Independent Submission
Request for Comments: 5794
Category: Informational
ISSN: 2070-1721

J. Lee
J. Lee
J. Kim
D. Kwon
C. Kim
NSRI

March 2010

A Description of the ARIA Encryption Algorithm

Abstract

This document describes the ARIA encryption algorithm. ARIA is a 128-bit block cipher with 128-, 192-, and 256-bit keys. The algorithm consists of a key scheduling part and data randomizing part.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This is a contribution to the RFC Series, independently of any other RFC stream. The RFC Editor has chosen to publish this document at its discretion and makes no statement about its value for implementation or deployment. Documents approved for publication by the RFC Editor are not a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc5794.

1. Introduction

1.1. ARIA Overview

ARIA is a general-purpose block cipher algorithm developed by Korean cryptographers in 2003. It is an iterated block cipher with 128-, 192-, and 256-bit keys and encrypts 128-bit blocks in 12, 14, and 16 rounds, depending on the key size. It is secure and suitable for most software and hardware implementations on 32-bit and 8-bit processors. It was established as a Korean standard block cipher algorithm in 2004 [ARIAKS] and has been widely used in Korea, especially for government-to-public services. It was included in PKCS #11 in 2007 [ARIAPKCS].

2. Algorithm Description

The algorithm consists of a key scheduling part and data randomizing part.

2.1. Notations

The following notations are used in this document to describe the algorithm.

```
^   bitwise XOR operation
<<< left circular rotation
>>> right circular rotation
||  concatenation of bit strings
0x  hexadecimal representation
```

2.2. Key Scheduling Part

Let K denote a master key of 128, 192, or 256 bits. Given the master key K, we first define 128-bit values KL and KR as follows.

```
KL || KR = K || 0 ... 0,
```

where the number of zeros is 128, 64, or 0, depending on the size of K. That is, KL is set to the leftmost 128 bits of K and KR is set to the remaining bits of K (if any), right-padded with zeros to a 128-bit value. Then, we define four 128-bit values (W0, W1, W2, and W3) as the intermediate round values appearing in the encryption of KL || KR by a 3-round, 256-bit Feistel cipher.

```
W0 = KL,
W1 = FO(W0, CK1) ^ KR,
W2 = FE(W1, CK2) ^ W0,
W3 = FO(W2, CK3) ^ W1.
```

Here, FO and FE, respectively called odd and even round functions, are defined in Section 2.4.1. CK1, CK2, and CK3 are 128-bit constants, taking one of the following values.

```
C1 =  0x517cc1b727220a94fe13abe8fa9a6ee0
C2 =  0x6db14acc9e21c820ff28b1d5ef5de2b0
C3 =  0xdb92371d2126e9700324977504e8c90e
```

These values are obtained from the first 128*3 bits of the fractional part of 1/PI, where PI is the circle ratio. Now the constants CK1, CK2, and CK3 are defined by the following table.

Key size	CK1	CK2	CK3
128	C1	C2	C3
192	C2	C3	C1
256	C3	C1	C2

For example, if the key size is 192 bits, CK1 = C2, CK2 = C3, and CK3 = C1.

Once W0, W1, W2, and W3 are determined, we compute encryption round keys ek1, ..., ek17 as follows.

```
ek1  = W0 ^(W1 >>> 19),
ek2  = W1 ^(W2 >>> 19),
ek3  = W2 ^(W3 >>> 19),
ek4  = (W0 >>> 19) ^ W3,
ek5  = W0 ^ (W1 >>> 31),
ek6  = W1 ^ (W2 >>> 31),
ek7  = W2 ^ (W3 >>> 31),
ek8  = (W0 >>> 31) ^ W3,
ek9  = W0 ^ (W1 <<< 61),
ek10 = W1 ^ (W2 <<< 61),
ek11 = W2 ^ (W3 <<< 61),
ek12 = (W0 <<< 61) ^ W3,
ek13 = W0 ^ (W1 <<< 31),
ek14 = W1 ^ (W2 <<< 31),
ek15 = W2 ^ (W3 <<< 31),
ek16 = (W0 <<< 31) ^ W3,
ek17 = W0 ^ (W1 <<< 19).
```

The number of rounds depends on the size of the master key as follows.

Key size	Number of Rounds
128	12
192	14
256	16

Due to an extra key addition layer in the last round, 12-, 14-, and 16-round algorithms require 13, 15, and 17 round keys, respectively.

Decryption round keys are derived from the encryption round keys.

```
dk1 = ek{n+1},
dk2 = A(ek{n}),
dk3 = A(ek{n-1}),
...,
dk{n}= A(ek2),
dk{n+1}= ek1.
```

Here, A and n denote the diffusion layer of ARIA and the number of rounds, respectively. The diffusion layer A is defined in Section 2.4.3.

2.3. Data Randomizing Part

The data randomizing part of the ARIA algorithm consists of the encryption and decryption processes. The encryption and decryption processes use functions FO, FE, A, SL1, and SL2. These functions are defined in Section 2.4.

2.3.1. Encryption Process

2.3.1.1. Encryption for 128-Bit Keys

Let P be a 128-bit plaintext and K be a 128-bit master key. Let ek1, ..., ek13 be the encryption round keys defined by K. Then the ciphertext C is computed by the following algorithm.

```
P1  = FO(P  , ek1 );              // Round 1
P2  = FE(P1 , ek2 );              // Round 2
P3  = FO(P2 , ek3 );              // Round 3
P4  = FE(P3 , ek4 );              // Round 4
P5  = FO(P4 , ek5 );              // Round 5
P6  = FE(P5 , ek6 );              // Round 6
P7  = FO(P6 , ek7 );              // Round 7
P8  = FE(P7 , ek8 );              // Round 8
P9  = FO(P8 , ek9 );              // Round 9
P10 = FE(P9 , ek10);              // Round 10
P11 = FO(P10, ek11);              // Round 11
C   = SL2(P11 ^ ek12) ^ ek13;     // Round 12
```

2.3.1.2. Encryption for 192-Bit Keys

Let P be a 128-bit plaintext and K be a 192-bit master key. Let ek1, ..., ek15 be the encryption round keys defined by K. Then the ciphertext C is computed by the following algorithm.

```
P1  = FO(P  , ek1 );              // Round 1
P2  = FE(P1 , ek2 );              // Round 2
P3  = FO(P2 , ek3 );              // Round 3
P4  = FE(P3 , ek4 );              // Round 4
P5  = FO(P4 , ek5 );              // Round 5
P6  = FE(P5 , ek6 );              // Round 6
P7  = FO(P6 , ek7 );              // Round 7
P8  = FE(P7 , ek8 );              // Round 8
P9  = FO(P8 , ek9 );              // Round 9
P10 = FE(P9 , ek10);              // Round 10
P11 = FO(P10, ek11);              // Round 11
P12 = FE(P11, ek12);              // Round 12
P13 = FO(P12, ek13);              // Round 13
C   = SL2(P13 ^ ek14) ^ ek15;     // Round 14
```

2.3.1.3. Encryption for 256-Bit Keys

Let P be a 128-bit plaintext and K be a 256-bit master key. Let ek1, ..., ek17 be the encryption round keys defined by K. Then the ciphertext C is computed by the following algorithm.

```
P1 = FO(P  , ek1 );              // Round 1
P2 = FE(P1 , ek2 );              // Round 2
P3 = FO(P2 , ek3 );              // Round 3
P4 = FE(P3 , ek4 );              // Round 4
P5 = FO(P4 , ek5 );              // Round 5
P6 = FE(P5 , ek6 );              // Round 6
P7 = FO(P6 , ek7 );              // Round 7
P8 = FE(P7 , ek8 );              // Round 8
P9 = FO(P8 , ek9 );              // Round 9
P10= FE(P9 , ek10);              // Round 10
P11= FO(P10, ek11);              // Round 11
P12= FE(P11, ek12);              // Round 12
P13= FO(P12, ek13);              // Round 13
P14= FE(P13, ek14);              // Round 14
P15= FO(P14, ek15);              // Round 15
C  = SL2(P15 ^ ek16) ^ ek17;     // Round 16
```

2.3.2. Decryption Process

The decryption process of ARIA is the same as the encryption process except that encryption round keys are replaced by decryption round keys. For example, encryption round keys ek1, ..., ek13 of the 12-round ARIA algorithm are replaced by decryption round keys dk1, ..., dk13, respectively.

2.4. Components of ARIA

2.4.1. Round Functions

There are two types of round functions for ARIA. One is called an odd round function and is denoted by FO. It takes as input a pair (D,RK) of two 128-bit strings and outputs

```
FO(D,RK) = A(SL1(D ^ RK)).
```

The other is called an even round function and is denoted by FE. It takes as input a pair (D,RK) of two 128-bit strings and outputs

```
FE(D,RK) = A(SL2(D ^ RK)).
```

Functions SL1 and SL2, called substitution layers, are described in Section 2.4.2. Function A, called a diffusion layer, is described in Section 2.4.3.

2.4.2. Substitution Layers

ARIA has two types of substitution layers that alternate between rounds. Type 1 is used in the odd rounds, and type 2 is used in the even rounds.

Type 1 substitution layer SL1 is an algorithm that takes a 16-byte string x0 || x1 ||...|| x15 as input and outputs a 16-byte string y0 || y1 ||...|| y15 as follows.

```
y0 = SB1(x0),  y1 = SB2(x1),  y2 = SB3(x2),  y3 = SB4(x3),
y4 = SB1(x4),  y5 = SB2(x5),  y6 = SB3(x6),  y7 = SB4(x7),
y8 = SB1(x8),  y9 = SB2(x9),  y10= SB3(x10), y11= SB4(x11),
y12= SB1(x12), y13= SB2(x13), y14= SB3(x14), y15= SB4(x15).
```

Type 2 substitution layer SL2 is an algorithm that takes a 16-byte string x0 || x1 ||...|| x15 as input and outputs a 16-byte string y0 || y1 ||...|| y15 as follows.

```
y0 = SB3(x0),  y1 = SB4(x1),  y2 = SB1(x2),  y3 = SB2(x3),
y4 = SB3(x4),  y5 = SB4(x5),  y6 = SB1(x6),  y7 = SB2(x7),
y8 = SB3(x8),  y9 = SB4(x9),  y10= SB1(x10), y11= SB2(x11),
y12= SB3(x12), y13= SB4(x13), y14= SB1(x14), y15= SB2(x15).
```

Here, SB1, SB2, SB3, and SB4 are S-boxes that take an 8-bit string as input and output an 8-bit string. These S-boxes are defined by the following look-up tables.

SB1:

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

SB2:

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

SB3:

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

SB4:

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

For example, SB1(0x23) = 0x26 and SB4(0xef) = 0xd3. Note that SB3 and SB4 are the inverse functions of SB1 and SB2, respectively, and accordingly SL2 is the inverse of SL1.

2.4.3. Diffusion Layer

Diffusion layer A is an algorithm that takes a 16-byte string x0 || x1 || ... || x15 as input and outputs a 16-byte string y0 || y1 ||...|| y15 by the following equations.

```
y0  = x3 ^ x4 ^ x6 ^ x8  ^ x9  ^ x13 ^ x14,
y1  = x2 ^ x5 ^ x7 ^ x8  ^ x9  ^ x12 ^ x15,
y2  = x1 ^ x4 ^ x6 ^ x10 ^ x11 ^ x12 ^ x15,
y3  = x0 ^ x5 ^ x7 ^ x10 ^ x11 ^ x13 ^ x14,
y4  = x0 ^ x2 ^ x5 ^ x8  ^ x11 ^ x14 ^ x15,
y5  = x1 ^ x3 ^ x4 ^ x9  ^ x10 ^ x14 ^ x15,
y6  = x0 ^ x2 ^ x7 ^ x9  ^ x10 ^ x12 ^ x13,
y7  = x1 ^ x3 ^ x6 ^ x8  ^ x11 ^ x12 ^ x13,
y8  = x0 ^ x1 ^ x4 ^ x7  ^ x10 ^ x13 ^ x15,
y9  = x0 ^ x1 ^ x5 ^ x6  ^ x11 ^ x12 ^ x14,
y10 = x2 ^ x3 ^ x5 ^ x6  ^ x8  ^ x13 ^ x15,
y11 = x2 ^ x3 ^ x4 ^ x7  ^ x9  ^ x12 ^ x14,
y12 = x1 ^ x2 ^ x6 ^ x7  ^ x9  ^ x11 ^ x12,
y13 = x0 ^ x3 ^ x6 ^ x7  ^ x8  ^ x10 ^ x13,
y14 = x0 ^ x3 ^ x4 ^ x5  ^ x9  ^ x11 ^ x14,
y15 = x1 ^ x2 ^ x4 ^ x5  ^ x8  ^ x10 ^ x15.
```

Note that A is an involution. That is, for any 16-byte input string x, x = A(A(x)) holds.

3. Security Considerations

ARIA is designed to be resistant to all known attacks on block ciphers [ARIA03]. Its security was analyzed by the COSIC group of K.U.Leuven in Belgium [ARIAEVAL] and no security flaw has been found.

4. Informative References

[ARIAEVAL] Biryukov, A., et al., "Security and Performance Analysis of ARIA", K.U.Leuven (2003), available at http://www.cosic.esat.kuleuven.be/publications/article-500.pdf

[ARIA03] Kwon, D., et al., "New Block Cipher: ARIA", ICISC 2003, pp. 432-445.

[ARIAKS] Korean Agency for Technology and Standards (KATS), "128 bit block encryption algorithm ARIA", KS X 1213:2004, December 2004 (In Korean).

[ARIAPKCS] RSA Laboratories, PKCS #11 v2.20 Amendment 3 Revision 1: Additional PKCS #11 Mechanisms, January 2007.

[X.680] ITU-T Recommendation X.680 (2002) | ISO/IEC 8824-1:2002, Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation.

[X.681] ITU-T Recommendation X.681 (2002) | ISO/IEC 8824-2:2002, Information technology - Abstract Syntax Notation One (ASN.1): Information object specification.

[X.682] ITU-T Recommendation X.682 (2002) | ISO/IEC 8824-3:2002, Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification.

[X.683] ITU-T Recommendation X.683 (2002) | ISO/IEC 8824-4:2002, Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications.

Appendix A. Example Data of ARIA

Here are test data for ARIA in hexadecimal form.

A.1. 128-Bit Key

```
- Key       : 000102030405060708090a0b0c0d0e0f
- Plaintext : 00112233445566778899aabbccddeeff
- Ciphertext: d718fbd6ab644c739da95f3be6451778

- Round key generators
   W0: 000102030405060708090a0b0c0d0e0f
   W1: 2afbea741e1746dd55c63ba1afcea0a5
   W2: 7c8578018bb127e02dfe4e78c288e33c
   W3: 6785b52b74da46bf181054082763ff6d

- Encryption round keys
   e1:  d415a75c794b85c5e0d2a0b3cb793bf6
   e2:  369c65e4b11777ab713a3e1e6601b8f4
   e3:  0368d4f13d14497b6529ad7ac809e7d0
   e4:  c644552b549a263fb8d0b50906229eec
   e5:  5f9c434951f2d2ef342787b1a781794c
   e6:  afea2c0ce71db6de42a47461f4323c54
   e7:  324286db44ba4db6c44ac306f2a84b2c
   e8:  7f9fa93574d842b9101a58063771eb7b
   e9:  aab9c57731fcd213ad5677458fcfe6d4
   e10: 2f4423bb06465abada5694a19eb88459
   e11: 9f8772808f5d580d810ef8ddac13abeb
   e12: 8684946a155be77ef810744847e35fad
   e13: 0f0aa16daee61bd7dfee5a599970fb35

- Intermediate round values
   P1:  7fc7f12befd0a0791de87fa96b469f52
   P2:  ac8de17e49f7c5117618993162b189e9
   P3:  c3e8d59ec2e62d5249ca2741653cb7dd
   P4:  5d4aebb165e141ff759f669e1e85cc45
   P5:  7806e469f68874c5004b5f4a046bbcfa
   P6:  110f93c9a630cdd51f97d2202413345a
   P7:  e054428ef088fef97928241cd3be499e
   P8:  5734f38ea1ca3ddd102e71f95e1d5f97
   P9:  4903325be3e500cccd52fba4354a39ae
   P10: cb8c508e2c4f87880639dc896d25ec9d
   P11: e7e0d2457ed73d23d481424095afdca0
```

A.2. 192-Bit Key

```
Key       : 000102030405060708090a0b0c0d0e0f
            1011121314151617
Plaintext : 00112233445566778899aabbccddeeff
Ciphertext: 26449c1805dbe7aa25a468ce263a9e79
```

A.3. 256-Bit Key

```
Key       : 000102030405060708090a0b0c0d0e0f
            101112131415161718191a1b1c1d1e1f
Plaintext : 00112233445566778899aabbccddeeff
Ciphertext: f92bd7c79fb72e2f2b8f80c1972d24fc
```

Appendix B. OIDs

Here is an ASN.1 module conforming to the 2002 version of ASN.1 [X.680][X.681][X.682][X.683].

```
AriaModesOfOperation {
iso(1) member-body(2) korea(400) nsri(200046) algorithm (1)
symmetric-encryption-algorithm(1) asn1-module(0) alg-oids(0) }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

OID ::= OBJECT IDENTIFIER

-- Synonyms --

id-algorithm OID ::=  { iso(1) member-body(2) korea(410) nsri(200046)
algorithm(1)}

id-sea OID ::= { id-algorithm symmetric-encryption-algorithm(1)}
id-pad OID ::= { id-algorithm pad(2)}

id-pad-null  RELATIVE-OID ::= {0} -- no padding algorithms identified
id-pad-1     RELATIVE-OID ::= {1}
-- padding method 2 of ISO/IEC 9797-1:1999

-- confidentiality modes:
-- ECB, CBC, CFB, OFB, CTR

id-aria128-ecb OID ::= { id-sea aria128-ecb(1)}
id-aria128-cbc OID ::= { id-sea aria128-cbc(2)}
id-aria128-cfb OID ::= { id-sea aria128-cfb(3)}
id-aria128-ofb OID ::= { id-sea aria128-ofb(4)}
id-aria128-ctr OID ::= { id-sea aria128-ctr(5)}
```

```
id-aria192-ecb OID ::= { id-sea aria192-ecb(6)}
id-aria192-cbc OID ::= { id-sea aria192-cbc(7)}
id-aria192-cfb OID ::= { id-sea aria192-cfb(8)}
id-aria192-ofb OID ::= { id-sea aria192-ofb(9)}
id-aria192-ctr OID ::= { id-sea aria192-ctr(10)}

id-aria256-ecb OID ::= { id-sea aria256-ecb(11)}
id-aria256-cbc OID ::= { id-sea aria256-cbc(12)}
id-aria256-cfb OID ::= { id-sea aria256-cfb(13)}
id-aria256-ofb OID ::= { id-sea aria256-ofb(14)}
id-aria256-ctr OID ::= { id-sea aria256-ctr(15)}

-- authentication modes: CMAC

id-aria128-cmac OID ::= { id-sea aria128-cmac(21)}
id-aria192-cmac OID ::= { id-sea aria192-cmac(22)}
id-aria256-cmac OID ::= { id-sea aria256-cmac(23)}

-- modes for both confidentiality and authentication
-- OCB 2.0, GCM, CCM, Key Wrap

id-aria128-ocb2 OID ::= { id-sea aria128-ocb2(31)}
id-aria192-ocb2 OID ::= { id-sea aria192-ocb2(32)}
id-aria256-ocb2 OID ::= { id-sea aria256-ocb2(33)}

id-aria128-gcm OID ::= { id-sea aria128-gcm(34)}
id-aria192-gcm OID ::= { id-sea aria192-gcm(35)}
id-aria256-gcm OID ::= { id-sea aria256-gcm(36)}

id-aria128-ccm OID ::= { id-sea aria128-ccm(37)}
id-aria192-ccm OID ::= { id-sea aria192-ccm(38)}
id-aria256-ccm OID ::= { id-sea aria256-ccm(39)}

id-aria128-kw OID ::= { id-sea aria128-kw(40)}
id-aria192-kw OID ::= { id-sea aria192-kw(41)}
id-aria256-kw OID ::= { id-sea aria256-kw(42)}

-- ARIA Key-Wrap with Padding Algorithm (AES version: RFC 5649)

id-aria128-kwp OID ::= { id-sea aria128-kwp(43)}
id-aria192-kwp OID ::= { id-sea aria192-kwp(44)}
id-aria256-kwp OID ::= { id-sea aria256-kwp(45)}
```

```
   AriaModeOfOperation ::= AlgorithmIdentifier
   { {AriaModeOfOperationAlgorithms} }

   AriaModeOfOperationAlgorithms ALGORITHM ::= {
   aria128ecb  |aria128cbc  |aria128cfb  |aria128ofb  |aria128ctr  |
   aria192ecb  |aria192cbc  |aria192cfb  |aria192ofb  |aria192ctr  |
   aria256ecb  |aria256cbc  |aria256cfb  |aria256ofb  |aria256ctr  |
   aria128cmac |aria192cmac |aria256cmac |
   aria128ocb2 |aria192ocb2 |aria256ocb2 |
   aria128gcm  |aria192gcm  |aria256gcm  |
   aria128ccm  |aria192ccm  |aria256ccm  |
   aria128kw   |aria192kw   |aria256kw   |
   aria128kwp  |aria192kwp  |aria256kwp ,
   ... --Extensible
   }

   aria128ecb  ALGORITHM ::=
   { OID id-aria128-ecb PARAMS AriaEcbParameters }
   aria128cbc  ALGORITHM ::=
   { OID id-aria128-cbc PARAMS AriaCbcParameters }
   aria128cfb  ALGORITHM ::=
   { OID id-aria128-cfb PARAMS AriaCfbParameters }
   aria128ofb  ALGORITHM ::=
   { OID id-aria128-ofb PARAMS AriaOfbParameters }
   aria128ctr  ALGORITHM ::=
   { OID id-aria128-ctr PARAMS AriaCtrParameters }

   aria192ecb  ALGORITHM ::=
   { OID id-aria192-ecb PARAMS AriaEcbParameters }
   aria192cbc  ALGORITHM ::=
   { OID id-aria192-cbc PARAMS AriaCbcParameters }
   aria192cfb  ALGORITHM ::=
   { OID id-aria192-cfb PARAMS AriaCfbParameters }

   aria192ofb  ALGORITHM ::=
   { OID id-aria192-ofb PARAMS AriaOfbParameters }
   aria192ctr  ALGORITHM ::=
   { OID id-aria192-ctr PARAMS AriaCtrParameters }
```

```
   aria256ecb  ALGORITHM ::=
   { OID id-aria256-ecb PARAMS AriaEcbParameters }
   aria256cbc  ALGORITHM ::=
   { OID id-aria256-cbc PARAMS AriaCbcParameters }
   aria256cfb  ALGORITHM ::=
   { OID id-aria256-cfb PARAMS AriaCfbParameters }
   aria256ofb  ALGORITHM ::=
   { OID id-aria256-ofb PARAMS AriaOfbParameters }
   aria256ctr  ALGORITHM ::=
   { OID id-aria256-ctr PARAMS AriaCtrParameters }

   aria128cmac ALGORITHM ::=
   { OID id-aria128-cmac PARAMS AriaCmacParameters }
   aria192cmac ALGORITHM ::=
   { OID id-aria192-cmac PARAMS AriaCmacParameters }
   aria256cmac ALGORITHM ::=
   { OID id-aria256-cmac PARAMS AriaCmacParameters }

   aria128ocb2 ALGORITHM ::=
   { OID id-aria128-ocb2 PARAMS AriaOcb2Parameters }
   aria192ocb2 ALGORITHM ::=
   { OID id-aria192-ocb2 PARAMS AriaOcb2Parameters }
   aria256ocb2 ALGORITHM ::=
   { OID id-aria256-ocb2 PARAMS AriaOcb2Parameters }

   aria128gcm  ALGORITHM ::=
   { OID id-aria128-gcm PARAMS AriaGcmParameters }
   aria192gcm  ALGORITHM ::=
   { OID id-aria192-gcm PARAMS AriaGcmParameters }
   aria256gcm  ALGORITHM ::=
   { OID id-aria256-gcm PARAMS AriaGcmParameters }

   aria128ccm  ALGORITHM ::=
   { OID id-aria128-ccm PARAMS AriaCcmParameters }
   aria192ccm  ALGORITHM ::=
   { OID id-aria192-ccm PARAMS AriaCcmParameters }
   aria256ccm  ALGORITHM ::=
   { OID id-aria256-ccm PARAMS AriaCcmParameters }

   aria128kw   ALGORITHM ::= { OID id-aria128-kw }
   aria192kw   ALGORITHM ::= { OID id-aria192-kw }
   aria256kw   ALGORITHM ::= { OID id-aria256-kw }

   aria128kwp   ALGORITHM ::= { OID id-aria128-kwp }
   aria192kwp   ALGORITHM ::= { OID id-aria192-kwp }
   aria256kwp   ALGORITHM ::= { OID id-aria256-kwp }
```

```
   AriaPadAlgo ::= CHOICE {
       specifiedPadAlgo   RELATIVE-OID,
       generalPadAlgo     OID
   }

   AriaEcbParameters ::= SEQUENCE {
       padAlgo   AriaPadAlgo  DEFAULT specifiedPadAlgo:id-pad-null
   }

   AriaCbcParameters ::= SEQUENCE {
       m         INTEGER       DEFAULT 1,
       -- number of stored ciphertext blocks
       padAlgo   AriaPadAlgo   DEFAULT specifiedPadAlgo:id-pad-1
   }

   AriaCfbParameters ::= SEQUENCE {
       r         INTEGER,
       -- bit-length of feedback buffer, 128<=r<=128*1024
       k         INTEGER,
       -- bit-length of feedback variable, 1<=k<=128
       j         INTEGER,
       -- bit-length of plaintext/ciphertext block, 1<=j<=k
       padAlgo   AriaPadAlgo  DEFAULT specifiedPadAlgo:id-pad-null
   }

   AriaOfbParameters ::= SEQUENCE {
       j         INTEGER,
       -- bit-length of plaintext/ciphertext block, 1<=j<=128
       padAlgo   AriaPadAlgo  DEFAULT specifiedPadAlgo:id-pad-null
   }

   AriaCtrParameters ::= SEQUENCE {
       j         INTEGER,
       -- bit-length of plaintext/ciphertext block, 1<=j<=128
       padAlgo   AriaPadAlgo  DEFAULT specifiedPadAlgo:id-pad-null
   }

   AriaCmacParameters ::= INTEGER -- bit-length of authentication tag

   AriaOcb2Parameters ::= INTEGER -- bit-length of authentication tag

   AriaGcmParameters  ::= SEQUENCE {
       s        INTEGER,   -- bit-length of starting variable
       t        INTEGER    -- bit-length of authentication tag
   }
```

```
   AriaCcmParameters  ::= SEQUENCE {
       w       INTEGER (2|3|4|5|6|7|8),
       -- length of message length field in octets
       t       INTEGER (32|48|64|80|96|112|128)
       -- bit-length of authentication tag
   }

   ALGORITHM ::= CLASS {
       &id     OBJECT IDENTIFIER UNIQUE,
       &Type   OPTIONAL
   }
   WITH SYNTAX { OID &id  [PARAMS &Type] }

   AlgorithmIdentifier { ALGORITHM:AlgoSet } ::= SEQUENCE {
       algorithm     ALGORITHM.&id( {AlgoSet} ),
       parameters ALGORITHM.&Type( {AlgoSet}{@algorithm} ) OPTIONAL
   }

   END
```

Authors' Addresses

Jungkeun Lee
National Security Research Institute
P.O.Box 1, Yuseong, Daejeon, 305-350, Korea

EMail: jklee@ensec.re.kr

Jooyoung Lee
National Security Research Institute
P.O.Box 1, Yuseong, Daejeon, 305-350, Korea

EMail: jlee05@ensec.re.kr

Jaeheon Kim

National Security Research Institute
P.O.Box 1, Yuseong, Daejeon, 305-350, Korea

EMail: jaeheon@ensec.re.kr

Daesung Kwon
National Security Research Institute
P.O.Box 1, Yuseong, Daejeon, 305-350, Korea

EMail: ds_kwon@ensec.re.kr

Choonsoo Kim
National Security Research Institute
P.O.Box 1, Yuseong, Daejeon, 305-350, Korea

EMail: jbr@ensec.re.kr