.2	256	23383.5	91.3
	512	37594.5	73.4	1024	66630.1	65.1
	1M	58011635	55.3	10M	579129650	55.2
ARIRANG-512	8	9507.7	1188.5	16	9528	595.5
	32	9409.1	294	64	9388.8	146.7
	128	16543.7	129.2	256	23501.2	91.8
	512	37771.6	73.8	1024	66768	65.2
	1M	58021755	55.3	10M	579129650	55.2
SHA-256	8	1767	220.9	16	1771	110.7
	32	1786.7	55.8	64	3415.5	53.4
	128	5044.1	39.4	256	8337.1	32.6
	512	14752.9	28.8	1024	27758.7	27.1
	1M	26592830	25.4	10M	266679710	25.4

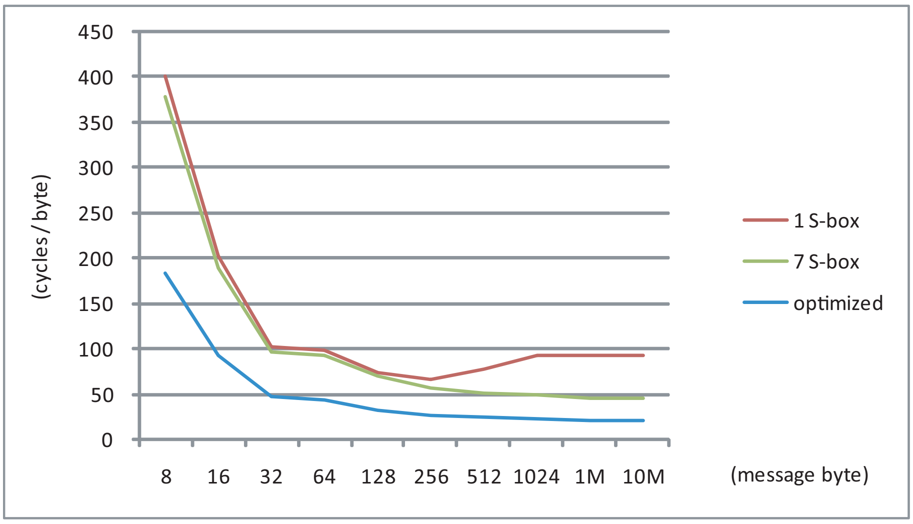

Figure 14: Cycles per byte in each version of ARIRANG-256

In each version, a required memory for S-box and algorithm set-up time are given in Table 12. If S-box was precomputed and stored internally like an 8-bit implementation, algorithm set-up takes no time.

Table 12: Set-up time and required memory of ARIRANG family on a 32-bit processor

Algorithm	Cycles	Required memory (Bytes)
ARIRANG family (optimized)	35660	20480
ARIRANG family using one S-box	4796	256
ARIRANG family using seven S-box	11945	1792

9.4 64-bit processors

In a 64-bit implementation, in the case of ARIRANG-256, the SubBytes and MDS transformations are simultaneously implemented with 3 XORings and 4 table lookups whose input and output sizes are 8 bits and 32 bits, respectively. In the case of ARIRANG-512, the SubBytes and MDS transformations are simultaneously implemented with 7 XORings and 8 table lookups whose input and output sizes are 8 bits and 64 bits, respectively.

ARIRANG was implemented on the same platform with the 32-bit platform, but running on Windows Vista Ultimate 64-bit (x64) Edition.

Table 13 lists our implementation results of ARIRANG and SHA-512 on a 64-bit processor.

The simulation results of ARIRANG-512 and -384 are almost same, showing that ARIRANG-512 and -384 are designed to be optimized on a 64-bit processor. As illustrated in Figure 15, the speed of ARIRANG-512 is about 25% faster than SHA-512.

Execution times of ARIRANG in cycles per message and cycles per byte for various message lengths and digest lengths are given in Table 14.

Table 13: Speed comparison of ARIRANG and SHA-512 on a 64-bit processor

Algorithm	ARIRANG-224	ARIRANG-256	ARIRANG-384	ARIRANG-512	SHA-512
Mbps	1197	1198	1503	1497	1195

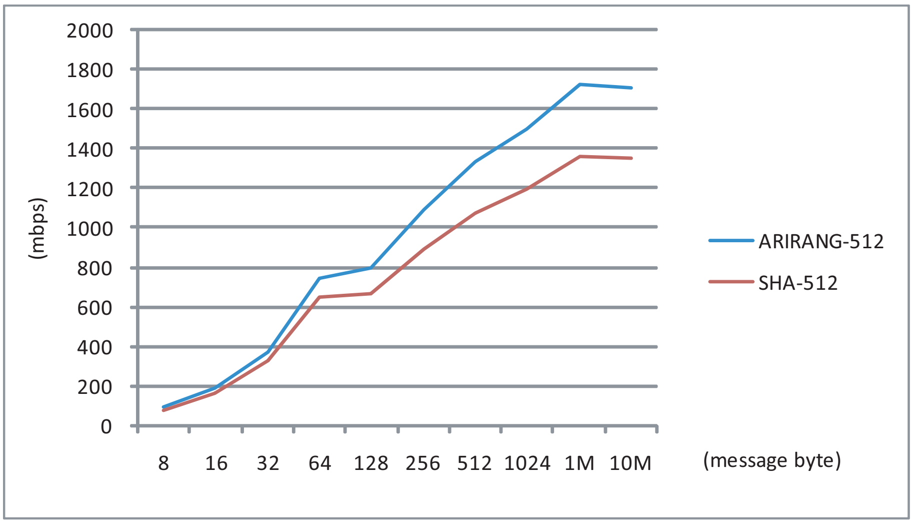

Figure 15: Performance of ARIRANG-512 and SHA-512

Similarly to the 32-bit implementation, we implemented for additional two versions. In the case of ARIRANG-512, it recorded 11 cycles per byte in optimized version, 344 cycles per byte in one S-box version and 204 cycles per byte in seven S-box version. Figure 16 represents our implementation result of each version of ARIRANG-512. The rest of our simulation results are given in Appendix C.

In each version, a required memory for S-box and algorithm set-up time of ARIRANG are given in Table 15. If S-box was precomputed and stored internally like the 8-bit and 32-bit implementations, algorithm set-up takes no time.

References

[1] E. Andreeva, C. Bouillaguet, P. Fouque, J. J. Hoch, J. Kelsey, A. Shamir and S. Zimmer, *Second Preimage Attacks on Dithered Hash Functions*, Advances in Cryptology – EUROCRYPT'08, LNCS 4965, Springer-Verlag, pp. 270–288, 2008.

[2] G. Bertoni, J. Daemen, M. Peeters and G. V. Assche, *On the Indifferentiability of the Sponge Construction*, Advances in Cryptology – EUROCRYPT'08, LNCS 4965, Springer-Verlag, pp. 181–197, 2008.

[3] M. Bellare, *N*ew Proofs for NMAC and HMAC: Security without Collision-Resistance, Advances in Cryptology – CRYPTO'06, LNCS 4117, Springer-Verlag, pp. 602–619, 2006.

Table 14: Execution times of various message sizes for ARIRANG and SHA-512 on a 64-bit processor

Algorithm	Message (Byte)	Cycles/Msg	Cycles/Byte	Message (Byte)	Cycles/Msg	Cycles/Byte
ARIRANG-224	8	1154.2	144.3	16	1138.5	71.2
	32	1154.2	36.1	64	2118.6	33.1
	128	3083.1	24.1	256	5004.3	19.5
	512	8855	17.3	1024	16515.8	16.1
	1M	15603016	14.9	10M	157138300	15
ARIRANG-256	8	1154.2	144.3	16	1178	73.6
	32	1154.2	36.1	64	2111	33
	128	3067.9	24	256	5036.2	19.7
	512	8859	17.3	1024	16492.6	16.1
	1M	15603016	14.9	10M	156822050	15
ARIRANG-384	8	1652.6	206.6	16	1730.5	108.2
	32	1644.5	51.4	64	1636.4	25.6
	128	3083.6	24.1	256	4491.3	17.5
	512	7384.3	14.4	1024	13147.9	12.8
	1M	11732875	11.2	10M	118317980	11.3
ARIRANG-512	8	1684	210.5	16	1675.9	104.7
	32	1675.9	52.4	64	1660.7	25.9
	128	3122.5	24.4	256	4538.3	17.7
	512	7404.3	14.5	1024	13203.6	12.9
	1M	11740971	11.2	10M	118355930	11.3
SHA-512	8	1928.9	241.1	16	1897.5	118.6
	32	1881.6	58.8	64	1897.5	29.6
	128	3719.9	29.1	256	5550.1	21.7
	512	9226.4	18	1024	16535.8	16.1
	1M	14907519	14.2	10M	150059360	14.3

Table 15: Set-up time and required memory of ARIRANG on a 64-bit processor

Algorithm	Cycles	Required memory (Bytes)
ARIRANG family (optimized)	32937	20480
ARIRANG family using one S-box	4723	256
ARIRANG family using seven S-box	11892	1792

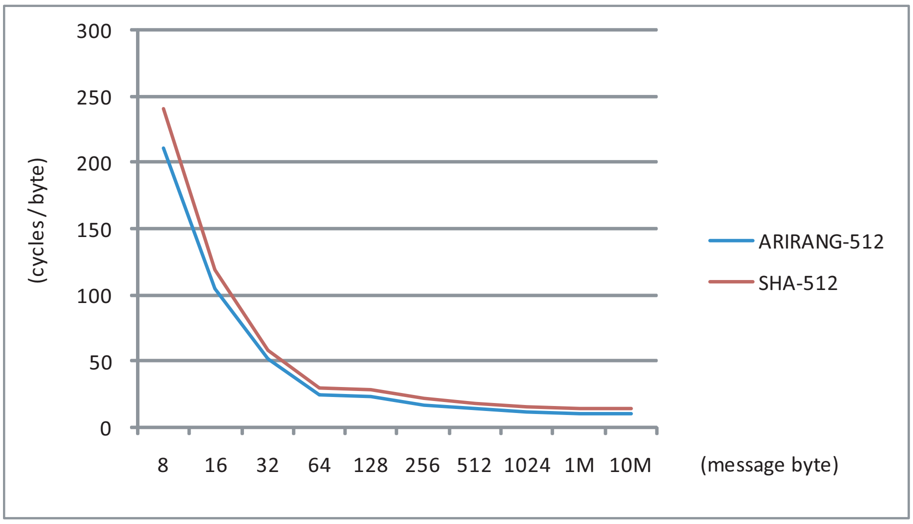

Figure 16: Cycles per byte of ARIRANG-512 and SHA-512

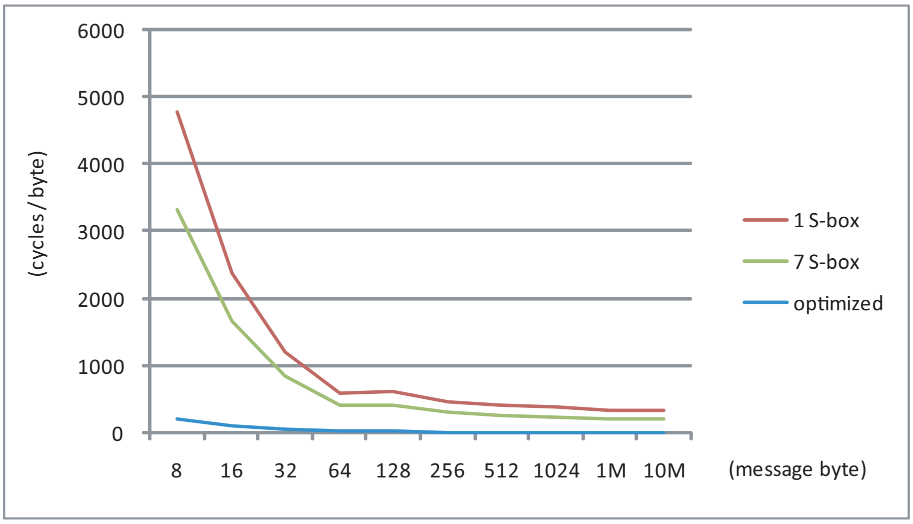

Figure 17: Cycles per byte of ARIRANG-512 and SHA-512

[4] M. Bellare, R. Canetti and H. Krawczyk, *Pseudorandom Functions Revisited: The Cascade Construction and its Concrete Security*, In the preceedings of the 37th Symposium on Foundations of Computer Science, IEEE, 1996.

[5] M. Bellare, R. Canetti and H. Krawczyk, *Keying Hash Functions for Message Authentication*, Advances in Cryptology – CRYPTO'96, LNCS 1109, Springer-Verlag, pp. 1–15, 1996.

[6] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, *On the Indifferentiability of the Sponge Construction*, Advances in Cryptology – EUROCRYPT'08, LNCS 4965, Springer-Verlag, pp. 181–197, 2008.

[7] M. Bellare and T. Kohno, *A Theoretical Treatment of Related-Key Attacks: RKA-PRPs, RKA-PRFs, and Applications*, Advances in Cryptology – EUROCRYPT'2003, LNCS 2656, Springer-Verlag, pp. 491–506, 2003.

[8] M. Bellare and P. Rogaway, *Random Oracles Are Practical : A Paradigm for Designing Efficient Protocols*, In 1st Conference on Computing and Communications Security, ACM, pages 62–73, 1993.

[9] M. Bellare and P. Rogaway, *The game-playing technique and its application to triple encryption*, Cryptology ePrint Archive: Report 2004/331, 2004.

[10] E. Biham and R. Chen, *Near-Collisions of SHA-0*, Advances in Cryptology – CRYPTO'04, LNCS 3152, pp. 290-305, 2004. /smallskip

[11] E. Biham, R, Chen, A. Joux, P. Carribault, C. Lemuet and W. Jalby, *Collisions of SHA-0 and Reduced SHA-1*, Advances in Cryptology – Eurocrypt'05, LNCS 3493, pp. 36-57. 2005. /smallskip

[12] B. den Boer and A. Bosselaers, *Collisions for the compression function of MD5*, Advances in Cryptology – Eurocrypt'93, LNCS 765, Springer Verlag, pp. 293–304, 1994./smallskip

[13] D. Chang and M. Nandi, *Improved Indifferentiability Security Proof of chopMD Hash Function*, FSE'2008, LNCS 5086, Springer-Verlag, pp. 429–443, 2008.

[14] D. Chang, J. Sung, S. Hong and S. Lee, *Indifferentiable Security Analysis of choppfMD, chopMD, a chopMDP, chopWPH, chopNI, chopEMD, chopCS, chopESh, a pfCM-chopMD Hash Domain Extensions*, Cryptology ePrint Archive: Report 2008/407, 2008.

[15] J. S. Coron, Y. Dodis, C. Malinaud and P. Puniya, *Merkle-Damgård Revisited: How to Construct a Hash Function*, Advances in Cryptology – CRYPTO'05, LNCS 3621, Springer-Verlag, pp. 430–448, 2005.

[16] I. B. Damgard, *A design principle for hash functions*, Advances in Cryptology – CRYPTO'89, LNCS 435, Springer-Verlag, pp. 416–427, 1990.

[17] M. Gorski, S. Lucks, T. Peyrin, *Slide Attacks on Hash Functions*, Cryptology ePrint Archive: Report 2008/263, 2008.

[18] S. Halevi and H. Krawczyk, *Strengthening Digital Signatures via Randomized Hashing*, Advances in Cryptology – CRYPTO'06, LNCS 4117, Springer-Verlag, pp. 41–59, 1996.

[19] S. Hirose, J. H. Park and A. Yun, *A Simple Variant of the Merkle-Damgård Scheme with a Permutation*, Advances in Cryptology – ASIACRYPT'07, LNCS 4833, Springer-Verlag, pp. 113–129, 2007.

[20] S. Hirose, *Security Analysis of DRBG Using HMAC in NIST SP 800-90*, WISA'08, to appear.

[21] D. Hong, D. Chang, J. Sung, S. Lee, S. Hong, J. Lee, D. Moon and S. Chee, *A New Dedicated 256-bit Hash Function : FORK-256*, FSE'06, LNCS 4047, Springer-Verlag, pp. 195–209, 2006.

[22] A. Joux, *Multicollisions in iterated hash functions. Application to cascaded constructions*, Advances in Cryptology – CRYPTO'04, LNCS 3152, Springer-Verlag, pp. 306–316, 2004.

[23] J. Kelsey and T. Kohno, *Herding Hash Functions and the Nostradamus Attack*, Advances in Cryptology – EUROCRYPT'06, LNCS 4004, Springer–Verlag, pp. 183-200, 2006.

[24] J. Kelsey and B. Schneier, *Second preimages on n-bit hash functions for much less than 2^n work*, Advances in Cryptology – EUROCRYPT'05, LNCS 3494, Springer-Verlag, pp. 474–490, 2005.

[25] X. Lai and J. L. Massey, *Hash Function Based on Block Ciphers*, Advances in Cryptology – EUROCRYPT'92, LNCS 658, Springer-Verlag, pp. 55–70, 1993.

[26] U. Maurer, R. Renner and C. Holenstein, *Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology*, TCC'04, LNCS 2951, Springer-Verlag, pp. 21–39, 2004.

[27] R. C. Merkle, *One Way Hash Functions and DES*, Advances in Cryptology – CRYPTO'89, LNCS 435, Springer-Verlag, pp. 428–446, 1990.

[28] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray and S. Vo, *A Statistical Test Suite for Random and Peudorandom Number Generators for Cryptographic Applications*, NIST Special Publication 800-22, 2001.

[29] X. Wang, X. Lai, D. Feng, H. Chen and X. Yu, *Cryptanalysis of the Hash Functions MD4 and RIPEMD*, Advances in Cryptology – EUROCRYPT'05, LNCS 3494, Springer-Verlag, pp. 1-18, 2005.

[30] L. Wang, K. Ohta and N. Kunihiro, *New Key-Recovery Attacks on HMAC/NMAC-MD4 and NMAC-MD5*, Advances in Cryptology – EUROCRYPT'08, LNCS 4965, Springer-Verlag, pp. 237–253, 2008.

[31] X. Wang and H. Yu, *How to Break MD5 and Other Hash Functions*, Advances in Cryptology – EUROCRYPT'05, LNCS 3494, Springer-Verlag, pp. 19–35, 2005.

[32] X. Wang, H. Yu and Y. L. Yin, *Efficient Collision Search Attacks on SHA-0*, Advances in Cryptology – CRYPTO'05, LNCS 3621, Springer-Verlag, pp. 1–16, 2005.

[33] X. Wang, Y. L. Yin and H. Yu, *Finding Collisions in the Full SHA-1*, Advances in Cryptology – CRYPTO'05, LNCS 3621, Springer-Verlag, pp. 17–36, 2005.

[34] Y. Sasaki, L. Wang, K. Ohta and N. Kunihiro,*Password Recovery on Challenge and Response: Impossible Differential Attack on Hash Function*, AFRICACRYPT'08, LNCS 5023, pp. 290–307, 2008. /smallskip

Appendix A

We denote the domain extension of ARIRANG by *a prefix-free-Counter-Masking-MD (pfCM-MD)*. In Appendix A.1, we introduce notations, definitions, and known results for security proofs. In Appendix A.2, we give the indifferentiable security proof on the *pfCM-MD*. In Appendix A.3, we provide a prf security of HMAC based on the *pfCM-MD*. In Appendix A.4, we define two prf constructions based on the *pfCM-MD* and prove the prf security of them. In Appendix A.5, we provide a prf security of NIST SP 800-56A key derivation function based on the *pfCM-MD*. In Appendix A.6, we provide eTCR security analysis of *pfCM-MD* with the message randomization function (in short, mr) in NIST SP 800-16. Our proof technique and most of notations follow those in [6, 3, 4].

A.1. Notations, Definitions and Known Results

Here we consider the compression function $f : \{0,1\}^n \times \{0,1\}^b \to \{0,1\}^n$. We write $||m||_b = k$ if $m \in \{0,1\}^{kb}$. That is, m is a message of k b-bit blocks. We denote the set of all functions from the domain $\mathcal{C}$ to the codomain $\mathcal{D}$ by $\mathsf{Maps}(\mathcal{C}, \mathcal{D})$.

Padding. We say any injective and length-consistent function $\mathsf{pad} : \{0,1\}^* \to (\{0,1\}^b)^*$ as a padding rule.

MD [27, 16]. The traditional Merkle-Damgård extension (MD) works as follow: for a message M, $\mathsf{pad}(M) = m_1||\cdots||m_t$ and $\mathrm{MD}^f_{\mathsf{pad}}(IV, M) = f(\cdots f(f(IV, m_1), m_2)\cdots, m_t)$, where f is a compression function and IV is the initial value.

pfCM-MD. CM-MD (MD with a counter-masking) works similar to MD as follow : for given a message M, $\mathsf{pad}(M) = m_1||\cdots||m_t$ and CM-MD$^f_{\mathsf{pad}}(IV, M)$=CM-MD$^f(IV,\mathsf{pad}(M)) = f(\cdots f(\, f(\, IV \oplus c_0, m_1) \oplus c_1, m_2) \oplus c_3, \cdots, m_t)$. For any two $c = c_0||\cdots||c_{t-1}$ and $c' = c'_0||\cdots||c'_{t'-1}$, if c is not a prefix of c', then we say its counter-masking is prefix-free. So, pfCM-MD means prefix-free-Counter-Masking-MD. In [14], they considered a special case that for any $c = c_0||\cdots||c_{t-1}$, where $c_0 = 0$ and $c_{i+1} = c_i + 1$ for $0 \le i \le t-3$ and $c_{t-1} = P$, where P is a fixed value bigger than other counter c_j's. For example, when the maximum bit-size of an input message is $2^{64}-1$, P can be any value larger than or equal to 2^{64}. When the maximum bit-size of an input message is $2^{128}-1$, any value P can be any value larger than or equal to 2^{128}. In this document, in the case that $c_0 = d$ and $c_{i+1} = c_i+1$ for $0 \le i \le t-3$ and $c_{t-1} = P$, we denote it by pfCM^d-MD.

chop. For $0 \le s \le n$ we define $\text{chop}_s(x) = x_L$ where $x = x_L \parallel x_R$ and $|x_R| = s$.

pfCM-chopMD. $\text{pfCM-chopMD}^f_{\mathsf{pad}}(IV, M) = \text{chop}_s(\text{pfCM-MD}^f_{\mathsf{pad}}(IV, M))$. Note that pfCM-chopMD with $s = 0$ is the same as pfCM-MD. That is, pfCM-MD is a special case of pfCM-chopMD. So, in the Appendix A.2, we focus on providing an indifferentiable security proof of pfCM-chopMD with any s.

NMAC and HMAC [5]. Let K_1 and K_2 be n bits. $\overline{K} = K||0^{b-n}$. opad is formed by repeating the byte '0x36' as many times as needed to get a b-bit block, and ipad is defined similarly using the byte '0x5c'. Then, NMAC and HMAC are defined as follows, where H is any hash function.

$$\begin{aligned} \text{NMAC}^H(K_2||K_1, M) &= H(K_2, H(K_1, M)) \\ \text{HMAC}^H_{IV}(K, M) &= H(IV, \overline{K} \oplus \mathsf{opad}||H(IV, \overline{K} \oplus \mathsf{ipad}||M))). \end{aligned}$$

In this document, we consider the case that H is $\text{pfCM}^0\text{-MD}^f_{\mathsf{pad}}(\star, \star)$. And it is clear that for any pad, there exists pad_1 such that $\text{NMAC}^{\text{pfCM}^1-\text{MD}^f_{\mathsf{pad}_1}}(K_2||K_1, M)$= $\text{HMAC}^{\text{pfCM}^0-\text{MD}^f_{\mathsf{pad}}}_{IV}(K, M)$, where K_2=$f(IV, \overline{K} \oplus \mathsf{opad})$ and K_1=$f(IV, \overline{K} \oplus \mathsf{ipad})$. *And we assume that in the case of NMAC, the outer hash function uses the compression function one time, and in the case of HMAC, the outer hash function uses the compression function two times.*

Two PRF Constructions based on a pfCM-MD.

1. $\text{pfCM}^{\text{i}} - \text{MD}^f_{\mathsf{pad}}(K, \star)$, where $K \overset{\$}{\leftarrow} \{0,1\}^n$.

2. $\text{pfCM}^{\text{i}} - \text{MD}^f_{\mathsf{pad}}(IV, K||0^{b-n}|| \star)$, where $K \overset{\$}{\leftarrow} \{0,1\}^n$.

It is clear that for any pad, K, and any M, there exists pad_1 such that $\text{pfCM}^1 - \text{MD}^f_{\mathsf{pad}_1}(K', M)$ $= \text{pfCM}^0 - \text{MD}^f_{\mathsf{pad}}(IV, K||0^{b-n}||M)$, where $K' = f(IV, K||0^{b-n})$.

Inequality. The following inequality will be used to prove Theorem 9.4.

Ineq 1. For any $0 \le a_i \le 1$, $\prod_{i=1}^q (1-a_i) \ge 1 - \sum_{i=1}^q a_i$. One can prove it by induction on q.

Random Oracle Model : f is said to be *a random oracle* from X to Y if for each $x \in X$ the value of $f(x)$ is chosen randomly from Y [8]. More precisely, $\Pr[f(x) = y \mid f(x_1) = y_1, f(x_2) = y_2, \ldots f(x_q) = y_q] = \frac{1}{T}$, where $x \notin \{x_1, \ldots, x_q\}$, $y, y_1, \cdots, y_q \in Y$ and $|Y| = T$. In the case that $X = \{0,1\}^d$ for a fixed value d, we say f is a FIL (Fixed Input Length) random oracle. In the case that $X = \{0,1\}^*$, we say f is a VIL (Variable Input Length) random oracle. A VIL random oracle is usually denoted by R.

The cost of Queries. The security bound of a scheme is usually described using the number q of queries and the maximum length l of each queries. On the other hand, in [6], the notion *cost* is used to describe the security bound of sponge construction. The notion *cost* denotes the total block length of q queries. The notion *cost* is significant because the unit of time complexity corresponds to the time of an underlying function call and the total time complexity depends on how many the underlying function is called. The notion *cost* exactly reflects how many the underlying function is called. So, we can consider two cases. The first case is that the number of queries is bounded by q. The second case is that the cost of queries is bounded by q. Without loss of generality, for describing notions and some results in this section, we assume that the number of queries is bounded by q.

Computational Distance. Let $F = (F_1, F_2, \cdots, F_t)$ and $G = (G_1, G_2, \cdots, G_t)$ be tuples of probabilistic oracle algorithms. We define the computational distance of a probabilistic attacker A distinguishing F from G as

$$\mathbf{Adv}_A(F, G) = |\Pr[A^F = 1] - \Pr[A^G = 1]|.$$

Statistical Distance. Let $F = (F_1, F_2, \cdots, F_t)$ and $G = (G_1, G_2, \cdots, G_t)$ be tuples of probabilistic oracle algorithms. We define the statistical distance of a deterministic attacker A distinguishing F from G as

$$\mathbf{Stat}_A(F, G) = \frac{1}{2} \sum_{v \in V_A} |\Pr[F = v] - \Pr[G = v]|,$$

where $\Pr[O = v]$ denotes $\Pr[O(c_i, x_i) = y_i, 1 \le i \le q, v = ((c_1, x_1, y_1), \cdots, (c_q, x_q, y_q))]$, where $O(c_i, x_i) = O_{c_i}(x_i)$. And we let the maximum statistical distance of F and G against any deterministic algorithm A be $\mathbf{Stat}(F, G)$, where the number of queries of A is bounded by q.

Computational Distance vs. Statistical Distance

Lemma 9.1. Let $F = (F_1, F_2, \cdots, F_t)$ and $G = (G_1, G_2, \cdots, G_t)$ be tuples of probabilistic oracle algorithms. For any probabilistic algorithm A which can make at most q queries

$$\mathbf{Adv}_A(F, G) \le \mathbf{Stat}(F, G).$$

Indifferentiability

We give a brief introduction of the indifferentiable security notion.

Definition 9.1. Indifferentiability. [26] A Turing machine H with oracle access to an ideal primitive f is said to be $(t_D, t_S, q, \varepsilon)$ indifferentiable from an ideal primitive R if there exists a simulator S such that for any distinguisher D it holds that :

$$|\Pr[D^{H,f} = 1] - \Pr[D^{R,S} = 1] < \varepsilon$$

The simulator has oracle access to R and runs in time at most t_S. The distinguisher runs in time at most t_D and makes at most q queries. Similarly, H^f is said to be (computationally) indifferentiable from R if ε is a negligible function of the security parameter k (for polynomially bounded by t_D and t_S).

The following Theorem [26] shows the relation between indifferentiable security notion and the security of a cryptosystem.

Theorem 9.2. [26] Let $\mathcal{P}$ be a cryptosystem with oracle access to an ideal primitive R. Let H be an algorithm such that H^f is indifferentiable from R. Then cryptosystem $\mathcal{P}$ is at least as secure in the f model with algorithm H as in the R model.

Above theorem says that if a domain extension (with a padding rule) based on a FIL random oracle f is indifferentiable from a VIL random oracle R, then a cryptosystem, which is proved in the VIL random oracle model, can use the domain extension (with a padding rule) based on a FIL random oracle f instead of R with negligible loss of security.

Definition 9.2 [prf-advantage]. The prf-advantage of A on $f : \{0,1\}^n \times \{0,1\}^b \rightarrow \{0,1\}^n$ is defined by

$$\mathbf{Adv}^{\mathsf{prf}}_{f(K,\star)}(A) = |\Pr[K \xleftarrow{\$} \{0,1\}^n : A^{f(K,\star)} = 1] - \Pr[g \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^b, \{0,1\}^n) : A^{g(\star)} = 1]|.$$
$$\mathbf{Adv}^{\mathsf{prf}}_{f(\star,K||0^{b-n})}(A) = |\Pr[K \xleftarrow{\$} \{0,1\}^n : A^{f(\star,K||0^{b-n})} = 1] - \Pr[g \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^n, \{0,1\}^n) : A^{g(\star)} = 1]|,$$

For any function, its prf-advantage can be similarly defined.

Definition 9.3 [rka-prf-advantage [7]]. Let Φ_1 be a set of functions mapping $\{0,1\}^b$ to $\{0,1\}^b$ and let Φ_2 be a set of functions mapping $\{0,1\}^n$ to $\{0,1\}^n$. Let A be an adversary whose queries have the form (X, ϕ) where $X \in \{0,1\}^n$ and $\phi \in \Phi_1$, or the form (ϕ, X) where $X \in \{0,1\}^b$ and $\phi \in \Phi_2$. For $i = 1$ or 2, the rka-prf-advantage of A in a Φ_i-restricted related-key attack (RKA) on $f : \{0,1\}^n \times \{0,1\}^b \rightarrow \{0,1\}^n$ is defined by

$$\begin{aligned}\mathbf{Adv}^{\mathsf{rka\text{-}prf}}_{f(\star,RK(\star,K||0^{b-n})),\Phi_1}(A) &= |\Pr[K \xleftarrow{\$} \{0,1\}^n : A^{f(\star,RK(\star,K||0^{b-n}))} = 1] \\ &\quad - \Pr[g \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^{n+b}, \{0,1\}^n); K \xleftarrow{\$} \{0,1\}^n : A^{g(\star,RK(\star,K||0^{b-n}))} = 1]|, \\ \mathbf{Adv}^{\mathsf{rka\text{-}prf}}_{f(RK(\star,K),\star),\Phi_2}(A) &= |\Pr[K \xleftarrow{\$} \{0,1\}^n : A^{f(RK(\star,K),\star)} = 1] \\ &\quad - \Pr[g \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^{n+b}, \{0,1\}^n); K \xleftarrow{\$} \{0,1\}^n : A^{g(RK(\star,K),\star)} = 1]|,\end{aligned}$$

where in the first case, on query (X, ϕ) of A, the oracle $O(\star, RK(\star, K||0^{b-n}))$ returns the value of $O(X, \phi(K||0^{b-n}))$ to A, and in the second case, on query (ϕ, X) of A, the oracle $O(RK(\star, K), \star)$ returns the value of $O(\phi(K), X)$ to A.

Definition 9.4 [multi-rka-prf-advantage]. Let A be an adversary whose queries have the form (i, X, ϕ) where $X \in \{0,1\}^n$ and $\phi \in \Phi_1$, or the form (i, ϕ, X) where $1 \leq i \leq q$ and $X \in \{0,1\}^b$ and $\phi \in \Phi_2$. For $i = 1$ and 2, the multi-rka-prf-advantage of A in a Φ_i-restricted related-key attack (RKA) on $f : \{0,1\}^n \times \{0,1\}^b \rightarrow \{0,1\}^n$ is defined by

$$\begin{aligned}\mathbf{Adv}^{\mathsf{multi\text{-}rka\text{-}prf}}_{f(\star,RK(\star,K_\star||0^{b-n})),\Phi_1}(A) &= |\Pr[K_1, \cdots, K_q \xleftarrow{\$} \{0,1\}^n : A^{f(\star,RK(\star,K_\star||0^{b-n}))} = 1] \\ &\quad - \Pr[g_1, \cdots, g_q \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^{n+b}, \{0,1\}^n); K \xleftarrow{\$} \{0,1\}^n : A^{g_\star(\star,RK(\star,K||0^{b-n}))} = 1]|, \\ \mathbf{Adv}^{\mathsf{multi\text{-}rka\text{-}prf}}_{f(RK(\star,K_\star),\star),\Phi_2}(A) &= |\Pr[K_1, \cdots, K_q \xleftarrow{\$} \{0,1\}^n : A^{f(RK(\star,K_\star),\star)} = 1] \\ &\quad - \Pr[g_1, \cdots, g_q \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^{n+b}, \{0,1\}^n); K \xleftarrow{\$} \{0,1\}^n : A^{g_\star(RK(\star,K),\star)} = 1]|,\end{aligned}$$

where in the first case, on query (i, X, ϕ) of A, $f(\star, RK(\star, K_\star||0^{b-n}))$ returns $f(X, \phi(K_i||0^{b-n}))$ to A, and $g_\star(\star, RK(\star, K||0^{b-n}))$ returns $g_i(X, \phi(K||0^{b-n}))$ to A. The second case is also similarly defined.

Definition 9.5 [au-advantage [3]]. For any almost universal (au) adversary A, the au-advantage of A on $F(K, \star)$ is defined as follows, where $F : \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^n$.

$$\mathbf{Adv}^{\mathsf{au}}_{F(K,\star)}(A) = \Pr[K \xleftarrow{\$} \{0,1\}^n; (M \neq M') \xleftarrow{\$} A : F(K, M) = F(K, M')].$$

Definition 9.6 [eTCR-advantage [18]]. For any eTCR-adversary A, the eTCR-advantage of A on a hash family $\mathbf{H} = \{H_r(IV, \star)\}_{r\in\mathcal{R}}$ is as follows,

$$\mathbf{Adv}^{\mathsf{eTCR}}_{\mathbf{H}}(A) = \Pr[(M, \mathtt{State}) \xleftarrow{\$} A; r \xleftarrow{\$} \mathcal{R}; (r', M') \xleftarrow{\$} A(r, M, \mathtt{State}) \\ : (r, M) \neq (r', M') \text{ and } H_r(IV, M) = H_{r'}(IV, M')].$$

Definition 9.7 [eSPR†-advantage]. Given a hash family $\mathbf{H} = \{H_r(IV, \star)\}_{r\in\mathcal{R}}$, for each r we let $H_r(IV, M)[i]$ be the input value of i-th compression function during the computation of $H_r(IV, M)$, that is, $H_r(IV, M)[i] = (c, m)$, where $c \in \{0,1\}^n$, $m \in \{0,1\}^b$, $M \in \{0,1\}^*$, and $H_r : \{IV\} \times \{0,1\}^* \to \{0,1\}^n$ is based on a compression function $f : \{0,1\}^n \times \{0,1\}^b \to \{0,1\}^n$. Then, for any eSPR†-adversary A, the eSPR†[7]-advantage of A on a hash family $\mathbf{H}$ is defined as follows,

$$\mathbf{Adv}^{\mathsf{eSPR}^\dagger}_{\mathbf{H}}(A) = \Pr[(M, \mathtt{State}) \xleftarrow{\$} A; r \xleftarrow{\$} \mathcal{R}; i \xleftarrow{\$} [1, l]; (c', m') \xleftarrow{\$} A(i, r, M, \mathtt{State}) \\ : (c, m) = H_r(IV, M)[i] \text{ and } (c, m) \neq (c', m') \text{ and } f(c, m) = f(c', m')],$$

where $l = \mathsf{Len}_f(H_r(IV, M))$ is the number of computations of the compression function f when computing $H_r(IV, M)$ for any r, where M is generated by the adversary A.

A.2. Indifferentiable Security Analysis of a pfCM-chopMD Domain Extension

In this Appendix A.2, we provide an indifferentiable security analysis of pfCM^0-chopMD, which is the domain extension of ARIRANG. In cases of ARIRANG-256 and ARIRANG-512, there is no chopped bit ($s = 0$). In cases of ARIRANG-224 and ARIRANG-384, the chopped bits are 32 bits ($s = 32$) and 128 bits ($s = 128$), respectively. For any i, the indifferentiable security analysis of pfCM^i-chopMD can be also similarly done.

Construction of the Simulator

Here, we define simulators as follows. the simulator S_{pfCM} will be used in order to prove the indifferentiable security of pfCM^0-chopMD. For defining the simulator, we follow the style of construction of the simulator in [13], where $R : \{0,1\}^* \to \{0,1\}^{n-s}$ is a VIL random oracle.

Definition of Simulator S_{pfCM}

Initialization :

[7] eSPR† is similar to eSPR defined in [18].

1. A partial function $e : \{0,1\}^{n+b} \to \{0,1\}^n$ initialized as empty,
2. a partial function $e^* = \text{CM-MD}^e : (\{0,1\}^b)^* \to \{0,1\}^n$ initialized as $e^*(\mathsf{null}) = \text{IV}$.
3. a set $I = \{\text{IV}\}$ and a set $U = \{\mathsf{null}\}$.

$\boxed{\text{On query } S^R_{pfCM}(x,m) :}$

```
001 if $(e(x,m) = x')$
        return $x'$;

002 else if $(\exists\ M'\ and\ M, e^*(M') = x \oplus P, ||M'||_b = i, \mathsf{pad}(M) = M'||m))$
        $y = R(M)$;
        choose $w \in_R \{0,1\}^s$;
        define $e(x,m) = z := y \parallel w$;
        return $z$;

003 else if $(\exists\ M', e^*(M') = x \oplus i, ||M'||_b = i)$
        choose $z \in_R \{0,1\}^n \setminus \{c \oplus (i+1) : c \in I\} \cup \{c \oplus P : c \in I\} \cup \{a : (i_a, a) \in U\}$
                    $\cup \{a \oplus P \oplus (i+1) : (i_a, a) \in U\} \cup \{a \oplus i_a \oplus (i+1) : (i_a, a) \in U\}$
                    $\cup \{a \oplus i_a \oplus P : (i_a, a) \in U\}$;
        define $e(x,m) = z$;
        define $U = U \cup \{(i+1, z)\}$;
        define $e^*(M'||m) = z$;
        return $z$;

004 else
        $z \in_R \{0,1\}^n$;
        define $e(x,m) = z$;
        define $I = I \cup \{x\}$;
        return $z$;
```

Some Important Observations on the Simulator S_{pfCM}

The bound of the number of queries. In line 003, the number q of queries of S should be bounded by $q < 2^n/6$ in order to choose z. If $q \geq 2^n/6$, the simulator may not work. So, we assume that $q < 2^n/6$.

The bound of the number of possible input message. Firstly, in 002 and 003, there exists at most one M' such that $e^*(M') = x \oplus i$ or $e^*(M') = x \oplus P$ by the process of selecting z unrelated to the set U in line 003. This first observation corresponds to Lemma 1 in [6]. Secondly, in line 002 and 003, by the process of selecting z unrelated to the set I in line 003, the following holds : if $e(x, m)$ is already defined under the assumption that $e^*(M'||m)$ is not defined for all M' previously defined on e^*, where $||M'||_b = i - 1$, then no $M(= M'||m)$ can be newly defined such that $e^*(M) = x \oplus i$ or $e^*(M) = x \oplus P$, where where $||M||_b = i$. This second observation corresponds to the second part of proof of Lemma 2 in [6].

Indifferentiable Security Analysis of pfCM^0-chopMD Hash Domain Extension

We will describe the indifferentiable security bound of each domain extension using the notion *cost* of queries. We let the cost be q. For example, with the cost q of queries, A can have access to O_2 q times and no access to O_1, where O_1 corresponds to a hash function or a VIL random oracle, and O_2 corresponds to a compression function or a FIL random oracle. By observations of simulators described above, the following Lemma holds.

Lemma 9.3. Let $q < 2^n/6$. When the total cost of queries to O_1 is t less than or equal to q, the queries to O_1 can be converted to t queries to O_2, where O_2 gives at least the same amount of information to an attacker A and has no higher cost than O_1.

Proof. The proof is the same as that of Lemma 3 in [6]. ■

The Lemma 9.3 says that to give all queries to O_2 and no query to O_1 is the best strategy to obtain better computational distance. That is, when the cost of queries is bound by q, for any A there is an attacker B such that the following holds :

$$\mathbf{Adv}_A((H^f, f), (R, S)) \leq \mathbf{Adv}_B(f, S),$$

where H^f=pfCM$^0 -$ chopMD$_g^f$, and S=S_{pfCM}. Therefore, we focus on computing the upper bound of the computational distance between f and S as shown in the following theorems.

Theorem 9.4. Let $q < (2^n - 1)/6$ be the number of queries and $0 \leq s < n$. $f : \{0,1\}^{n+b} \rightarrow \{0,1\}^n$ is a FIL random oracle. S_{pfCM} is the simulator defined in the previous section. Then for any (deterministic or probabilistic) algorithm A

$$\mathbf{Adv}_A(f, S) \leq \tfrac{q(3q-1)}{2^n}.$$

Proof. Let S be S_{pfCM}. By Lemma 9.1, we only focus on computing an upper bound of $\mathbf{Stat}(f, S)$. Note that $\mathbf{Stat}(f, S)$ is defined over all deterministic algorithms. So when the oracle is f, the number of possible views is 2^{nq}. And for any deterministic algorithm A, each view occurs with probability $1/2^{nq}$. We let the set of 2^{nq} possible views be V_A. On the other hand, when the oracle is S, the number of possible views is at least $(2^n - 2)(2^n - 8) \cdots (2^n - 6q + 4)$. We let the set of the smallest possible views be T_S and the size of T_S be r_q. Assume that each of T_S views occurs with probability $1/r_q$. Therefore,

$$\begin{aligned}
\mathbf{Stat}_A(f, S) &= \tfrac{1}{2}\textstyle\sum_{v \in V_A} |\Pr[f = v] - \Pr[S = v]| \\
&= \tfrac{1}{2}\textstyle\sum_{v \in V_A \setminus T_S} |\Pr[f = v] - \Pr[S = v]| + \tfrac{1}{2}\textstyle\sum_{v \in T_S} |\Pr[f = v] - \Pr[S = v]| \\
&\leq \tfrac{1}{2}\textstyle\sum_{v \in V_A \setminus T_S} |\tfrac{1}{2^{nq}} - 0| + \tfrac{1}{2}\textstyle\sum_{v \in T_S} |\tfrac{1}{2^{nq}} - \tfrac{1}{r_q}| \\
&= \tfrac{1}{2} \cdot \tfrac{2^{nq} - r_q}{2^{nq}} + \tfrac{1}{2} \cdot |\tfrac{r_q}{2^{nq}} - \tfrac{r_q}{r_q}| \\
&= \tfrac{1}{2} \cdot (1 - \tfrac{r_q}{2^{nq}}) + \tfrac{1}{2} \cdot (1 - \tfrac{r_q}{2^{nq}}) \\
&= 1 - \tfrac{r_q}{2^{nq}}
\end{aligned}$$

$= 1 - \prod_{i=1}^{q}(1 - \frac{6i-4}{2^n})$
$\leq \sum_{i=1}^{q}(\frac{6i-4}{2^n})$ (by Ineq 1.)
$= \frac{q(3q-1)}{2^n}$. ■

From Lemma 9.3 and Theorem 9.4, we can get indiffrerentiable security bound of pfCM^0-chopMD as the following corollary.

Corollary 9.5. Let $q < (2^n - 1)/6$ be the cost of queries and $0 \leq s < n$. $f : \{0,1\}^{n+b} \to \{0,1\}^n$ is a FIL random oracle. S_{pfCM} is the simulator defined in the previous section. Then for any attacker A

$$\mathbf{Adv}_A((\mathrm{pfCM}^0 - \mathrm{chopMD}^f_{\mathsf{pad}}, f), (R, S_{pfCM})) \leq \frac{q(3q-1)}{2^n}.$$

A.3. PRF Security Analysis of HMAC based on a pfCM-MD Domain Extension

In this section, with game-based proof technique, we provide a prf security analysis of HMAC based on a pfCM^0-MD domain extension. Our proof follows the proof technique for HMAC in [3]. For any i, HMAC based on a pfCM^i-MD domain extension can be also proved in the similar way.

Lemma 9.6. For any rka-prf-adversary A with q queries, there exists an adversary B_A such that

$$|\Pr[A^{G_7} = 1] - \Pr[A^{G_6} = 1]| = \mathbf{Adv}^{\mathsf{rka\text{-}prf}}_{f(\star, RK(\star, K||0^{b-n})), \Phi_1}(B_A),$$

where G_7 and G_6 are games defined in Fig. 18, B_A is defined in Fig. 19. B_A can only make two $(IV, \phi_{\mathsf{ipad}})$ and $(IV, \phi_{\mathsf{opad}})$ queries. $\Phi_1 = \{\phi_{\mathsf{ipad}}, \phi_{\mathsf{opad}}\}$ where $\phi_{\mathsf{ipad}}(x) = x \oplus \mathsf{ipad}$ and $\phi_{\mathsf{opad}}(x) = x \oplus \mathsf{opad}$.

Proof. Since $\Pr[A^{G_7} = 1] = \Pr[K \xleftarrow{\$} \{0,1\}^n : B_A^{f(\star, RK(\star, K||0^{b-n}))} = 1]$ and $\Pr[A^{G_6} = 1] = \Pr[g \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^{n+b}, \{0,1\}^n); K \xleftarrow{\$} \{0,1\}^n : B_A^{g(\star, RK(\star, K||0^{b-n}))} = 1]$, this lemma holds. ■

Lemma 9.7. For any prf-adversary A, the following equality holds :

$$\Pr[A^{G_6} = 1] = \Pr[A^{G_5} = 1],$$

where G_6 and G_5 are games defined in Fig. 18.

Proof. We already assumed that in the case of NMAC, the outer hash function uses the compression function one time, and in the case of HMAC, the outer hash function uses the compression function two times. So, this lemma is clear. ■

Lemma 9.8. For any prf-adversary A with q queries, there exists a prf-adversary C_A such that

$$|\Pr[A^{G_5} = 1] - \Pr[A^{G_4} = 1]| = \mathbf{Adv}^{\mathsf{prf}}_{f(K, \star)}(C_A),$$

where G_5 and G_4 are games defined in Fig. 18, and C_A is defined in Fig. 19. C_A can make at most q queries.

```
Game $G_1$

On query $M$
    $Z \xleftarrow{\$} \{0,1\}^n$
    Return $Z$
```

```
Game $G_2$

$K_1 \xleftarrow{\$} \{0,1\}^n$; $s \leftarrow 0$
$Z_1, \cdots, Z_q \xleftarrow{\$} \{0,1\}^n$
On query $M$
    $s \leftarrow s+1$; $M_s \leftarrow M$
    $Y_s \leftarrow \text{pfCM}^1\text{-MD}^f_{\mathsf{pad}_1}(K_1, M_s)$
    If $(\exists\, r < s : Y_r = Y_s)$ then
        $\mathsf{bad} \leftarrow \mathsf{true}$;
    Return $Z_s$
```

```
Game $G_3$

$K_1 \xleftarrow{\$} \{0,1\}^n$; $s \leftarrow 0$
$Z_1, \cdots, Z_q \xleftarrow{\$} \{0,1\}^n$
On query $M$
    $s \leftarrow s+1$; $M_s \leftarrow M$
    $Y_s \leftarrow \text{pfCM}^1\text{-MD}^f_{\mathsf{pad}_1}(K_1, M_s)$
    If $(\exists\, r < s : Y_r = Y_s)$ then
        $\mathsf{bad} \leftarrow \mathsf{true}$; $Z_s \leftarrow Z_r$
    Return $Z_s$
```

```
Game $G_4$

$K_1 \xleftarrow{\$} \{0,1\}^n$
$g \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^b, \{0,1\}^n)$
On query $M$
    Return $g(\mathsf{pad}_1(\text{pfCM}^1 - \text{MD}^f_{\mathsf{pad}_1}(K_1, M_s)))$
```

```
Game $G_5$

$K_2, K_1 \xleftarrow{\$} \{0,1\}^n$
On query $M$
    Return $f((K_2 \oplus P), \mathsf{pad}_1(\text{pfCM}^1 - \text{MD}^f_{\mathsf{pad}_1}(K_1, M_s)))$
```

```
Game $G_6$

$K_2, K_1 \xleftarrow{\$} \{0,1\}^n$
On query $M$
    Return $\text{NMAC}^{\text{pfCM}^1 - \text{MD}^f_{\mathsf{pad}_1}}(K_2 || K_1, M)$
```

```
Game $G_7$

$K \xleftarrow{\$} \{0,1\}^n$
$\overline{K} \leftarrow K || 0^{b-n}$
$K_1 \leftarrow f(IV, \overline{K} \oplus \mathsf{ipad})$
$K_2 \leftarrow f(IV, \overline{K} \oplus \mathsf{opad})$
On query $M$
    Return $\text{NMAC}^{\text{pfCM}^1 - \text{MD}^f_{\mathsf{pad}_1}}(K_2 || K_1, M)$
```

Figure 18: Game $G_1 \sim G_7$

```
Adversary $B_A^{O(\star,RK(\star,K||0^{b-n}))}$, where $O$ is $f(\star,K||0^{b-n})$ or $g(\star,K||0^{b-n})$.

$K_1 \leftarrow O(IV, RK(\phi_{\mathsf{ipad}}, K||0^{b-n}))$
$K_2 \leftarrow O(IV, RK(\phi_{\mathsf{opad}}, K||0^{b-n}))$
Run $A$ as follows:
    On query $M$ of $A$, reply $\mathrm{NMAC}^{\mathrm{pfCM}^1-\mathrm{MD}^f_{\mathsf{pad}_1}(\star,\star)}(K_2||K_1, M)$ to $A$
    Let $T$ be the final output of $A$
Return $T$
```

```
Adversary $C_A^O$, where $O$ is $f(K,\star)$ or $g(\star)$.

$K_1 \xleftarrow{\$} \{0,1\}^n$
Run $A$ as follows:
    On query $M$ of $A$, reply $O(\mathsf{pad}_1(\mathrm{pfCM}^1 - \mathrm{MD}^f_{\mathsf{pad}_1}(K_1, M))))$ to $A$
    Let $T$ be the final output of $A$
Return $T$
```

```
Adversary $D_A$

$s \leftarrow 0$ and $Z_1, \cdots, Z_q \xleftarrow{\$} \{0,1\}^n$
Run $A$ as follows:
    On query $M$ of $A$, $s \leftarrow s+1$ and $M_s \leftarrow M$ and reply $Z_s$ to $A$
$i, j \xleftarrow{\$} [1,q]$ with $i \neq j$
Return $M_i$ and $M_j$
```

Figure 19: Adversary B_A, C_A, D_A

```
Adversary $E_A^{O(RK(\star,K),\star)}$, where $O$ is $f(K,\star)$ or $g(K,\star)$.

100 Run $A$, and obtain $M$, $M'$ from $A$, and let $m = ||\mathsf{pad}_1(M)||_b$, $m' = ||\mathsf{pad}_1(M')||_b$
200 Let $\mathsf{pad}_1(M) = M_1||\cdots||M_m$ and $\mathsf{pad}_1(M') = M'_1||\cdots||M'_{m'}$ and $r = LCP(\mathsf{pad}_1(M), \mathsf{pad}_1(M'))$
        /* $r$ is the $b$-bit block length of the largest common prefix of $\mathsf{pad}_1(M)$ and $\mathsf{pad}_1(M')$ */
300 Randomly choose $(l, l')$ from $I(\mathsf{pad}_1(M), \mathsf{pad}_1(M'))$
        /* total number of cases is at most $m + m' - 1$.
           $I(\mathsf{pad}_1(M), \mathsf{pad}_1(M'))$ is a sequence of $(1,1)||\cdots||(r,r)||(r+1,r+1)$
           $||(r+2,r+1)||\cdots||(m,r+1)||(m,r+2)||\cdots||(m,m')$. */
400 If $(l, l') \in I_1(\mathsf{pad}_1(M), \mathsf{pad}_1(M')) \cup \{(r+1,r+1)\} \cup I_2(\mathsf{pad}_1(M), \mathsf{pad}_1(M'))$
          /* $I_1(\mathsf{pad}_1(M), \mathsf{pad}_1(M')) = \{(1,1), \cdots, (r,r)\}$ and $I_2 = \{(r+2,r+1), \cdots, (m,r+1)\}$ */
401     then if $l = m$ then $a_l \leftarrow O(\phi_P, M_l)$ else $a_l \leftarrow O(\phi_l, M_l)$
402     else $a_l \xleftarrow{\$} \{0,1\}^n$
500 If $(l, l') \in I_1(\mathsf{pad}_1(M), \mathsf{pad}_1(M')) \cup \{(r+1,r+1)\} \cup I_3(\mathsf{pad}_1(M), \mathsf{pad}_1(M'))$
          /* $I_3 = \{(m,r+2), \cdots, (m,m')\}$ */
501     then if $l' = m'$ then $a'_{l'} \leftarrow O(\phi_P, M'_{l'})$ else $a'_{l'} \leftarrow O(\phi_{l'}, M'_{l'})$
502     else $a'_{l'} \xleftarrow{\$} \{0,1\}^n$
600 For $i = l + 1$ to $m$ do
601     if $i < m$ then $a_i \leftarrow f(a_{i-1} \oplus i, M_i)$
602     if $i = m$ then $a_i \leftarrow f(a_{i-1} \oplus P, M_i)$
700 For $i = l' + 1$ to $m'$ do
701     if $i < m'$ then $a'_i \leftarrow f(a'_{i-1} \oplus i, M'_i)$
702     if $i = m'$ then $a'_i \leftarrow f(a'_{i-1} \oplus P, M'_i)$
800 If $a_m = a'_{m'}$ then return 1 else return 0.
```

Figure 20: Adversary E_A: P is the last counter value of pfCM^1-MD.

Proof. Since $\Pr[A^{G_5} = 1] = \Pr[K \xleftarrow{\$} \{0,1\}^n : C_A^{f(K,\star)} = 1]$ and $\Pr[A^{G_4} = 1] = \Pr[g \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^b, \{0,1\}^n) : C_A^{g(\star)} = 1]$, this lemma holds. ■

Lemma 9.9. For any prf-adversary A with q queries, the following equality holds :

$$\Pr[A^{G_4} = 1] = \Pr[A^{G_3} = 1],$$

where G_4 and G_3 are games defined in Fig. 18.

Proof. By the definitions of G_3 and G_4, it is clear. ■

Lemma 9.10. For any prf-adversary A with q queries, the following inequality holds :

$$|\Pr[A^{G_3} = 1] - \Pr[A^{G_2} = 1]| \leq \Pr[A^{G_2} \text{ sets } \mathsf{bad}],$$

where G_3 and G_2 are games defined in Fig. 18.

Proof. As described in [9], this lemma follows from the Fundamental Lemma of Game Playing. ■

Lemma 9.11. For any prf-adversary A with q queries, the following equality holds :

$$\Pr[A^{G_2} = 1] = \Pr[A^{G_1} = 1],$$

where G_2 and G_1 are games defined in Fig. 18.

Proof. By the definitions of G_1 and G_2, it is clear. ■

Lemma 9.12. For any prf-adversary A with q queries, there exists an au-adversary D_A such that

$$\Pr[A^{G_2} \text{ sets } \mathsf{bad}] \leq \frac{q(q-1)}{2}\mathbf{Adv}^{\mathsf{au}}_{\mathrm{pfCM}^1-\mathrm{MD}^f_{\mathsf{pad}_1}(K,\star)}(D_A),$$

where G_2 is a game defined in Fig. 18, and D_A is defined in Fig. 19.

Proof. We let $F(K,\star)$ be $\mathrm{pfCM}^1 - \mathrm{MD}^f_{\mathsf{pad}_1}(K,\star)$. Without loss of generality, we assume that A makes q different queries.

$$\begin{aligned}
&\mathbf{Adv}^{\mathsf{au}}_{F(K,\star)}(D_A)\\
&\quad = \textstyle\sum_{i<j} \Pr[K \xleftarrow{\$} \{0,1\}^n; M_1, \cdots, M_q \xleftarrow{\$} A^D : F(K, M_i) = F(K, M_j)]\Pr[M_i, M_j \xleftarrow{\$} D_A]\\
&\quad = \textstyle\sum_{i<j} \Pr[K \xleftarrow{\$} \{0,1\}^n; M_1, \cdots, M_q \xleftarrow{\$} A^{G_2} : F(K, M_i) = F(K, M_j)]\frac{2}{q(q-1)}\\
&\quad \geq \Pr[K \xleftarrow{\$} \{0,1\}^n; M_1, \cdots, M_q \xleftarrow{\$} A^{G_2} : \exists M_i, M_j \text{ s.t. } F(K, M_i) = F(K, M_j)]\frac{2}{q(q-1)}\\
&\quad = \Pr[A^{G_2} \text{ sets } \mathsf{bad}]\frac{2}{q(q-1)}.
\end{aligned}$$

■

Lemma 9.13. For given M and M', where $||\mathsf{pad}_1(M)||_b = m \leq t$ and $||\mathsf{pad}_1(M')||_b = m' \leq t'$, if (α', β') is the predecessor of (α, β) in the sequence of $I(\mathsf{pad}_1(M), \mathsf{pad}_1(M'))$, then the following holds.

$$\begin{aligned}&\Pr[K \xleftarrow{\$} \{0,1\}^n : E_{A(M,M')}^{f(RK(\star,K),\star)} = 1 | (l, l') = (\alpha, \beta) \leftarrow E_{A(M,M')}^{f(RK(\star,K),\star)}]\\ =&\Pr[g \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^{n+b}, \{0,1\}^n); K \xleftarrow{\$} \{0,1\}^n : E_{A(M,M')}^{g(RK(\star,K),\star)} = 1 | (l, l') = (\alpha', \beta') \leftarrow E_{A(M,M')}^{g(RK(\star,K),\star)}],\end{aligned}$$

Here, $E_A^{O(RK(\star,K),\star)}$, I_1, I_2, I_3 and I are defined in Fig. 20. In a sequence $((\alpha_1, \beta_1), \cdots, (\alpha_n, \beta_n))$, (α_i, β_i) is called the predecessor of $(\alpha_{i+1}, \beta_{i+1})$. For example, in the sequence I, the predecessor of $(r+2, r+1)$ is $(r+1, r+1)$ and the predecessor of $(m, r+2)$ is $(m, r+1)$.

Proof. It follows from the definition of E_A in Fig. 20. ■

Lemma 9.14. For any au-adversary A, the following holds.

$$\begin{aligned}&\Pr[K \xleftarrow{\$} \{0,1\}^n : E_{A(M,M')}^{f(RK(\star,K),\star)} = 1 | (l, l') = (1, 1) \leftarrow E_{A(M,M')}^{f(RK(\star,K),\star)}]\\ &\qquad = \Pr[K \xleftarrow{\$} \{0,1\}^n : F(K, M) = F(K, M')],\\ &\Pr[g \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^{n+b}, \{0,1\}^n); K \xleftarrow{\$} \{0,1\}^n : E_{A(M,M')}^{g(RK(\star,K),\star)} = 1 | (l, l') = (m, m') \leftarrow E_{A(M,M')}^{g(RK(\star,K),\star)}]\\ &\qquad = 2^{-n},\end{aligned}$$

where $F(K, \star)$ denotes $\text{pfCM}^1 - \text{MD}_{\mathsf{pad}_1}^f(K, \star)$.

Proof. It is clear by the construction of E_A in Fig. 20. ■

Lemma 9.15. For any au-adversary A, there exists a rka-prf-adversary E_A such that

$$\mathbf{Adv}_{\text{pfCM}^1 - \text{MD}_{\mathsf{pad}_1}^f(K,\star)}^{\mathsf{au}}(A) \leq (t + t' - 1)\mathbf{Adv}_{f(K,\star),\Phi_2}^{\mathsf{rka\text{-}prf}}(E_A) + 2^{-n},$$

where E_A is defined in Fig. 20.For any output (M, M') of A, $||\mathsf{pad}_1(M)||_b \leq t$ and $||\mathsf{pad}_1(M')||_b \leq t'$. When $t^* = max(t, t')$, $\Phi_2 = \{\phi_1, \cdots, \phi_{t^*}, \phi_P\}$ where $\phi_i(x) = x \oplus i$. E_A can only make at most two (M_i, ϕ) and (M'_j, ϕ') queries, where M_i and M'_j are any value of b-bit, and $\phi, \phi' \in \Phi_2$.

Proof. We let $F(K, \star)$ be $\text{pfCM}^1 - \text{MD}_{\mathsf{pad}_1}^f(K, \star)$.

$$\begin{aligned}&\mathbf{Adv}_{f(RK(\star,K),\star),\Phi_2}^{\mathsf{rka\text{-}prf}}(E_A)\\ &\quad = |\Pr[K \xleftarrow{\$} \{0,1\}^n : E_A^{f(RK(\star,K),\star)} = 1]\\ &\qquad - \Pr[g \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^{n+b}, \{0,1\}^n); K \xleftarrow{\$} \{0,1\}^n : E_A^{g(RK(\star,K),\star)} = 1]|\\ &\quad = |\textstyle\sum_{M \neq M'} \Pr[K \xleftarrow{\$} \{0,1\}^n : E_{A(M,M')}^{f(RK(\star,K),\star)} = 1]\Pr[(M, M') \leftarrow A]\\ &\qquad - \textstyle\sum_{M \neq M'} \Pr[g \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^{n+b}, \{0,1\}^n); K \xleftarrow{\$} \{0,1\}^n : E_{A(M,M')}^{g(RK(\star,K),\star)} = 1]\Pr[(M, M') \leftarrow A]|\\ &\quad \geq |\textstyle\sum_{M \neq M'} \frac{\Pr[K \xleftarrow{\$} \{0,1\}^n : F(K,M) = F(K,M')] - 2^{-n}}{t+t'-1}\Pr[M, M' \leftarrow A]| \text{ by Lemma 9.13, 9.14}\\ &\quad = |\tfrac{1}{t+t'-1}[(\textstyle\sum_{M \neq M'} \Pr[K \xleftarrow{\$} \{0,1\}^n : F(K, M) = F(K, M')]\Pr[M, M' \leftarrow A]) - 2^{-n}]|\\ &\quad = |\tfrac{1}{t+t'-1}(\mathbf{Adv}_{F(K,\star)}^{\mathsf{au}}(A) - 2^{-n})|\\ &\quad \geq \tfrac{1}{t+t'-1}(\mathbf{Adv}_{F(K,\star)}^{\mathsf{au}}(A) - 2^{-n}).\end{aligned}$$

■

Theorem 9.16. For any prf-adversary A, there exist adversaries B_A, C_A, D_A, E_{D_A} such that

$$\begin{aligned}\mathbf{Adv}^{\mathsf{prf}}_{\mathrm{HMAC}_{IV}^{\mathrm{pfCM}^0-\mathrm{MD}^f_{\mathsf{pad}}}}(A) &\leq \mathbf{Adv}^{\mathsf{rka\text{-}prf}}_{f(\star,RK(\star,K\|0^{b-n})),\Phi_1}(B_A) + \mathbf{Adv}^{\mathsf{prf}}_{f(K,\star)}(C_A)\\ &+ \tfrac{q(q-1)(t+t'-1)}{2}\mathbf{Adv}^{\mathsf{rka\text{-}prf}}_{f(RK(\star,K),\star),\Phi_2}(E_{D_A}) + \tfrac{q(q-1)}{2^{n+1}},\end{aligned}$$

where B_A, C_A, D_A, E_{D_A}, Φ_1, and Φ_2 are defined as before.

Proof. By the definition of the prf-advantage, $\mathbf{Adv}^{\mathsf{prf}}_{\mathrm{HMAC}_{IV}^{\mathrm{pfCM}-\mathrm{MD}^f_{\mathsf{pad}}}}(A) = |\Pr[A^{G_7} = 1] - \Pr[A^{G_1} = 1]|$. So, we can get the above theorem with Lemma 9.6 ∼ Lemma 9.15. ∎

A.4. Security Analysis of Two PRF Constructions based on a pfCM-MD Domain Extension

In this section, we provide prf security analysis of pfCM^0-MD$^f_{\mathsf{pad}}(IV, K\|0^{b-n}\|\star)$ and pfCM^1-MD$^f_{\mathsf{pad}}(K,\star)$, where $K \xleftarrow{\$} \{0,1\}^n$. Our analysis follows the analysis technique of Bellare *et al.*' paper [4]. For any d and d', pfCM^d-MD$^f_{\mathsf{pad}}(IV, K\|0^{b-n}\|\star)$ and pfCM$^{d'}$-MD$^f_{\mathsf{pad}}(K,\star)$ can be also proved in the similar way.

Lemma 9.17. For any prf-adversary A with q queries, there exists a prf-adversary F_A such that

$$|\Pr[A^{G'_3} = 1] - \Pr[A^{G'_2} = 1]| = \mathbf{Adv}^{\mathsf{prf}}_{f(\star,K\|0^{b-n})}(F_A)$$

where G'_3 and G'_2 are games defined in Fig. 21, and F_A is defined in Fig. 22. F_A can only make the query IV.

Proof. Since $\Pr[A^{G'_3} = 1] = \Pr[K \xleftarrow{\$} \{0,1\}^n : F_A^{f(\star,K\|0^{b-n})} = 1]$ and $\Pr[A^{G'_2} = 1] = \Pr[g \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^n,\{0,1\}^n) : F_A^{g(\star)} = 1]$, the lemma holds. ∎

Lemma 9.18. For any prf-adversary A, the following equality holds :

$$|\Pr[A^{G'_2} = 1] - \Pr[A^{G'_1} = 1]| = \mathbf{Adv}^{\mathsf{prf}}_{\mathrm{pfCM}^1-\mathrm{MD}^f_{\mathsf{pad}_1}(K,\star)}(A)$$

where G'_2 and G'_1 are games defined in Fig. 21.

Proof. By the definition of the prf-advantage, the lemma holds. ∎

Lemma 9.19. For any $2 \leq j \leq l$, the following holds.

$$\begin{aligned}&\Pr[K_1,\cdots,K_q \xleftarrow{\$} \{0,1\}^n : H_{A,i\leftarrow j}^{f(RK(\star,K_\star),\star)} = 1]\\ &{=}\Pr[g_1,\cdots,g_q \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^{n+b},\{0,1\}^n);K \xleftarrow{\$} \{0,1\}^n : H_{A,i\leftarrow j-1}^{g_\star(RK(\star,K),\star)} = 1],\end{aligned}$$

```
Game $G'_1$

On query $M$
    $Z \overset{\$}{\leftarrow} \{0,1\}^n$
    Return $Z$
```

```
Game $G'_2$

$K' \overset{\$}{\leftarrow} \{0,1\}^n$
On query $M$
    Return pfCM$^1$-MD$^f_{\mathsf{pad}_1}(K', M)$
```

```
Game $G'_3$

$K \overset{\$}{\leftarrow} \{0,1\}^n$
$K' \leftarrow f(IV, K||0^{b-n})$
On query $M$
    Return pfCM$^1$-MD$^f_{\mathsf{pad}_1}(K', M)$
```

Figure 21: Game $G'_1 \sim G'_3$

where H_A is defined in Fig. 22, and $i \leftarrow j$ is described in line 10000 in Fig. 22. If A makes q queries, then H_A can make at most q queries. We assume that for each query M of A, the b-bit block length of $\mathsf{pad}_1(M)$ is at most l. $\Phi_3 = \{\phi_1, \cdots, \phi_l, \phi_P\}$, where $\phi_i(X) = X \oplus i$ and P is the last counter of pfCM-MD. When we denote t-th query of H_A by (i^t, ϕ^t, X^t), we assume that $\{\phi^1, \cdots, \phi^q\} \subset \{\phi_P, \phi_j\}$ for some j. In other words, even though H_A can make queries to any one of $\{O_1, O_2, \cdots, O_q\}$, H_A can use at most two related-key-deriving (RKD) functions ϕ's from Φ_3.

Proof. It follows from the definition of $H_A^{O_1,\cdots,O_q}$ in Fig. 22. ■

Lemma 9.20. For any prf-adversary A with q queries, the following holds.

$$\Pr[K_1, \cdots, K_q \overset{\$}{\leftarrow} \{0,1\}^n : H_{A,i\leftarrow 1}^{f(RK(\star,K_\star),\star)} = 1] = \Pr[K \overset{\$}{\leftarrow} \{0,1\}^n : A^{F(K,\star)} = 1],$$
$$\Pr[K_1,\cdots,K_q \overset{\$}{\leftarrow} \{0,1\}^n : H_{A,i\leftarrow l}^{f(RK(\star,K_\star),\star)} = 1] = \Pr[g \overset{\$}{\leftarrow} \mathsf{Maps}(\{0,1\}^*,\{0,1\}^n) : A^{g(\star)} = 1],$$

where $F(K, \star)$ denotes $\mathrm{pfCM}^1 - \mathrm{MD}^f_{\mathsf{pad}_1}(K, \star)$.

Proof. It is clear by the construction of H_A in Fig. 22. ■

Theorem 9.21. For any prf-adversary A with q queries, there exists a multi-rka-prf-adversary H_A such that

$$\mathbf{Adv}^{\mathsf{prf}}_{\mathrm{pfCM}^1-\mathrm{MD}^f_{\mathsf{pad}_1}(K,\star)}(A) = l \cdot \mathbf{Adv}^{\mathsf{multi\text{-}rka\text{-}prf}}_{f(RK(\star,K_\star),\star),\Phi_3}(H_A),$$

where H_A is defined as before.

Proof. We let $F(K, \star)$ be $\mathrm{pfCM}^1 - \mathrm{MD}^f_{\mathsf{pad}_1}(K, \star)$.

$$\begin{aligned}&\mathbf{Adv}^{\mathsf{multi\text{-}rka\text{-}prf}}_{f(RK(\star,K_\star),\star),\Phi_3}(H_A)\\ &\quad = |\Pr[K_1, \cdots, K_q \overset{\$}{\leftarrow} \{0,1\}^n : H_A^{f(RK(\star,K_\star),\star)} = 1]\end{aligned}$$

```
Adversary $F_A^{O(\star,K||0^{b-n})}$, where $O$ is $f(\star,K||0^{b-n})$ or $g(\star)$.

$K' \leftarrow O(IV)$
Run $A$ as follows:
    On query $M$ of $A$, reply $\mathrm{pfCM}^1 - \mathrm{MD}^f_{\mathsf{pad}_1}(K',M)$ to $A$
    Let $T$ be the final output of $A$
Return $T$
```

```
Adversary $H_A^{O_1,\cdots,O_q}$, where $O_i$ is $f(RK(\star,K_i),\star)$ or $g_i(RK(\star,K),\star)$.

Randomly choose $j$ from $[1,l]$ and $i \leftarrow j$ and $s \leftarrow 0$
Run $A$ as follows:
    On query $t$-th query $M^t$ of $A$,   // $1 \leq t \leq q$
      $m \leftarrow ||\mathsf{pad}_1(M^t)||_b$ and let $\mathsf{pad}_1(M^t) = M_1^t||\cdots||M_m^t$     // $||\mathsf{pad}_1(M^t)||_b \leq l$
      if $m \leq i-1$ then pick at random an $n$-bit string $a^t$ and return $a^t$ to $A$
      else (namely $m \geq i$),
        if $(M_1^t,\cdots,M_{i-1}^t) \neq (M_1^r,\cdots,M_{i-1}^r)$ for all $r < t$
        then $s \leftarrow s+1$ and let $c^t = s$
        else if $(M_1^t,\cdots,M_{i-1}^t)=(M_1^r,\cdots,M_{i-1}^r)$ & $||\mathsf{pad}_1(M^r)||_b \neq i-1$ for a $r$ s.t. $r < t$
            then let $c^t = c^r$
            else $s \leftarrow s+1$ and let $c^t = s$
        if $m > i$ then $a^t = O_{c^t}(\phi_i, M_i^t)$ else $a^t = O_{c^t}(\phi_P, M_i^t)$
        return $\mathrm{pfCM}^{\mathrm{i}+1} - \mathrm{MD}^f(a^t, M_{i+1}^t||\cdots||M_m^t)$ to $A$
Let $T$ be the final output of $A$
Return $T$
```

Figure 22: Adversary F_A and H_A: P is the last counter value of pfCM-MD.

$$
\begin{aligned}
&\quad - \Pr[g_1,\cdots,g_q \overset{\$}{\leftarrow} \mathsf{Maps}(\{0,1\}^{n+b},\{0,1\}^n); K \overset{\$}{\leftarrow} \{0,1\}^n : H_A^{g_\star(RK(\star,K),\star)} = 1]| \\
&= |\textstyle\sum_{j=1}^{l} \Pr[K_1,\cdots,K_q \overset{\$}{\leftarrow} \{0,1\}^n : H_{A,i=j}^{f(RK(\star,K_\star),\star)} = 1] \cdot \frac{1}{l} \\
&\quad - \textstyle\sum_{j=1}^{l} \Pr[g_1,\cdots,g_q \overset{\$}{\leftarrow} \mathsf{Maps}(\{0,1\}^{n+b},\{0,1\}^n); K \overset{\$}{\leftarrow} \{0,1\}^n : H_{A,i=j}^{g_\star(RK(\star,K),\star)} = 1] \cdot \frac{1}{l}| \\
&= \frac{1}{l}|\Pr[K \overset{\$}{\leftarrow} \{0,1\}^n : A^{F(K,\star)} = 1] - \Pr[g \overset{\$}{\leftarrow} \mathsf{Maps}(\{0,1\}^*,\{0,1\}^n) : A^{g(\star)} = 1]| \\
&= \frac{1}{l}\mathbf{Adv}^{\mathsf{prf}}_{F(K,\star)}(A). \qquad \blacksquare
\end{aligned}
$$

The second equality follows from the definition of H_A in Fig. 22 and the third equality follows from Lemma 9.19 and Lemma 9.26.

Theorem 9.22. For any prf-adversary A with q queries, there exists a prf-adversary F_A such that

$$
\mathbf{Adv}^{\mathsf{prf}}_{\text{pfCM}^0-\text{MD}^f_{\mathsf{pad}}(IV,K||\star)}(A) \le \mathbf{Adv}^{\mathsf{prf}}_{\text{pfCM}^1-\text{MD}^f_{\mathsf{pad}_1}(K,\star)}(A) + \mathbf{Adv}^{\mathsf{prf}}_{f(\star,K||0^{b-n})}(F_A),
$$

where F_A can only make the query IV and is defined in Fig. 22.

Proof. By the definition of the prf-advantage, $\mathbf{Adv}^{\mathsf{prf}}_{\text{pfCM}^0-\text{MD}^f_{\mathsf{pad}}(IV,K||\star)}(A) = |\Pr[A^{G'_3} = 1] - \Pr[A^{G'_1} = 1]|$. So, we can get above theorem with Lemma 9.17 $\sim$ Lemma 9.18. $\blacksquare$

Corollary 9.23. For any prf-adversary A with q queries, there exist adversaries F_A and H_A such that

$$
\mathbf{Adv}^{\mathsf{prf}}_{\text{pfCM}^0-\text{MD}^f_{\mathsf{pad}}(IV,K||\star)}(A) \le l \cdot \mathbf{Adv}^{\mathsf{multi\text{-}rka\text{-}prf}}_{f(RK(\star,K_\star),\star),\Phi_3}(H_A) + \mathbf{Adv}^{\mathsf{prf}}_{f(\star,K||0^{b-n})}(F_A),
$$

where F_A, H_A and Φ_3 are defined as before.

Proof. This holds by Theorem 9.21 and 9.22. $\blacksquare$

A.5. PRF Security Analysis of NIST SP 800-56A Key Derivation Function based on a pfCM-MD Domain Extension

In this section, we provide prf security analysis of pfCM^0-MD$^f_{\mathsf{pad}}(IV, \ \star_{32}\ ||K||\ \star)$, where $\star_{32}$ is any 32-bit string, and $K \overset{\$}{\leftarrow} \{0,1\}^n$. pfCM^0-MD$^f_{\mathsf{pad}}(IV, \ \star_{32}\ ||K||\ \star)$ corresponds to NIST SP 800-56A key derivation function based on ARIRANG. Our analysis follows the analysis technique of Bellare *et al.*' paper [4]. For any i, the prf security of pfCM^i-MD$^f_{\mathsf{pad}}(IV, \ \star_{32}\ ||K||\ \star)$ can be also proved in the similar way.

Lemma 9.24. For any prf-adversary A with q queries, there exists a prf-adversary Q_A such that

$$
|\Pr[A^{G''_3} = 1] - \Pr[A^{G''_2} = 1]| = \mathbf{Adv}^{\mathsf{prf}}_{f(\star,\star_{32}||K||\star_{b-n-32})}(Q_A)
$$

where G''_3 and G''_2 are games defined in Fig. 23, and Q_A is defined in Fig. 24. And Q_A can make q queries of the form $(IV||\star_{32}\ ||\star_{b-n-32})$, and $\star_i$ means any i-bit string.

Proof. Since $\Pr[A^{G_3''} = 1] = \Pr[K \xleftarrow{\$} \{0,1\}^n : Q_A^{f(\star_n, \star_{32}||K||\star_{b-n-32})} = 1]$ and $\Pr[A^{G_2'} = 1] = \Pr[g \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^n, \{0,1\}^n) : Q_A^{g(\star_n||\star_{32}||\star_{b-n-32})} = 1]$, the lemma holds. ■

Lemma 9.25. For any prf-adversary A, the following equality holds :

$$|\Pr[A^{G_2''} = 1] - \Pr[A^{G_1''} = 1]| = \mathbf{Adv}^{\mathsf{prf}}_{\mathrm{pfCM}^1-\mathrm{MD}^f_{\mathsf{pad}_1}(g(IV, \star_{32}||K||\star_{b-n-32}),\ \star)}(A)$$

where G_2'' and G_1'' are games defined in Fig. 23, and $g \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^b, \{0,1\}^n)$.

Proof. By the definition of the prf-advantage, the lemma holds. ■

Lemma 9.26. For any $2 \leq j \leq l-1$, the following holds.

$$\begin{aligned}&\Pr[K_1, \cdots, K_q \xleftarrow{\$} \{0,1\}^n : V_{A,i=j}^{f(RK(\star, K_\star), \star)} = 1] \\ &= \Pr[g_1, \cdots, g_q \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^{n+b}, \{0,1\}^n); K \xleftarrow{\$} \{0,1\}^n : V_{A,i=j-1}^{g_\star(RK(\star,K),\star)} = 1],\end{aligned}$$

where V_A is defined in Fig. 24.

Proof. It is clear by the definition of V_A. ■

```
Game $G_1''$

100 On query $M$
101     $Z \xleftarrow{\$} \{0,1\}^n$
102     Return $Z$
```

```
Game $G_2''$

100 $g \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^b, \{0,1\}^n)$
300 On query $M = M_1||M_2$ // $|M_1| = 32$ and $|M_2| = t$ where $t$ is any value.
200     $K' \leftarrow g(IV, M_1||M_2[1, b-n-32])$ // $M_2[1, x]$ denotes the first $x$-bit of $M_2$
301     Return pfCM$^1$-MD$^f_{\mathsf{pad}_1}(K', M[b-n-31, t])$
```

```
Game $G_3''$

100 $K \xleftarrow{\$} \{0,1\}^n$
300 On query $M = M_1||M_2$ // $|M_1| = 32$ and $|M_2| = t$ where $t$ is any value.
200     $K' \leftarrow f(IV, M_1||K||M_2[1, b-n-32])$ // $M_2[1, x]$ denotes the first $x$-bit of $M_2$
301     Return pfCM$^1$-MD$^f_{\mathsf{pad}_1}(K', M[b-n-31, t])$
```

Figure 23: Game $G_1'' \sim G_3''$

Lemma 9.27. For any prf-adversary A of q queries, the following holds.

```
Adversary $Q_A^{O(\star_n,\star_{32}||\star_{b-n-32})}$, where $O$ is $f(\star_n,\star_{32}||K||\star_{b-n-32})$ or $g(\star_n||\star_{32}||\star_{b-n-32})$.

Run $A$ as follows:
    On query $M$ of $A$, $K' \leftarrow O(IV, M[1, b-n])$ // $|M| = t$
    Reply $\mathrm{pfCM}^1 - \mathrm{MD}^f_{\mathsf{pad}_1}(K', M[b-n, t])$ to $A$
    Let $T$ be the final output of $A$
Return $T$
```

```
Adversary $V_A^{O_1,\cdots,O_q}$, where $O_i$ is $f(RK(\star,K_i)||\star)$ or $g_i(RK(\star,K)||\star)$.

Randomly choose $j$ from $[1, l-1]$ and $i \leftarrow j$ and $s \leftarrow 0$
Run $A$ as follows:
    On query $t$-th query $M^t$ of $A$,    // $1 \le t \le q$
      Let $\mathsf{pad}(M^t) = M_1^t||\cdots||M_{m_t}^t$ where $|M_1^t| = b-n$, $|M_j^t| = b$ for $2 \le j \le m_t$    // $m_t \le l$
      if $m_t \le i-1$ then $a^t \xleftarrow{\$} \{0,1\}^n$ and return $a^t$ to $A$
      else (namely $m_t \ge i$),
        if $(M_1^t,\cdots,M_i^t) \neq (M_1^r,\cdots,M_i^r)$ for all $r < t$
        then $s \leftarrow s+1$ and let $c^t = s$ and $a^t \xleftarrow{\$} \{0,1\}^n$
        else if $(M_1^t,\cdots,M_i^t) = (M_1^r,\cdots,M_i^r)$ & $((m_t = i$ & $m_r = i)$ or $(m_t \neq i$ & $m_r \neq i))$ for some $r$ with $r < t$
            then let $c^t \leftarrow c^r$ and $a^t \leftarrow a^r$
            else if $(M_1^t,\cdots,M_i^t) = (M_1^r,\cdots,M_i^r)$ & $(m_t = i$ or $m_r = i)$ for some $r$ with $r < t$
                then $s \leftarrow s+1$ and let $c^t = s$ and $a^t \xleftarrow{\$} \{0,1\}^n$
        if $m^t > i+1$ then $a^t \leftarrow O_{c^t}(\phi_i, M_{i+1}^t)$
        if $m^t = i+1$ then $a^t \leftarrow O_{c^t}(\phi_P, M_{i+1}^t)$
        return $\mathrm{pfCM}^{\mathrm{i}+1} - \mathrm{MD}^f(a^t, M_{i+2}^t||\cdots||M_{m_t}^t)$ to $A$    // $\mathrm{pfCM}^{\mathrm{i}+1} - \mathrm{MD}^f(a^t, \mathsf{null}) = a^t$
Let $T$ be the final output of $A$
Return $T$
```

Figure 24: Adversary Q_A and V_A: P is the last counter value of pfCM-MD.

$$\Pr[K_1,\cdots,K_q \xleftarrow{\$} \{0,1\}^n : V_{A,i=1}^{f(RK(\star,K_\star),\star)} = 1] = \Pr[K \xleftarrow{\$} \{0,1\}^n : A^{F(K,\star)} = 1],$$
$$\Pr[K_1,\cdots,K_q \xleftarrow{\$} \{0,1\}^n : V_{A,i=l-1}^{f(RK(\star,K_\star),\star)} = 1] = \Pr[g \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^*,\{0,1\}^n) : A^{g(\star)} = 1],$$

where $F(K,\star)$ denotes $\mathrm{pfCM}^1 - \mathrm{MD}^f_{\mathsf{pad}_1}(g(IV,\star_{32}||K||\star_{b-n-32}),\ \star)$.

Proof. It is clear by the construction of V_A in Fig. 24. ■

Theorem 9.28. For any prf-adversary A with q queries, there exist adversaries V_A such that

$$\mathbf{Adv}^{\mathsf{prf}}_{\mathrm{pfCM}^1 - \mathrm{MD}^f_{\mathsf{pad}_1}(g(IV,\star_{32}||K||\star_{b-n-32}),\ \star)}(A) = (l-1)\cdot \mathbf{Adv}^{\mathsf{multi\text{-}rka\text{-}prf}}_{f(RK(\star,K_\star),\star),\Phi_4}(V_A),$$

where V_A is defined in Fig. 24 and V_A can make at most q queries. $g \xleftarrow{\$} \mathsf{Maps}(\{0,1\}^b,\{0,1\}^n)$. We assume that for each query M of A, the b-bit block length of $\mathsf{pad}_1(M)$ is at most l. $\Phi_4 = \{\phi_1,\cdots,\phi_l,\phi_P\}$, where $\phi_i(X) = X \oplus i$ and P is the last counter of pfCM-MD. We assume that $\{\phi^1,\cdots,\phi^q\} \subset \{\phi_P,\phi_j\}$ for some j, where (i^t,ϕ^t,X^t) is t-th query of V_A. In other words, even though V_A can make queries to any one of $\{O_1,O_2,\cdots,O_q\}$, V_A can use at most two related-key-deriving (RKD) functions ϕ's from Φ_4.

Proof. We let $F(K,\star)$ be $\mathrm{pfCM}^1 - \mathrm{MD}^f_{\mathsf{pad}_1}(g(IV,\star_{32}||K||\star_{b-n-32}),\ \star)$.

$$\mathbf{Adv}^{\mathsf{multi\text{-}rka\text{-}prf}}_{f(RK(\star,K_\star),\star),\Phi_4}(V_A)$$
$$= |\Pr[K_1,\cdots,K_q \xleftarrow{\$} \{0,1\}^n : V_A^{f(RK(\star,K_\star),\star)} = 1]$$

$$
\begin{aligned}
&\quad - \Pr[g_1,\cdots,g_q \overset{\$}{\leftarrow} \mathsf{Maps}(\{0,1\}^{n+b},\{0,1\}^n);K\overset{\$}{\leftarrow}\{0,1\}^n:V_A^{g_\star(RK(\star,K),\star)}=1]| \\
&= |\textstyle\sum_{j=1}^{l-1}\Pr[K_1,\cdots,K_q \overset{\$}{\leftarrow} \{0,1\}^n : V_{A,i=j}^{f(RK(\star,K_\star),\star)} = 1]\cdot\frac{1}{l-1} \\
&\quad - \textstyle\sum_{j=1}^{l-1}\Pr[g_1,\cdots,g_q\overset{\$}{\leftarrow}\mathsf{Maps}(\{0,1\}^{n+b},\{0,1\}^n);K\overset{\$}{\leftarrow}\{0,1\}^n:V_{A,i=j}^{g_\star(RK(\star,K),\star)}=1]\cdot\frac{1}{l-1}| \\
&= \tfrac{1}{l-1}|\Pr[K \overset{\$}{\leftarrow} \{0,1\}^n : V^{F(K,\star)} = 1] - \Pr[g \overset{\$}{\leftarrow} \mathsf{Maps}(\{0,1\}^*,\{0,1\}^n) : V^{g(\star)} = 1]| \\
&= \tfrac{1}{l-1}\mathbf{Adv}_{F(K,\star)}^{\mathsf{prf}}(A). \qquad \blacksquare
\end{aligned}
$$

The second equality follows from the definition of V_A in Fig. 24 and the third equality follows from Lemma 9.26 and Lemma 9.27.

Theorem 9.29. For any prf-adversary A with q queries, there exist a prf-adversary Q_A such that

$$
\begin{aligned}
\mathbf{Adv}^{\mathsf{prf}}_{\mathrm{pfCM}^0-\mathrm{MD}^f_{\mathsf{pad}}(IV,\ \star_{32}||K||\ \star)}(A) \le\ & \mathbf{Adv}^{\mathsf{prf}}_{\mathrm{pfCM}^1-\mathrm{MD}^f_{\mathsf{pad}_1}(g(IV,\star_{32}||K||\star_{b-n-32}),\ \star)}(A) \\
&+ \mathbf{Adv}^{\mathsf{prf}}_{f(\star,\star_{32}||K||\star_{b-n-32})}(Q_A),
\end{aligned}
$$

where Q_A can make q queries of the form $(IV||\star_{32}||\star_{b-n-32})$ and is defined in Fig. 24, $\star_i$ means any i-bit string, and $g \overset{\$}{\leftarrow} \mathsf{Maps}(\{0,1\}^b,\{0,1\}^n)$.

Proof. By the definition of the prf-advantage, $\mathbf{Adv}^{\mathsf{prf}}_{\mathrm{pfCM}^0-\mathrm{MD}^f_{\mathsf{pad}}(IV,\ \star_{32}||K||\ \star)}(A) = |\Pr[A^{G_3''} = 1] - \Pr[A^{G_1''} = 1]|$. So, we can get above theorem with Lemma 9.24 $\sim$ Lemma 9.25. ■

Corollary 9.30. For any adversary A with q queries, there exist adversaries Q_A and V_A such that

$$
\mathbf{Adv}^{\mathsf{prf}}_{\mathrm{pfCM}^0-\mathrm{MD}^f_{\mathsf{pad}}(IV,\ \star_{32}||K||\ \star)}(A) \le (l-1)\cdot\mathbf{Adv}^{\mathsf{multi\text{-}rka\text{-}prf}}_{f(RK(\star,K_\star),\star),\Phi_4}(V_A)+\mathbf{Adv}^{\mathsf{prf}}_{f(\star,\star_{32}||K||\star_{b-n-32})}(Q_A),
$$

Q_A, V_A and Φ_4 are defined as before.

Proof. This holds by Theorem 9.28 and 9.29. ■

A.6. eTCR Security Analysis of a pfCM-MD Domain Extension with the Message Randomization Function in NIST SP 800-16

Here, we provide eTCR security analysis of pfCM^0-MD with the message randomization (in short, mr) in NIST SP 800-16, where pfCM^0-MD is the domain extension of ARIRANG. More precisely, we define a hash family $\mathbf{H} = \{\mathrm{pfCM}^0 - \mathrm{MD}^f_{\mathsf{pad}}(IV, \mathsf{mr}(r, M))\}_{r\in\cup_{80\le i\le 1024}\{0,1\}^i}$, where mr is the message randomization function in NIST SP 800-16, and $M \in \{0,1\}^*$. And for ARIRANG, we let $\mathsf{pad}(M) = M||10^t||\mathsf{bin}_d(|M|)$, where $\mathsf{bin}_d(|M|)$ is the d-bit representation of the bit-length of M and t is the smallest non-negative integer such that $\mathsf{pad}(M)$ is a multiple of b-bit block. In cases of ARIRANG-224 and ARIRANG-256, $d = 64$, and in cases of ARIRANG-384 and ARIRANG-512, $d = 128$.

Message Randomization (mr) in NIST SP 800-16

$\boxed{\mathsf{mr}(r, M) = M' :}$

```
1  If ($|M| \geq |r| - 1$) then $padding = 1$ else $padding = 1||0^{|r|-|M|-1}$
2  $m = M||padding$
3  Let $n = |r|$
4  If ($n > 1024$) then stop and output an error indicator
5  $counter = \lfloor |m|/n \rfloor$
6  $remainder = (|m|\ mod\ n)$
7  Concatenate $counter$ copies of the $r$ to the $remainder$ left-most bits of the $r$ to get $R$ such that $|R| = |m|$
       $R = r||r|| \cdots ||r||r[0 \ldots (remainder - 1)]$
8  $r\_length\_indicator = r\_length\_indicator\_generation(n)$
9  $M' = r||(m \oplus R)||r\_length\_indicator$
10 Return $M'$;
```

$\boxed{r\_length\_indicator\_generation(n) :}$ // 80$\leq n \leq$1024 and the output is 16-bit.

```
1  $A = n$ and $B = A\ mod\ 2$
2  If  $B = 0$ then $b_{15} = 0$ else $b_{15} = 1$
3  For $i = 14$ to 0
       3.1 $A = \lfloor A/2 \rfloor$ and $B = A\ mod\ 2$
       3.2 If  $B = 0$ then $b_i = 0$ else $b_i = 1$
4  $r\_length\_indicator = b_0||b_1|| \cdots ||b_{15}$
5  Return $r\_length\_indicator$;
```

eTCR Security Analysis of pfCM^0-MD with mr in NIST SP 800-16

Lemma 9.31. For any $(r, M) \neq (r', M')$, $\mathsf{mr}(r, M) \neq \mathsf{mr}(r', M')$,

where mr is the message randomization in NIST SP 800-16.

Proof. If $\mathsf{mr}(r, M) = \mathsf{mr}(r', M')$, then by the definition of mr the following equality hold.

$$r||(m \oplus R)||r\_length\_indicator = r'||(m' \oplus R')||r'\_length\_indicator. \qquad (1)$$

Since $|r\_length\_indicator| = |r'\_length\_indicator| = 16$ by the definition of mr, $r\_length\_indicator$ should be equal to $r'\_length\_indicator$, which means that $|r| = |r'|$. And since r and r' are located in the first some bits in the equality (1), we know that $r = r'$, which means also that $m = m'$ and $R = R'$, where R and R' are generated from the identical r (=r'). Finally, by the padding method defined in line 1 and 2 of mr, $m = m'$ means that $M = M'$. Therefore, the lemma holds. ■

In the following theorem, it is shown that the eTCR-advantage of A on the pfCM^0-MD with mr is bounded by the eSPR†-advantage of A on the pfCM^0-MD with mr.

Theorem 9.32. For any eTCR-adversary A, there exists a SPR†-adversary B_A such that

$$\mathbf{Adv}_{\mathbf{H}}^{\mathsf{eTCR}}(A) \le l \cdot \mathbf{Adv}_{\mathbf{H}}^{\mathsf{eSPR}^\dagger}(B_A),$$

where $\mathbf{H} = \{\text{pfCM}^0 - \text{MD}_{\mathsf{pad}}^f(IV, \mathsf{mr}(r, \star))\}_{r \in \cup_{80 \le i \le 1024}\{0,1\}^i}$, and mr is the message randomization function in NIST SP 800-16. B_A is defined in Fig. 25. l is defined in Fig. 25.

Proof. Let $H_r(IV, \star)$ be $\text{pfCM}^0 - \text{MD}_{\mathsf{pad}}^f(IV, \mathsf{mr}(r, \star))$. Δ is the statement that "$(M, \texttt{State}) \overset{\$}{\leftarrow} A; r \overset{\$}{\leftarrow} \mathcal{R}; (r', M') \overset{\$}{\leftarrow} A(r, M, \texttt{State}) : (r, M) \ne (r', M')$ and $H_r(IV, M) = H_{r'}(IV, M')$". Υ is the statement that "$(M, \texttt{State}) \overset{\$}{\leftarrow} B_A; r \overset{\$}{\leftarrow} \cup_{80 \le j \le 1024}\{0,1\}^j; i \overset{\$}{\leftarrow} [1, l]; (c', m') \overset{\$}{\leftarrow} B_A(i, r, M, \texttt{State}) : (c, m) = H_r(IV, M)[i]$ and $(c, m) \ne (c', m')$ and $f(c, m) = f(c', m')$".

$$\begin{aligned}\mathbf{Adv}_{\mathbf{H}}^{\mathsf{eTCR}}(A) &= \Pr[\Delta] = \Pr[\Delta \wedge (|\mathsf{mr}(r, M)| = |\mathsf{mr}(r', M')|)] + \Pr[\Delta \wedge (|\mathsf{mr}(r, M)| \ne |\mathsf{mr}(r', M')|)] \\ &\le l \cdot \Pr[\Upsilon \wedge (|\mathsf{mr}(r, M)| = |\mathsf{mr}(r', M')|)] + l \cdot \Pr[\Upsilon \wedge (|\mathsf{mr}(r, M)| \ne |\mathsf{mr}(r', M')|)] \\ &= l \cdot \Pr[\Upsilon] = l \cdot \mathbf{Adv}_{\mathbf{H}}^{\mathsf{eSPR}^\dagger}(B_A).\end{aligned}$$

The equality of the second line is guaranteed by Claim 1 and Claim 2.

Claim 1. $\Pr[\Delta \wedge (|\mathsf{mr}(r, M)| = |\mathsf{mr}(r', M')|)] \le l \cdot \Pr[\Upsilon \wedge (|\mathsf{mr}(r, M)| = |\mathsf{mr}(r', M')|)]$.

Proof. Since $\text{pfCM}^0 - \text{MD}_{\mathsf{pad}}^f(IV, \star)$ preserves the collision-resistance of f and $|\mathsf{mr}(r, M)| = |\mathsf{mr}(r', M')|$, if $(\mathsf{mr}(r, M), \mathsf{mr}(r', M'))$ is a collision pair of $\text{pfCM}^0 - \text{MD}_{\mathsf{pad}}^f(IV, \star)$, there exists a i such that $f(c, x) = f(c', x')$, where $(c, x) = \text{pfCM}^0 - \text{MD}_{\mathsf{pad}}^f(IV, \mathsf{mr}(r, M))[i]$, $(c', x') = \text{pfCM}^0 - \text{MD}_{\mathsf{pad}}^f(IV, \mathsf{mr}(r', M'))[i]$, and $(c, x) \ne (c', x')$. In the definition of B_A in Fig. 25, the probability that i is correctly guessed is $1/l$. So, the Claim 1 holds.

Claim 2. $\Pr[\Delta \wedge (|\mathsf{mr}(r, M)| \ne |\mathsf{mr}(r', M')|)] = l \cdot \Pr[\Upsilon \wedge (|\mathsf{mr}(r, M)| \ne |\mathsf{mr}(r', M')|)]$.

Proof. Since $\mathsf{pad}(M) = M||10^t||\mathsf{bin}_d(|M|)$, if $|\mathsf{mr}(r, M)| \ne |\mathsf{mr}(r', M')|$, and $(\mathsf{mr}(r, M), \mathsf{mr}(r', M'))$ is a collision pair of $\text{pfCM}^0 - \text{MD}_{\mathsf{pad}}^f(IV, \star)$, then $f(c, x) = f(c', x')$, where $(c, x) = \text{pfCM}^0 - \text{MD}_{\mathsf{pad}}^f$ $(IV, \mathsf{mr}(r, M))[l]$, $(c', x') = \text{pfCM}^0 - \text{MD}_{\mathsf{pad}}^f$ $(IV, \mathsf{mr}(r', M'))[l']$, and $(c, x) \ne (c', x')$. In the definition of B_A in Fig. 25, the probability that $i = l$ is $1/l$. So, the Claim 2 holds. ■

```
Adversary $B_A$.

000 Run $A$ and obtain $M$ from $A$ and Choose $M$ as a target message.
100 Given $r \xleftarrow{\$} \cup_{80 \le i \le 1024}\{0,1\}^i$
200 Given $i \xleftarrow{\$} [1, l]$        // $l = \mathsf{Len}_f(\mathrm{pfCM}^0 - \mathrm{MD}^f_{\mathsf{pad}}(IV, \mathsf{mr}(r, M)))$
300 Forward $r$ to $A$.
400 Obtain $(r', M')$ from $A$ and let $l' = \mathsf{Len}_f(\mathrm{pfCM}^0 - \mathrm{MD}^f_{\mathsf{pad}}(IV, \mathsf{mr}(r', M')))$.
500 if $|\mathsf{mr}(r, M)| = |\mathsf{mr}(r', M')|$ then $(c', m') \leftarrow \mathrm{pfCM}^0 - \mathrm{MD}^f_{\mathsf{pad}}(IV, \mathsf{mr}(r', M'))[i]$
600 if $|\mathsf{mr}(r, M)| \neq |\mathsf{mr}(r', M')|$ then $(c', m') \leftarrow \mathrm{pfCM}^0 - \mathrm{MD}^f_{\mathsf{pad}}(IV, \mathsf{mr}(r', M'))[l']$
700 Return $(c', m')$
```

Figure 25: Adversary B_A: $l' = \mathsf{Len}_f(\mathrm{pfCM}^0 - \mathrm{MD}^f_{\mathsf{pad}}(IV, \mathsf{mr}(r', M')))$ is the number of computations of the compression function f when computing $\mathrm{pfCM}^0 - \mathrm{MD}^f_{\mathsf{pad}}(IV, \mathsf{mr}(r', M'))$ for any r, where M is generated by the adversary A. mr is the message randomization function in NIST SP 800-16.

Appendix B

In this section, we demonstrate local collision patterns and each conditions with maximum probability. Figure 26 is shape of local collision patterns and table 16 is condition and probability for each local collision pattern. In figure 26 and table 16, each variable of form $X - Y - Z - W$ is as follows:

- X means the number of step to construct local collision.
- Y means the shape of the local collision pattern.
- Z means position of perturbation message difference of the step to be started the local collision.
- W means the step to be started the local collision.

Specially the conditions of each local collision pattern means conditions that expanded message words which are made from original input message words should be satisfy. The local collision patterns of table 16 are pattern with maximize probability which we have done exhaustive search through simulation. In table 16, we considered two probability which one is probability to satisfy local collision in first round and the other is probability to satisfy local collision in the other rounds. In case first round, attacker can control message value and message difference to construct local collision, but he can not control message value and message difference because these values are already fixed in first round.

Table 16: Local collision patterns condition of ARIRANG.

pattern	condition ($t = 0, 20, 40, 60$)	probability	
		$t = 0$	$t \neq 0$
$5-A-L-2$	$\Delta W_{\sigma(2+t)} = \Delta W_{\sigma(11+t)}$ $\Delta W_{\sigma(5+t)} \oplus \Delta W_{\sigma(7+t)} \oplus \Delta W_{\sigma(9+t)} = \Delta W_{\sigma(10+t)} = \texttt{0x00000000}$	0	0
$5-A-L-5$	$(\Delta W_{\sigma(8+t)} \lll 31) = \Delta W_{\sigma(11+t)}$ $\Delta W_{\sigma(13+t)} \oplus \Delta W_{\sigma(15+t)} \oplus \Delta W_{\sigma(17+t)} = \Delta W_{\sigma(0+t)} = \texttt{0x00000000}$	0	0
$5-A-R-2$	$\Delta W_{\sigma(3+t)} = \Delta W_{\sigma(10+t)}$ $\Delta W_{\sigma(4+t)} \oplus \Delta W_{\sigma(6+t)} \oplus \Delta W_{\sigma(8+t)} = \Delta W_{\sigma(11+t)} = \texttt{0x00000000}$	0	0
$5-A-R-5$	$(\Delta W_{\sigma(9+t)} \lll 19) = \Delta W_{\sigma(10+t)}$ $\Delta W_{\sigma(12+t)} \oplus \Delta W_{\sigma(14+t)} \oplus \Delta W_{\sigma(16+t)} = \Delta W_{\sigma(1+t)} = \texttt{0x00000000}$	0	0
$6-A-L-3$	$(\Delta W_{\sigma(4+t)} \oplus \Delta W_{\sigma(6+t)} \lll 31) = \Delta W_{\sigma(11+t)}$ $\Delta W_{\sigma(7+t)} \oplus \Delta W_{\sigma(9+t)} = \Delta W_{\sigma(10+t)} = \texttt{0x00000000}$ $\Delta W_{\sigma(13+t)} \oplus \Delta W_{\sigma(15+t)} = \Delta W_{\sigma(0+t)} = \texttt{0x00000000}$	2^{-64}	2^{-224}
$6-A-R-3$	$(\Delta W_{\sigma(5+t)} \oplus \Delta W_{\sigma(7+t)} \lll 19) = \Delta W_{\sigma(10+t)}$ $\Delta W_{\sigma(6+t)} \oplus \Delta W_{\sigma(8+t)} = \Delta W_{\sigma(11+t)} = \texttt{0x00000000}$ $\Delta W_{\sigma(12+t)} \oplus \Delta W_{\sigma(14+t)} = \Delta W_{\sigma(1+t)} = \texttt{0x00000000}$	2^{-64}	2^{-224}
$7-A-L-1$	$(\Delta W_{\sigma(13+t)} \lll 5) = \Delta W_{\sigma(0+t)}$ $(\Delta W_{\sigma(4+t)} \lll 31) = \Delta W_{\sigma(11+t)}$ $\Delta W_{\sigma(3+t)} \oplus \Delta W_{\sigma(5+t)} \oplus \Delta W_{\sigma(7+t)} \oplus \Delta W_{\sigma(9+t)} = \Delta W_{\sigma(10+t)} = \texttt{0x00000000}$	2^{-64}	2^{-224}
$7-A-L-4$	$(\Delta W_{\sigma(9+t)} \lll 19) = \Delta W_{\sigma(10+t)}$ $(\Delta W_{\sigma(6+t)} \lll 31) = \Delta W_{\sigma(11+t)}$ $\Delta W_{\sigma(13+t)} \oplus \Delta W_{\sigma(15+t)} \oplus \Delta W_{\sigma(17+t)} \oplus \Delta W_{\sigma(19+t)} = \Delta W_{\sigma(0+t)} = \texttt{0x00000000}$	2^{-64}	2^{-224}
$7-A-R-1$	$(\Delta W_{\sigma(12+t)} \lll 11) = \Delta W_{\sigma(1+t)}$ $(\Delta W_{\sigma(5+t)} \lll 19) = \Delta W_{\sigma(10+t)}$ $\Delta W_{\sigma(2+t)} \oplus \Delta W_{\sigma(4+t)} \oplus \Delta W_{\sigma(6+t)} \oplus \Delta W_{\sigma(8+t)} = \Delta W_{\sigma(11+t)} = \texttt{0x00000000}$	2^{-64}	2^{-224}
$7-A-R-4$	$(\Delta W_{\sigma(7+t)} \lll 19) = \Delta W_{\sigma(10+t)}$ $(\Delta W_{\sigma(8+t)} \lll 31) = \Delta W_{\sigma(11+t)}$ $\Delta W_{\sigma(12+t)} \oplus \Delta W_{\sigma(14+t)} \oplus \Delta W_{\sigma(16+t)} \oplus \Delta W_{\sigma(18+t)} = \Delta W_{\sigma(1+t)} = \texttt{0x00000000}$	2^{-64}	2^{-224}
$7-B-L-1$	$\Delta W_{\sigma(13+t)} \lll 5) = \Delta W_{\sigma(0+t)}$ $(\Delta W_{\sigma(2+t)} \oplus \Delta W_{\sigma(4+t)} \lll 31) = \Delta W_{\sigma(11+t)}$ $\Delta W_{\sigma(3+t)} \oplus \Delta W_{\sigma(5+t)} \oplus \Delta W_{\sigma(7+t)} \oplus \Delta W_{\sigma(9+t)} = \Delta W_{\sigma(10+t)} = \texttt{0x00000000}$	2^{-32}	2^{-224}
$7-B-L-4$	$\Delta W_{\sigma(9+t)} \lll 19) = \Delta W_{\sigma(10+t)}$ $(\Delta W_{\sigma(6+t)} \oplus \Delta W_{\sigma(8+t)} \lll 31) = \Delta W_{\sigma(11+t)}$ $\Delta W_{\sigma(13+t)} \oplus \Delta W_{\sigma(15+t)} \oplus \Delta W_{\sigma(17+t)} \oplus \Delta W_{\sigma(19+t)} = \Delta W_{\sigma(0+t)} = \texttt{0x00000000}$	2^{-64}	2^{-224}
$7-B-R-1$	$\Delta W_{\sigma(12+t)} \lll 11) = \Delta W_{\sigma(1+t)}$ $(\Delta W_{\sigma(3+t)} \oplus \Delta W_{\sigma(5+t)} \lll 19) = \Delta W_{\sigma(10+t)}$ $\Delta W_{\sigma(2+t)} \oplus \Delta W_{\sigma(4+t)} \oplus \Delta W_{\sigma(6+t)} \oplus \Delta W_{\sigma(8+t)} = \Delta W_{\sigma(11+t)} = \texttt{0x00000000}$	2^{-32}	2^{-224}
$7-B-R-4$	$\Delta W_{\sigma(8+t)} \lll 31) = \Delta W_{\sigma(11+t)}$ $(\Delta W_{\sigma(7+t)} \oplus \Delta W_{\sigma(9+t)} \lll 19) = \Delta W_{\sigma(10+t)}$ $\Delta W_{\sigma(12+t)} \oplus \Delta W_{\sigma(14+t)} \oplus \Delta W_{\sigma(16+t)} \oplus \Delta W_{\sigma(18+t)} = \Delta W_{\sigma(1+t)} = \texttt{0x00000000}$	2^{-64}	2^{-224}

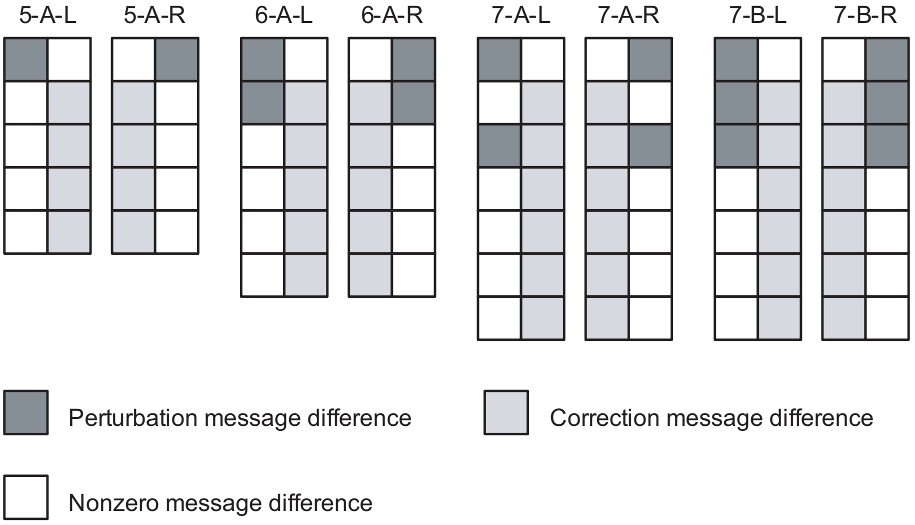
5-A-L
5-A-R
6-A-L
6-A-R
7-A-L
7-A-R
7-B-L
7-B-R
Perturbation message difference
Correction message difference
Nonzero message difference

Figure 26: Local collision patterns of ARIRANG.

Appendix C

In this section, we represent our simulation results of ARIRANG in various versions. Table 17 and Table 18 are the results on a 32-bit processor. Table 19 and Table 20 are the results on a 64-bit processor.

Table 17: Execution times of various message sizes for ARIRANG using one S-box on a 32-bit processor

Algorithm	Message (Byte)	Cycles/Msg	Cycles/Byte	Message (Byte)	Cycles/Msg	Cycles/Byte
ARIRANG-224	8	3220.7	402.6	16	3201.7	200.1
	32	3261.2	101.9	64	6284.5	98.2
	128	9348.4	73	256	16938.4	66.2
	512	38798.8	75.8	1024	93569.5	91.4
	1M	97645350	93.1	10M	976421875	93.1
ARIRANG-256	8	3203	400.4	16	3222	201.4
	32	3240.9	101.3	64	6285.8	98.2
	128	9368.6	73.2	256	16938.4	66.2
	512	39947.4	78	1024	93926.3	91.7
	1M	97544150	93	10M	976124600	93.1
ARIRANG-384	8	31268.3	3908.5	16	31111.4	1944.5
	32	30992.5	968.5	64	31625	494.1
	128	63981.2	499.9	256	99836.3	390
	512	178523.1	348.7	1024	332359.8	324.6
	1M	306174275	292	10M	3061198800	291.9
ARIRANG-512	8	31230.3	3903.8	16	31210.1	1950.6
	32	30992.5	968.5	64	31170.9	487
	128	64356.9	502.8	256	100409.4	392.2
	512	178285.3	348.2	1024	332280.1	324.5
	1M	306167950	292	10M	3061198800	291.9

Table 18: Execution times of various message sizes for ARIRANG using seven S-box on a 32-bit processor

Algorithm	Message (Byte)	Cycles/Msg	Cycles/Byte	Message (Byte)	Cycles/Msg	Cycles/Byte
ARIRANG-224	8	3024.1	378	16	3008.2	188
	32	3047.6	95.2	64	5902	92.2
	128	8763.9	68.5	256	14598.9	57
	512	25956	50.7	1024	48780.9	47.6
	1M	46797157	44.6	10M	468844420	44.7
ARIRANG-256	8	3027.9	378.5	16	3020.3	188.8
	32	3043.8	95.1	64	5897.9	92.2
	128	8783.9	68.6	256	14519.7	56.7
	512	25912.8	50.6	1024	48743	47.6
	1M	46785266	44.6	10M	468682500	44.7
ARIRANG-384	8	23616	2952	16	23604.1	1475.3
	32	23600.3	737.5	64	23584.4	368.5
	128	46710.1	364.9	256	69843.7	272.8
	512	115992.7	226.5	1024	208289.8	203.4
	1M	189152920	180.4	10M	1890147820	180.3
ARIRANG-512	8	23623.6	2953	16	23651.5	1478.2
	32	23623.9	738.2	64	23587.9	368.6
	128	46717.7	365	256	69820.2	272.7
	512	115988.6	226.5	1024	208408.8	203.5
	1M	189181003	180.4	10M	1890385640	180.3

Table 19: Execution times of various message sizes for ARIRANG using one S-box on a 64-bit processor

Algorithm	Message (Byte)	Cycles/Msg	Cycles/Byte	Message (Byte)	Cycles/Msg	Cycles/Byte
ARIRANG-224	8	3142.3	392.8	16	3103	193.9
	32	3240.9	101.3	64	6048	94.5
	128	8933.4	69.8	256	16582.9	64.8
	512	39886.7	77.9	1024	95072.3	92.8
	1M	100409375	95.8	10M	999546075	95.3
ARIRANG-256	8	3142.3	392.8	16	3103	193.9
	32	3142.3	98.2	64	6068.2	94.8
	128	8954.9	70	256	16346.3	63.9
	512	39373.1	76.9	1024	93768.1	91.6
	1M	99814825	95.2	10M	998856650	95.3
ARIRANG-384	8	38582.5	4822.8	16	38443.4	2402.7
	32	39017.7	1219.3	64	38502.8	601.6
	128	79477.4	620.9	256	121440	474.4
	512	213883.7	417.7	1024	396478.8	387.2
	1M	361214425	344.5	10M	3610291025	344.3
ARIRANG-512	8	38206.8	4775.8	16	38306.7	2394.2
	32	38385.2	1199.5	64	38463.6	601
	128	79280.1	619.4	256	122132	477.1
	512	213883.7	417.7	1024	397011.4	387.7
	1M	361012025	344.3	10M	3610986775	344.4

Table 20: Execution times of various message sizes for ARIRANG using seven S-boxes on a 64-bit processor

Algorithm	Message (Byte)	Cycles/Msg	Cycles/Byte	Message (Byte)	Cycles/Msg	Cycles/Byte
ARIRANG-224	8	2438.9	304.9	16	2383.5	149
	32	2423.2	75.7	64	4593.5	71.8
	128	6759.7	52.8	256	11149.7	43.6
	512	19903.8	38.9	1024	37552.8	36.7
	1M	35787609	34.1	10M	359022180	34.2
ARIRANG-256	8	2391.6	299	16	2395.7	149.7
	32	2419.2	75.6	64	4581.6	71.6
	128	6783.7	53	256	11147.7	43.5
	512	19864.5	38.8	1024	37358	36.5
	1M	35755731	34.1	10M	358546540	34.2
ARIRANG-384	8	26675.8	3334.5	16	26663.9	1666.5
	32	26430.7	826	64	26683.7	416.9
	128	52778.1	412.3	256	78860.9	308.1
	512	131085.6	256	1024	235446.9	229.9
	1M	213792843	203.9	10M	2138363590	203.9
ARIRANG-512	8	26648.2	3331	16	26651.8	1665.7
	32	26683.4	833.9	64	26675.6	416.8
	128	52766.2	412.2	256	78856.8	308
	512	131069.9	256	1024	235451.9	229.9
	1M	213824468	203.9	10M	2138285160	203.9