Encryption

CS6025 Data Encoding

Yizong Cheng

2-24-15

PENGUIN CLASSICS

EDGAR ALLAN POE

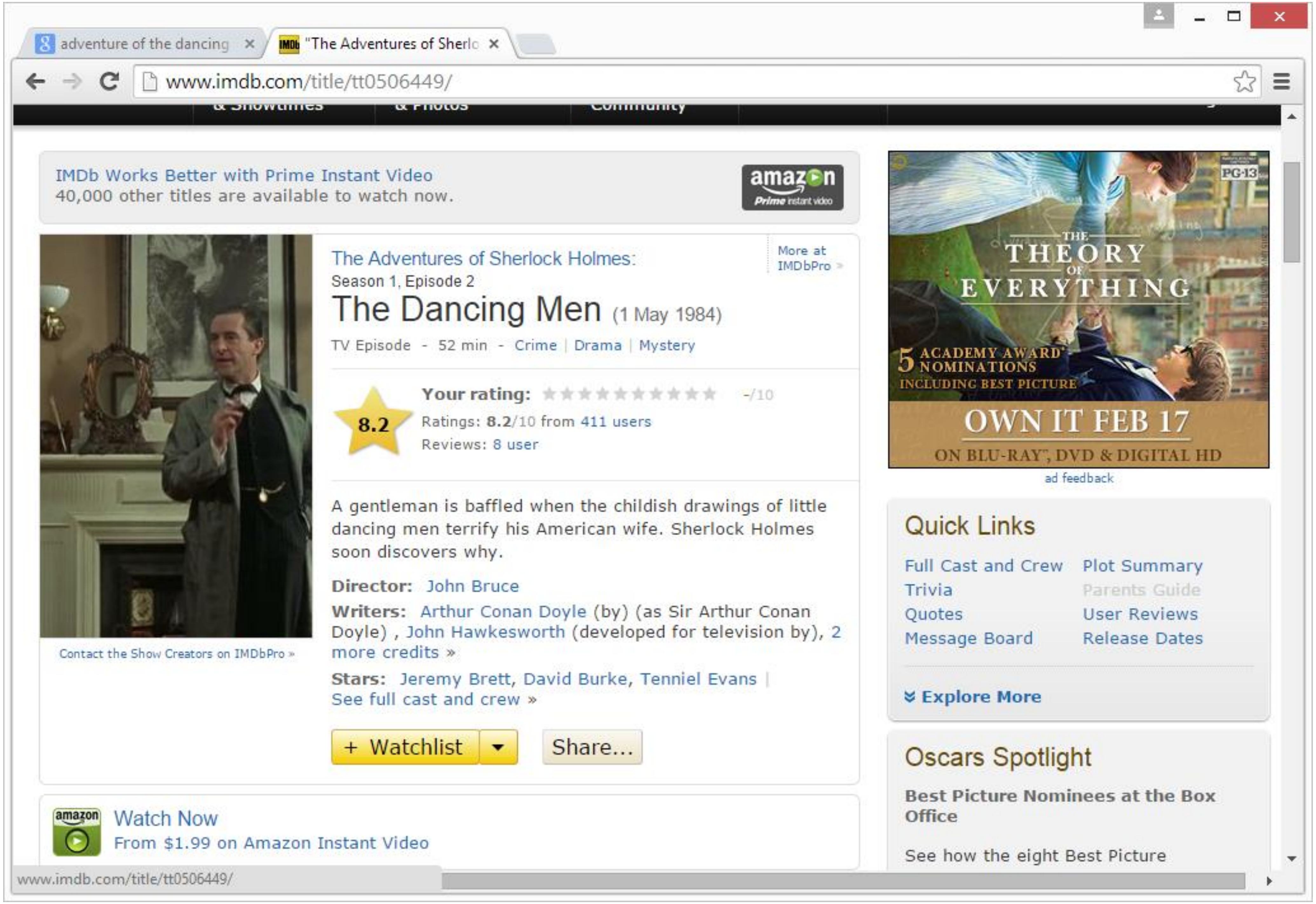
THE FALL OF THE HOUSE OF USHER

AND OTHER WRITINGS

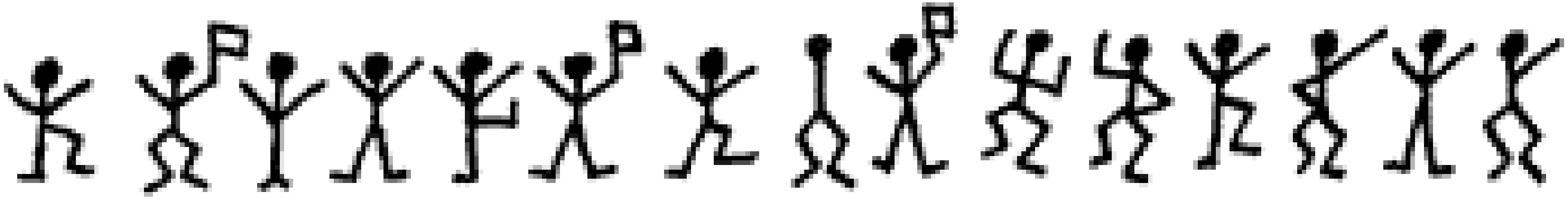
Edgar A. Poe’s “The Gold-Bug” (1843)

53‡‡†305))6*;4826)4‡.)4‡);806*;48†8¶60))85;]8*:‡*8†83(88) 5*†;
46(;88*96*?;8)*‡(;485);5*†2:*‡(;4956*2(5*—4)8¶8*;4069285);)6†8)
4‡‡;1(‡9;48081;8:8‡1;48†85;4)485†528806*81(‡9;48;(88;4(‡?34;48)
4‡;161;:188;‡?;

‘“*A good glass in the bishop’s hostel in the devil’s seat twenty-one degrees and thirteen minutes northeast and by north main branch seventh limb east side shoot from the left eye of the death’s-head a bee-line from the tree through the shot fifty feet out.*”’


adventure of the dancing
"The Adventures of Sherlo
www.imdb.com/title/tt0506449/
IMDb Works Better with Prime Instant Video
40,000 other titles are available to watch now.
amazon Prime instant video
The Adventures of Sherlock Holmes:
Season 1, Episode 2
The Dancing Men (1 May 1984)
TV Episode - 52 min - Crime | Drama | Mystery
More at IMDbPro »
Your rating: -/10
8.2
Ratings: 8.2/10 from 411 users
Reviews: 8 user
A gentleman is baffled when the childish drawings of little dancing men terrify his American wife. Sherlock Holmes soon discovers why.
Director: John Bruce
Writers: Arthur Conan Doyle (by) (as Sir Arthur Conan Doyle) , John Hawkesworth (developed for television by), 2 more credits »
Stars: Jeremy Brett, David Burke, Tenniel Evans | See full cast and crew »
+ Watchlist
Share...
Contact the Show Creators on IMDbPro »
Watch Now
From $1.99 on Amazon Instant Video
PG-13
THE THEORY OF EVERYTHING
5 ACADEMY AWARD NOMINATIONS
INCLUDING BEST PICTURE
OWN IT FEB 17
ON BLU-RAY, DVD & DIGITAL HD
ad feedback
Quick Links
Full Cast and Crew
Plot Summary
Trivia
Parents Guide
Quotes
User Reviews
Message Board
Release Dates
Explore More
Oscars Spotlight
Best Picture Nominees at the Box Office
See how the eight Best Picture
www.imdb.com/title/tt0506449/

The Adventure of the Dancing Men (1898)



Substitute Bytes (SubBytes) in AES

Table 5.2 AES S-Boxes

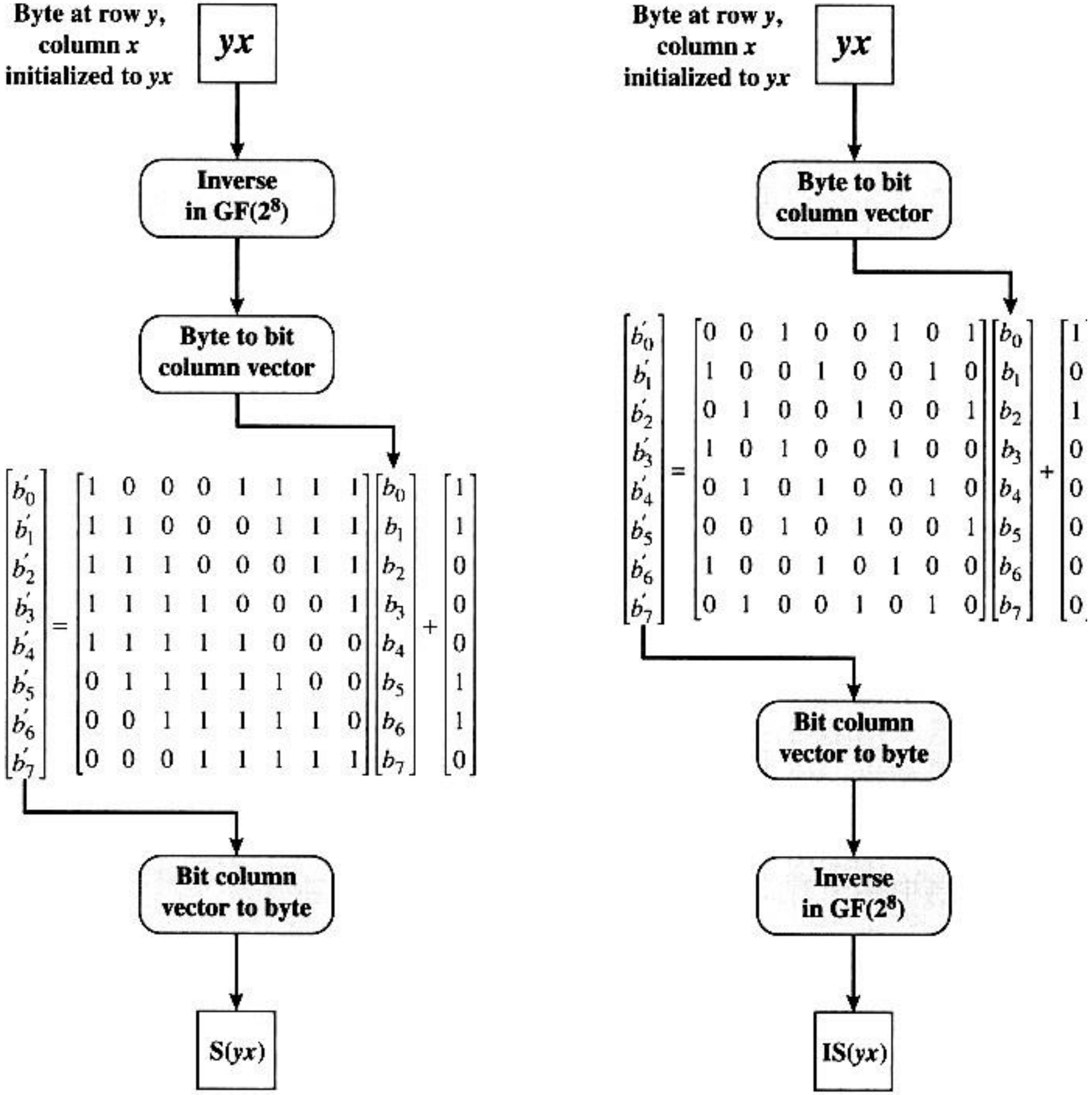
	y																
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	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

(a) S-box

Inverse S-Box for Decoder

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

(b) Inverse S-box



(a) Calculation of byte at row *y*, column *x* of S-box

(a) Calculation of byte at row *y*, column *x* of IS-box

Figure 5.6 Constuction of S-Box and IS-Box

Global Constants and Variables

```
static final int numberOfBits = 8;
static final int fieldSize = 1 << numberOfBits;
static final int irreducible = 0x11b;
static final int logBase = 3;
static final byte[][] A = new byte[][] {
            {1, 1, 1, 1, 1, 0, 0, 0},
            {0, 1, 1, 1, 1, 1, 0, 0},
            {0, 0, 1, 1, 1, 1, 1, 0},
            {0, 0, 0, 1, 1, 1, 1, 1},
            {1, 0, 0, 0, 1, 1, 1, 1},
            {1, 1, 0, 0, 0, 1, 1, 1},
            {1, 1, 1, 0, 0, 0, 1, 1},
            {1, 1, 1, 1, 0, 0, 0, 1}
     };
static final byte[] B = new byte[] { 0, 1, 1, 0, 0, 0, 1, 1};
int[] alog = new int[fieldSize];
int[] log = new int[fieldSize];
int[] S = new int[fieldSize];
```

Making the S-Box

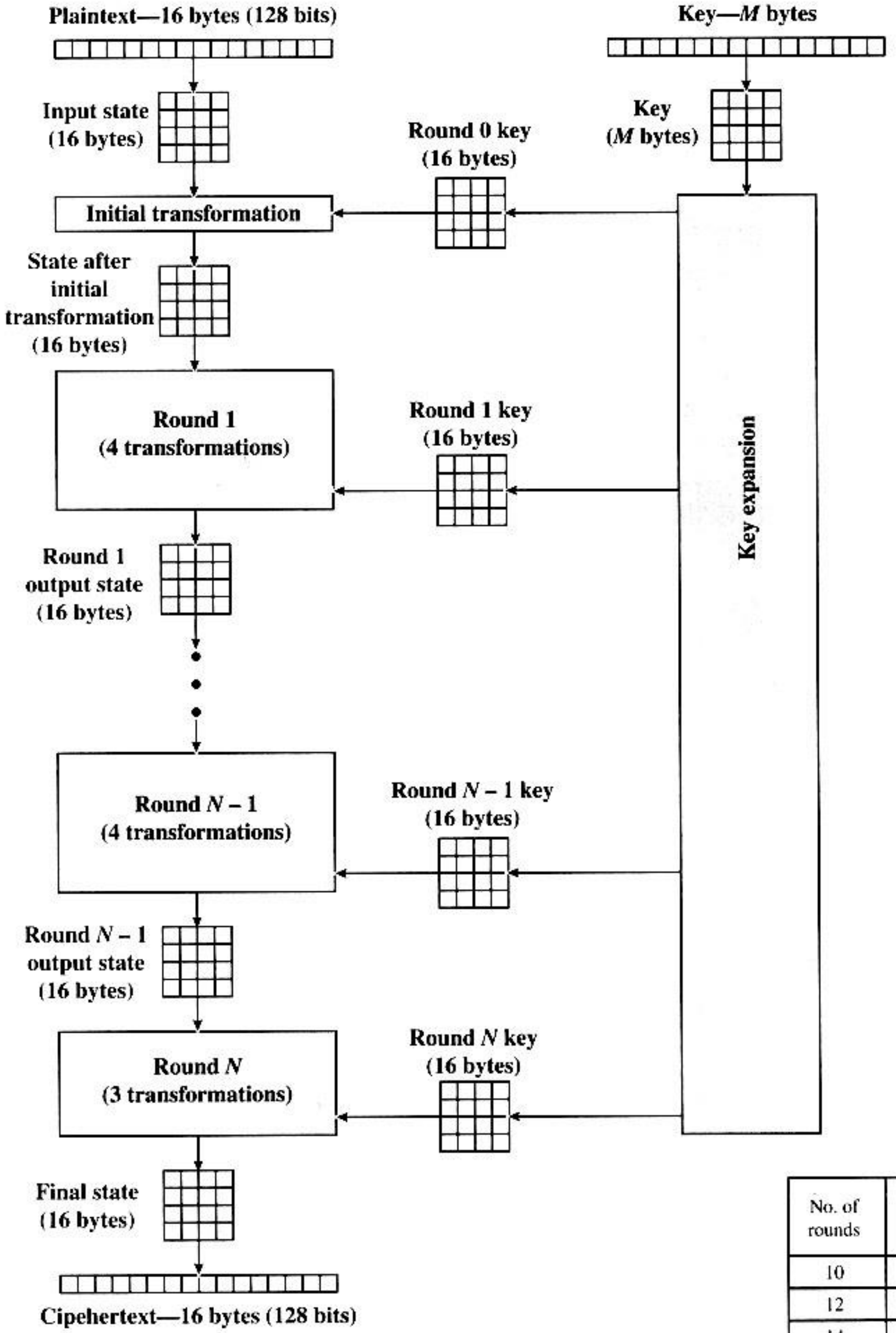
```
void buildS(){
     int[] bitColumn = new int[8];
     for (int i = 0; i < fieldSize; i++){
       int inverse = i < 2 ? i : multiplicativeInverse(i);
       for (int k = 0; k < 8; k++)
           bitColumn[k] = inverse >> (7 - k) & 1;
       S[i] = 0;
       for (int k = 0; k < 8; k++){
          int bit = B[k];
          for (int l = 0; l < 8; l++)
            if (bitColumn[l] == 1) bit ^= A[k][l];
          S[i] ^= bit << 7 - k;
       }
   }
```

Linear Code

- One step in building the S box is $b' = Ab + B$. b', b, and B are vectors in Z_2 and A is a matrix in Z_2.
- The inverse is $b = A'b' + B'$. Substituting $b' = Ab + B$, we have $b = A'Ab + A'B + B'$. The solution is $A'A = I$ and $B' = A'B$.

Advanced Encryption Standard (AES)

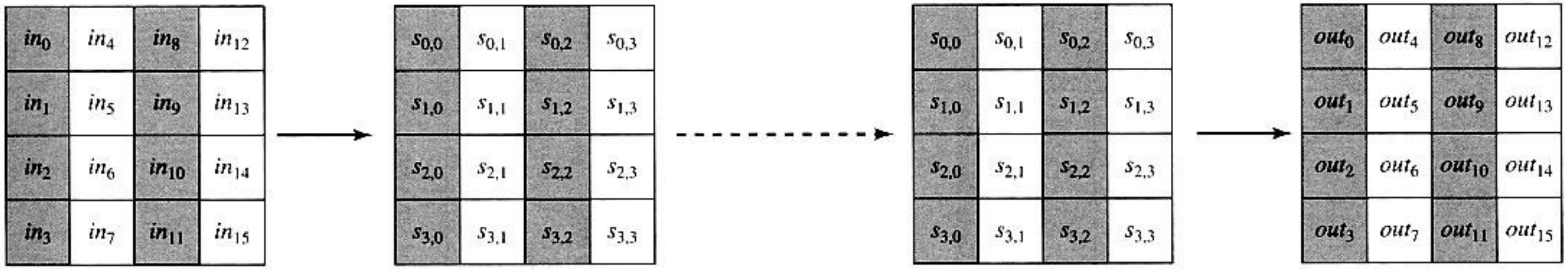
- Adopted and published by NIST 12/4/2001
- Selected from 15 candidate ciphers
- Invented by Belgians Daemen and Rijmen
- Also named Rijindael
- Plaintext/ciphertext element: 16-byte blocks



No. of rounds	Key Length (bytes)
10	16
12	24
14	32

Figure 5.1 AES Encryption Process

The 16-Byte Block and Key



(a) Input, state array, and output

(b) Key and expanded key

Figure 5.2 AES Data Structures

Four Transformations and Iteration

- AddRoundKey (A)
- SubBytes (B)
- ShiftRow (S)
- MixColumn (M)
- ABSMABSMA…BSMABSA
- Initiation: A
- 10 rounds of BSMA
- Final round: BSA

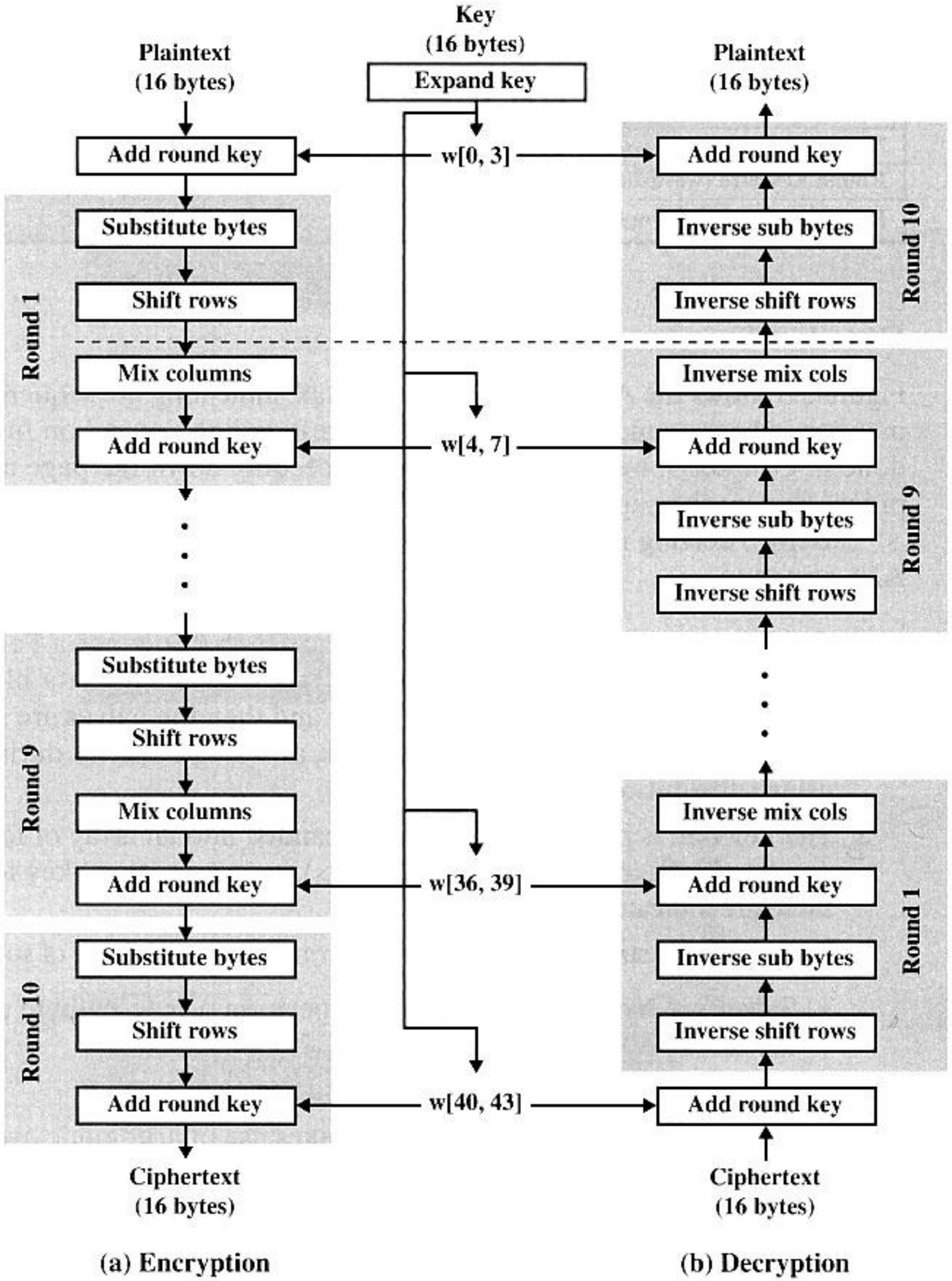


Figure 5.3 AES Encryption and Decryption

```
static final int blockSize = 16;
int[] state = new int[blockSize];

int readBlock(){
 byte[] data = new byte[blockSize];
 int len = 0;
 try {
   len = System.in.read(data);
 } catch (IOException e){
   System.err.println(e.getMessage());
   System.exit(1);
 }
 if (len <= 0) return len;
 for (int i = 0; i < len; i++){
   if (data[i] < 0) state[i] = data[i] + fieldSize;
   else state[i] = data[i];
 }
 for (int i = len; i < blockSize; i++) state[i] = 0;
  return len;
}

void subBytes(){
  for (int i = 0; i < blockSize; i++) state[i] = S[state[i]];
}
```

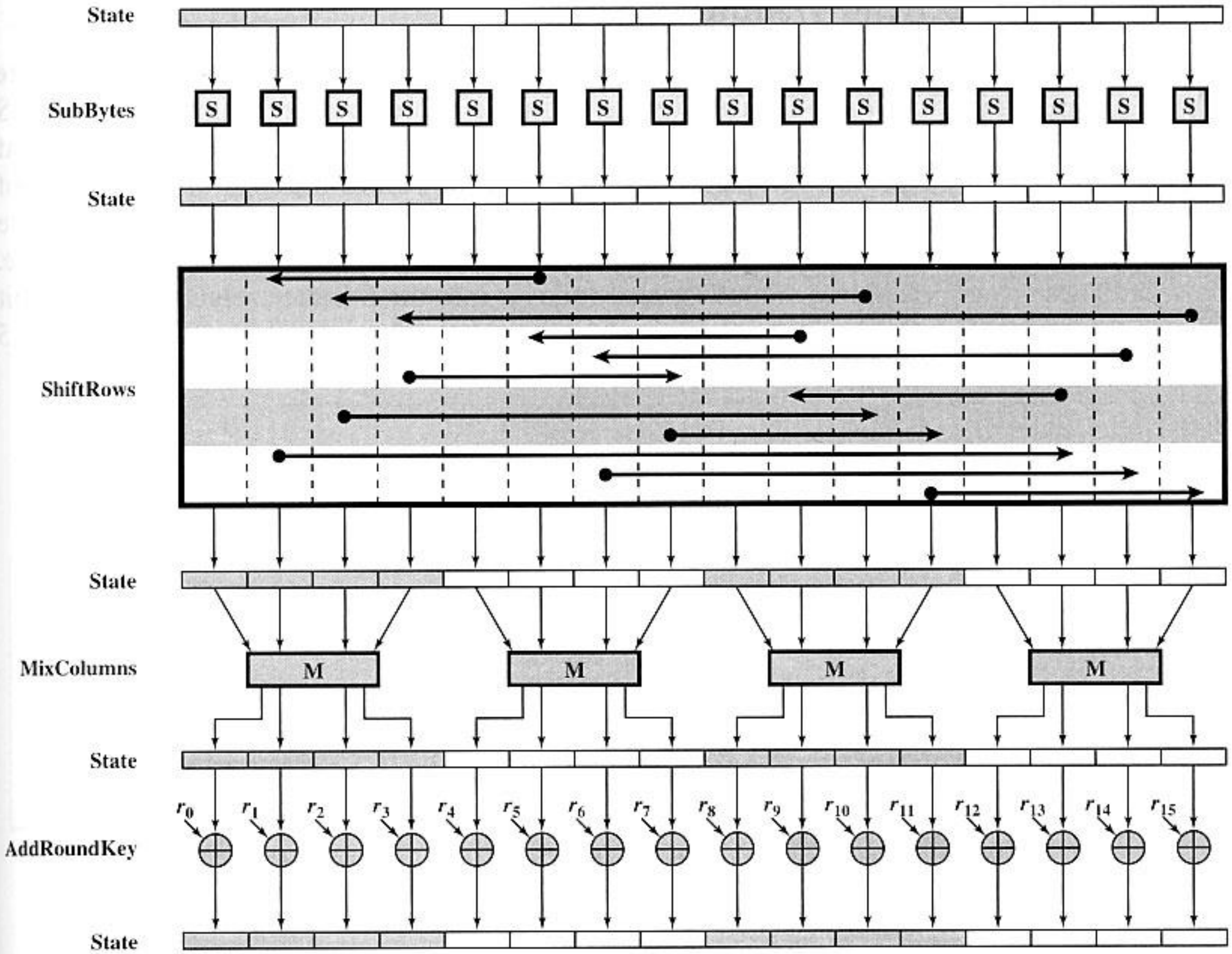


Figure 5.4 AES Encryption Round

Example of SubBytes

EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5

→

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

Example of ShiftRows

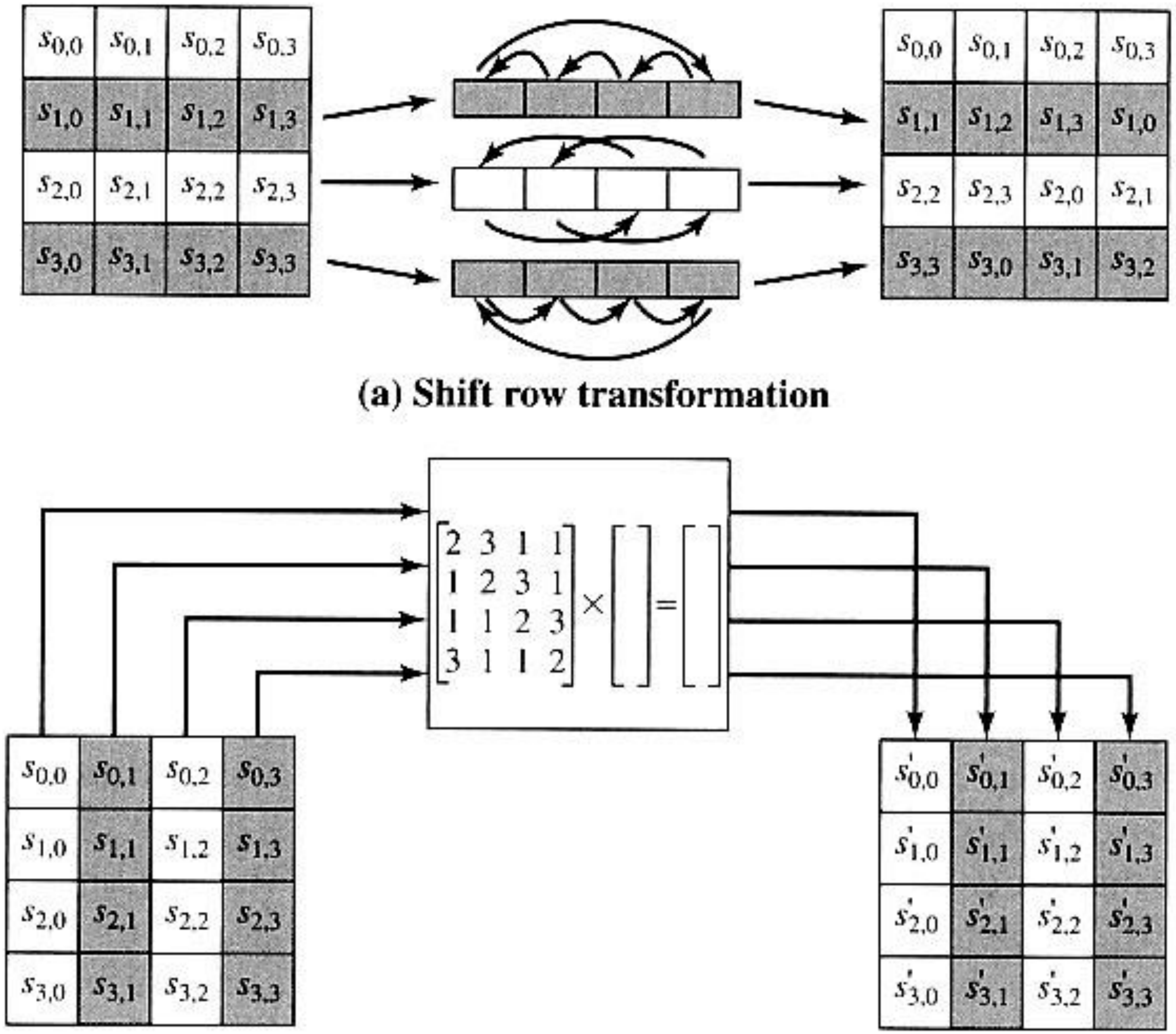
87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

Shift Rows

```
void shiftRows(){
  int temp = state[2]; state[2] = state[10]; state[10] = temp;
  temp = state[6]; state[6] = state[14]; state[14] = temp;
  temp = state[1]; state[1] = state[5]; state[5] = state[9];
  state[9] = state[13]; state[13] = temp;
  temp = state[3]; state[3] = state[15]; state[15] = state[11];
  state[11] = state[7]; state[7] = temp;
}
```



(b) Mix column transformation

Figure 5.7 AES Row and Column Operations

MixColumns

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

MixColumns

$$s'_{0,j} = (2 \bullet s_{0,j}) \oplus (3 \bullet s_{1,j}) \oplus s_{2,j} \oplus s_{3,j}$$

$$s'_{1,j} = s_{0,j} \oplus (2 \bullet s_{1,j}) \oplus (3 \bullet s_{2,j}) \oplus s_{3,j}$$

$$s'_{2,j} = s_{0,j} \oplus s_{1,j} \oplus (2 \bullet s_{2,j}) \oplus (3 \bullet s_{3,j})$$

$$s'_{3,j} = (3 \bullet s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \bullet s_{3,j})$$

The following is an example of MixColumns:

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

→

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

mixColumns

```
static final byte[][] G = new byte[][] {
            {2, 1, 1, 3},
            {3, 2, 1, 1},
            {1, 3, 2, 1},
            {1, 1, 3, 2}
        };

void mixColumns(){
  int[] temp = new int[4];
  for (int k = 0; k < 4; k++){
   for (int i = 0; i < 4; i++){
     temp[i] = 0;
     for (int j = 0; j < 4; j++)
       temp[i] ^= logMultiply(G[j][i], state[k * 4 + j]);
   }
   for (int i = 0; i < 4; i++) state[k * 4 + i] = temp[i];
  }
 }
```

AddRoundKey (A)

- There are 11 roundKeys from expansion of the 128-bit crypt key
- Each roundKey is byte[16]
- AddRoundKey bitwise XORs the roundKey to the block as byte[16].
- AddRoundKey is its own inverse.

```
void addRoundKey(int round){
   for (int k = 0; k < blockSize; k++)
      state[k] ^= roundKey[round][k];
}
```

AddRoundKey

$$\begin{array}{|c|c|c|c|}\hline 47 & 40 & A3 & 4C \\ \hline 37 & D4 & 70 & 9F \\ \hline 94 & E4 & 3A & 42 \\ \hline ED & A5 & A6 & BC \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|}\hline AC & 19 & 28 & 57 \\ \hline 77 & FA & D1 & 5C \\ \hline 66 & DC & 29 & 00 \\ \hline F3 & 21 & 41 & 6A \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline EB & 59 & 8B & 1B \\ \hline 40 & 2E & A1 & C3 \\ \hline F2 & 38 & 13 & 42 \\ \hline 1E & 84 & E7 & D6 \\ \hline \end{array}$$

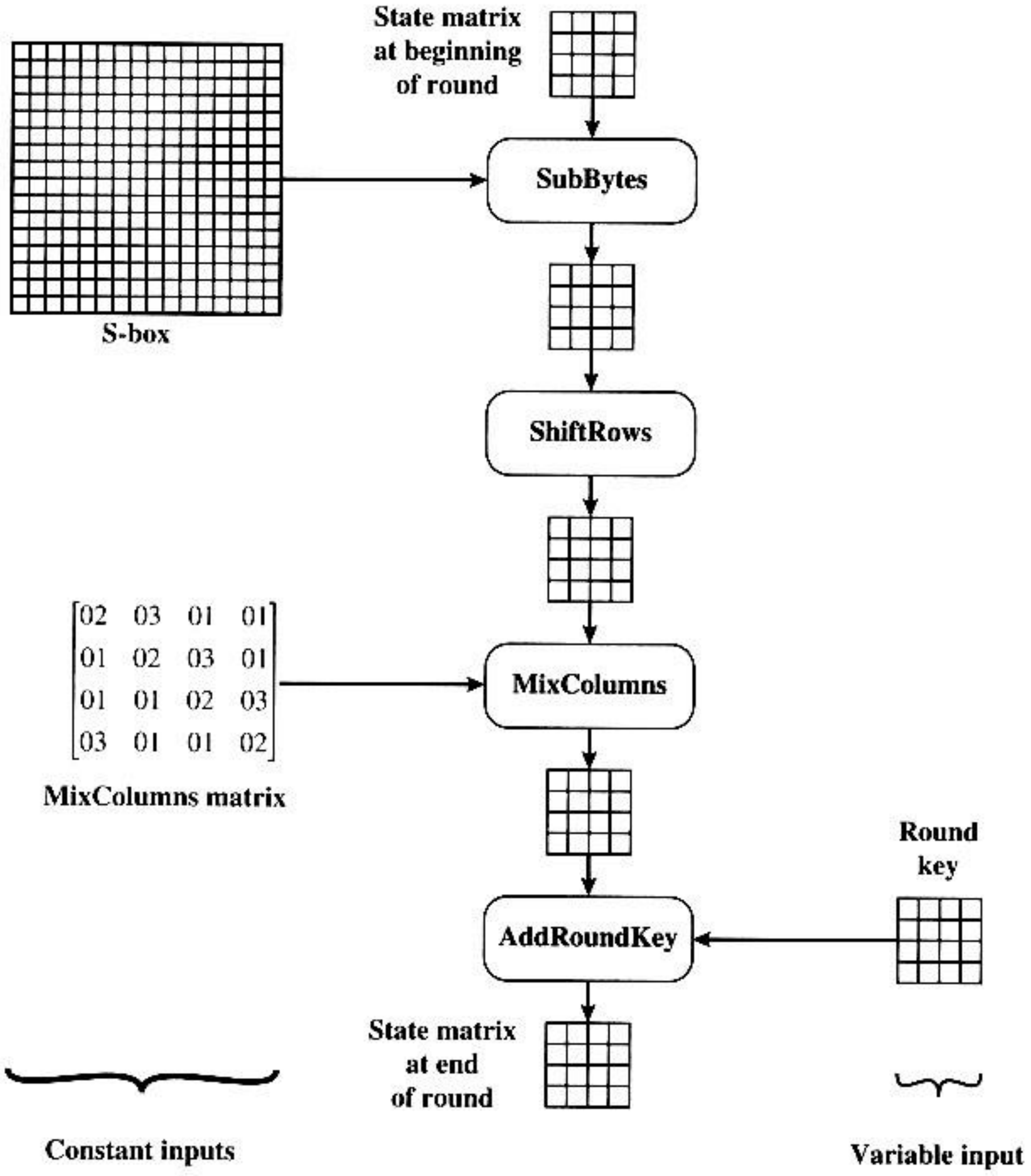
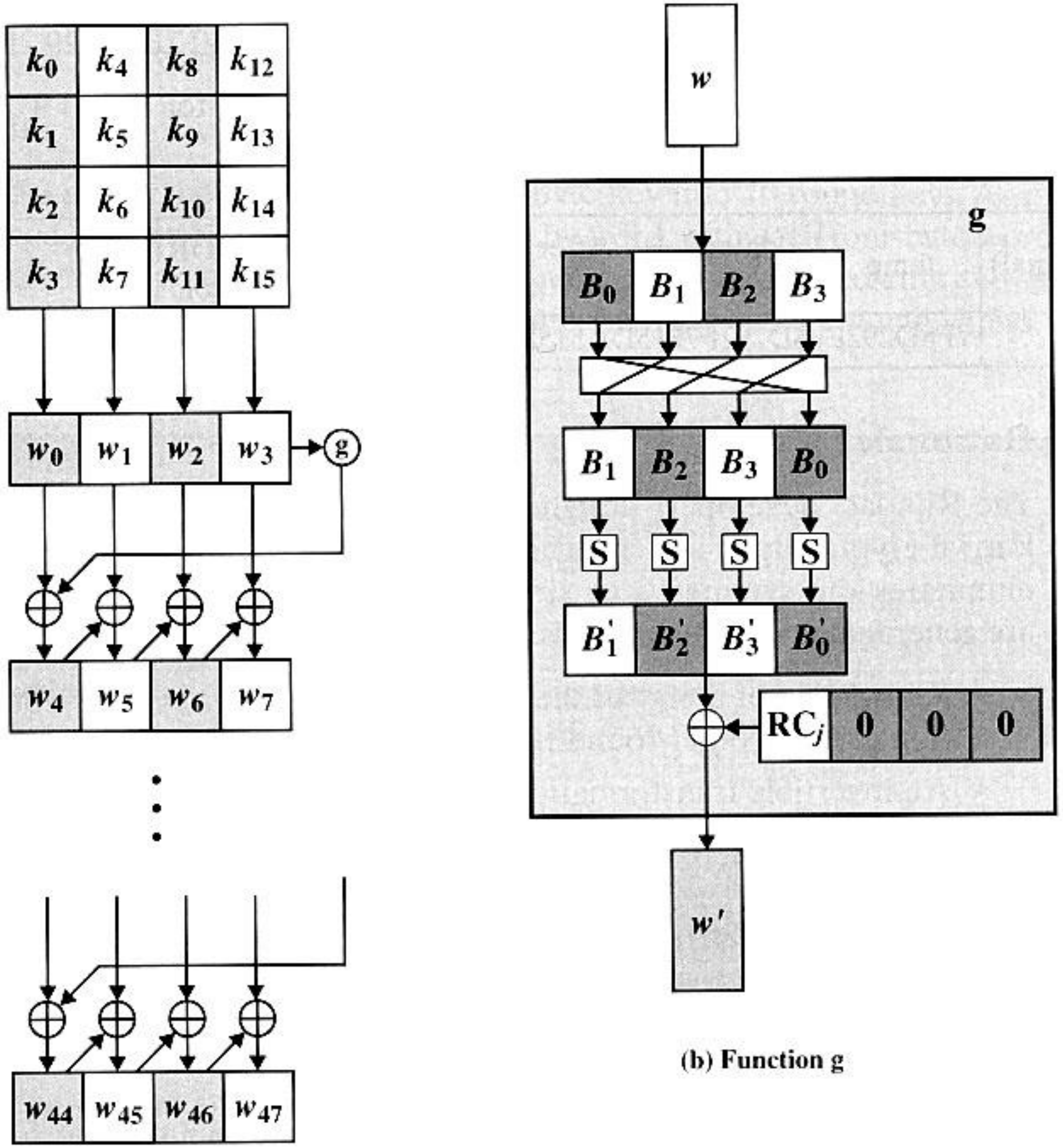


Figure 5.8 Inputs for Single AES Round

readKey and stallingskey.txt

```
void readKey(String filename){
   Scanner in = null;
   try {
     in = new Scanner(new File(filename));
   } catch (FileNotFoundException e){
     System.err.println(filename + " not found");
     System.exit(1);
   }
   hexkey = in.nextLine();
   in.close();
 }
```

```
0f1571c947d9e8590cb7add6af7f6798
```



(b) Function g

(a) Overall algorithm

Figure 5.9 AES Key Expansion

Key Expansion

1. RotWord performs a one-byte circular left shift on a word. This means that an input word $[B_0, B_1, B_2, B_3]$ is transformed into $[B_1, B_2, B_3, B_0]$.
2. SubWord performs a byte substitution on each byte of its input word, using the S-box (Table 5.2a).
3. The result of steps 1 and 2 is XORed with a round constant, Rcon[j].

The round constant is a word in which the three rightmost bytes are always 0. Thus, the effect of an XOR of a word with Rcon is to only perform an XOR on the left-most byte of the word. The round constant is different for each round and is defined as $Rcon[j] = (RC[j], 0, 0, 0)$, with $RC[1] = 1$, $RC[j] = 2 \cdot RC[j-1]$ and with multiplication defined over the field $GF(2^8)$. The values of RC[j] in hexadecimal are

j	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36

expandKey

```
void expandKey(){
   for (int i = 0; i < 16; i++) roundKey[0][i] =
     Integer.parseInt(hexkey.substring(i * 2, (i + 1) * 2), 16);
   int rcon = 1;
   for (int i = 1; i < numberOfRounds; i++){
     roundKey[i][0] = S[roundKey[i-1][13]] ^ rcon;
     rcon <<= 1; if (rcon > 0xFF) rcon ^= 0x11B;
     roundKey[i][1] = S[roundKey[i-1][14]];
     roundKey[i][2] = S[roundKey[i-1][15]];
     roundKey[i][3] = S[roundKey[i-1][12]];
     for (int k = 0; k < 4; k++)
        roundKey[i][k] ^= roundKey[i-1][k];
     for (int k = 4; k < 16; k++)
        roundKey[i][k] = roundKey[i][k-4] ^ roundKey[i-1][k];
   }
}
```

AES Block Cipher

```
void blockCipher(){
   addRoundKey(0);
   for (int i = 1; i < numberOfRounds; i++){
     subBytes();
     shiftRows();
     if (i < numberOfRounds - 1) mixColumns();
     addRoundKey(i);
   }
 }
```

Main and encrypt()

```
void encrypt(){
    while (readBlock() > 0){
      blockCipher();
      writeBlock();
    }
    System.out.flush();
  }
```

```
public static void main(String[] args){
   H11A h11 = new H11A();
   h11.makeLog();
   h11.buildS();
   h11.readKey(args[0]);
   h11.expandKey();
   h11.encrypt();
}
```

Homework 11: due 3-2-15

- Complete the “inverse” versions of the steps in H11B.java so H11B is the inverse of H11A.
- Test your H11B with stallingskey.txt on test11.aes and get the original file to submit along with your code.

```
void decrypt(){
   while (readBlock() > 0){
     blockDecipher();
     writeBlock();
   }
   System.out.flush();
 }


public static void main(String[] args){
   if (args.length < 1){
     System.err.println("Usage: java H11B key < encrypted > original");
     return;
   }
   H11B h11 = new H11B();
   h11.makeLog();
   h11.buildS();
   h11.readKey(args[0]);
   h11.expandKey();
   h11.decrypt();
}
```

blockDecipher and inverseAddRoundKey

```
void inverseAddRoundKey(int round){
   for (int k = 0; k < blockSize; k++)
      state[k] ^= roundKey[?][k];  // which roundkey?
}

void blockDecipher(){
   inverseAddRoundKey(0);
   for (int i = 1; i < numberOfRounds; i++){
      inverseSubBytes();
      inverseShiftRows();
      inverseAddRoundKey(i);
      if (i < numberOfRounds - 1) inverseMixColumns();
   }
}
```

```
void inverseSubBytes(){
   for (int i = 0; i < blockSize; i++)
     state[i] = Si[state[i]];
 }

void inverseShiftRows(){
  // your code here
}

 void inverseMixColumns(){
  int[] temp = new int[4];
  for (int k = 0; k < 4; k++){
   for (int i = 0; i < 4; i++){
     temp[i] = 0;
     for (int j = 0; j < 4; j++)
       temp[i] ^= logMultiply(Gi[j][i], state[k * 4 + j]);
   }
   for (int i = 0; i < 4; i++) state[k * 4 + i] = temp[i];
  }
 }
```

Building Si Table too

```
void buildS(){
     int[] bitColumn = new int[8];
     for (int i = 0; i < fieldSize; i++){
       int inverse = i < 2 ? i : multiplicativeInverse(i);
       for (int k = 0; k < 8; k++)
           bitColumn[k] = inverse >> (7 - k) & 1;
       S[i] = 0;
       for (int k = 0; k < 8; k++){
          int bit = B[k];
          for (int l = 0; l < 8; l++)
            if (bitColumn[l] == 1) bit ^= A[k][l];
          S[i] ^= bit << 7 - k;
       }
       Si[S[i]] = i;
   }
 }
```

Propagation of a Single-Bit Flip (H11C.java)

```
void randomBlock(){
   for (int i = 0; i < blockSize; i++)
      state2[i] = state1[i] = random.nextInt(fieldSize);
   int index = random.nextInt(blockSize);
   state2[index]++;  if (state2[index] >= fieldSize) state2[index] = 0;
 }
```

Blockcipher on Two Blocks

```
void blockCipher(){
   addRoundKey(state1, 0); addRoundKey(state2, 0);
   System.out.print("0 "); numberOfFlips();
   for (int i = 1; i < numberOfRounds; i++){
     subBytes(state1);  subBytes(state2);
     System.out.print(i + " "); numberOfFlips();
     shiftRows(state1); shiftRows(state2);
     System.out.print(i + " "); numberOfFlips();
     if (i < numberOfRounds - 1){ mixColumns(state1); mixColumns(state2); }
     System.out.print(i + " "); numberOfFlips();
     addRoundKey(state1, i);  addRoundKey(state2, i);
   }
 }
```

Comparing Two States

```
void numberOfFlips(){
  int total = 0;
  for (int i = 0; i < blockSize; i++){
    int diff = state1[i] ^ state2[i];
    int nf = 0;
    for (int mask = 0x80; mask > 0; mask >>= 1)
      if ((diff & mask) > 0) nf++;
    total += nf;
    System.out.print(nf + " ");
  }
  System.out.println(total);
}
```

0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	4
1	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	4
1	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	4
1	0	0	0	0	0	0	0	0	4	4	4	4	0	0	0	0	16
2	0	0	0	0	0	0	0	0	5	2	5	6	0	0	0	0	18
2	0	0	5	0	0	2	0	0	5	0	0	0	0	0	0	6	18
2	5	3	4	5	4	2	2	2	4	5	5	3	6	6	3	7	66
3	5	4	4	3	1	4	3	3	5	4	6	2	7	3	2	5	61
3	5	4	6	5	1	4	2	3	5	3	4	3	7	4	3	2	61
3	5	0	5	4	2	4	3	5	7	4	4	4	4	7	4	3	65
4	6	0	3	6	4	1	1	6	2	5	5	6	4	5	4	3	61
4	6	1	5	3	4	5	4	6	2	5	3	6	4	0	1	6	61
4	5	6	1	3	7	5	1	6	2	5	4	5	5	4	4	4	67
5	5	4	5	4	6	5	5	5	5	3	3	3	3	6	6	4	72
5	5	5	3	4	6	3	6	4	5	6	5	5	3	4	5	3	72
5	3	3	2	7	3	4	3	5	3	4	4	2	2	3	4	4	56
6	5	4	4	2	6	1	4	3	5	4	4	7	3	5	4	5	66
6	5	1	4	5	6	4	4	2	5	5	4	3	3	4	4	7	66
6	6	4	5	6	5	4	3	2	3	2	6	4	4	6	4	0	64

7	4	4	2	3	3	3	3	3	4	2	4	4	3	6	3	0	51
7	4	3	4	0	3	2	3	3	4	6	2	3	3	4	3	4	51
7	6	5	2	6	2	3	3	3	5	4	5	3	7	6	4	5	69
8	5	2	2	6	5	4	6	4	7	5	4	5	5	5	3	4	72
8	5	4	4	4	5	5	3	6	7	5	2	4	5	2	6	5	72
8	4	5	3	1	6	4	5	6	4	4	6	4	3	6	3	4	68
9	5	3	5	4	5	5	3	4	3	4	4	3	3	5	4	1	61
9	5	5	4	1	5	4	4	4	3	5	5	4	3	3	3	3	61
9	3	4	5	3	6	6	4	5	4	3	3	5	4	4	3	3	65
10	3	3	5	3	6	4	2	5	4	3	5	5	5	1	2	4	60
10	3	4	5	4	6	3	2	3	4	1	5	5	5	3	2	5	60
10	3	4	5	4	6	3	2	3	4	1	5	5	5	3	2	5	60