4조 보안 SW 보고서

20151764 천정휘 20161943 최장혁

20161926 이승연 20182209 김수빈

목차

1. 암호모드 구현

1.1. 블록암호 LEA 구현

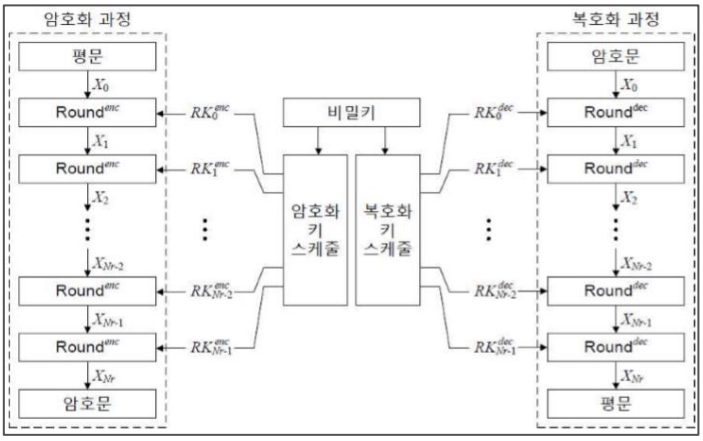
1.1.1. 구성

■ LEA 규격

	LEA-128
블록 길이 (bit)	128
비밀키 길이 (bit)	128
라운드 수	24

<표 1> LEA 규격

■ 전체적인 동작 과정



<그림 1> 암호화 및 복호화 과정

■ Little endian

- A가 바이트 배열이고 B가 워드 배열인 경우

B = (B[0], B[1], … , B[n - 1]), A = (A [0], A[1], … , A[4n - 1])의 비트열들이

B[i] = A[4i + 3] || A[4i + 2] || A[4i + 1] || A[4i] (0 ≤ i ≤ (n - 1))로 대입된다.

바이트 배열, 워드 배열 및 워드 내 비트 색인의 관계를 정리하면 <표 2>와 같다.

바이트 배열	3	2	1	0	7	6	5	4	…	4n-1	4n-2	4n-3	4n-4
워드 배열	0				1				…	n-1			
워드 내 비트 색인	31 … 0				31 … 0				…	31 … 0			

<표 2> 바이트 배열, 워드 배열, 워드 내 비트 색인의 관계

1.1.2. 소스코드

■ 라이브러리

```
library("dplyr")
```

■ 함수

1) delta : 키 스케줄에 사용되는 상수.

delta[[1]]=c3efe9db, delta[[2]] = 44626b02, delta[[3]] = 79e27c8a, delta[[4]] = 78df30ec

- 소스코드는 다음과 같다.

```
delta <- list(as.raw(c(0xc3, 0xef, 0xe9, 0xdb)), as.raw(c(0x44, 0x62, 0x6b, 0x02)),
              as.raw(c(0x79, 0xe2, 0x7c, 0x8a)), as.raw(c(0x78 ,0xdf, 0x30, 0xec)))
```

R에서는 32비트를 지원하지 않기에 바이트열 4개를 합쳐서 워드 배열을 만든다.

2) ROR(n, i) : 32bit인 2진수열 n의 i비트 우측 순환이동 함수.

- 소스코드는 다음과 같다.

```
ROR <- function(n, i){
  bitwXor(lag(n, i, default = 0), lead(n, 32-i, default = 0))
}
```

- 내장함수

lag(n, i, default = 0) : i비트만큼 좌측으로 이동한다. 빈 곳은 0으로 채운다.

lead(n, 32-i, default = 0) : 32-i비트만큼 우측으로 이동한다.

bitwXor : 두 비트열을 xor해준다. 그러면 i비트만큼 우측으로 순환 이동한다.

3) ROL(n, i) : 32비트 2진수열 n의 i비트 좌측 순환이동.

- 소스코드는 다음과 같다.

```
ROL <- function(n, i){
  bitwXor(lag(n, 32-i, default = 0), lead(n, i, default = 0))
}
```

- 내장함수

lag(n, 32-i, default = 0) : 32-i비트만큼 좌측으로 이동한다.

lead(n, i, default = 0) : i비트만큼 우측으로 이동한다.

bitwXor : 두 비트열을 xor해준다. 그러면 i비트만큼 좌측으로 순환 이동한다

4) split_n(a, n) : 벡터 a를 원소 n개를 가지는 리스트 배열로 나눈다.

- 소스코드는 다음과 같다.

```
split_n <- function(a, n){
  split(a, ceiling(seq_along(a)/n))
}
```

- 내장함수

split : 규칙에 따라 리스트 배열로 나눈다.

seq_along(a) : a를 1부터 a의 길이만큼 순서대로 재정의한다.

ceiling : 숫자를 올림해준다.

- 입출력 예시

입력 : a = c(4, 8, 3, 9, 1, 2), n = 2라고 가정해보자.

seq_along(a) : 1 2 3 4 5 6 이 출력된다.

ceiling(seq_along(a)/n) : 1 1 2 2 3 3 이 출력된다.

split(a, ceiling(seq_along(a)/n)) : a[[1]] = 4 8, a[[2]] = 3 9, a[[3]] = 1 2 가 출력된다.

5) raw_to_bin : 16진수인 n을 2진수은 n으로 변환.

- 소스코드는 다음과 같다.

```
raw_to_bin <- function(n){
  a<-as.integer(rawToBits(n))
  b<-split_n(a, 8)
  c<-list()
  for(i in 1:(length(a)/8)){
    c<-append(c, rev(split_n(b[[i]], 4)))
  }
  d<-vector()
  for(i in 1:(length(a)/4)){
    d<-append(d, rev(c[[i]]))
  }
  d
}
```

- 내장함수

  rawToBits : raw를 bit형태로 변환시킨다. 하지만 순서가 바뀌어서 출력된다.

  append : 리스트나 벡터에 원소를 추가한다.

  rev : 원소 순서를 거꾸로 뒤집는다.

- 입출력 예시

  입력 : n = 0xc1라고 가정해보자. 원래 0xc1은 1 1 0 0 0 0 0 1이다.

  a <- as.integer(rawToBits(n)) : a = 1 0 0 0 0 0 1 1 이 출력된다.

  b <- split_n(a, 8) : b[[1]] = 1 0 0 0 0 0 1 1 이 출력된다.

  c <- append(c, rev(split_n(b[[i]], 4))) : c[[1]] = 0 0 1 1, c[[2]] = 1 0 0 0 이 출력된다.

  d <- append(d, rec(c[[i]])) : d = 1 1 0 0 0 0 0 1 이 출력된다.

6) bin_to_dec : 2진수인 n을 10진수인 n으로 변환.

- 소스코드는 다음과 같다.

```
bin_to_dec <- function(n){
  dec = 0
  for(i in 1:length(n)){
    dec = dec + n[length(n)-i+1]*(2^(i-1))
  }
  dec
}
```

R에서는 32bit를 지원하지 않기에 하나하나 계산해준다.

각 자리 수에 각 자리수의 가중치를 곱하는 방식이다. 여기에서 각 자리수의 가중치란2^n(자리수-1)을 말한다.

7) bin_to_raw : 2진수인 n을 16진수인 n으로 변환.

- 소스코드는 다음과 같다.

```
bin_to_raw<-function(a){
  b<-split_n(a, 8)
  c<-vector()
  for(i in 1:(length(a)/8)){
    c<-append(c, bin_to_dec(b[[i]]))
  }
  as.raw(c)
}
```

R에서는 32bit를 지원하지 않기에 8bit씩 나눠서 변환하고 다시 붙인다.

먼저 bin_to_dec 함수를 이용해 2진수를 10진수로 바꾼다.

다음에 내장함수 as.raw를 이용해 10진수를 16진수로 바꾼다.

8) Plus(a, b) : 32 비트 2 진수열 a 와 b 에 대해 (a + b mod 2^32)로 정의되는 연산.

- 소스코드는 다음과 같다.

```
Plus <- function(a,b){
  P<-a+b
  for(i in 0:31){
    if(P[32-i] == 2){
      P[32-i] <- 0
      P[32-i-1] <- P[32-i-1] + 1
    }
    if(P[32-i] == 3){
      P[32-i] <- 1
      P[32-i-1] <- P[32-i-1] + 1
    }
  }
  P
}
```

R 에서는 32 비트를 지원하지 않기에 비트 하나하나 계산해준다.

- 입출력 예시

입력 : a = 0 0 1 1, b = 1 0 1 1 이라고 가정하자.

P <- a+b : 1 0 2 2 가 출력된다.

2 를 0 으로 바꾸고 올림 : 1 0 3 0 이 출력된다.

3 을 1 로 바꾸고 올림 : 1 1 1 0 이 출력된다.

9) Minus(a, b) : 32 비트 2 진수열 a 와 b 에 대해 (a – b mod 2^32)로 정의되는 연산.

- 소스코드는 다음과 같다.

```
Minus<-function(a,b){
  M<-a-b
  for(i in 0:31){
    if(M[32-i] == -1){
      M[32-i] <- 1
      M[32-i-1] <- (M[32-i-1] - 1)
    }
    if(M[32-i] == -2){
      M[32-i] <- 0
      M[32-i-1] <- M[32-i-1] - 1
    }
  }
  M
}
```

R 에서는 32 비트를 지원하지 않기에 비트 하나하나 계산해준다.

- 입출력 예시

입력 : a = 1 0 1 0, b = 0 1 0 1 이라고 가정하자.

M <- a – b : 1 1 -1 -1 이 출력된다.

-1 을 1 로 바꾸고 내림 : 1 1 -2 1 이 출력된다.

-2 를 0 으로 바꾸고 내림 : 1 0 0 1 이 출력된다.

■ KeySchedule

- 키 스케줄 알고리즘

알고리즘 3 LEA-128 암호화 키 스케줄 함수: $(RK_0^{enc}, \cdots, RK_{23}^{enc}) \leftarrow KeySchedule_{128}^{enc}(K)$
입력: 128비트 비밀키 K
출력: 24개의 192비트 암호화 라운드키 RK_i^{enc} $(0 \le i \le 23)$
1: $T \leftarrow K$
2: for i = 0 to 23 do
3: $T[0] \leftarrow ROL_1(T[0] \boxplus ROL_i(\delta[i \bmod 4]))$
4: $T[1] \leftarrow ROL_3(T[1] \boxplus ROL_{i+1}(\delta[i \bmod 4]))$
5: $T[2] \leftarrow ROL_6(T[2] \boxplus ROL_{i+2}(\delta[i \bmod 4]))$
6: $T[3] \leftarrow ROL_{11}(T[3] \boxplus ROL_{i+3}(\delta[i \bmod 4]))$
7: $RK_i^{enc} \leftarrow (T[0], T[1], T[2], T[1], T[3], T[1])$
8: end for

비밀키 lea_key로부터 암호화, 복호화 과정에 필요한 24개의 192비트 라운드 키 RK를 생성하는 키 스케줄 과정을 설명한다.

RK : 암호화, 복호화 과정의 라운드 함수에 사용되는 192 비트 라운드 키.

32 비트 배열 RK = (RK[[1]], RK[[2]], RK[[3]], RK[[4]], RK[[5]], RK[[6]])로 표기된다.

T : 키 스케줄의 라운드키 생성에 사용되는 내부상태 변수.

비밀키와 동일한 길이를 가지며, 워드 배열 T = (T[[1]], T[[2]], T[[3]], T[[4]])로 표기된다.

- 소스코드는 다음과 같다.

```
lea_key <- as.raw(0:15)
lea_key

T<-split_n(lea_key, 4)
T<-lapply(T, function(n) raw_to_bin(rev(n)))
```

lea_key = 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

T <- split_n(lea_key, 4) : 바이트 배열인 lea_key를 워드 배열로 나눠 T에 저장

T <- lapply(T, function(n) (rev(n))) : Little endian

T[[1]] = 03 02 01 00, T[[2]] = 07 06 05 04, T[[3]] = 0b 0a 09 08, T[[4]] = 0f 0e 0d 0c

```
RK <- list()

for(i in 0:23){
  T[[1]] <- ROL(Plus(T[[1]], ROL(raw_to_bin(delta[[i%%4+1]]),i)),1)
  T[[2]] <- ROL(Plus(T[[2]], ROL(raw_to_bin(delta[[i%%4+1]]),i+1)),3)
  T[[3]] <- ROL(Plus(T[[3]], ROL(raw_to_bin(delta[[i%%4+1]]),i+2)),6)
  T[[4]] <- ROL(Plus(T[[4]], ROL(raw_to_bin(delta[[i%%4+1]]),i+3)),11)
  RK[[i*6+1]]<-T[[1]]
  RK[[i*6+2]]<-T[[2]]
  RK[[i*6+3]]<-T[[3]]
  RK[[i*6+4]]<-T[[2]]
  RK[[i*6+5]]<-T[[4]]
  RK[[i*6+6]]<-T[[2]]
}
```

키 스케줄 알고리즘과 동일하게 코드를 작성한다.

내부변수 T를 이용하여 라운드키 RK를 생성한다.

RK는 24라운드 각각의 키로 사용된다.

■ Encrypt

- 알고리즘 1 : 암호화 알고리즘

알고리즘 1 암호화 함수: $C \leftarrow Encrypt(P, RK_0^{enc}, RK_1^{enc}, \cdots, RK_{Nr-1}^{enc})$
입력: 128비트 평문 P, Nr개의 192비트 라운드키 $RK_0^{enc}, RK_1^{enc}, \cdots, RK_{Nr-1}^{enc}$
출력: 128비트 암호문 C
1: $X_0 \leftarrow P$
2: **for** i = 0 to (Nr − 1) **do**
3: $X_{i+1} \leftarrow Round^{enc}(X_i, RK_i^{enc})$
4: **end for**
5: $C \leftarrow X_{Nr}$

LEA 의 암호화 함수 Encrypt 는 KeyShedule 을 수행하여 생성된

24 개의 128 비트 라운드 키 RK 와 128 비트 평문 P 를 입력 받아,

알고리즘 1 을 수행하여 128 비트 암호문 C 를 출력한다.

- 알고리즘 2 : 라운드 함수 알고리즘

알고리즘 2 암호화 과정의 i번째 라운드 함수: $X_{i+1} \leftarrow Round^{enc}(X_i, RK_i^{enc})$
입력: 128비트 내부상태 변수 X_i, 192비트 라운드키 RK_i^{enc}
출력: 128비트 내부상태 변수 X_{i+1}
1: $X_{i+1}[0] \leftarrow ROL_9((X_i[0] \oplus RK_i^{enc}[0]) \boxplus (X_i[1] \oplus RK_i^{enc}[1]))$
2: $X_{i+1}[1] \leftarrow ROR_5((X_i[1] \oplus RK_i^{enc}[2]) \boxplus (X_i[2] \oplus RK_i^{enc}[3]))$
3: $X_{i+1}[2] \leftarrow ROR_3((X_i[2] \oplus RK_i^{enc}[4]) \boxplus (X_i[3] \oplus RK_i^{enc}[5]))$
4: $X_{i+1}[3] \leftarrow X_i[0]$

128비트 내부상태 변수 X_Round와 192비트 라운드키 RK로부터

알고리즘 2를 수행하여 새로운 128비트 내부상태 변수 X_NextRound를 생성한다.

워드 배열 X_Round = (X_Round[[1]], X_Round[[2]], X_Round[[3]], X_Round[[4]])로

표기된다. X_NextRound도 동일하게 표기된다.

- 소스코드는 다음과 같다.

```
##### LEA_Encrypt #####

msg_chr <- "Hong kong has been rocked by pro-democracy,
anti-government demonstrations for months, with escalating
anger and violence on all sides. CNN takes a look at how
the protests evolved over the course of the year"
msg_raw <- charToRaw(msg_chr)
msg_raw

Plain_text <- split_n(msg_raw, 16)
len <- length(Plain_text)
Cipher_text <- list()
```

msg_raw <- charToRaw(msg_chr) : character 형태인 메시지를 raw형태로 바꿔준다.

Plain_text <- split_n(msg_raw, 16) : LEA는 블록암호로, 16byte 블록으로 나눠

암호화한다.

len : 13 개

```
Cipher_text <- list()

for(j in 1:len){
  X_Round <- split_n(Plain_text[[j]], 4)
  X_Round <- lapply(X_Round, function(n) raw_to_bin(rev(n)))
  X_NextRound <- list()

  for(i in 0:23){
    X_NextRound[[1]] <- ROL(Plus(bitwXor(X_Round[[1]], RK[[i*6+1]]),
                                 bitwXor(X_Round[[2]], RK[[i*6+2]])), 9)
    X_NextRound[[2]] <- ROR(Plus(bitwXor(X_Round[[2]], RK[[i*6+3]]),
                                 bitwXor(X_Round[[3]], RK[[i*6+4]])), 5)
    X_NextRound[[3]] <- ROR(Plus(bitwXor(X_Round[[3]], RK[[i*6+5]]),
                                 bitwXor(X_Round[[4]], RK[[i*6+6]])), 3)
    X_NextRound[[4]] <- X_Round[[1]]

    X_Round <- lapply(X_NextRound, function(x) x)
  }
  block_text <- as.vector(sapply(X_Round, function(n) rev(bin_to_raw(n))))
  Cipher_text <- unlist(append(Cipher_text, block_text))
}
Cipher_text
```

X_Round <- split_n(Plain_text[[j]], 4) : 바이트 배열인 Plain_text 를

워드 배열로 나눠 X_Round 에 저장

X_Round <- lapply(X_Round, function(n) (rev(n))) : Little endian

암호화 알고리즘, 라운드 알고리즘과 동일하게 코드를 작성한다.

내부변수 X_Round 과, 라운드 키 RK 를 이용하여 라운드 알고리즘을 수행하여

24 라운드를 반복하면 메시지가 암호화된다.

block_text <- as.vector(sapply(X_Round, function(n) (rev(n)))) : 워드 배열인 X_Round 를 바이트 배열인 block_text 에 저장, Little endian

Cipher_text <- unlist(append(Cipher_text, block_text)) : block_text 를 len 만큼 for 문을 반복해서 Cipher_text 에 추가해서 벡터로 만든다.

■ Decrypt

- 알고리즘 1 : 복호화 함수

알고리즘 6 복호화 함수: $P \leftarrow Decrypt(C, RK_0^{dec}, RK_1^{dec}, \cdots, RK_{Nr-1}^{dec})$
입력: 128비트 암호문 C, Nr개의 192비트 라운드키 $RK_0^{dec}, RK_1^{dec}, \cdots, RK_{Nr-1}^{dec}$
출력: 128비트 평문 P
1: $X_0 \leftarrow C$
2: **for** i = 0 to (Nr − 1) **do**
3: $X_{i+1} \leftarrow Round^{dec}(X_i, RK_i^{dec})$
4: **end for**
5: $P \leftarrow X_{Nr}$

LEA의 복호화 함수 Decrypt는 KeySchedule을 수행하여 생성된

24개의 192비트 RK와 128비트 암호문 C를 입력 받아

알고리즘 1을 수행하여 128 비트 평문 P을 출력한다.

- 알고리즘 2 : 라운드 함수

알고리즘 7 복호화 과정의 i번째 라운드 함수: $X_{i+1} \leftarrow Round^{dec}(X_i, RK_i^{dec})$
입력: 128비트 내부상태 변수 X_i, 192비트 라운드키 RK_i^{dec}
출력: 128비트 내부상태 변수 X_{i+1}
1: $X_{i+1}[0] \leftarrow X_i[3]$
2: $X_{i+1}[1] \leftarrow (ROR_9(X_i[0]) \boxminus (X_{i+1}[0] \oplus RK_i^{dec}[0])) \oplus RK_i^{dec}[1]$
3: $X_{i+1}[2] \leftarrow (ROL_5(X_i[1]) \boxminus (X_{i+1}[1] \oplus RK_i^{dec}[2])) \oplus RK_i^{dec}[3]$
4: $X_{i+1}[3] \leftarrow (ROL_3(X_i[2]) \boxminus (X_{i+1}[2] \oplus RK_i^{dec}[4])) \oplus RK_i^{dec}[5]$

128비트 내부상태 변수 X_Round와 192비트 라운드키 RK로부터

알고리즘 2를 수행하여 새로운 128비트 내부상태 변수 X_NextRound를 생성한다.

워드 배열 X_Round = (X_Round[[1]], X_Round[[2]], X_Round[[3]], X_Round[[4]])로

표기된다. X_NextRound 도 동일하게 표기된다.

- 암호화 RK 와 복호화 RK 의 관계

$$RK_i^{dec} = RK_{Nr-i-1}^{enc} \quad (0 \le i \le (Nr-1))$$

- 소스코드는 다음과 같다.

```
##### LEA_Decrypt #####

Cipher_text <- split_n(Cipher_text, 16)
Decrypt_text <- list()
```

LEA 는 블록암호로, 16byte 블록으로 나눠 복호화 한다.

```
for(j in 1:len){
  X_Round <- split_n(Cipher_text[[j]], 4)
  X_Round <- lapply(X_Round, function(n) raw_to_bin(rev(n)))
  X_NextRound <- list()

  for(i in 0:23){
    X_NextRound[[1]] <- X_Round[[4]]
    X_NextRound[[2]] <-
      bitwXor(Minus(ROR(X_Round[[1]], 9),
                    bitwXor(X_NextRound[[1]], RK[[((24-i-1)*6)+1]])),
              RK[[((24-i-1)*6) + 2]])
    X_NextRound[[3]] <-
      bitwXor(Minus(ROL(X_Round[[2]], 5),
                    bitwXor(X_NextRound[[2]], RK[[((24-i-1)*6)+3]])),
              RK[[((24-i-1)*6) + 4]])
    X_NextRound[[4]] <-
      bitwXor(Minus(ROL(X_Round[[3]], 3),
                    bitwXor(X_NextRound[[3]], RK[[((24-i-1)*6)+5]])),
              RK[[((24-i-1)*6)+6]])

    X_Round <- lapply(X_NextRound, function(x) x)
  }
  block_text <- as.vector(sapply(X_Round, function(n) rev(bin_to_raw(n))))
  Decrypt_text <- unlist(append(Decrypt_text, block_text))
}
Decrypt_text
```

X_Round <- split_n(Cipher_text[[j]], 4) : 바이트 배열인 Cipher_text 를

워드 배열로 나눠 X_Round 에 저장

X_Round <- lapply(X_Round, function(n) (rev(n))) : Little endian

복호화 알고리즘, 라운드 알고리즘과 동일하게 코드를 작성한다.

내부변수 X_Round 과, 라운드 키 RK 를 이용하여 라운드 알고리즘을 수행하여

24 라운드를 반복하면 메시지가 복호화 된다.

block_text <- as.vector(sapply(X_Round, function(n) (rev(n)))) : 워드 배열인 X_Round 를 바이트 배열인 block_text 에 저장, Little endian

Decrypt_text <- unlist(append(Decrypt_text, block_text) : block_text 를 len 만큼 for 문을 반복해서 Decrypt_text 에 추가해서 벡터로 만든다.

```
Plain_text <- rawToChar(Decrypt_text)
Plain_text
```

raw 형태인 메시지를 character 형태로 바꿔준다.

1.1.2 출력 결과

- 메시지

```
> msg_chr <- "Hong kong has been rocked by pro-democracy,
 anti-government demonstrations for months, with escalatin
g anger and violence on all sides. CNN takes a look at how
 the protests evolved over the course of the year"
```

- 메시지 raw 형 변환

```
> msg_raw
  [1] 48 6f 6e 67 20 6b 6f 6e 67 20 68 61 73 20 62 65
 65 6e 20 72 6f 63 6b 65 64 20 62 79 20 70 72 6f 2d
 [34] 64 65 6d 6f 63 72 61 63 79 2c 20 0a 61 6e 74 69
 2d 67 6f 76 65 72 6e 6d 65 6e 74 20 64 65 6d 6f 6e
 [67] 73 74 72 61 74 69 6f 6e 73 20 66 6f 72 20 6d 6f
 6e 74 68 73 2c 20 0a 77 69 74 68 20 65 73 63 61 6c
[100] 61 74 69 6e 67 20 61 6e 67 65 72 20 61 6e 64 20
 76 69 6f 6c 65 6e 63 65 20 6f 6e 20 61 6c 6c 20 73
[133] 69 64 65 73 2e 20 0a 43 4e 4e 20 74 61 6b 65 73
 20 61 20 6c 6f 6f 6b 20 61 74 20 68 6f 77 20 74 68
[166] 65 20 70 72 6f 74 65 73 74 73 20 65 76 6f 6c 76
 65 64 20 6f 76 65 72 20 74 68 65 20 63 6f 75 72 73
[199] 65 20 6f 66 20 74 68 65 20 79 65 61 72
```

- 메시지 암호화

```
> Cipher_text
  [1] fc 22 32 15 b8 ea 52 03 1f ad 69 70 c8 2e 96
 e5 9e d3 50 76 4f 64 18 d7 31 24 2e 31 32 b3 37
 [32] 6a fc 63 65 9e 45 0c 17 e9 4d bc 09 49 7a 06
 43 ee 0d f4 12 85 a2 ea f1 81 2d d8 e2 a3 58 38
 [63] e0 18 f7 cd 4a 70 2d 75 b0 4e a2 8f c8 21 6a
 96 6e 96 c7 67 e5 f1 ce 36 56 78 ab a9 94 52 7d
 [94] 7b 23 f0 d2 59 fb d8 62 3f 55 68 83 60 12 1c
 ea 4c eb 5d 8d e3 70 df 05 61 ad c5 32 ee 68 a7
[125] f2 de fd 24 04 35 1f 01 43 a3 16 85 75 72 8d
 57 93 ef 08 0b 18 90 e6 3a d8 aa c0 a5 2c 55 2d
[156] d7 b3 f9 08 45 8b 62 c4 b4 f7 6d 0e cd c5 28
 fb 67 be 25 9d 9c 26 43 92 bb 9b 7a 13 09 5f 2c
[187] 85 21 56 f6 91 01 10 12 39 79 35 71 b1 0b 25
 60 96 fc fc 8e 4c e8
```

- 메시지 복호화

```
> Decrypt_text
  [1] 48 6f 6e 67 20 6b 6f 6e 67 20 68 61 73 20 62
 65 65 6e 20 72 6f 63 6b 65 64 20 62 79 20 70 72
 [32] 6f 2d 64 65 6d 6f 63 72 61 63 79 2c 20 61 6e
 74 69 2d 67 6f 76 65 72 6e 6d 65 6e 74 20 64 65
 [63] 6d 6f 6e 73 74 72 61 74 69 6f 6e 73 20 66 6f
 72 20 6d 6f 6e 74 68 73 2c 20 77 69 74 68 20 65
 [94] 73 63 61 6c 61 74 69 6e 67 20 61 6e 67 65 72
 20 61 6e 64 20 76 69 6f 6c 65 6e 63 65 20 6f 6e
[125] 20 61 6c 6c 20 73 69 64 65 73 2e 20 43 4e 4e
 20 74 61 6b 65 73 20 61 20 6c 6f 6f 6b 20 61 74
[156] 20 68 6f 77 20 74 68 65 20 70 72 6f 74 65 73
 74 73 20 65 76 6f 6c 76 65 64 20 6f 76 65 72 20
[187] 74 68 65 20 63 6f 75 72 73 65 20 6f 66 20 74
 68 65 20 79 65 61 72
```

- 메시지 character 형 변환

```
> Plain_text
[1] "Hong kong has been rocked by pro-democracy, anti-government demonstrations
 for months, with escalating anger and violence on all sides. CNN takes a look
 at how the protests evolved over the course of the year"
```

1.2. R openssl을 통한 AES CTR 모드 구현

1.2.1. 소스코드

```
#CTR MODE

library("digest")
```

AES openssl을 구현하기 위해 라이브러리 설치

```
msg_chr <- "Hong kong has been rocked by pro-democracy,
anti-government demonstrations for months, with escalating
anger and violence on all sides. CNN takes a look at how
the protests evolved over the course of the year"
msg_raw <- charToRaw(msg_chr)
```

character 형태인 메시지를 16진수 raw 형태로 변환.

```
key <- as.raw(0:15)
iv <- sample(0:255, 16, replace = TRUE)
aes <- AES(key, mode = "CTR", iv)
cipher_msg <- aes$encrypt(msg_raw)
cipher_msg
```

암호화 키 = 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

AES CTR 모드로 설정 후 CTR모드에 사용되는 initial vector 설정

cipher_msg에 msg를 AES_CTR모드로 암호화한 암호문을 저장

```
aes <- AES(key, mode = "CTR", iv)
dec_msg <- aes$decrypt(cipher_msg, raw = TRUE)
dec_msg
```

dec_msg에 암호문(cipher_msg)를 복호화 한 것을 저장

```
msg <- rawToChar(dec_msg)
msg
```

16진수 raw형태를 character 형태인 메시지로 변환.

소스코드의 CTR모드를 ECB, CBC로 바꾸어서 운영모드 변경 가능

1.2.2. 출력 결과

- 메시지

```
> msg_chr <- "Hong kong has been rocked by pro-democracy,
 anti-government demonstrations for months, with escalatin
g anger and violence on all sides. CNN takes a look at how
 the protests evolved over the course of the year"
```

- raw 형태의 메시지

```
> msg_raw
  [1] 48 6f 6e 67 20 6b 6f 6e 67 20 68 61 73 20 62 65
 [17] 65 6e 20 72 6f 63 6b 65 64 20 62 79 20 70 72 6f
 [33] 2d 64 65 6d 6f 63 72 61 63 79 2c 20 61 6e 74 69
 [49] 2d 67 6f 76 65 72 6e 6d 65 6e 74 20 64 65 6d 6f
 [65] 6e 73 74 72 61 74 69 6f 6e 73 20 66 6f 72 20 6d
 [81] 6f 6e 74 68 73 2c 20 77 69 74 68 20 65 73 63 61
 [97] 6c 61 74 69 6e 67 20 61 6e 67 65 72 20 61 6e 64
[113] 20 76 69 6f 6c 65 6e 63 65 20 6f 6e 20 61 6c 6c
[129] 20 73 69 64 65 73 2e 20 43 4e 4e 20 74 61 6b 65
[145] 73 20 61 20 6c 6f 6f 6b 20 61 74 20 68 6f 77 20
[161] 74 68 65 20 70 72 6f 74 65 73 74 73 20 65 76 6f
[177] 6c 76 65 64 20 6f 76 65 72 20 74 68 65 20 63 6f
[193] 75 72 73 65 20 6f 66 20 74 68 65 20 79 65 61 72
```

- 암호화 메시지

```
> cipher_msg
  [1] a2 ff eb 91 af a5 cc 23 51 18 1c 28 58 44 95 e2
 [17] 5a 43 5d 4b b0 89 05 5e a2 00 8a bc 1a 4a 5d 47
 [33] b9 40 a0 97 f7 8d c9 c7 47 02 28 4b 9f b1 95 5d
 [49] 3d 0d 24 84 a2 75 4e e7 85 43 da b7 c9 d7 46 4e
 [65] ec 94 d3 a4 92 9a e0 f8 e1 04 31 99 77 a4 5d 7b
 [81] 34 c6 41 a9 e0 1b 9d 86 b9 20 32 5c 58 62 fd 7b
 [97] 39 3a 72 33 01 fe f1 dd be 08 d0 21 c8 bc f1 07
[113] 5e 6c 69 b4 2f 7c 27 0e dc b0 14 72 1f 7c f2 85
[129] 06 7a 65 35 07 c2 bd 5b 53 6e 88 37 14 1f fe 0b
[145] ab ed d9 68 8f a4 4d 70 24 c6 ca 9f c5 53 75 17
[161] 36 06 4e 95 3c da b0 a2 c5 ab 1b 9d b9 de e6 8f
[177] b3 34 86 fa 6b f8 21 5c 15 b6 6d 4d 06 45 48 fc
[193] 6b cb 3f dd 56 3b d9 a0 86 39 b2 2b 33 7f 4e b9
```

- 복호화 메시지

```
> dec_msg
  [1] 48 6f 6e 67 20 6b 6f 6e 67 20 68 61 73 20 62 65
 [17] 65 6e 20 72 6f 63 6b 65 64 20 62 79 20 70 72 6f
 [33] 2d 64 65 6d 6f 63 72 61 63 79 2c 20 61 6e 74 69
 [49] 2d 67 6f 76 65 72 6e 6d 65 6e 74 20 64 65 6d 6f
 [65] 6e 73 74 72 61 74 69 6f 6e 73 20 66 6f 72 20 6d
 [81] 6f 6e 74 68 73 2c 20 77 69 74 68 20 65 73 63 61
 [97] 6c 61 74 69 6e 67 20 61 6e 67 65 72 20 61 6e 64
[113] 20 76 69 6f 6c 65 6e 63 65 20 6f 6e 20 61 6c 6c
[129] 20 73 69 64 65 73 2e 20 43 4e 4e 20 74 61 6b 65
[145] 73 20 61 20 6c 6f 6f 6b 20 61 74 20 68 6f 77 20
[161] 74 68 65 20 70 72 6f 74 65 73 74 73 20 65 76 6f
[177] 6c 76 65 64 20 6f 76 65 72 20 74 68 65 20 63 6f
[193] 75 72 73 65 20 6f 66 20 74 68 65 20 79 65 61 72
```

- 문자로 변환한 메시지

```
> msg
[1] "Hong kong has been rocked by pro-democracy, anti-government
 demonstrations for months, with escalating anger and violence on
 all sides. CNN takes a look at how the protests evolved over the
 course of the year"
```

2. 각 암호 운영 모드의 암호화/복호화 시간 측정 및 비교

2.1. 소스 코드

■ 시간 측정에 쓰인 함수 - proc.time()

proc.time은 세가지 값을 도출해 내는데 순서대로 user time, system time, elapsed time을 의미합니다. 이 중 elapsed time은 코드의 총 소요 시간으로, 코드를 시작한 직후로부터 코드 수행이 끝날 때까지의 시간을 초시계로 잰 것과 같이 얼마나 걸렸는지를 초 단위로 알려줍니다. User time과 system time은 elapsed time을 구성하는 요소로, 각각 프로그램 코드 자체를 수행하는데 걸린 시간과 프로그램이 운영체제의 명령을 호출했다면 그때 운영체제가 명령을 수행하는데 걸린 시간을 의미합니다. 이 보고서에서는 elapsed time을 기준으로 시간을 비교했습니다.

■ 소스코드

```
ptm <- proc.time
<<암호구현>>
proc.time() – ptm
```

■ 암호화 한 평문

Hong kong has been rocked by pro-democracy, anti-government demonstrations for months, with escalating anger and violence on all sides. CNN takes a look at how the protests evolved over the course of the year

글자수(공백 포함): 16 * 13

2.2. 시간 측정

■ AES_CBC.r

```
> proc.time() - ptm
 사용자   시스템 elapsed
   0.02     0.02     0.05
```

0.05초

■ AES_CTR.r

```
> proc.time() - ptm
 사용자   시스템 elapsed
   0.02     0.00     0.03
```

0.03초

■ LEA.r

```
> proc.time() - ptm
 사용자   시스템 elapsed
   0.58     0.08     0.69
```

0.69초

2.3. 결과

Openssl 로 구현한 AES의 CBC, CTR모드는 0.01 시간이 걸렸으며 LEA는 0.67초의 시간이 걸렸습니다. LEA는 경량 암호로써 AES 보다 1.5~ 2배 바르다고 알려져 있지만 결과는 정 반대로 나왔습니다. 이는 AES openssl 패키지의 R 최적화가 잘 되어있는 반면 LEA 코드는 R에 최적화되지 않은 for 문의 잦은 사용으로 시간이 느리다고 판단되었습니다.

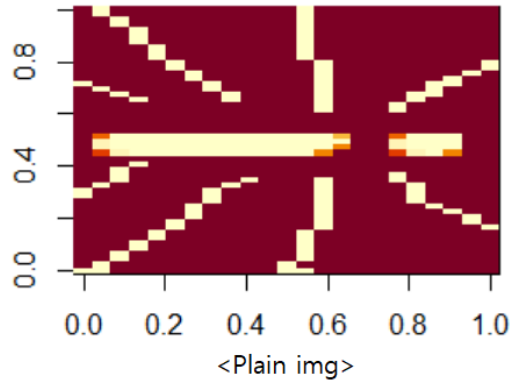
3. 이미지 암호화

3.1. 소스코드

■ 이미지 저장

```
library("png")
library("imager")
library("keras")
library("digest")
my_img2 <- readPNG("C:/Users/pizza/Desktop/what2.png")
my_img2
```

my_img2에 readPNG로 불러드린 png파일 저장



■ 이미지 raw형 변환

```
a <- array_reshape(my_img2[,,2], c(1,23*50))
a <-as.raw(a)
a
```

my_img를 1차 배열로 변환 후 as.raw를 통해 바이트의 16진 표현으로 변환

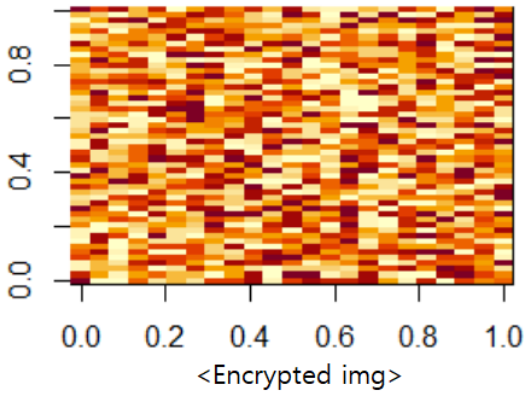
```
a
  [1] 00 01 01 01 01 01 01 01 01 01 01 01 01 01 01 00 00 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
 [34] 01 01 00 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 00 00 01 01 01 01 01 01 01 01 01 01 01 01
 [67] 00 01 01 01 01 01 00 00 00 00 01 01 01 01 01 01 01 01 00 01 01 01 01 01 01 01 01 01 01 01 01 01 00
[100] 00 01 01 00 00 01 01 01 01 01 01 01 01 01 01 01 01 01 01 00 00 00 01 01 00 00 00 00 01 01 01 01 01
[133] 01 00 01 01 01 01 01 01 01 01 01 01 01 01 01 00 00 01 01 01 01 01 01 00 00 01 01 01 01 01 01 01 01
[166] 01 01 01 01 01 01 00 01 00 00 00 00 01 01 01 01 01 01 00 01 01 01 01 01 01 01 01 01 01 00 00 00 01
[199] 01 01 01 01 01 01 01 01 01 00 00 00 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 00 00 00 00 01
[232] 01 01 01 01 01 01 01 01 01 01 00 00 00 00 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 00 00 01
[265] 01 01 01 01 01 01 01 01 01 00 00 00 00 01 01 01 01 01 01 01 01 01 01 01 01 01 01 00 00 01 01 01 01
[298] 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 00 00 00 01 01 01 01 01 01 01 01 01 01 00 00 00 00 01
[331] 01 01 01 01 01 01 01 01 00 00 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
[364] 00 00 00 00 01 01 01 01 01 01 00 00 00 00 01 01 01 01 01 01 01 01 00 00 01 01 01 01 01 01 01 01 01
[397] 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 00 01 01 01 01 01 00 00 00 00
[430] 01 01 01 01 00 00 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
[463] 01 01 01 01 01 01 00 01 01 01 01 01 00 00 00 00 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
[496] 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 00 00 00
[529] 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 00 00 01 01 01 01 01 01 01
[562] 01 01 01 01 01 01 01 01 01 01 01 01 01 00 00 00 00 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
[595] 01 01 01 01 01 01 01 00 01 00 00 00 00 00 00 00 01 01 01 01 01 01 01 01 01 01 01 01 01 01 00 00 00
[628] 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 00 00 00 00 00 00 00 00 00 00 01 01 01 01 01 01 01 00
[661] 00 00 00 00 00 00 00 00 00 00 01 01 01 01 01 00 00 00 00 01 01 01 01 00 00 00 00 00 00 00 00 00 01
[694] 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 00 00
[727] 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
[760] 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
[793] 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
[826] 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
[859] 01 01 01 01 01 01 01 01 01 01 00 00 01 01 01 00 00 00 00 01 01 01 01 00 00 01 01 01 01 01 01 01 01
[892] 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 00 00 00 00 01 01 01 01 01 00 00
[925] 00 00 01 01 01 01 01 01 00 00 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
[958] 01 01 01 01 01 00 01 01 01 01 01 01 01 01 01 00 00 00 00 01 01 01 01 01 01 01 00 00 00 01 01 01 01
[991] 01 01 01 01 01 01 01 01 01 01 01
```

■ 이미지 암호화(AES_CTR모드)

```
key <- as.raw(0:15)
iv <- sample(0:255, 16, replace = TRUE)
aes <- AES(key, mode = "CTR", iv)
cipher_msg <- aes$encrypt(a)
cipher_msg2 <- array_reshape(cipher_msg, c(23, 50))
image(cipher_msg2)
```

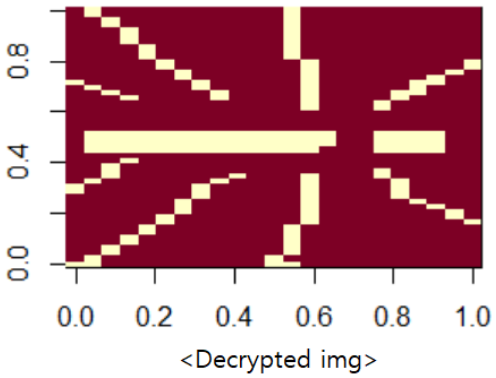
Cipher_msg를 원래의 모양인 c(23,50)으로 변형해 준 후 이미지 출력

```
> cipher_msg
   [1] fc c5 29 16 b3 a6 85 0a 1f 21 92 65 a5 f7 54 3d 58 c8 74 f3 29 72 cc b3 25 2a 6f 4a 39 2f 33 5b 4a
  [34] 36 7c a8 b6 81 5f 67 7b 07 6d e7 1e 76 59 0b 64 ec 07 6e 5f 65 80 e7 81 dd e4 00 06 6e 90 a3 df 17
  [67] 51 a8 77 a7 eb ae 2d 37 e5 dd 45 fd ba 4c 36 97 c1 d1 f9 51 b5 5a 38 af ef 8c ab 57 54 b5 1a 73 46
 [100] ad 0a 19 61 37 14 29 14 0b b3 7e 23 ab 8f 0c 5c b1 2c 4d f8 65 6a 8c 4f 6a 54 16 57 08 e5 0b 63 b9
 [133] 39 78 17 66 ca aa 3a cd 7c af 4b 6d 94 04 31 db 0e dd a0 76 9e cc 6b b9 5e 87 03 2d 45 9b 02 c9 b5
 [166] d3 07 5f da da 9c 74 a0 a2 9c 1d 63 3a 19 25 13 9c 28 ee 71 a1 cb bb 66 c9 c6 1b 5b d2 93 83 46 f2
 [199] 64 69 33 9f 05 db b9 0d ba 34 fe a0 6c 3c e2 54 e0 a3 cd 60 41 7b 45 3d f0 d6 de 45 89 ae 84 01 d9
 [232] df 1e 9b 0d bd d6 79 4f 39 10 07 62 70 60 aa e7 23 84 c7 64 69 3c b7 ba 0e ad 3a 3a 36 6f 81 7e 01
 [265] a4 24 64 af 88 2c c7 4a d6 51 28 21 97 cc 48 ab 8c f9 da ba 9b 2c 87 1a 3c 70 53 f5 b7 17 d8 85 d2
 [298] c7 8f 00 2e 69 cb 6e a2 bd ae 82 00 2c 42 6a 45 de bf cc 49 b4 b5 07 31 c9 e6 a4 4b d9 2e 76 78 f4
 [331] f1 f0 f3 a6 f6 ce 82 c2 7a ec 0c fd 8c 86 22 d5 7d a4 d5 7b ab 0d 73 18 8e 48 53 6b 58 ca ef bd f0
 [364] df 4a a4 55 d8 c6 e5 81 9e 32 ab 8a d4 3a 6f 92 90 9f 7e 0a 0f 65 90 72 fd 9c db 1a e0 15 e9 b2 db
 [397] 90 52 cd d1 ac 78 7a df 49 3c d9 24 82 9a 1e 84 fd 16 4e 51 3d 52 d6 92 e3 73 fb dc 0d 1a dc c0 58
 [430] 7d 95 c6 64 74 52 d4 2d 84 06 17 35 94 23 d4 e8 a1 82 7e 9c 94 db dc 34 c8 95 b7 08 54 8d 4c 06 45
 [463] fe a8 35 d8 df f6 15 91 6d ea c4 11 1b 32 37 d4 cf c8 ae a2 a6 ef 42 e7 59 16 a3 12 4d 74 fa 2e 71
 [496] 63 8d 49 50 c9 1e 0c 11 e7 ee d8 02 9f 7c ab ca 77 49 fc 6b 4e c5 2e 04 32 fd 28 2e 9d f2 b5 a8 ff
 [529] 19 0f da 95 b7 36 be 22 cd b5 b9 13 b5 26 6d e1 78 3f 83 32 23 65 b8 f9 46 a5 f2 aa 70 bc 84 0f e1
 [562] d9 e0 88 f0 d2 11 e5 07 51 3c 8d 08 59 33 f3 4f b8 2e b9 5f 74 a7 b8 e7 74 ae 52 47 c6 fb a2 45 fb
 [595] 9a 40 31 84 fd d8 e2 fe 55 1b 91 e4 e6 5f 6d cc a9 7d 64 ec 81 da 1c 3a e2 ac a6 b5 09 6a 57 e6 37
 [628] 3e 3c 79 fb 51 03 e0 88 0a 27 a8 3c 0e 08 e0 fe 2b 65 7c 4d 7d cf 3d c5 17 44 5e ba 7c b2 95 c5 2d
 [661] 01 b3 7f c5 30 74 82 a8 a7 0d 4b 21 42 28 8f 14 2f b1 88 78 a9 2e a5 88 da 37 0f 01 57 00 5c a4 34
 [694] b2 27 ab 46 17 12 29 d2 50 3a 81 6a 99 aa f2 ba c1 f9 cb f5 f0 ae 08 65 bc fe e4 f9 72 f6 c9 5e 95
 [727] 58 03 f6 06 4b 17 12 00 12 2e 72 43 39 e2 95 a7 64 a1 9b 94 ef 5f ff 2a e1 e4 8a e2 80 d1 2c 1b 56
 [760] 79 25 20 1c a4 16 58 22 71 2c 03 fd db bc cc 95 a3 b7 d5 ff 10 d0 11 02 05 ea 65 a3 11 e4 83 9f 40
 [793] 48 8f 67 25 57 f5 3c 65 71 32 b5 06 37 d1 af 8d 86 32 05 07 e9 07 e1 64 23 3a 3b 7c 0b af 0f 56 86
 [826] f9 e0 29 54 5f cc c0 2f 42 c9 e1 1a 91 07 94 c5 47 55 e3 b6 5f 7a 89 fb 38 1e 0f e9 33 b3 e1 75 42
 [859] e8 8f cf c2 b5 b6 00 89 bd 50 15 fe a4 45 33 cd bc 2b 6f 51 b0 f2 19 c5 d2 87 2e 38 60 36 71 73 6c
 [892] 71 b1 b8 31 08 71 eb 00 c9 c8 02 8f b7 3f fd 7b 15 ff 00 dc 64 ed 79 00 90 0e ce 91 8c d3 b1 fc 48
 [925] 9f c0 07 1a ea 2b 5d 93 28 30 6e df ea 5e a4 cd 8e c3 c6 ec e8 41 bc e4 20 93 09 e7 f7 98 bb f1 68
 [958] 97 c5 81 5b f3 4b 70 68 8f 3c 60 a5 be 46 33 7b 0a c9 a8 29 31 63 3d 61 72 7b 7b 2f 6e 25 9d e9 33
 [991] 79 7d 00 25 85 5c 1b 94 db 06
```

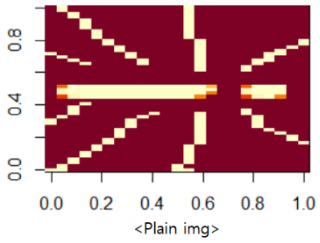


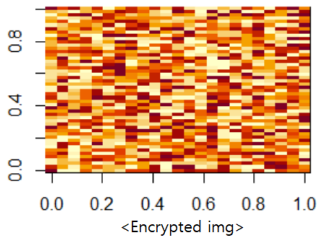
■ 이미지 복호화

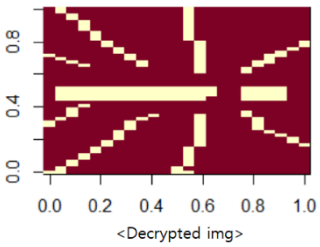
```
aes <- AES(key, mode="CTR", iv)
dec_msg <- aes$decrypt(cipher_msg, raw=TRUE)
msg2<-array_reshape(dec_msg, c(23,50))
image(msg2)
```



3.2. 결과







#참고

모드를 ECB, CBC로 바꾸었을 때

```
Error in aes$encrypt(a) : Text length must be a multiple of 16 bytes
```

다음과 같은 에러문이 뜨는 것을 알 수 있었습니다.

이는 CTR 모드는 패딩이 필요 없는 데에 반해 ECB, CBC 모드는 텍스트의 길이가

16바이트의 배수가 필요하다고 요구하는 것을 보아 AES Openssl ECB, CBC 모드에는

패딩이 구현되어 있지 않음을 알 수 있었습니다.