ARIA 검증시스템

Mode of Operations Validation System for ARIA

2019. 7.

암호모듈 시험기관

개정이력

버전	변경일	변경사유	변경내용
3.0	2019-07-01	최초작성	최초작성

목 차

암호알고리즘 검증기준 V3.0

1. 범위

❑ 본 장에서는 KS X 1213-1에 기술된 블록 암호 ARIA를 KS X 1213-2에 기술된 운영모드 중 암·복호화 전용 운영모드로 구현한 경우의 정확성 절차를 명시한 ARIA 운영모드 검증시스템(Mode of Operations Validation System for ARIA, 이하 ARIA MOVS)에 대해 기술한다.

- 암·복호화 전용 운영모드: ECB, CBC, CFB, OFB, CTR

❑ ARIA MOVS는 기지 답안 검사(Known Answer Test), 다중 블록 메시지 검사(Multi-block Message Test), 그리고 몬테 카를로 검사(Monte Carlo Test)의 세 가지 시험으로 구성된다.

2. 관련 표준

❑ 본 장에서는 다음 표준들이 인용되었다.

- KS X 1213-1:2014, 128비트 블록 암호 알고리즘 ARIA - 제1부: 일반
- KS X 1213-2:2014, 128비트 블록 암호 알고리즘 ARIA - 제2부: 운영모드
- TTAK.KO-12.0271-Part1/R1:2016, n비트 블록암호 운영모드 - 제1부: 일반
- TTAK.KO-12.0271-Part3:2017, n비트 블록암호 운영모드 - 제3부: 블록암호 ARIA
- IETF RFC 5794, A Description of ARIA Encryption Algorithm (2010)

3. 기호

기호	의 미
BitJ[A]	A의 J번째 비트
ByteJ[A]	A의 J번째 바이트
CT64[A]	A의 하위 64 비트
MSBJ[A]	A의 최상위(Most Significant Bit) J비트
LSBJ[A]	A의 최하위(Least Significant Bit) J비트

4. ARIA

❑ ARIA는 경량 환경 및 하드웨어 구현을 위해 최적화된 Involutional SPN 구조를 갖는 범용 블록 암호알고리즘이다.

❑ ARIA의 주요 특성은 다음과 같다.

- 블록 크기: 128 비트
- 키 길이: 128/192/256 비트
- 전체 구조: ISPN(Involutional Substitution-Permutation Network)
- 라운드 수: 12/14/16 (키 길이에 따라 결정됨)
- 특징: 경량 환경 및 하드웨어에서의 효율성 향상을 위해, ARIA가 사용하는 대부분의 연산은 XOR과 같은 단순한 바이트 단위 연산으로 구성되어 있다.

구분	입·출력 크기[bytes]	입력 키 길이[bytes]	라운드 수
ARIA-128	16	16	12
ARIA-192	16	24	14
ARIA-256	16	32	16

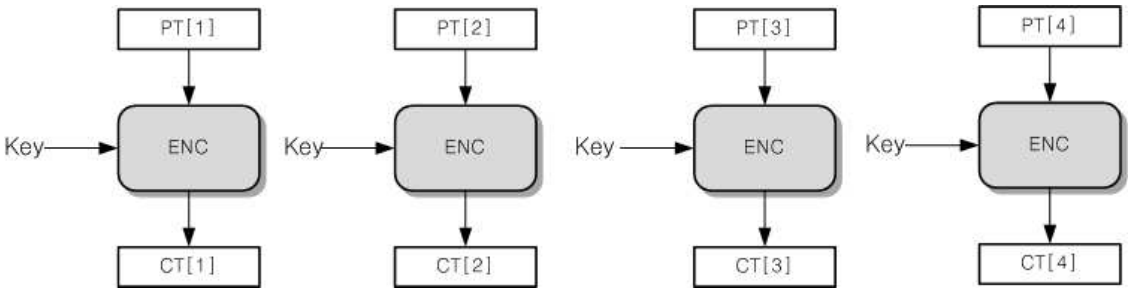
5. ARIA 운영모드

❑ ARIA는 128비트 블록 단위로 암·복호화를 수행하는 블록 암호알고리즘이다. 그러나 일반적으로 암호화하고자 하는 평문은 128비트 이상이며 매우 다양한 크기로 구성된다. 이처럼 임의의 크기를 갖는 데이터를 암·복호화하기 위해서 일정한 입·출력 크기를 갖는 블록 암호의 기본 알고리즘을 적용하는 방식을 블록 암호알고리즘의 운영모드라고 한다.

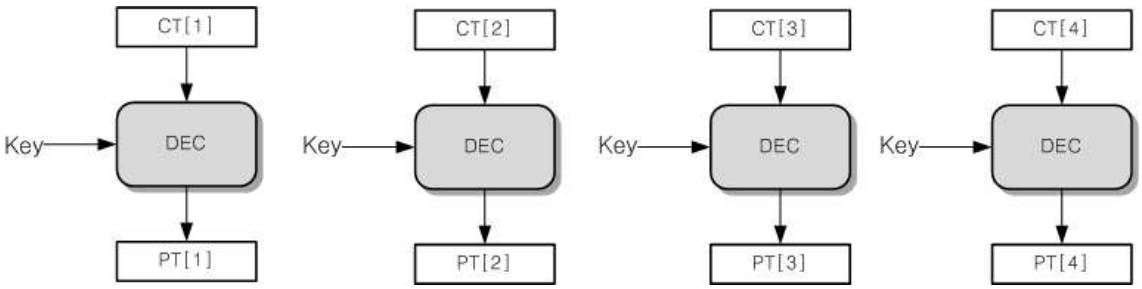
5.1. ECB

❑ ECB모드는 주어진 평문을 블록의 크기로 나누어서 차례로 암호화하는 방식으로 평문 블록과 암호문 블록이 일대일 대응관계를 가지는 가장 단순한 운영모드이다.

- 암호화: CT[i] = ENC(Key, PT[i]), (i=1, …, t)
- 복호화: PT[i] = DEC(Key, CT[i]), (i=1, …, t)



[ECB모드 암호화 과정]

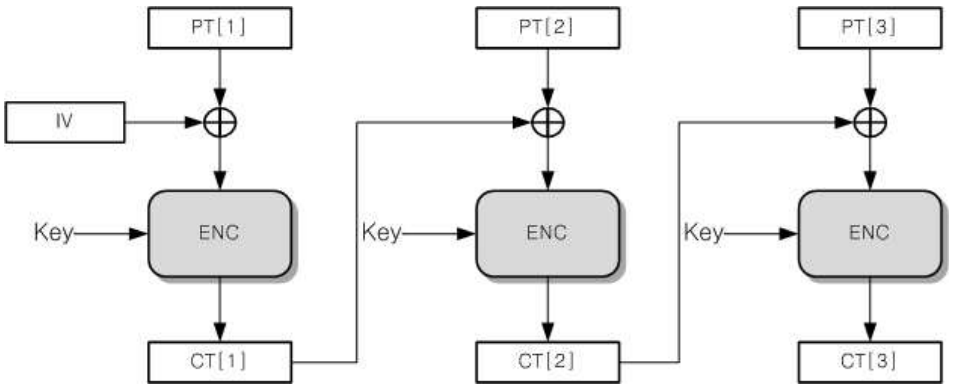


[ECB모드 복호화 과정]

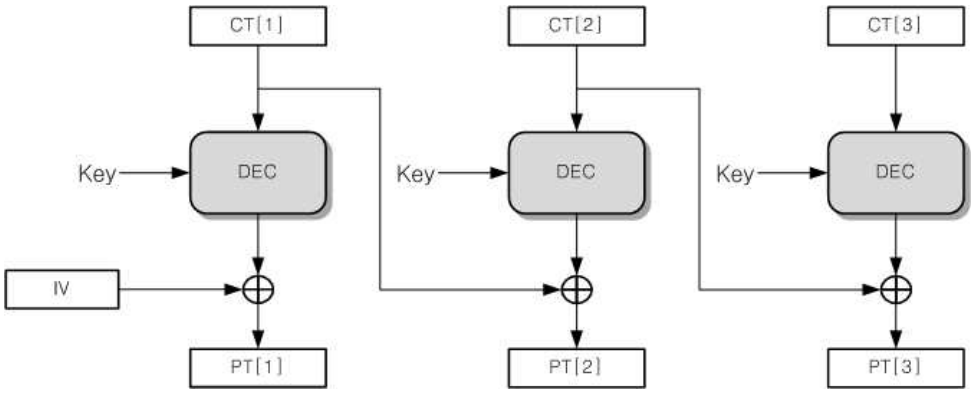
5.2. CBC

❑ CBC모드는 이전 암호문 블록과 현재 평문 블록을 XOR한 블록을 암호알고리즘의 입력으로 사용하여 암호문 블록을 생성하는 방법이다. 이 때, 첫 번째 평문 블록 PT[1]은 초기벡터와 XOR하여 사용한다.

- 암호화: CT[i] = ENC(Key, CT[i-1]⊕PT[i]), (i=1, …, t, CT[0]=IV)
- 복호화: PT[i] = CT[i-1]⊕DEC(Key, CT[i]), (i=1, …, t, CT[0]=IV)



[CBC모드 암호화 과정]



[CBC모드 복호화 과정]

5.3. CFB

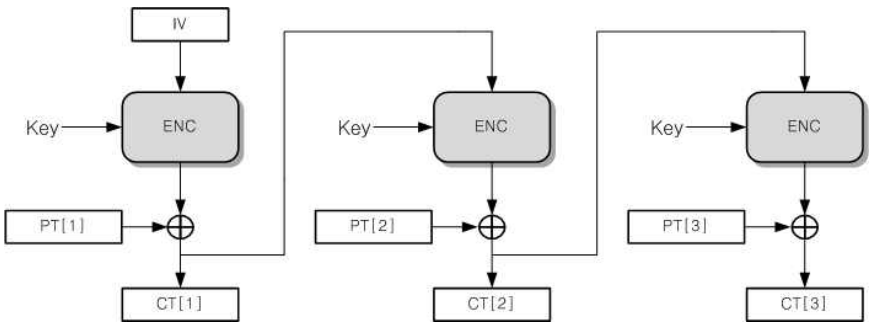
❑ CFB모드는 이전 블록의 암호문을 암호알고리즘에 입력하여 얻은 결과값과 현재 평문 블록을 XOR하여 암호문 블록을 생성한다. 이 때, 첫 번째 평문 블록의 암호화 과정에서는 초기벡터를 암호알고리즘의 입력으로 이용한다.

- 암호화: 암호알고리즘의 입력 블록을 CF라고 했을 때, J비트 단위의 평문을 암호화하기 위하여 암호문의 J비트가 피드백되는 CFB 운영 방식은 다음의 단계를 따른다. (CF[i]는 i번째 입력 상태)

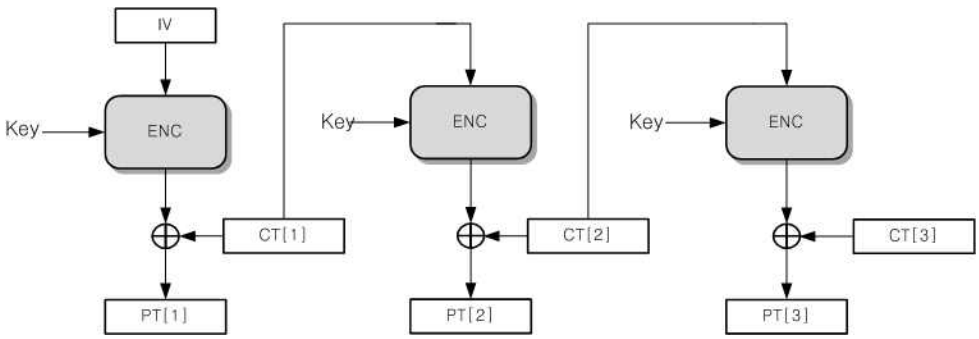
  - (단계 1) CF[1] = IV로 두고 (단계 2)에서 (단계 4)를 $1 \le i \le t$인 CF에 대해서 반복한다. (단, i=t인 경우에 (단계 4)는 생략)
  - (단계 2) MSBJ[i] = ENC(KEY, CF[i])의 최상위 J비트
  - (단계 3) CT[i] = PT[i]⊕MSBJ[i]
  - (단계 4) CF[i+1] = (CF[i] << J)⊕(0···0 ‖ CT[i])

- 복호화: 암호화에 사용된 것과 동일한 IV를 사용하여 (단계 3)을 제외하고는 동일하게 이루어진다. 복호화 과정의 (단계 3)은 아래와 같다.

  - (단계 3) PT[i] = CT[i]⊕MSBJ[i]



[CFB모드 암호화 과정]

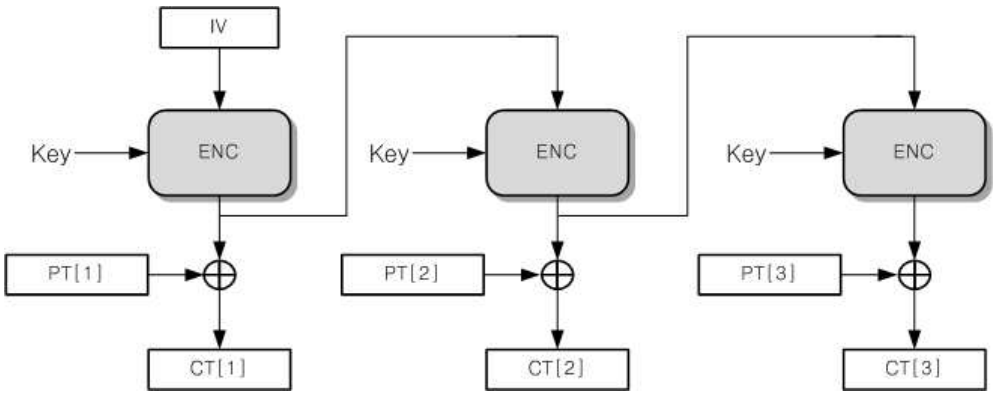


[CFB모드 복호화 과정]

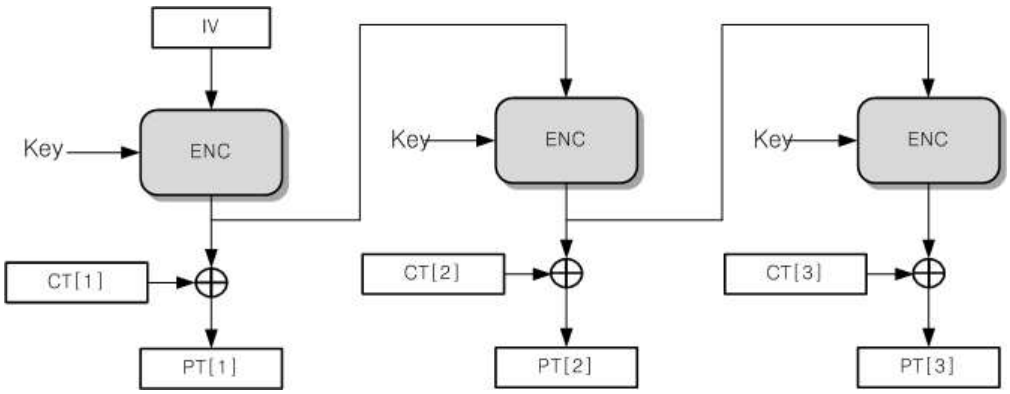
5.4. OFB

❑ OFB모드는 이전 블록의 암호알고리즘 출력을 암호알고리즘에 입력하여 얻은 결과값과 현재 평문 블록을 XOR하여 암호문 블록을 생성한다. 이 때, 첫 번째 평문 블록의 암호화 과정에서는 초기벡터를 암호알고리즘의 입력으로 이용한다.

- 암호화: CT[i] = PT[i]⊕OT[i], OT[i]=ENC(Key, OT[i-1]), (i=1, …, t, OT[0]=IV)
- 복호화: PT[i] = CT[i]⊕OT[i], OT[i]=ENC(Key, OT[i-1]), (i=1, …, t, OT[0]=IV)



[OFB모드 암호화 과정]

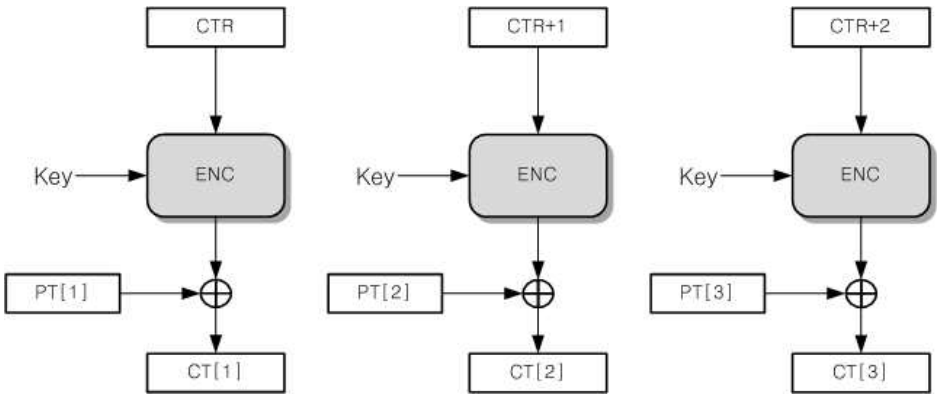


[OFB모드 복호화 과정]

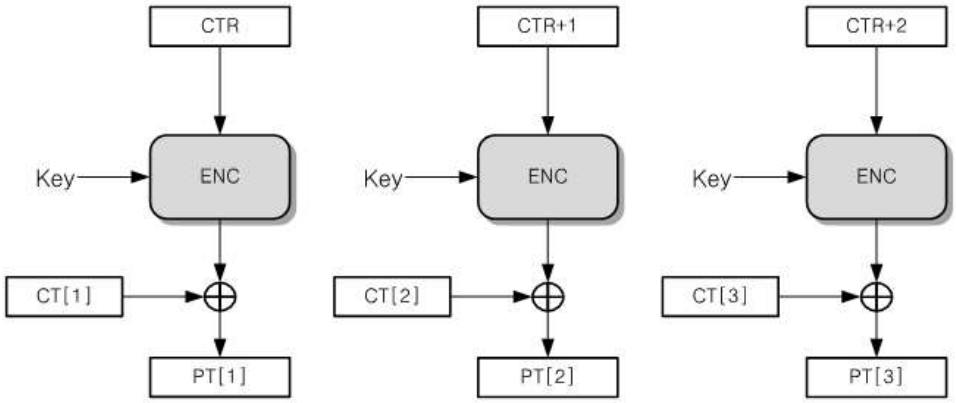
5.5. CTR

❑ CTR모드는 1씩 증가하는 카운터를 입력으로 하는 암호알고리즘의 출력과 평문 블록을 XOR하여 암호문 블록을 생성한다. 여기서 카운터 초기값은 암호화 하는 경우마다 다른 값을 이용한다.

- 암호화: CT[i]=PT[i]⊕ENC(Key, CTR[0]+i-1) (i=1, …, t, CTR[0]=CTR)
- 복호화: PT[i]=CT[i]⊕ENC(Key, CTR[0]+i-1) (i=1, …, t, CTR[0]=CTR)



[CTR모드 암호화 과정]



[CTR모드 복호화 과정]

6. ARIA MOVS 검사

❑ ARIA MOVS는 다음의 운영모드를 검사한다.

- ECB
- CBC
- OFB
- CFB1 (데이터 블록이 1비트인 CFB)
- CFB8 (데이터 블록이 8비트인 CFB)
- CFB128 (데이터 블록이 128비트인 CFB)
- CTR

❑ 각 운영모드 검증은 128, 192, 256 비트의 키 길이에 따른 검증으로 다시 나뉜다.

❑ 운영모드, 키 길이 등이 결정되면 ARIA MOVS는 검사에 필요한 입력값을 저장하고 있는 REQUEST파일과 해당 운영모드 등에 대한 결과값인 FACTS파일을 생성한다. IUT는 ARIA MOVS에게서 받은 REQUEST파일에 저장되어 있는 입력값을 이용하여 생성한 출력값을 저장하고 있는 RESPONSE파일을 생성한다. ARIA MOVS는 IUT에게서 받은 RESPONSE파일과 FACTS파일을 비교하여 값의 일치여부를 확인한다.

6.1. 구성 정보

❑ ARIA MOVS의 검증 과정을 시작할 때, 신청인은 시험기관에 IUT를 설명하는 기본 정보 외 ARIA 검사에 필요한 다음 정보를 제공해야 한다.

○ 지원되는 운영모드
○ 키 길이
○ CTR 모드가 지원되는 경우 CTR 모드에서 카운터 초기값 생성방식

6.2. 기지 답안 검사

❑ 기지 답안 검사(KAT)는 ARIA 알고리즘의 세부 구성요소가 정확하게 구현되었는가를 확인하는 검사로써, 고정된 입력값에 대한 출력값이 올바른지 검사한다. ARIA에 대한 기지 답안 검사는 다음의 두 가지 형태로 구분하여 검사한다.

- Variable Key KAT
- Variable Text KAT

ARIA MOVS:

A. 다음 내용을 포함하는 REQUEST 파일을 생성한다.
  ○ 파일명: ARIA[키길이][운영모드명]KAT.req
  1. 해당 운영모드에 입력될 값
    - 키
    - 초기벡터(ECB모드는 제외, CTR모드는 카운터 초기값에 해당)
    - 암호화/복호화를 위한 평문/암호문

B. 다음의 정보가 포함된 FACTS 파일을 생성한다.
  ○ 파일명: ARIA[키길이][운영모드명]KAT.fax
  1. REQUEST 파일에 포함된 정보
  2. 해당 운영모드에 의해 생성된 결과값

IUT:

A. REQUEST 파일에 포함된 정보를 이용하여 출력값을 생성한다.

B. 다음 사항을 포함하는 RESPONSE 파일을 생성한다.
  ○ 파일명: ARIA[키길이][운영모드명]KAT.rsp
  1. REQUEST 파일에 포함된 정보
  2. A 과정에서 생성한 출력값

ARIA MOVS:

A. RESPONSE 파일과 FACTS 파일의 내용을 비교한다.

B. 만약 IUT가 생성한 모든 값이 정확하다면 결과로 PASS를 출력하고, 그렇지 않으면 FAIL을 출력한다.

6.3. 다중 블록 메시지 검사

❑ 다중 블록 메시지 검사(MMT)는 ECB, CBC, OFB, CFB, CTR 운영모드를 이용하여 긴 메시지의 암호화/복호화를 올바르게 처리하는지 체크하는 구현 정확성 확인 검사이다. 이 검사는 한 블록의 정보가 다음 블록에 연속적으로 영향을 미치는지를 확인한다. ECB, CBC, OFB, CFB128, CTR의 블록크기는 128비트, CFB1의 블록크기는 1비트, CFB8의 블록크기는 8비트이다. 각각의 모드에 대해 입력 메시지 블록크기가 i*블록크기(1≤i≤10)인 10개의 메시지를 시험한다.

ARIA MOVS:

A. 다음 내용을 포함하는 REQUEST 파일을 생성한다.
  ○ 파일명: ARIA[키길이][운영모드명]MMT.req
  1. 해당 운영모드에 입력될 값
    - 키
    - 초기벡터(ECB모드는 제외, CTR모드는 카운터 초기값에 해당)
    - 암호화/복호화를 위한 평문/암호문

B. 다음의 정보가 포함된 FACTS 파일을 생성한다.
  ○ 파일명: ARIA[키길이][운영모드명]MMT.fax
  1. REQUEST 파일에 포함된 정보
  2. 해당 운영모드에 의해 생성된 결과값

IUT:

A. REQUEST 파일에 포함된 정보를 이용하여 출력값을 생성한다.

B. 다음 사항을 포함하는 RESPONSE 파일을 생성한다.
  ○ 파일명: ARIA[키길이][운영모드명]MMT.rsp
  1. REQUEST 파일에 포함된 정보
  2. A 과정에서 생성한 출력값

ARIA MOVS:

A. RESPONSE 파일과 FACTS 파일의 내용을 비교한다.

B. 만약 IUT가 생성한 모든 값이 정확하다면 결과로 PASS를 출력하고, 그렇지 않으면 FAIL을 출력한다.

6.4. 몬테 카를로 검사

❑ 몬테 카를로 검사(MCT)는 구현 과정의 결함 여부를 확인하기 위해서 각각의 운영모드에 대해 100개의 임의의 평문을 생성하여 처리한다. 구체적인 검사 방법은 다음에서 각 운영모드별로 명세한다. 초기 벡터를 사용하는 모드의 경우 각각의 평문과 함께 초기벡터가 사용된다.

ARIA MOVS:

A. 다음 내용을 포함하는 REQUEST 파일을 생성한다.
   ○ 파일명: ARIA[키길이][운영모드명]MCT.req
   1. 해당 운영모드에 입력될 값
      - 키
      - 초기벡터(ECB모드는 제외, CTR모드는 카운터 초기값에 해당)
      - 암호화/복호화를 위한 평문/암호문

B. 다음의 정보가 포함된 FACTS 파일을 생성한다.
   ○ 파일명: ARIA[키길이][운영모드명]MCT.fax
   1. REQUEST 파일에 포함된 정보
   2. 해당 운영모드에 의해 생성된 결과값

IUT:

A. REQUEST 파일에 포함된 정보를 이용하여 출력값을 생성한다.

B. 다음 사항을 포함하는 RESPONSE 파일을 생성한다.
   ○ 파일명: ARIA[키길이][운영모드명]MCT.rsp
   1. REQUEST 파일에 포함된 정보
   2. A 과정에서 생성한 출력값

ARIA MOVS:

A. RESPONSE 파일과 FACTS 파일의 내용을 비교한다.

B. 만약 IUT가 생성한 모든 값이 정확하다면 결과로 PASS를 출력하고, 그렇지 않으면 FAIL을 출력한다.

■ 몬테 카를로 검사 - ECB 테스트

❑ MCT-ECB 검사의 REQUEST파일은 하나의 키와 평문이 주어지고, RESPONSE 파일은 이에 대한 MCT-ECB 검사를 수행하여 생성되는 100개의 키, 평문, 암호문으로 작성된다.

[ARIA MCT-ECB 검사]

```
Key[0] = Key;
PT[0] = PT;
for(i=0 to 99) {
  Output Key[i];
  Output PT[0];
  for(j=0 to 999) {
      CT[j] = ARIA(Key[i], PT[j]);
      PT[j+1] = CT[j];
    }
  Output CT[j];
  If ( keylen = 128 )
       Key[i+1] = Key[i] xor CT[j];
  If ( keylen = 192 )
       Key[i+1] = Key[i] xor (CT64[j-1]||CT[j]);
  If ( keylen = 256 )
       Key[i+1] = Key[i] xor (CT[j-1]||CT[j]);
  PT[0] = CT[j];
}
```

■ 몬테 카를로 검사 - CBC 테스트

❑ MCT-CBC 검사의 REQUEST파일은 하나의 키, 초기벡터, 평문이 주어지고 RESPONSE파일은 이에 대한 MCT-CBC 검사를 수행하여 생성되는 100개의 키, 초기벡터, 평문, 암호문으로 작성된다.

[ARIA MCT-CBC 검사]

```
Key[0] = Key;
IV[0] = IV;
PT[0] = PT;
for(i=0 to 99){
    Output Key[i];
    Output IV[i];
    Output PT[0];
    for(j=0 to 999){
        if ( j=0 ){
            CT[j] = ARIA(Key[i], PT[j] xor IV[i]);
            PT[j+1] = IV[i];
        }
        else{
            CT[j] = ARIA(Key[i], PT[j] xor CT[j-1]);
            PT[j+1] = CT[j-1];
         }
    }
    Output CT[j];
    If ( keylen = 128 )
         Key[i+1] = Key[i] xor CT[j];
    If ( keylen = 192 )
         Key[i+1] = Key[i] xor (CT64[j-1]||CT[j]);
    If ( keylen = 256 )
         Key[i+1] = Key[i] xor (CT[j-1]||CT[j]);
    IV[i+1] = CT[j];
    PT[0] = CT[j-1];
}
```

■ 몬테 카를로 검사 - CFB1 테스트

❑ MCT-CFB1 검사의 REQUEST파일은 하나의 키, 초기벡터, 평문이 주어지고 RESPONSE파일은 이에 대한 MCT-CFB1 검사를 수행하여 생성되는 100개의 키, 초기벡터, 평문, 암호문으로 작성된다.

[ARIA MCT-CFB1 검사]

```
Key[0] = Key;
IV[0] = IV;
PT[0] = PT;
for(i=0 to 99){
    Output Key[i];
    Output IV[i];
    Output PT[0];
    for(j=0 to 999){
        if ( j=0 ){
            CT[j] = PT[j] xor MSB1(ARIA(Key[i], IV[i]));
            PT[j+1] = BitJ(IV[i]);
            CF[j+1] = LSB127(IV[i]) || CT[j];
        }
        else{
            CT[j] = PT[j] xor MSB1(ARIA(Key[i], CF[j]));
            if (j<128)
                PT[j+1] = BitJ(IV[i]);
            else
                PT[j+1] = CT[j-128];
            CF[j+1] = LSB127(CF[j]) || CT[j];
         }
   }
      Output CT[j];
    If ( keylen = 128 )
        Key[i+1] = Key[i] xor (CT[j-127]||CT[j-126]||...||CT[j]);
    If ( keylen = 192 )
        Key[i+1] = Key[i] xor (CT[j-191]||CT[j-190]||...||CT[j]);
    If ( keylen = 256 )
        Key[i+1] = Key[i] xor (CT[j-255]||CT[j-254]||...||CT[j]);
    IV[i+1] = (CT[j-127] || CT[j-126] || ... || CT[j]);
    PT[0] = CT[j-128];
}
```

■ 몬테 카를로 검사 - CFB8 테스트

❑ MCT-CFB8 검사의 REQUEST파일은 하나의 키, 초기벡터, 평문이 주어지고 REQUEST파일은 이에 대한 MCT-CFB8 검사를 수행하여 생성되는 100개의 키, 초기벡터, 평문, 암호문으로 작성된다.

[ARIA MCT-CFB8 검사]

```
Key[0] = Key;
IV[0] = IV;
PT[0] = PT;
for(i=0 to 99){
    Output Key[i];
    Output IV[i];
    Output PT[0];
    for(j=0 to 999){
        if ( j=0 ){
            CT[j] = PT[j] xor MSB8(ARIA(Key[i], IV[i]));
            PT[j+1] = ByteJ(IV[i]);
            CF[j+1] = LSB120(IV[i]) || CT[j];
        }
        else{
            CT[j] = PT[j] xor MSB8(ARIA(Key[i], CF[j]));
            if (j<16)
                PT[j+1] = ByteJ(IV[i]);
            else
                PT[j+1] = CT[j-16];
            CF[j+1] = LSB120(CF[j]) || CT[j];
         }
    }
    Output CT[j];
    If ( keylen = 128 )
        Key[i+1] = Key[i] xor (CT[j-15]||CT[j-14]||...||CT[j]);
    If ( keylen = 192 )
        Key[i+1] = Key[i] xor (CT[j-23]||CT[j-22]||...||CT[j]);
    If ( keylen = 256 )
        Key[i+1] = Key[i] xor (CT[j-31]||CT[j-30]||...||CT[j]);
   IV[i+1] = (CT[j-15] || CT[j-14] || ... || CT[j]);
   PT[0] = CT[j-16];
}
```

■ 몬테 카를로 검사 - CFB128 테스트

❑ MCT-CFB128 검사의 REQUEST파일은 하나의 키, 초기벡터, 평문이 주어지고 RESPONSE파일은 이에 대한 MCT-CFB128 검사를 수행하여 생성되는 100개의 키, 초기벡터, 평문, 암호문으로 작성된다.

[ARIA MCT-CFB128 검사]

```
Key[0] = Key;
IV[0] = IV;
PT[0] = PT;
for(i=0 to 99){
    Output Key[i];
    Output IV[i];
    Output PT[0];
    for(j=0 to 999){
        if ( j=0 ){
            CT[j] = PT[j] xor ARIA(Key[i], IV[i]);
            PT[j+1] = IV[i];
        }
        else{
            CT[j] = PT[j] xor ARIA(Key[i], CT[j-1]);
            PT[j+1] = CT[j-1];
         }
    }
    Output CT[j];
    If ( keylen = 128 )
         Key[i+1] = Key[i] xor CT[j];
    If ( keylen = 192 )
         Key[i+1] = Key[i] xor (CT64[j-1]||CT[j]);
    If ( keylen = 256 )
         Key[i+1] = Key[i] xor (CT[j-1]||CT[j]);
    IV[i+1] = CT[j] ;
    PT[0] = CT[j-1];
}
```

■ 몬테 카를로 검사 - OFB 테스트

❑ MCT-OFB 검사의 REQUEST파일은 하나의 키, 초기벡터, 평문이 주어지고 RESPONSE파일은 이에 대한 MCT-OFB 검사를 수행하여 생성되는 100개의 키, 초기벡터, 평문, 암호문으로 작성된다.

[ARIA MCT-OFB 검사]

```
Key[0] = Key;
IV[0] = IV;
PT[0] = PT;
for(i=0 to 99){
    Output Key[i];
    Output IV[i];
    Output PT[0];
    for(j=0 to 999){
        if ( j=0 ){
            OT[j] = ARIA(Key[i], IV[i]);
            CT[j] = PT[j] xor OT[j];
            PT[j+1] = IV[i];
        }
        else{
            OT[j] = ARIA(Key[i], OT[j-1]);
            CT[j] = PT[j] xor OT[j];
            PT[j+1] = CT[j-1]
         }
    }
    Output CT[j];
    If ( keylen = 128 )
        Key[i+1] = Key[i] xor CT[j];
    If ( keylen = 192 )
        Key[i+1] = Key[i] xor (CT64[j-1]||CT[j]);
    If ( keylen = 256 )
        Key[i+1] = Key[i] xor (CT[j-1]||CT[j]);
    IV[i+1] = CT[j];
    PT[0] = CT[j-1];
}
```

■ 몬테 카를로 검사 - CTR 테스트

❑ MCT-CTR 검사의 REQUEST파일은 하나의 키, 카운터 초기값, 평문이 주어지고 RESPONSE파일은 이에 대한 MCT-CTR 검사를 수행하여 생성되는 100개의 키, 카운터 초기값, 평문, 암호문으로 작성된다.

[ARIA MCT-CTR 검사]

```
Key[0] = Key;
CTR[0] = CTR;
PT[0] = PT;
for(i=0 to 99){
    Output Key[i];
    Output CTR[0];
    Output PT[0];
    for(j=0 to 999){
            CT[j] = PT[j] xor ARIA(Key[i], CTR[0]);
            CTR[0] = (CTR[0] + 1) mod 2^128
            PT[j+1] = CT[j];
    }
    Output CT[j];
    If ( keylen = 128 )
         Key[i+1] = Key[i] xor CT[j];
    If ( keylen = 192 )
         Key[i+1] = Key[i] xor (CT64[j-1]||CT[j]);
    If ( keylen = 256 )
         Key[i+1] = Key[i] xor (CT[j-1]||CT[j]);
    PT[0] = CT[j];
}
```