LECTURE 3

DES and AES

Telecommunication systems department

Lecturer: assistant professor Persikov Anatoliy Valentinovich

DATA ENCRYPTION STANDARD

The algorithm is designed to encipher and decipher blocks of data consisting of 64 bits under control of a 64-bit key. Deciphering must be accomplished by using the same key as for enciphering, but with the schedule of addressing the key bits altered so that the deciphering process is the reverse of the enciphering process. A block to be enciphered is subjected to an initial permutation IP, then to a complex key-dependent computation and finally to a permutation which is the inverse of the initial permutation IP^{-1}. The key-dependent computation can be simply defined in terms of a function f, called the **cipher function**, and a function KS, called the **key schedule**. A description of the computation is given first, along with details as to how the algorithm is used for encipherment. Next, the use of the algorithm for decipherment is described. Finally, a definition of the cipher function f is given in terms of primitive functions which are called the **selection functions** S_i and the **permutation function** P.

The following notation is convenient: Given two blocks L and R of bits, LR denotes the block consisting of the bits of L followed by the bits of R. Since concatenation is associative, $B_1B_2 \ldots B_8$, for example, denotes the block consisting of the bits of B_1 followed by the bits of B_2...followed by the bits of B_8.

DES ALGORITHM ENCIPHERING

The 64 bits of the input block to be enciphered are first subjected to the following permutation, called the initial permutation IP:

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

That is the permuted input has bit 58 of the input as its first bit, bit 50 as its second bit, and so on with bit 7 as its last bit. The permuted input block is then the input to a complex key-dependent computation described below.

DES ALGORITHM ENCIPHERING

The computation which uses the permuted input block as its input to produce the pre-output block consists, but for a final interchange of blocks, of 16 iterations of a calculation that is described below in terms of the **cipher function** f which operates on two blocks, one of 32 bits and one of 48 bits, and produces a block of 32 bits.

Let the 64 bits of the input block to an iteration consist of a 32 bit block L followed by a 32 bit block R. Using the notation defined in the introduction, the input block is then LR.

Let K be a block of 48 bits chosen from the 64-bit key. Then the output $L'R'$ of an iteration with input LR is defined by:

$$L' = R$$
$$R' = L \oplus f(R, K) \qquad (1)$$

where $\oplus$ denotes bit-by-bit addition modulo 2.

As remarked before, the input of the first iteration of the calculation is the **permuted input block**. If $L'R'$ is the output of the 16th iteration then $R'L'$ is the pre-output block. At each iteration a different block K of key bits is chosen from the 64-bit key designated by KEY.

DES ALGORITHM ENCIPHERING

With more notations we can describe the iterations of the computation in more detail. Let KS be a function which takes an integer n in the range from 1 to 16 and a 64-bit block KEY as input and yields as output a 48-bit block K_n which is a permuted selection of bits from KEY.
That is

$$K_n = KS(n, KEY) \tag{2}$$

with K_n determined by the bits in 48 distinct bit positions of KEY. KS is called the **key schedule** because the block K used in the n'th iteration of (1) is the block K_n determined by (2).

As before, let the permuted input block be LR. Finally, let L_0 and R_0 be respectively L and R and let L_n and R_n be respectively L' and R' of (1) when L and R are respectively L_{n-1} and R_{n-1} and K is K_n; that is, when n is in the range from 1 to 16,

$$\begin{aligned} L_n &= R_{n-1} \\ R_n &= L_{n-1} \oplus f(R_{n-1}, K_n) \end{aligned} \tag{3}$$

The pre-output block is then $R_{16}L_{16}$. The key schedule produces the 16 K_n which are required for the algorithm.

DES ALGORITHM ENCIPHERING

The output of that computation, called the pre-output, is then subjected to the following permutation which is the inverse of the initial permutation IP^{-1}:

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

That is, the output of the algorithm has bit 40 of the pre-output block as its first bit, bit 8 as its second bit, and so on, until bit 25 of the pre-output block is the last bit of the output.

DES ALGORITHM DECIPHERING

The permutation IP^{-1} applied to the preoutput block is the inverse of the initial permutation IP applied to the input. Further, from (1) it follows that:

$$\begin{aligned} R &= L' \\ L &= R' \oplus f(L', K) \end{aligned} \tag{4}$$

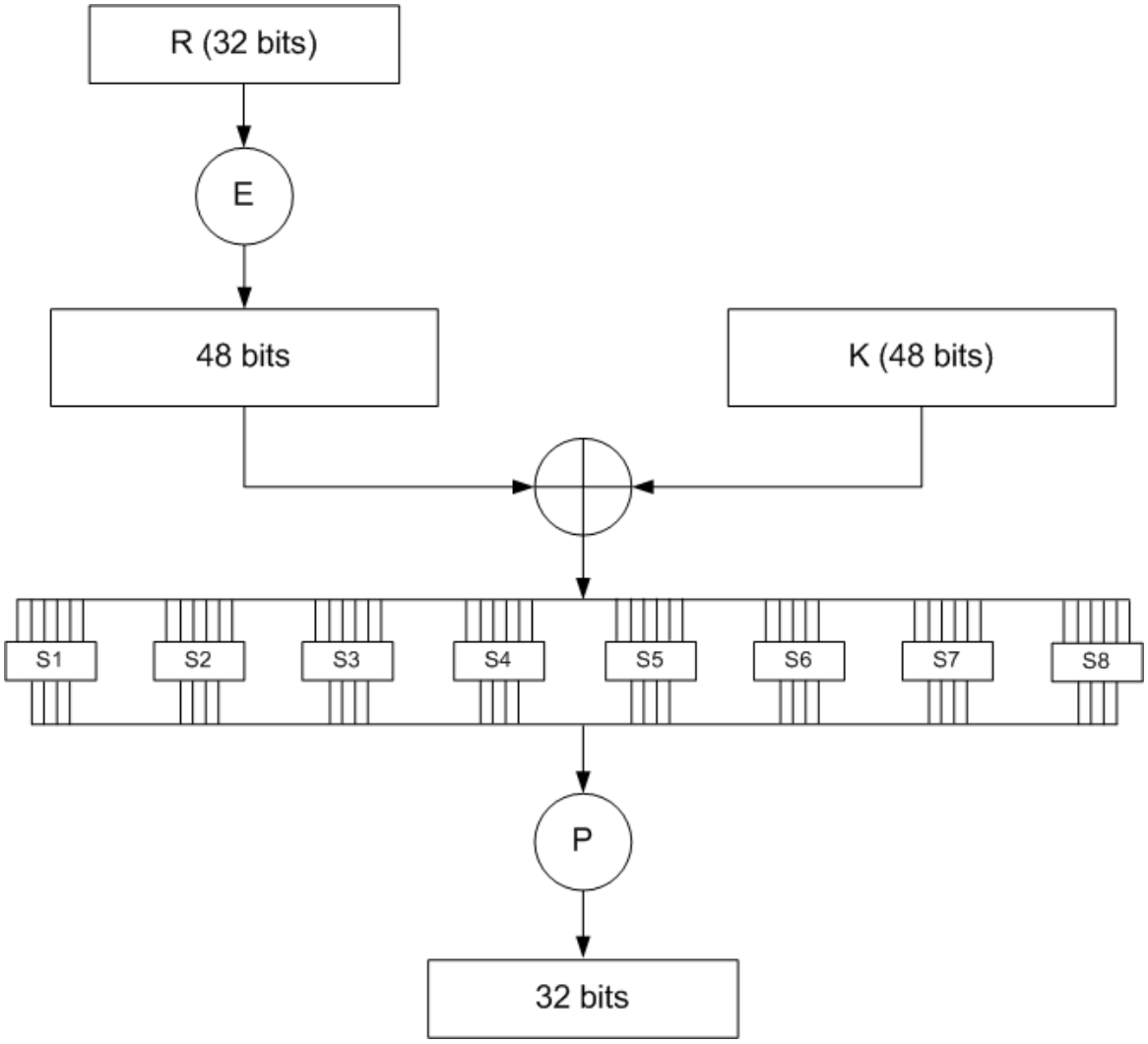
Consequently, to decipher it is only necessary to apply the very same algorithm to an enciphered message block, taking care that at each iteration of the computation the same block of key bits K is used during decipherment as was used during the encipherment of the block.

Using the notation of the previous section, this can be expressed by the equations:

$$\begin{aligned} R_{n-1} &= L_n \\ L_{n-1} &= R_n \oplus f(L_n, K_n) \end{aligned} \tag{5}$$

where now $R_{16}L_{16}$ is the permuted input block for the deciphering calculation and L_0R_0 is the pre-output block. That is, for the decipherment calculation with $R_{16}L_{16}$ as the permuted input, K_{16} is used in the first iteration, K_{15} in the second, and so on, with K_1 used in the 16th iteration.

DES CIPHER FUNCTION



DES CIPHER FUNCTION

Let E denote a function which takes a block of 32 bits as input and yields a block of 48 bits as output. Let E be such that the 48 bits of its output, written as 8 blocks of 6 bits each, are obtained by selecting the bits in its inputs in order according to the following table:

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Thus the first three bits of $E(R)$ are the bits in positions 32, 1 and 2 of R while the last 2 bits of $E(R)$ are the bits in positions 32 and 1.

DES CIPHER FUNCTION

		Column																
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Row	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	S0
	1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	
	2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	
	3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
	0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	S1
	1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	
	2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	
	3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	
	0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	S2
	1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	
	2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7	
	3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	
	0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	S3
	1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	
	2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	
	3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	
	0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	S4
	1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	
	2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	
	3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3	
	0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11	S5
	1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	
	2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6	
	3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13	
	0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	S6
	1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	
	2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	
	3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12	
	0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	S7
	1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	
	2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	
	3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11	

DES CIPHER FUNCTION

Each of the unique selection functions S_1,S_2,...,S_8, takes a 6-bit block as input and yields a 4-bit block as output and is illustrated by using a table containing the recommended S_1:

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

If S_1 is the function defined in this table and B is a block of 6 bits, then $S_1(B)$ is determined as follows: The first and last bits of B represent in base 2 a number in the range 0 to 3. Let that number be i. The middle 4 bits of B represent in base 2 a number in the range 0 to 15. Let that number be j. Look up in the table the number in the i'th row and j'th column. It is a number in the range 0 to 15 and is uniquely represented by a 4 bit block. That block is the output $S_1(B)$ of S_1 for the input B. For example, for input 011011 the row is 01, that is row 1, and the column is determined by 1101, that is column 13. In row 1 column 13 appears 5 so that the output is 0101.

DES CIPHER FUNCTION

The permutation function P yields a 32-bit output from a 32-bit input by permuting the bits of the input block. Such a function is defined by the following table:

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

The output $P(L)$ for the function P defined by this table is obtained from the input L by taking the 16th bit of L as the first bit of $P(L)$, the 7th bit as the second bit of $P(L)$, and so on until the 25th bit of L is taken as the 32nd bit of $P(L)$.

DES CIPHER FUNCTION

Now let S_1,...,S_8 be eight distinct selection functions, let P be the permutation function and let E be the function defined above.

To define $f(R, K)$ we first define B_1,...,B_8 to be blocks of 6 bits each for which

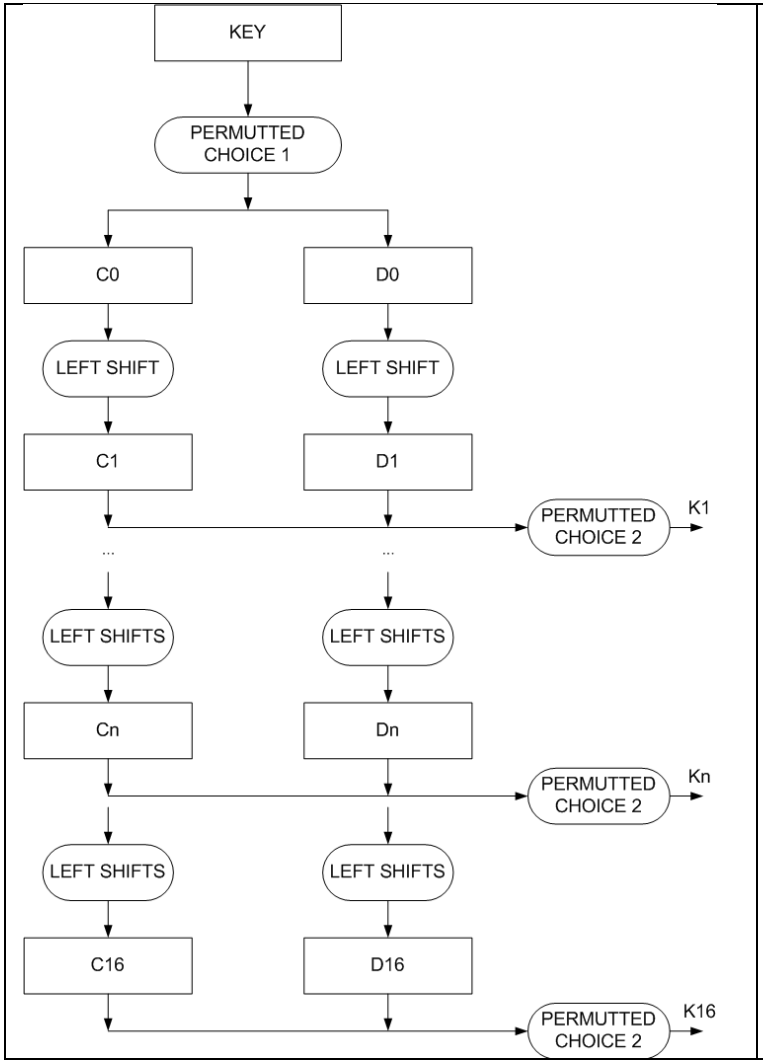
$$B_1 B_2 \ldots B_8 = K \oplus E(R) \tag{6}$$

The block $f(R, K)$ is then defined to be

$$P(S_1(B_1) S_2(B_2) \ldots S_8(B_8)) \tag{7}$$

Thus $K \oplus E(R)$ is first divided into the 8 blocks as indicated in (6). Then each B_i is taken as an input to S_i and the 8 blocks $S_1(B_1)$, $S_2(B_2)$, ..., $S_8(B_8)$ of 4 bits each are consolidated into a single block of 32 bits which forms the input to P. The output (7) is then the output of the function f for the inputs R and K.

DES KEY SCHEDULE



Recall that K_n, for $1 \leq n \leq 16$, is the block of 48 bits in (2) of the algorithm. Hence, to describe KS, it is sufficient to describe the calculation of K_n from KEY for $n = 1, 2, \ldots, 16$.

To complete the definition of KS it is therefore sufficient to describe the two permuted choices, as well as the schedule of left shifts. One bit in each 8-bit byte of the KEY may be utilized for error detection in key generation, distribution and storage. Bits 8, 16,..., 64 are for use in assuring that each byte is of odd parity.

DES KEY SCHEDULE

Permuted choice 1 is determined by the following table:

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

The table has been divided into two parts, with the first part determining how the bits of $C_{()}$ are chosen, and the second part determining how the bits of $D_{()}$ are chosen. The bits of KEY are numbered 1 through 64. The bits of $C_{()}$ are respectively bits 57, 49, 41,..., 44 and 36 of KEY, with the bits of $D_{()}$ being bits 63, 55, 47,..., 12 and 4 of KEY.

DES KEY SCHEDULE

With $C_{()}$ and $D_{()}$ defined, we now define how the blocks C_n and D_n are obtained from the blocks C_{n-1} and D_{n-1}, respectively, for $n = 1, 2, \ldots, 16$. That is accomplished by adhering to the following schedule of left shifts of the individual blocks:

Round number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number of shifts	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

For example, C_3 and D_3 are obtained from C_2 and D_2, respectively, by two left shifts, and C_{16} and D_{16} are obtained from C_{15} and D_{15}, respectively, by one left shift. In all cases, by a single left shift is meant a rotation of the bits one place to the left, so that after one left shift the bits in the 28 positions are the bits that were previously in positions 2, 3,..., 28, 1.

DES KEY SCHEDULE

Permuted choice 2 is determined by the following table:

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Therefore, the first bit of K_n is the 14th bit of $C_n D_n$, the second bit the 17th, and so on with the 47th bit the 29th, and the 48th bit the 32nd.

AES

The Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2002. Originally called Rijndael, the cipher was developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, who submitted to the AES selection process.

AES has been adopted by the U.S. government and is now used worldwide. It supersedes the Data Encryption Standard (DES). The algorithm described by AES is a symmetric-key algorithm, meaning the same key is used for both encrypting and decrypting the data.

All bytes in the AES algorithm are interpreted as finite field elements. Finite field elements can be added and multiplied, but these operations are different from those used for numbers.

ALGORITHM SPECIFICATION

For the AES algorithm, the length of the input block, the output block and the State is 128 bits. This is represented by $Nb = 4$, which reflects the number of 32-bit words (number of columns) in the State.

For the AES algorithm, the length of the Cipher Key, K, is 128, 192, or 256 bits. The key length is represented by Nk = 4, 6, or 8, which reflects the number of 32-bit words (number of columns) in the Cipher Key.

For the AES algorithm, the number of rounds to be performed during the execution of the algorithm is dependent on the key size. The number of rounds is represented by Nr, where Nr = 10 when Nk = 4, Nr = 12 when Nk = 6, and Nr = 14 when Nk = 8.

	Key length (in words)	Block size (in words)	Round number (in words)
AES-128	4	4	10
AES -192	6	4	12
AES -256	8	4	14

ALGORITHM SPECIFICATION

At the start of the Cipher, the input is copied to the State array. After an initial Round Key addition, the State array is transformed by implementing a round function 10, 12, or 14 times (depending on the key length), with the final round differing slightly from the first $Nr - 1$ rounds. The final State is then copied to the output.

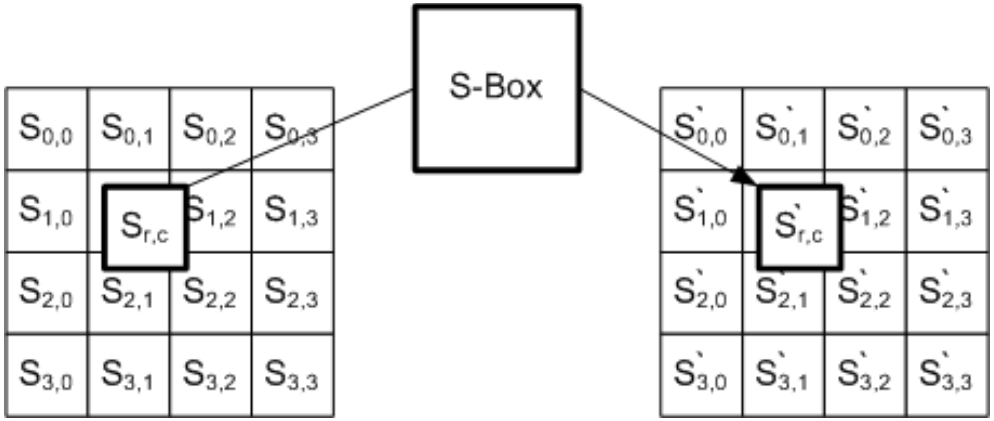
The round function is parameterized using a key schedule that consists of a one-dimensional array of four-byte words derived using the Key Expansion routine. The array w[] contains the key schedule.

ALGORITHM SPECIFICATION

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
  begin
    byte state[4,Nb]
    state = in

    AddRoundKey(state, w[0, Nb-1])

    for round = 1 step 1 to Nr-1
      SubBytes(state)
      ShiftRows(state)
      MixColumns(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    end for

    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
    out = state
  end
```

All Nr rounds are identical with the exception of the final round, which does not include the MixColumns() transformation.

SubBytes() TRANSFORMATION



		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

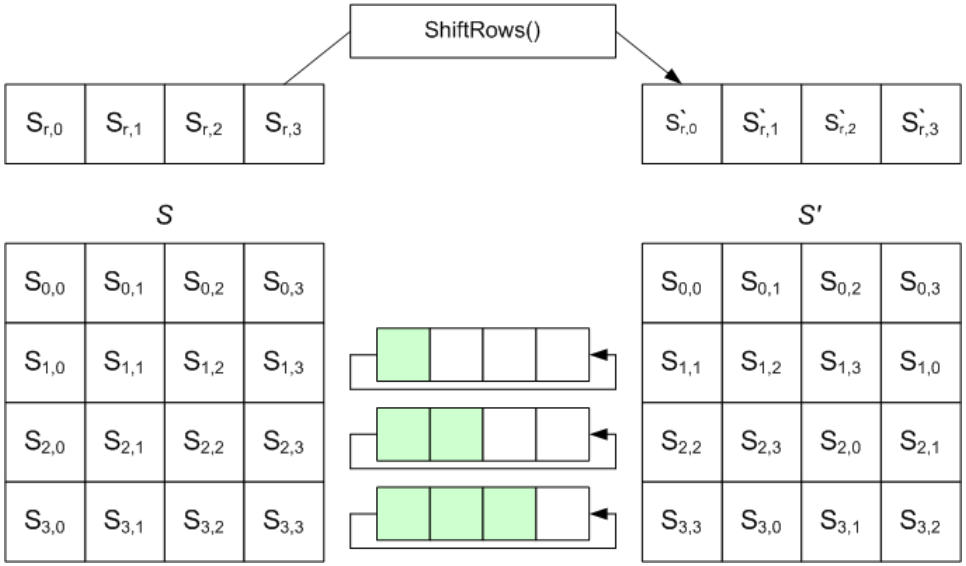
ShiftRows() TRANSFORMATION

In the ShiftRows() transformation, the bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, $r = 0$, is not shifted. Specifically, the ShiftRows() transformation proceeds as follows:

$$s'_{r,c} = s_{r,(c+shift(r,Nb)) \bmod Nb} \text{ for } 0 < r < 4 \text{ and } 0 \leq c < Nb, \qquad (18)$$

where the shift value $shift(r, Nb)$ depends on the row number, r, as follows (recall that $Nb = 4$):

$$shift(1,4) = 1; \quad shift(2,4) = 2; \quad shift(3,4) = 3. \qquad (19)$$



MixColumns() TRANSFORMATION

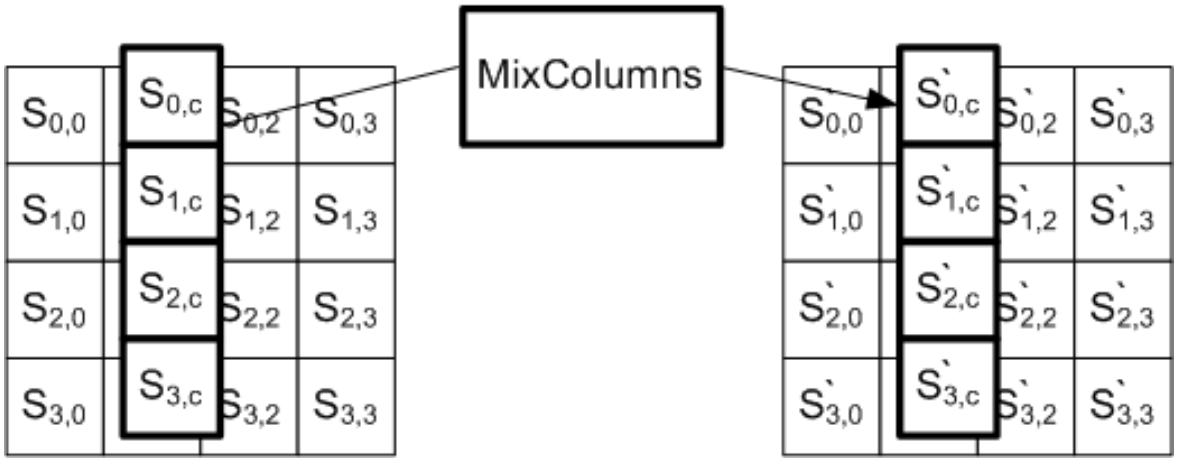
The MixColumns() transformation operates on the State column-by-column, treating each column as a four-term polynomial. The columns are considered as polynomials over $GF(2^8)$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $a(x)$, given by

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}. \tag{20}$$

Let $s'(x) = a(x) \oplus s(x)$:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \leq c < Nb \tag{21}$$

MixColumns() TRANSFORMATION



As a result of this multiplication, the four bytes in a column are replaced by the following:

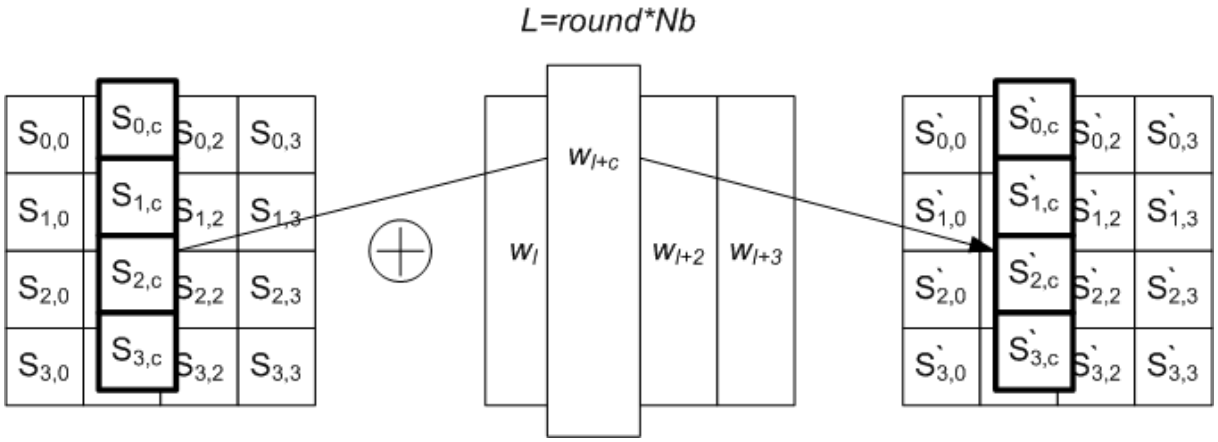
$$s'_{0,c} = (\{02\} \circ s_{0,c}) \oplus (\{03\} \circ s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$
$$s'_{1,c} = s_{0,c} \oplus (\{02\} \circ s_{1,c}) \oplus (\{03\} \circ s_{2,c} \oplus s_{3,c})$$
$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \circ s_{2,c}) \oplus (\{03\} \circ s_{3,c})$$
$$s'_{3,c} = (\{03\} \circ s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \circ s_{3,c})$$

AddRoundKey() TRANSFORMATION

In the AddRoundKey() transformation, a Round Key is added to the State by a simple bitwise XOR operation. Each Round Key consists of Nb words from the key schedule. Those Nb words are each added into the columns of the State, such that

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{round*Nb+c}] \text{ for } 0 \leq c < Nb \qquad (22)$$

where $[w_i]$ are the key schedule words, and $round$ is a value in the range $0 \leq round \leq Nr$.



In the Cipher, the initial Round Key addition occurs when round = 0, prior to the first application of the round function. The application of the AddRoundKey() transformation to the Nr rounds of the Cipher occurs when $1 \leq round \leq Nr$.

In the Cipher, the initial Round Key addition occurs when $round = 0$, prior to the first application of the round function. The application of the AddRoundKey() transformation to the Nr rounds of the Cipher occurs when $1 \leq round \leq Nr$.

KEY EXPANSION

The AES algorithm takes the Cipher Key, K, and performs a Key Expansion routine to generate a key schedule. The Key Expansion generates a total of $Nb\ (Nr\ +\ 1)$ words: the algorithm requires an initial set of Nb words, and each of the Nr rounds requires Nb words of key data. The resulting key schedule consists of a linear array of 4-byte words, denoted $[w_i]$, with i in the range $0 \leq i\ <\ Nb(Nr\ +\ 1)$.

SubWord() is a function that takes a four-byte input word and applies the S-box to each of the four bytes to produce an output word. The function RotWord() takes a word $[a_0, a_1, a_2, a_3]$ as input, performs a cyclic permutation, and returns the word $[a_1, a_2, a_3, a_0]$. The round constant word array, $Rcon[i]$, contains the values given by $[x^{i-1}, \{00\}, \{00\}, \{00\}]$, with x^{i-1} being powers of x (x is denoted as $\{02\}$) in the field $GF(2^8)$ (note that i starts at 1, not 0).

It can be seen that the first Nk words of the expanded key are filled with the Cipher Key. Every following word, $w[i]$, is equal to the XOR of the previous word, $w[i-1]$, and the word Nk positions earlier, $w[i-Nk]$. For words in positions that are a multiple of Nk, a transformation is applied to $w[i-1]$ prior to the XOR, followed by an XOR with a round constant, $Rcon[i]$. This transformation consists of a cyclic shift of the bytes in a word ($RotWord()$), followed by the application of a table lookup to all four bytes of the word ($SubWord()$).

KEY EXPANSION

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word temp

   i = 0

   while (i < Nk)
     w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
     i = i+1
   end while

   i = Nk

   while (i < Nb * (Nr+1)]
     temp = w[i-1]
     if (i mod Nk = 0)
       temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
     else if (Nk > 6 and i mod Nk = 4)
       temp = SubWord(temp)
     end if
     w[i] = w[i-Nk] xor temp
     i = i + 1
   end while
end
```

THANKS FOR ATTENTION