Práctica 4
El algoritmo AES

Índice

Para entregar

- Carpeta "AES" con el código de las funciones $KeyExpansion()$, $cifAES()$, y $decAES()$ completado.

1. Introducción

El objetivo de esta práctica es implementar el estándar de cifrado simétrico AES (*"Advanced Encryption Standard"*).

El algoritmo AES cifra bloques de 128, 192 o 256 bits, utilizando claves iniciales de 128, 192 o 256 bits. La clave inicial debe ser previamente expandida para cifrar o descifrar el bloque de mensaje.

El bloque de mensaje a cifrar se estructura en palabras de 4 bytes, que constituyen el *estado inicial*. Este estado inicial sufre una serie de transformaciones dando lugar a estados intermedios. Cada transformación opera sobre el *estado* resultado de la transformación anterior. Las transformaciones que utiliza el AES son:

Fichero.R	Transformación
SubByte.R	SubByte(), InvSubByte()
ByteSub.R	ByteSub(), InvByteSub()
ShiftRow.R	ShiftRow(), InvShiftRow()
MixColumn.R	MixColumn(), InvMixColumn()
AddRoundKey.R	AddRoundKey()
RotByte.R	RotByte()

Estas transformaciones se suministran al alumno ya programadas.

Con las funciones anteriores se programa la expansión de la clave inicial, el cifrado de un bloque y su descifrado. Denominaremos a los correspondientes ficheros

KeyExpansion.R	KeyExpansion()
cifAES.R	cifAES()
decAES.R	decAES()

Los ficheros "Key.Expansion.R", "cifAES.R" y "decAEs.R" se entregan al alumno incompletos. La tarea de esta práctica consiste en completar la implementación de estas transformaciones.

Los ficheros "SubByte.R", "ByteSub.R", "ShiftRow.R" y "MixColumn.R" implementan la operación para cifrar y para descifrar. La distinción se realiza con el segundo parámetro de su entrada. Se ha optado por el criterio: 0 para el sentido directo (cifrar) y 1 para el inverso (descifrar).

La implementación elegida convierte a $SubByte()$ y $ByteSub()$ en una caja de sustitución o S-caja, como en el caso del cifrado simétrico DES. La única diferencia entre estas dos funciones es que $SubByte()$ opera sobre un vector de bytes, mientras que $ByteSub()$ lo hace sobre una matriz.

En la implementación de $MixColumn()$ y $AddRoundKey()$ se han utilizado las funciones auxiliares $xorbit()$, $xtime()$ y $prodbytes()$.

```
xorbit <- function(a,b)
  # Entrada: a, b vectores de bytes expresados en decimal.
  # Salida: vector de bytes expresados en decimal, calculados a partir
  #            de la suma XOR bit a bit de los bytes de a y b.
  # a, b deben tener la misma longitud.
```

```
xtime <- function(a)
  # Entrada: a (byte expresado en decimal).
  # Salida: byte expresado en decimal, calculado a partir del
  #         producto del byte a por x ("02") en el cuerpo GF(2^8)
  #         generado por el polinomio irreducible m(x)=x^8+x^4+x^3+x+1
```

```
bytesprod <- function(a, b)
  #Entrada: a, b (bytes expresado en decimal).
  #Salida: byte expresado en decimal, calculado a partir del
  #        producto de los bytes a y b en el cuerpo GF(2^8)
  #        generado por el polinomio irreducible m(x)=x^8+x^4+x^3+x+1
```

Se recomienda encarecidamente al alumno que lea el código (o al menos las cabeceras) de todos los ficheros.R para conocer cómo opera cada transformación antes de proceder a la implementación.

Se cifrarán bloques de 16 bytes ($N_b = 4$ palabras) utilizando una clave inicial de 16 bytes ($N_k = 4$ palabras). El número de rondas para estos tamaños de bloque y clave es $n = 10$. El estado se representará por una matriz de 4 filas y $N_b = 4$ columnas.

2. Programas

2.1. Expansión de Clave

Como el número de rondas es $n = 10$, se necesitan 11 subclaves $K_0, K_1, \ldots, K_{10}$ de cuatro palabras cada una. Es decir, 44 palabras. Hemos de expandir, por tanto, la clave inicial.

La clave expandida será una matriz 4×44 de bytes, denotada por W, obtenida a partir de la clave inicial K_0.

En la función de expansión se utilizan las funciones $SubByte()$ y $RotByte()$, y unas constantes $Rcon(j)$, $1 \leq j \leq 10$, que se definen de la siguiente forma :

- $Rcon(j) = (RC(j), 00, 00, 00)$.
- $RC(j)$ es el elemento de GF(2^8) correspondiente a x^{j-1} módulo $m(x) = x^8 + x^4 + x^3 + x + 1$.

Observación: almacenaremos las constantes $Rcon(j)$ en formato decimal en la matriz $Rcon$.

```
> Rcon
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    2    4    8   16   32   64  128   27    54
[2,]    0    0    0    0    0    0    0    0    0     0
[3,]    0    0    0    0    0    0    0    0    0     0
[4,]    0    0    0    0    0    0    0    0    0     0
```

Algoritmo de expansión de clave para $N_k \leq 6$

Entrada: *claveinicial* (vector de $4N_k$ bytes que representa la clave inicial).

Salida: Matriz $4 \times N_b(n+1)$ de bytes que representa la clave expandida.

Paso 1. Para $i = 1, \ldots, N_k$

Paso 1.1. Hacer $W[\ ,i] = claveinicial[(4(i-1)+1) : (4i)]$.

Paso 2. Para $i = N_k + 1, \ldots, N_b(n+1)$

Paso 2.1. Hacer $tmp = W[\ ,i-1]$

Paso 2.2. Si $i \equiv 1 \mod N_k$

Paso 2.2.1.
Hacer $tmp = SubByte(RotByte(tmp)) \oplus Rcon[\ ,(i-1)/N_k]$.

Paso 2.3. Hacer $W[\ ,i] = W[\ ,i-N_k] \oplus tmp$.

Paso 3. Salida W.

- Programar el algoritmo anterior ($KeyExpansion()$). La entrada es un vector de 16 bytes expresados en hexadecimal que representa la clave inicial, y la salida una matriz 4×44 de bytes expresados en decimal que representa la clave expandida.

  Observación: para efectuar la suma $\oplus$ podemos utilizar la función $xorbit()$.

EJERCICIO. Obtener las claves expandidas para las siguientes claves iniciales (las dos claves están almacenadas en el fichero "prueba.R"):

1. *claveinicial*1

   ("00", "01", "02", "03", "04", "05", "06", "07", "08", "09", "0a", "0b", "0c", "0d", "0e", "0f").

2. *claveinicial*2

   ("2b", "7e", "15", "16", "28", "ae", "d2", "a6", "ab", "f7", "15", "88", "09", "cf", "4f", "3c").

Solución:

1. *claveexpandida*1:

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
[1,]    0    4    8   12  214  210  218  214  182   100   190   104   182   210
[2,]    1    5    9   13  170  175  166  171  146    61   155    48   255   194
[3,]    2    6   10   14  116  114  120  118  207   189   197   179   116   201
[4,]    3    7   11   15  253  250  241  254   11   241     0   254    78   191
     [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
[1,]   108     4    71   149   249   253    60   169    80   173    94   247
[2,]    89   105   247    53   108     5   170   159   243   246    57   166
[3,]    12   191   247    62    50   141   163   157   175    34    15   146
[4,]   191    65   188     3   188   253   232   235    87   170   125   150
     [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38]
[1,]   167    10    20   227    68    78    71   164   224   174    84   240
[2,]    85   163   249    95    10   169    67    28    22   191   153   133
[3,]    61    31   112   226   223   192   135   101   186   122    50    87
[4,]   193   107    26   140    77    38    53   185   244   210   209   104
     [,39] [,40] [,41] [,42] [,43] [,44]
[1,]    16   190    19   227   243    77
[2,]   147    44    17   148     7    43
[3,]   237   151    29    74   167    48
[4,]   156    78   127    23   139   197
```

2. *claveexpandida*2:

```
   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
[1,]   43   40  171    9  160  136   35   42  242   122    89   115    61    71
[2,]  126  174  247  207  250   84  163  108  194   150    53    89   128    22
[3,]   21  210   21   79  254   44   57  118  149   185   128   246    71   254
[4,]   22  166  136   60   23  177   57    5  242    67   122   127   125    62
     [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
[1,]    30   109   239   168   182   219   212   124   202    17   109    17
[2,]    35   122    68    82   113    11   209   131   242   249   136    11
[3,]   126   136   165    91    37   173   198   157   184    21   163    62
[4,]    68    59    65   127    59     0   248   135   188   188   122   253
     [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38]
[1,]   219   202    78    95   132    78   234   181    49   127   172    25
[2,]   249     0    84    95   166   166   210   141    43   141   119   250
[3,]   134   147   247   201    79   220   115   186   245    41   102   220
[4,]    65   253    14   243   178    79    33   210    96    47   243    33
     [,39] [,40] [,41] [,42] [,43] [,44]
[1,]    40    87   208   201   225   182
[2,]   209    92    20   238    63    99
[3,]    41     0   249    37    12    12
[4,]    65   110   168   137   200   166
```

2.2. Cifrado

Algoritmo de cifrado AES con n rondas

Entrada: *mensaje* (vector de $4N_b$ bytes que representa el mensaje en claro).
claveinicial (vector de $4N_k$ bytes que representa la clave inicial).

Salida: Vector de $4N_b$ bytes que representa el mensaje cifrado.

Paso 1. Calcular las clave expandida W a partir de *claveinicial*.

Paso 2. Copiar en sentido columnas *mensaje* en la matriz *estado* (matriz $4 \times N_b$) (las 4 primeras componentes de *mensaje* formarán la primera columna de *estado*, las 4 siguientes, la segunda, etc).

Paso 3. Hacer $clave.ronda = W[\ ,1 : N_b]$.

Paso 4. Hacer $estado = AddRoundKey(estado, clave.ronda)$.

Paso 5. Para $i = 1, \ldots, n$

Paso 5.1. Extraer de W la clave de ronda actual ($clave.ronda$).

Paso 5.2. Hacer $estado = ByteSub(estado)$.

Paso 5.3. Hacer $estado = ShiftRow(estado)$.

Paso 5.4. Si $i \neq n$

Paso 5.4.1. Hacer $estado = Mixcolumn(estado)$.

Paso 5.5. Hacer $estado = AddRoundKey(estado, clave.ronda)$.

Paso 6. Hacer $cifrado = as.hexmode(estado)$.

Paso 7. Salida $cifrado$.

- Programar el algoritmo anterior ($cifAES()$). La entrada son dos vectores de 16 bytes expresados en hexadecimal que representan el mensaje en claro y la clave inicial, y la salida será un vector de 16 bytes expresados en hexadecimal que representa el mensaje cifrado.

EJERCICIO. Cifrar los mensajes con las claves iniciales dadas (los mensajes y las claves están almacenados en el fichero "prueba.R"):

1. $mensaje1$

   ("00", "11", "22", "33", "44", "55", "66", "77", "88", "99", "aa", "bb", "cc", "dd", "ee", "ff"),

   $claveinicial1$

   ("00", "01", "02", "03", "04", "05", "06", "07", "08", "09", "0a", "0b", "0c", "0d", "0e", "0f").

2. $mensaje2$

   ("32", "43", "f6", "a8", "88", "5a", "30", "8d", "31", "31", "98", "a2", "e0", "37", "07", "34"),

   $claveinicial2$

   ("2b", "7e", "15", "16", "28", "ae", "d2", "a6", "ab", "f7", "15", "88", "09", "cf", "4f", "3c").

Solución:

1. $cifrado1$:

```
 [1] "69" "c4" "e0" "d8" "6a" "7b" "04" "30" "d8" "cd" "b7" "80" "70" "b4" "c5"
[16] "5a"
```

2. $cifrado2$:

```
 [1] "39" "25" "84" "1d" "02" "dc" "09" "fb" "dc" "11" "85" "97" "19" "6a" "0b"
[16] "32"
```

2.3. Descifrado

Para descifrar un cifrado AES podemos invertir el orden de las subclaves empleadas en el cifrado y el de las funciones en cada ronda. Las funciones que se aplican son las inversas de las utilizadas en el cifrado. Observemos que la función inversa de AddRoundKey es AddRoundKey.

Cifrado:

$$AK \mid BS \quad SR \quad MC \quad AK \mid \ldots \mid BS \quad SR \quad MC \quad AK \mid BS \quad SR \quad AK$$

Descifrado:

$$AK \quad ISR \quad IBS \mid AK \quad IMC \quad ISR \quad IBS \mid \ldots \mid AK \quad IMC \quad ISR \quad IBS \mid AK$$

Existe otra manera de encadenar las funciones en el descifrado. Observemos que en el cifrado y descifrado están intercambiados $ByteSub$ con $ShiftRow$ por una parte y $AddRoundKey$ con $MixColumn$ por otra.

$$\begin{array}{cccccccccccc} AK & \underbrace{BS} & \underbrace{SR} & \underbrace{MC} & \underbrace{AK} & \ldots & \underbrace{BS} & \underbrace{SR} & \underbrace{MC} & \underbrace{AK} & \underbrace{BS} & \underbrace{SR} & AK \\ AK & ISR & IBS & AK & IMC & \ldots & ISR & IBS & AK & IMC & ISR & IBS & AK \end{array}$$

- Como $ByteSub$ opera sobre bytes y $ShiftRow$ sólo los cambia de lugar, las dos operaciones pueden intercambiarse.
- La secuencia

  $$AddRoundKey(S, RoundK)$$
  $$InvMixColumn(S)$$

  puede cambiarse por

  $$InvMixColumn(S)$$
  $$AddRoundKey(S, InvMixColumn(RoundK))$$

  donde $InvMixColumn(RoundK)$ se obtiene aplicando $InvMixColumn$ a $RoundK$.

De esta forma, la estructura del descifrado es idéntica a la del cifrado, usando las funciones inversas (como antes, las claves debe ser usadas en orden inverso y modificadas según acabamos de indicar).

Denotaremos por AK^I la transformación $AddRoundKey(S, InvMixColumn(RoundK))$.

$$\begin{array}{c|cccc|c|cccc|ccc} AK & BS & SR & MC & AK & \ldots & BS & SR & MC & AK & BS & SR & AK \\ AK & IBS & ISR & IMC & AK^I & \ldots & IBS & ISR & IMC & AK^I & IBS & ISR & AK \end{array}$$

Algoritmo de descifrado AES con n rondas

Entrada: $cifrado$ (vector de $4N_b$ bytes que representa el mensaje cifrado).
$claveinicial$ (vector de $4N_k$ bytes que representa la clave inicial).

Salida: Vector de $4N_b$ bytes que representa el mensaje en claro.

Paso 1. Calcular la clave expandida W a partir de $claveinicial$.

Paso 2. Copiar en sentido columnas $cifrado$ en la matriz $estado$ (matriz $4 \times N_b$).

Paso 3. Hacer $clave.ronda = W[\ , (nN_b + 1) : (n + 1)N_b]$.

Paso 4. Hacer $estado = AddRoundKey(estado, clave.ronda)$.

Paso 5. Para $i = 1, \ldots, n$

Paso 5.1. Extraer de W la clave de ronda actual ($clave.ronda$).

Paso 5.2. Hacer $estado = InvByteSub(estado)$.

Paso 5.3. Hacer $estado = InvShiftRow(estado)$.

Paso 5.4. Si $i \neq n$

Paso 5.4.1. Hacer $estado = InvMixcolumn(estado)$.

Paso 5.4.2. Hacer $clave.ronda = InvMixcolumn(clave.ronda)$.

Paso 5.5. Hacer $estado = AddRoundKey(estado, clave.ronda)$.

Paso 6. Hacer $mensaje = as.hexmode(estado)$.

Paso 7. Salida $mensaje$.

- Programar el algoritmo anterior ($decAES()$). La entrada son dos vectores de 16 bytes expresados en hexadecimal que representan el mensaje cifrado y la clave inicial, y la salida será un vector de 16 bytes expresados en hexadecimal que representa el mensaje en claro.

EJERCICIO. Descifrar los mensajes que han sido cifrados en el ejercicio de la sección de cifrado y comprobar que el resultado coincide con el mensaje en claro original.