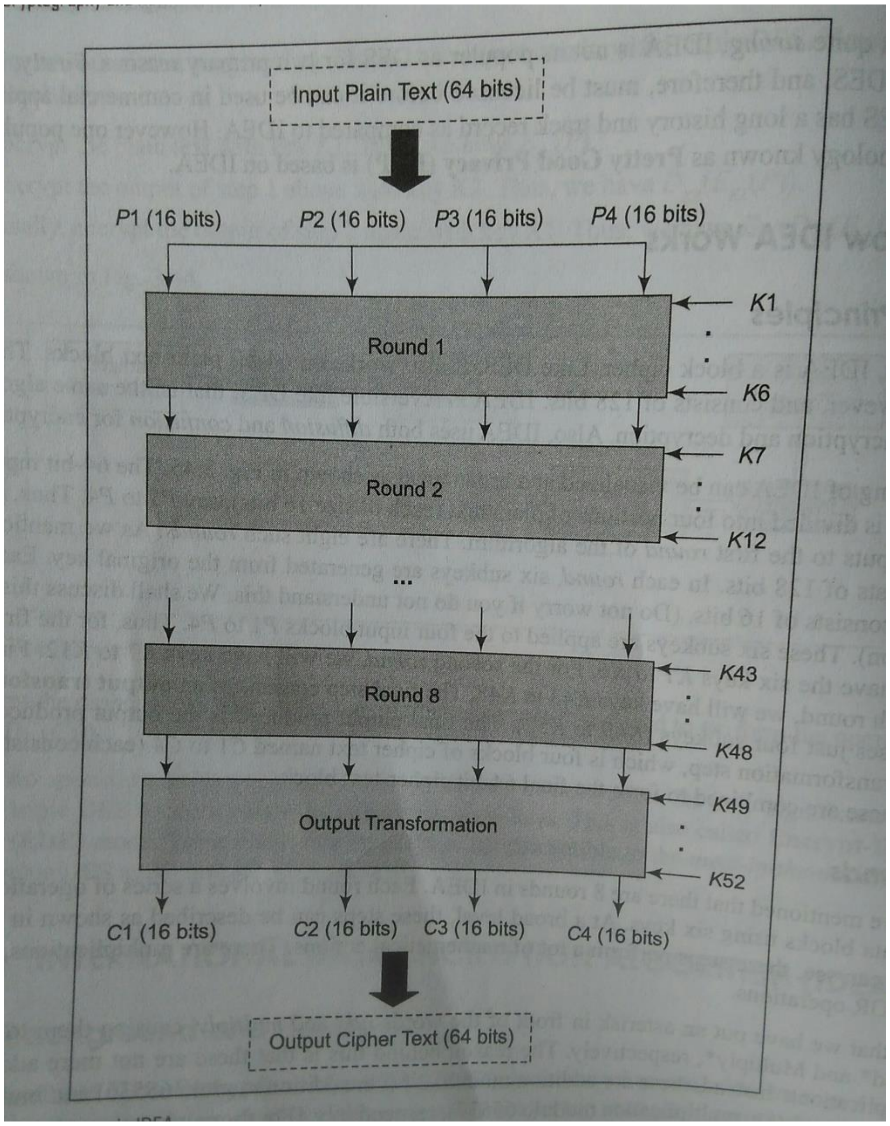
MODULE 2

IDEA: Primitive operations- Key expansions- One round, Odd round, Even Round- Inverse keys for decryption.

AES: Basic Structure- Primitive operation- Inverse Cipher- Key Expansion, Rounds, Inverse Rounds. Stream Cipher –RC4.

INTERNATIONAL DATA ENCRYPTION ALGORITHM (IDEA)

- International Data Encryption Algorithm is one of the strongest cryptographic algorithms launched in 1990..
- IDEA is not as popular as DES for two primary reasons:
  - It is patented unlike DES and therefore must be licensed before it can be used in commercial applications.
  - DES has a long history as compared to IDEA.
- One popular e-mail privacy technology known as pretty good privacy (PGP) is based on IDEA.
- **BASIC PRINCIPLES**
  - IDEA is a block cipher
  - Works on 64 bit plain text blocks.
  - Key consists of 128 bits
  - Same algorithm is used for encryption and decryption



Steps in IDEA

  - 64 bit plain text block is divided into four portions of plain text each of size 16 bits(say P1

to P4).
- P1 to P4 are the inputs to the first round of the algorithm.
- There are eight such rounds.
- The key consists of 128 bits. In each round 6 subkeys are used and after the round operation an output transformation is performed using subkeys(K49 to K52).
- The final output produced is the output produced by the output transformation step which is the four blocks of cipher text named C1 to C4(16 bits each).
- These are combined to form the final 64 bit cipher text block.

❖ **SUB KEY GENERATION**

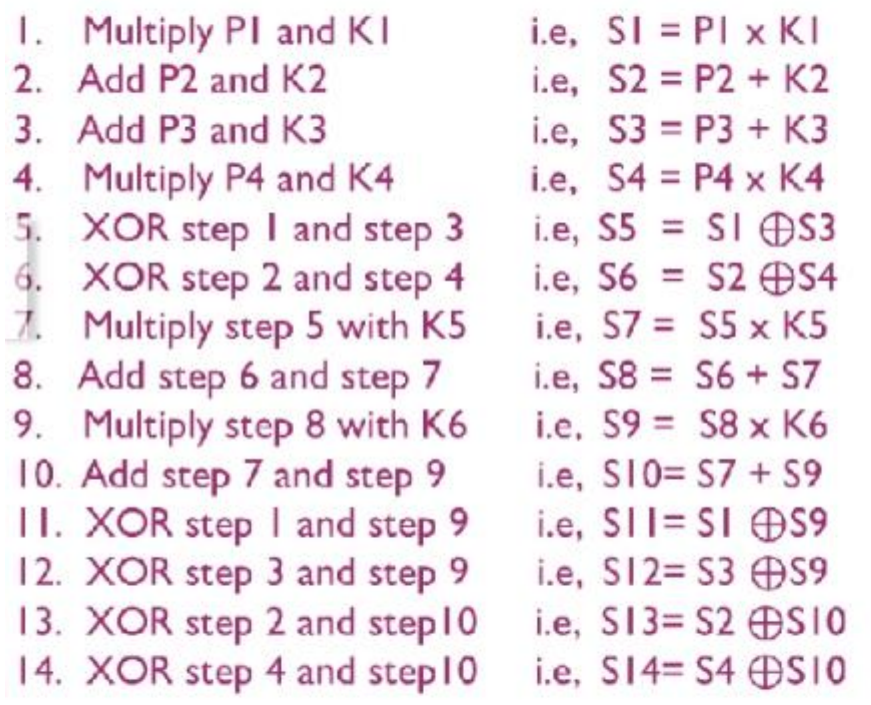
- Each rounds makes use of 6 subkeys (8X6=48) and the final output transformation uses four subkeys(48+4 =52 subkeys).
- 128 bit key is divided into 8 parts K1 to K8 each of 16 bits.
- Of which K1 to K6 is used for ROUND 1 operation.
- For the remaining round subkeys are generated by performing key shifting.
- The orginal key is shifted left circularly by 25 bits.
- And hence 52 subkeys are generated.

❖ **OPERATIONS**

- Operations performed are
  - Addition,
  - Multiplication,
  - XOR

❖ **ROUNDS**

- Each round involves a series of operation on the four data blocks using six keys.

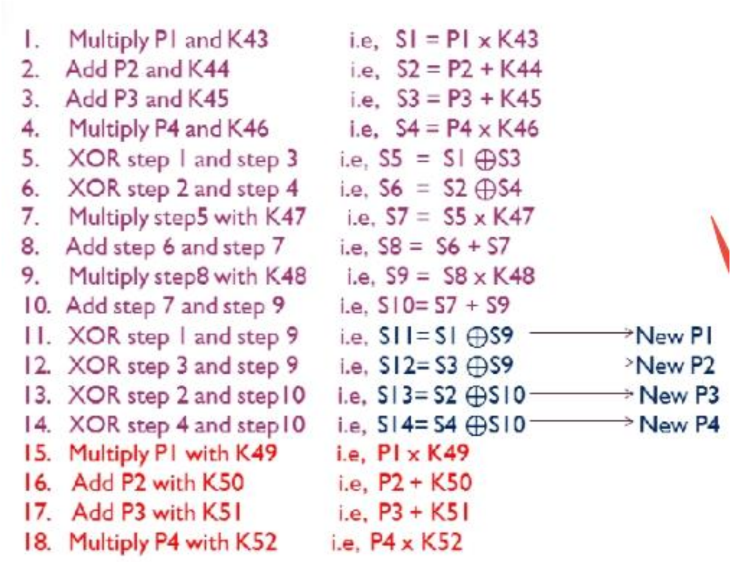


1.	Multiply P1 and K1	i.e, S1 = P1 x K1
2.	Add P2 and K2	i.e, S2 = P2 + K2
3.	Add P3 and K3	i.e, S3 = P3 + K3
4.	Multiply P4 and K4	i.e, S4 = P4 x K4
5.	XOR step 1 and step 3	i.e, S5 = S1 ⊕S3
6.	XOR step 2 and step 4	i.e, S6 = S2 ⊕S4
7.	Multiply step 5 with K5	i.e, S7 = S5 x K5
8.	Add step 6 and step 7	i.e, S8 = S6 + S7
9.	Multiply step 8 with K6	i.e, S9 = S8 x K6
10.	Add step 7 and step 9	i.e, S10= S7 + S9
11.	XOR step 1 and step 9	i.e, S11= S1 ⊕S9
12.	XOR step 3 and step 9	i.e, S12= S3 ⊕S9
13.	XOR step 2 and step10	i.e, S13= S2 ⊕S10
14.	XOR step 4 and step10	i.e, S14= S4 ⊕S10

Details of one ROUND in IDEA

- Similarly same set of operations are performed for each round by changing the key.
- The input blocks for the next round is obtained from the previous round.

- For the next round P1, P2, P3, P4 as **S11, S13, S12, S14** respectively.
- 8th round



1.	Multiply P1 and K43	i.e, S1 = P1 x K43	
2.	Add P2 and K44	i.e, S2 = P2 + K44	
3.	Add P3 and K45	i.e, S3 = P3 + K45	
4.	Multiply P4 and K46	i.e, S4 = P4 x K46	
5.	XOR step 1 and step 3	i.e, S5 = S1 $\oplus$S3	
6.	XOR step 2 and step 4	i.e, S6 = S2 $\oplus$S4	
7.	Multiply step5 with K47	i.e, S7 = S5 x K47	
8.	Add step 6 and step 7	i.e, S8 = S6 + S7	
9.	Multiply step8 with K48	i.e, S9 = S8 x K48	
10.	Add step 7 and step 9	i.e, S10= S7 + S9	
11.	XOR step 1 and step 9	i.e, S11=S1 $\oplus$S9	$\rightarrow$ New P1
12.	XOR step 3 and step 9	i.e, S12= S3 $\oplus$S9	$\rightarrow$ New P2
13.	XOR step 2 and step10	i.e, S13= S2 $\oplus$S10	$\rightarrow$ New P3
14.	XOR step 4 and step10	i.e, S14= S4 $\oplus$S10	$\rightarrow$ New P4
15.	Multiply P1 with K49	i.e, P1 x K49	
16.	Add P2 with K50	i.e, P2 + K50	
17.	Add P3 with K51	i.e, P3 + K51	
18.	Multiply P4 with K52	i.e, P4 x K52	

- In the 8th round steps 15 to 18 is called output tramsformation to obtain C1, C2, C3, C4 combined to form cipher text C.

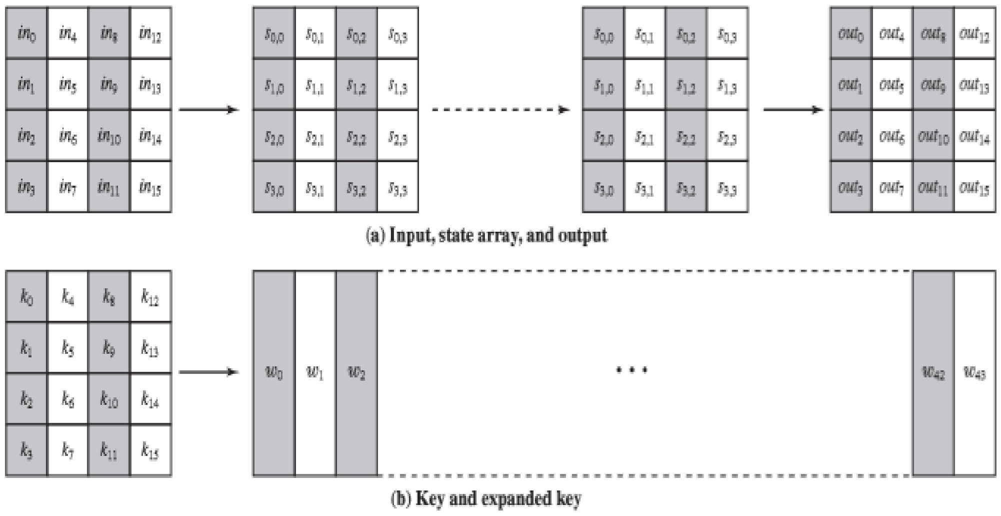
❖ **OUTPUT TRANSFORMATION**
- It is a one time operation
- Takes place at the end of 8th round.
- The output of this process is the final 64 bit cipher text which is the combination of four four cipher text blocks C1 to C4.

❖ **DECRYPTION**
- The process is exactly same as that of encryption process.
- There are some altertions in the generation and patterns of subkeys.
- The decryption subkeys are actually an inverse of the encryption subkeys.That is while subkey generation an right circular shift is performed.

❖ **STRENGTH OF IDEA**
- IDEA uses 128 bit key which is double than the size of DES.
- Thus to break IDEA 2 exp 128 encryption operations would be required.
- Thus IDEA is more secured.

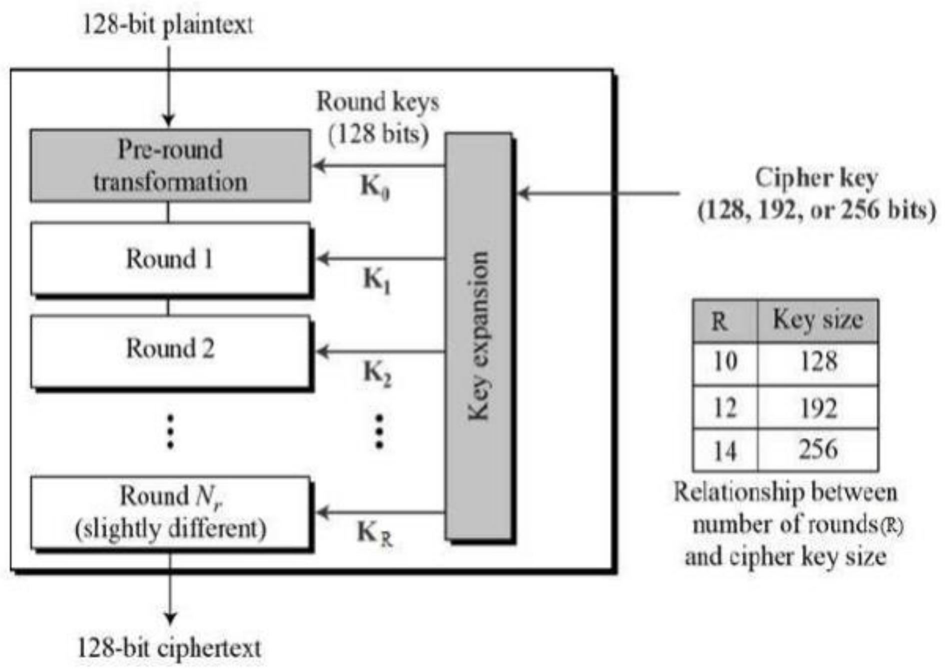
ADVANCED ENCRYPTION STANDARD

- Designed by Rijmen-Daemen in Belgium
- Has 128/192/256 bit keys, 128 bit data
- an **iterative** rather than **feistel** cipher
  - It is based on 'substitution–permutation network'.
  - processes data as block of 4 columns of 4 bytes
  - operates on entire data block in every round
- designed to be:
  - resistant against known attacks
  - speed and code compactness on many CPUs
  - design simplicity
- **AES DATA STRUCTURE**
  - Input array
  - State array
  - Output array
  - Key and expanded key

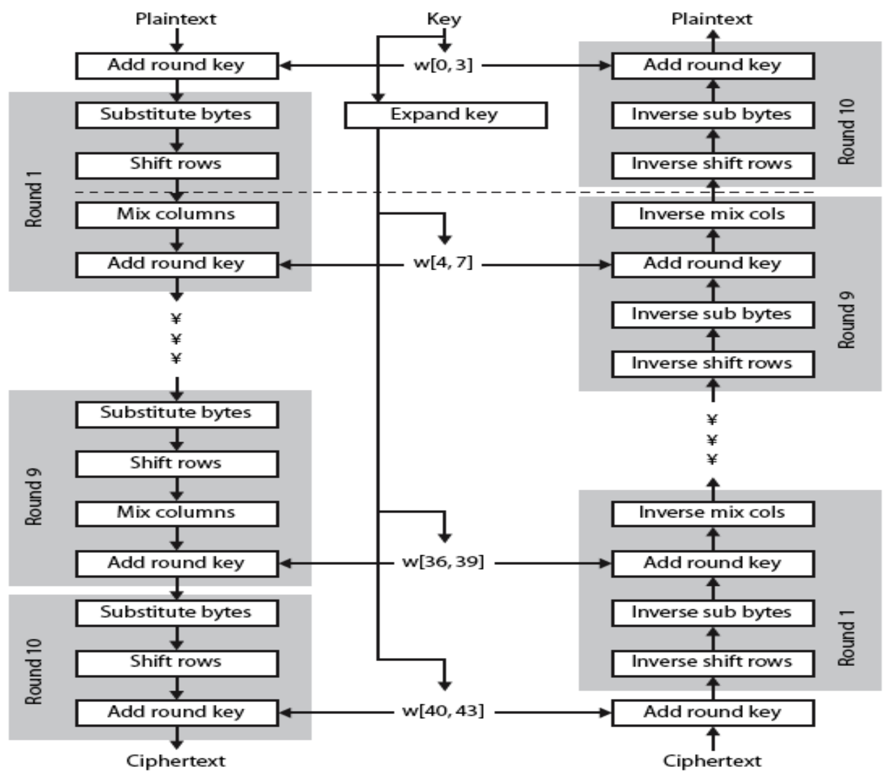


in_0 in_4 in_8 in_{12} in_1 in_5 in_9 in_{13} in_2 in_6 in_{10} in_{14} in_3 in_7 in_{11} in_{15}
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$ $s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$ $s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
out_0 out_4 out_8 out_{12} out_1 out_5 out_9 out_{13} out_2 out_6 out_{10} out_{14} out_3 out_7 out_{11} out_{15}
(a) Input, state array, and output
k_0 k_4 k_8 k_{12} k_1 k_5 k_9 k_{13} k_2 k_6 k_{10} k_{14} k_3 k_7 k_{11} k_{15}
w_0 w_1 w_2 • • • w_{42} w_{43}
(b) Key and expanded key

- **AES STRUCTURE**
  - Encryption consists of 10 rounds of processing for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys.
  - Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

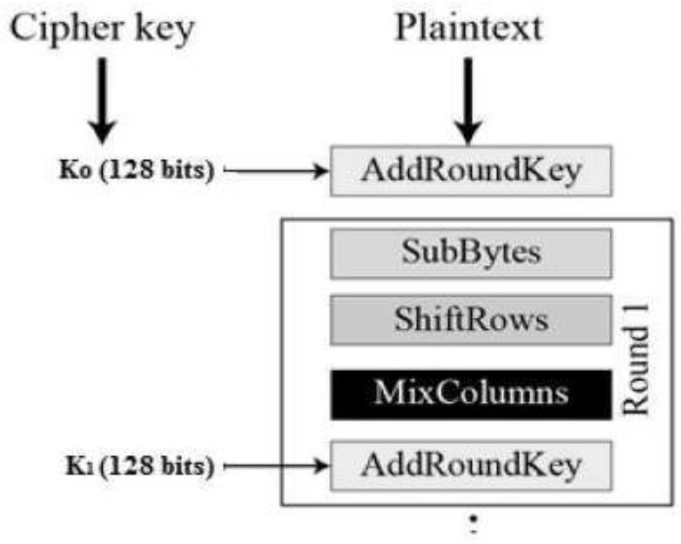


R	Key size
10	128
12	192
14	256



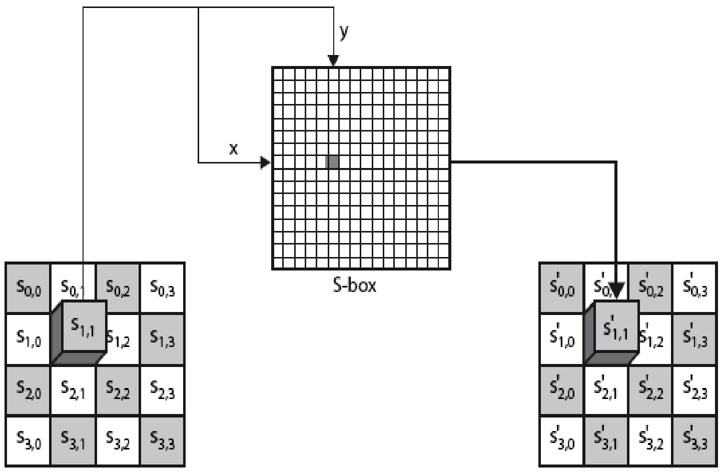
(a) Encryption (b) Decryption

- **<u>AES ENCRYPTION PROCESS</u>**



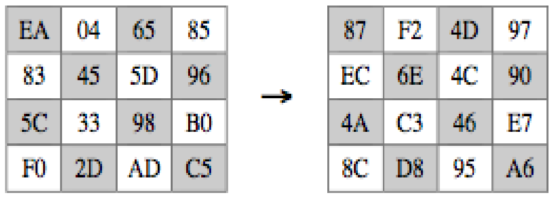
first round process

- **Substitute Bytes**
  - a simple substitution of each byte
  - uses one table of 16x16 bytes containing a permutation of all 256 8-bit values
  - each byte of state is replaced by byte indexed by row (left 4-bits) & column (right 4-bits)
    - eg. byte {95} is replaced by byte in row 9 column 5
  - designed to be resistant to all known attacks
  - The Byte Substitution operates on each byte of state independently, with the input byte used to index a row/col in the table to retrieve the substituted value.
  - The corresponding substitution step used during decryption is called InvSubBytes.)



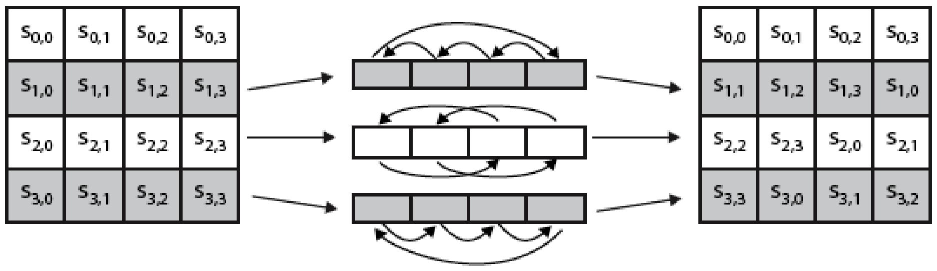
y
x
S-box
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

- **Substitute Bytes Example**

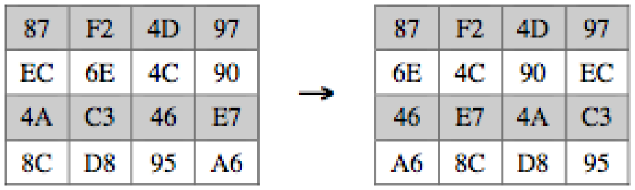


EA	04	65	85		87	F2	4D	97
83	45	5D	96	→	EC	6E	4C	90
5C	33	98	B0		4A	C3	46	E7
F0	2D	AD	C5		8C	D8	95	A6

- **Shift Rows**
  - a circular byte shift in each
    - ➢ 1st row is unchanged
    - ➢ 2nd row does 1 byte circular shift to left
    - ➢ 3rd row does 2 byte circular shift to left
    - ➢ 4th row does 3 byte circular shift to left
  - decrypt inverts using shifts to right
  - since state is processed by columns, this step permutes bytes between the columns



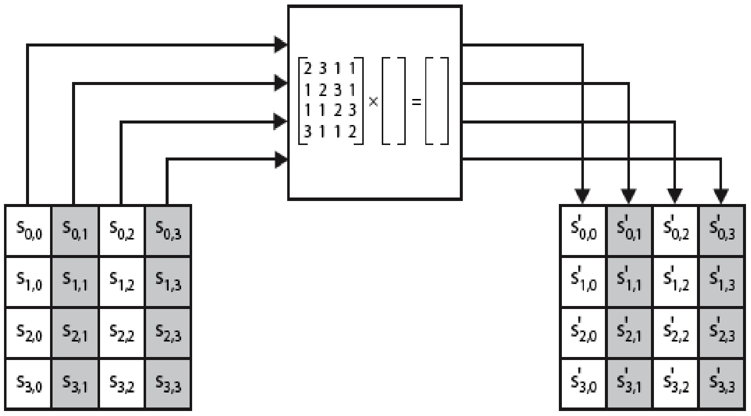
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{1,0}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,1}$
$s_{3,3}$ $s_{3,0}$ $s_{3,1}$ $s_{3,2}$



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

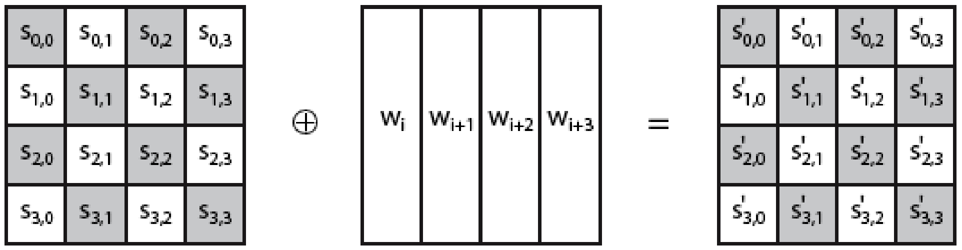
87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

- **Mix Columns**
  - each column is processed separately
  - each byte is replaced by a value dependent on all 4 bytes in the column
  - Each column of four bytes is now transformed using a special mathematical function.
  - This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column.
  - The result is another new matrix consisting of 16 new bytes.
  - **It should be noted that this step is not performed in the last round.**



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

- **Add Round Key**
  - XOR state with 128-bits of the round key
  - again processed by column (though effectively a series of byte operations)
  - The 16 bytes of the matrix are now considered as 128 bits and are XORed to the 128 bits of the round key.
  - If this is the last round then the output is the ciphertext.
  - Otherwise, the resulting 128 bits are interpreted as 16 bytes and we begin another similar round.


$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$\oplus$
w_i w_{i+1} w_{i+2} w_{i+3}
$=$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

- AES Encryption round

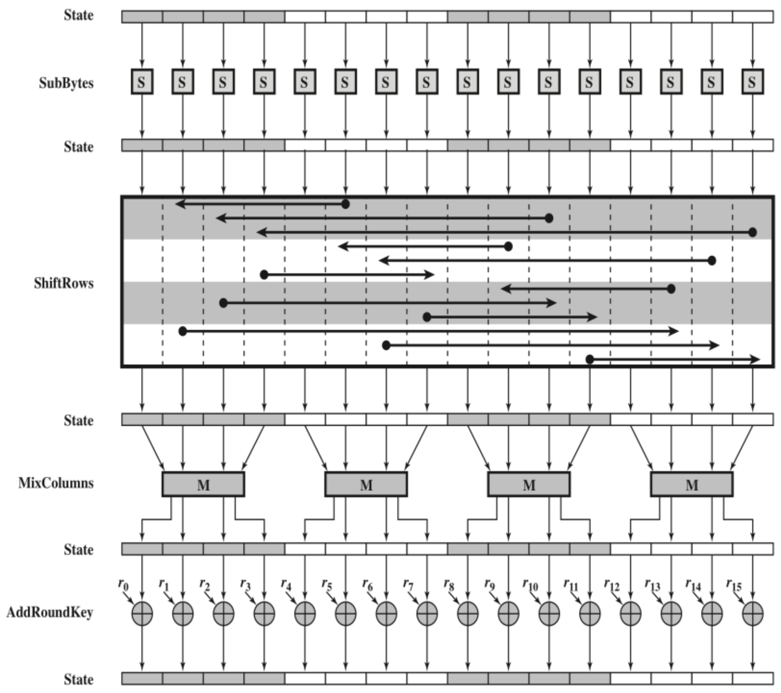

State
SubBytes
S
State
ShiftRows
State
MixColumns
M
State
AddRoundKey
r_0 r_1 r_2 r_3 r_4 r_5 r_6 r_7 r_8 r_9 r_{10} r_{11} r_{12} r_{13} r_{14} r_{15}
State

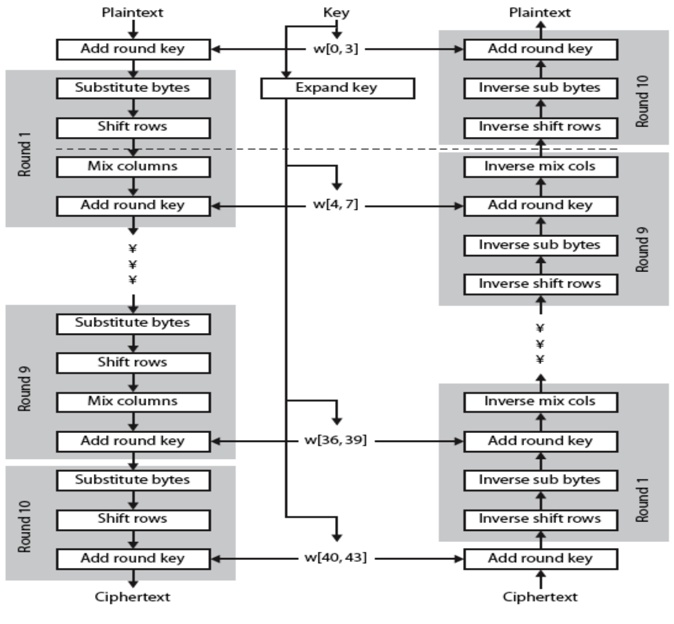
Figure 5.4 AES Encryption Round

- <u>**AES DECRYPTION PROCESS**</u>
  - The process of decryption of an AES ciphertext is similar to the encryption process in the reverse order.
  - Each round consists of the four processes conducted in the reverse order
    - Add round key
    - Mix columns
    - Shift rows
    - Byte substitution

- Since sub-processes in each round are in reverse manner, unlike for a Feistel Cipher, the encryption and decryption algorithms needs to be separately implemented, although they are very closely related.

INVERSE CIPHER

- AES decryption cipher is not identical to the encryption cipher
- Sequence of transformations for decryption differs from that for encryption, although the form of the key schedules for encryption and decryption is the same.

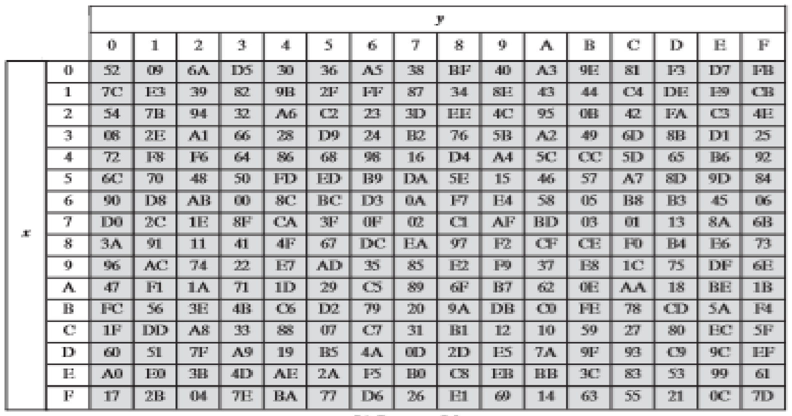


(a) Encryption (b) Decryption

- It has the disadvantage that two separate software or firmware modules are needed for applications that require both encryption and decryption.
- An equivalent version of the decryption algorithm that has the same structure as the encryption algorithm.
- The equivalent version has the same sequence of transformations as the encryption algorithm (with transformations replaced by their inverses).
- To achieve this equivalence, a change in key schedule is needed.
- An encryption round has the structure SubBytes, ShiftRows, MixColumns, AddRoundKey.
- The standard decryption round has the structure InvShiftRows, InvSubBytes, AddRoundKey, InvMixColumns.
- Thus, the first two stages of the decryption round need to be interchanged, and the second two stages of the decryption round need to be interchanged.
- **INTERCHANGING INVSHIFTROWS AND INVSUBBYTES**
  - InvShiftRows affects the sequence of bytes in State but does not alter byte contents and does not depend on byte contents to perform its transformation.
  - InvSubBytes affects the contents of bytes in State but does not alter byte sequence and does not depend on byte sequence to perform its transformation.
  - For a given State Si
    - InvShiftRows [InvSubBytes (Si)] = InvSubBytes [InvShiftRows (Si)]
- **INTERCHANGING ADDROUNDKEY AND INVMIXCOLUMNS**

- The transformations AddRoundKey and InvMixColumns do not alter the sequence of bytes in State.
- If we view the key as a sequence of words,then both AddRoundKey and InvMixColumns operate on State one column at a time
- for a given State Si and a given round key wj, InvMixColumns (Si XOR wj) = [InvMixColumns (Si)] XOR [InvMixColumns (wj)]

INVSUBBYTES



		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
x	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

INVSHIFTROWS

- a circular byte shift in each
  - 1st row is unchanged
  - 2nd row does 1 byte circular shift to right
  - 3rd row does 2 byte circular shift to right
  - 4th row does 3 byte circular shift to right

INVMIXCOLUMNS

- Inverse mix columns operation
- Take each column(word) and multiply with a predefined 4X4 matrix(inverse)

we can interchange AddRoundKey and InvMixColumns, provided that we first apply InvMixColumns to the round key.

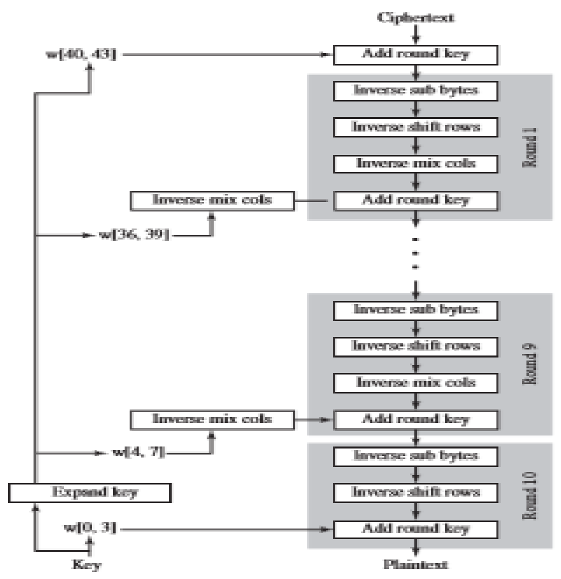


Figure 5.10 Equivalent Inverse Cipher

Equivalent decryption algorithm.

KEY EXPANSION

- Takes as input a 4-word (16 byte) key and produces a linear array of 44 words (176) bytes
- In other words, the orginal 16-byte key array is expanded into a key containing 11X4X4= 176 bytes.

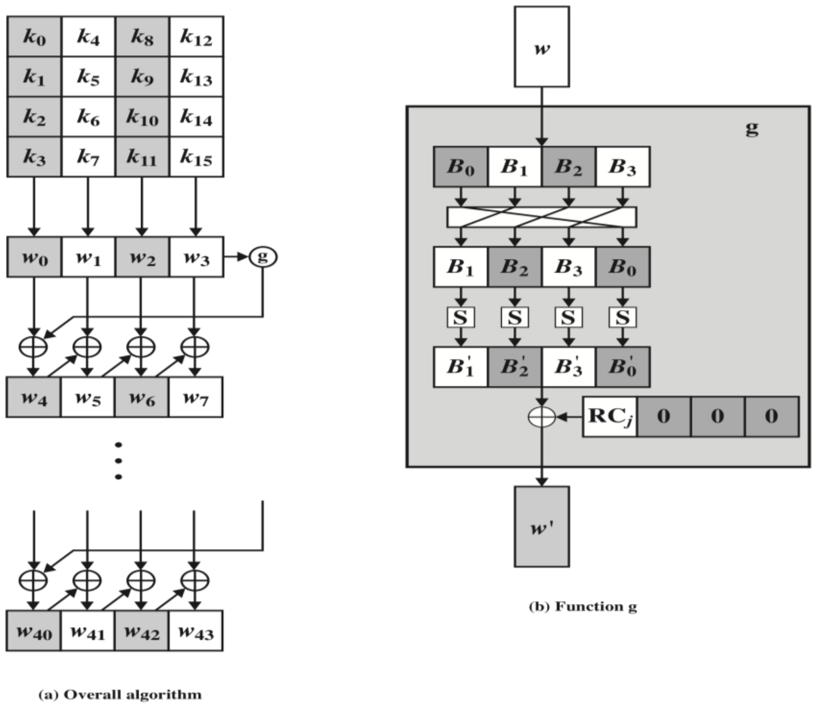

k_0 k_4 k_8 k_{12}
k_1 k_5 k_9 k_{13}
k_2 k_6 k_{10} k_{14}
k_3 k_7 k_{11} k_{15}
w_0 w_1 w_2 w_3
g
w_4 w_5 w_6 w_7
w_{40} w_{41} w_{42} w_{43}
(a) Overall algorithm
w
g
B_0 B_1 B_2 B_3
B_1 B_2 B_3 B_0
S S S S
B'_1 B'_2 B'_3 B'_0
RC_j 0 0 0
w'
(b) Function g

Figure 5.9 AES Key Expansion

- **STEPS**
  - ❖ The orginal 16 byte key is copied into the first 4 words of the expanded key.

        - (After filling the first array (W0 to W3) of the expanded key block, the remaining 10 arrays(for words numbered from W4 to W43) are filled one by one.
        - Every added key array block depends on the immediately preceding block and the block 4 positions earlier.
    - Function **ROTATE** performs a circular left shift on the contents of the word by one byte.If an input word contains four bytes numbered[B1,B2,B3,B4] then the output word would contain [B2,B3,B4,B1].
    - Function **SUBSTITUTE** performs a byte substitution on each byte of the input word. For this purpose it uses an S- box.
    - In the function **CONSTANT(Rcon),** the output of the above steps is XORed with a constant. This constant is a word consisting of 4 bytes. The value of the constant depends on the round number. The last three bytes of a constant word always contain 0.

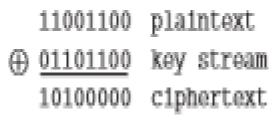
Round number	1	2	3	4	5	6	7	8	9	10
Value of a constant be used in Hex	01	02	04	08	10	20	40	80	1B	36

- **Key Expansion Rationale**
    - The expansion key algorithm is designed to be resistant to known cryptanalytic attacks
    - The specific criteria that were used are:
        - Knowledge of a part of the cipher key or round key does not enable calculation of many other round-key bits.
        - An invertible transformation [i.e.,knowledge of any consecutive words of the expanded key enables regeneration the entire expanded key].
        - Speed on a wide range of processors.
        - Usage of round constants to eliminate symmetries.
        - Diffusion of cipher key differences into the round keys; that is, each key bit affects many round key bits.
        - Simplicity of description.

STREAM CIPHER

- A typical stream cipher encrypts plaintext one byte (or bit) at a time, usually by XOR'ing with a pseudo-random keystream.
- Have a pseudo random **keystream**
- randomness of **stream key** completely destroys statistically properties in message
    - $C_i = M_i$ XOR $StreamKey_i$

    Encryption:

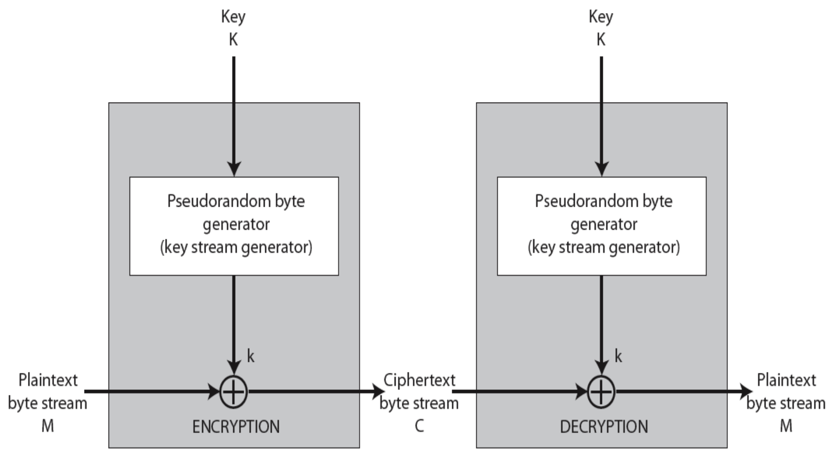


```
  11001100  plaintext
⊕ 01101100  key stream
  10100000  ciphertext
```

    Decryption:

```
   10100000  ciphertext
⊕  01101100  key stream
   11001100  plaintext
```

- **STREAM CIPHER DIAGRAM**


Key
K
Pseudorandom byte generator (key stream generator)
k
Plaintext byte stream M
ENCRYPTION
Ciphertext byte stream C
Key
K
Pseudorandom byte generator (key stream generator)
k
DECRYPTION
Plaintext byte stream M

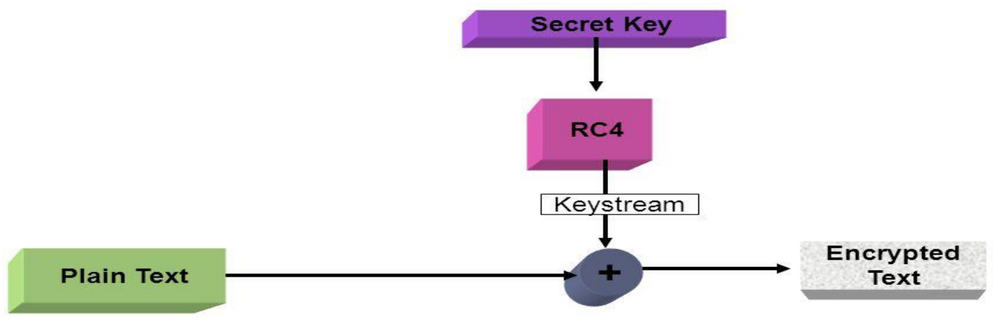
  - general structure of a stream cipher illustrates that, where a key is input to a pseudorandom bit generator that produces an apparently random keystream of bits, and which are XOR'd with message to encrypt it, and XOR'd again to decrypt it by the receiver.
- **DESIGN CONSIDERATIONS FOR A STREAM CIPHER:**
  - The encryption sequence should have a large period, the longer the period of repeat the more difficult it will be to do cryptanalysis.
  - The keystream should approximate the properties of a true random number stream as close as possible, the more random-appearing the keystream is, the more randomized the ciphertext is, making cryptanalysis more difficult.
  - To guard against brute-force attacks, the key needs to be sufficiently long. The same considerations as apply for block ciphers are valid here .Thus, with current technology, a key length of at least 128 bits is desirable.
  - With a properly designed pseudorandom number generator, a stream cipher can be as secure as block cipher of comparable key length. The primary advantage of a stream cipher is that stream ciphers are almost always faster and use far less code than do block ciphers.

RC4

- ❖ RC4 is a stream cipher designed in 1987 by Ron Rivest for RSA Security.
- ❖ It is a variable key-size stream cipher with byte-oriented operations.
- ❖ The algorithm is based on the use of a random permutation.
- ❖ RC4 is probably the most widely used stream cipher.
- ❖ It is used in the SSL/TLS secure web protocol, & in the WEP & WPA wireless LAN security protocols.
- ❖ RC4 was kept as a trade secret by RSA Security, but in September 1994 was anonymously posted on the Internet on the Cypherpunks anonymous remailers list.

- The RC4 key is used to form a random permutation of all 8-bit values, it then uses that permutation to scramble input info processed a byte at a time.
- RC4 generates a pseudorandom tream of bits called keystream. This is combined wih the plaintext using XOR for encryption.

RC4 Block Diagram



Cryptographically very strong and easy to implement

- Two parts are included:
    - Key schedule algorithm(KSA)
        - starts with an array S of numbers: 0..255
        - use key to well and truly shuffle
        - S forms **internal state** of the cipher

```
for i = 0 to 255 do
    S[i] = i
    T[i] = K[i mod keylen])
j = 0
for i = 0 to 255 do
    j = (j + S[i] + T[i]) (mod 256)
    swap (S[i], S[j])
```

        - The RC4 key schedule initialises the state S to the numbers 0..255, and then walks through each entry in turn, using its current value plus the next byte of key to pick another entry in the array, and swaps their values over. After doing this 256 times, the result is a well and truly shuffled array. The total number of possible states is 256! - a truly enormous number, much larger even than the 2048-bit (256*8) max key allowed can select.
    - Pseudo random number generation
        - encryption continues shuffling array values

        - sum of shuffled pair selects "stream key" value from permutation
        - XOR S[t] with next byte of message to en/decrypt

```
i = j = 0
for each message byte $M_i$
i = (i + 1) (mod 256)
j = (j + S[i]) (mod 256)
swap(S[i], S[j])
t = (S[i] + S[j]) (mod 256)
$C_i$ = $M_i$ XOR S[t]
```

        - To form the stream key for en/decryption (which are identical), RC4 continues to shuffle the permutation array S by continuing to swap each element in turn with some other entry, and using the sum of these two entry values to select another value from the permutation to use as the stream key, which is then XOR'd with the current message byte.

- **DECRYPTION**
    - Decryption is also processed in the same manner.