Advance Encryption Standard

Why AES ?

- DES broken for small key size
- Demand: replacement of DES
- Triple-DES is secured……. ?

- NIST issued call for ciphers in 1997
- 15 candidates accepted in Jun 1998
- 5 were shortlisted in Aug 1999

NIST's requirements Competition

- Private key symmetric block cipher
- 128-bit data, 128-bit keys
- Stronger & faster than Triple-DES
- Provide full specification & design details
- Both C & Java implementations

NIST Evaluation Criteria

- Criteria:
  - Security – effort for practical cryptanalysis
  - Cost – in terms of computational efficiency
  - Algorithm & implementation characteristics
  - ease of software & hardware implementation
  - flexibility in encryption-decrypt and Keying

Shortlisted Candidates

- After testing and evaluation, shortlist in Aug-1999
  - MARS - complex, fast, high security margin
  - RC6 - simple, fast, low security margin
  - Rijndael - fast, good security margin
  - Serpent - slow, clean, high security margin
  - Twofish - complex, fast, high security margin

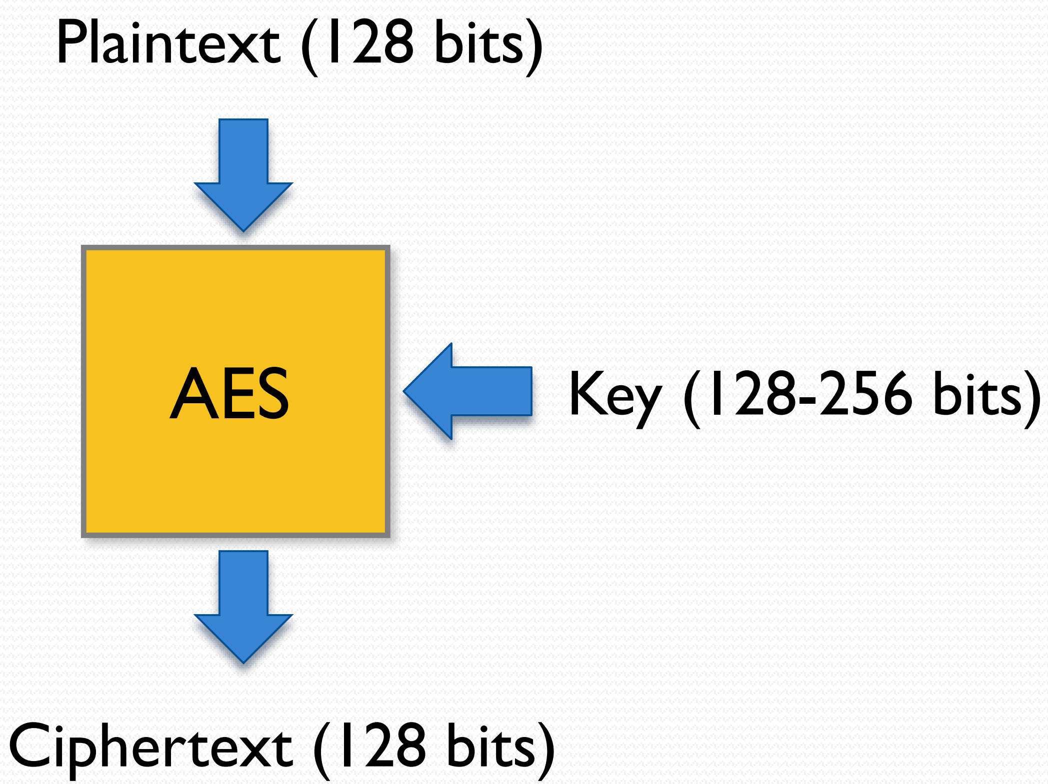
The AES Cipher - Rijndael

- Rijndael was selected as the AES in Oct-2000
  - Designed by Vincent Rijmen and Joan Daemen in Belgium

- An **iterative** rather than **Feistel** cipher
  - processes data (128 bits) as bytes (16 bytes)
  - 10 rounds

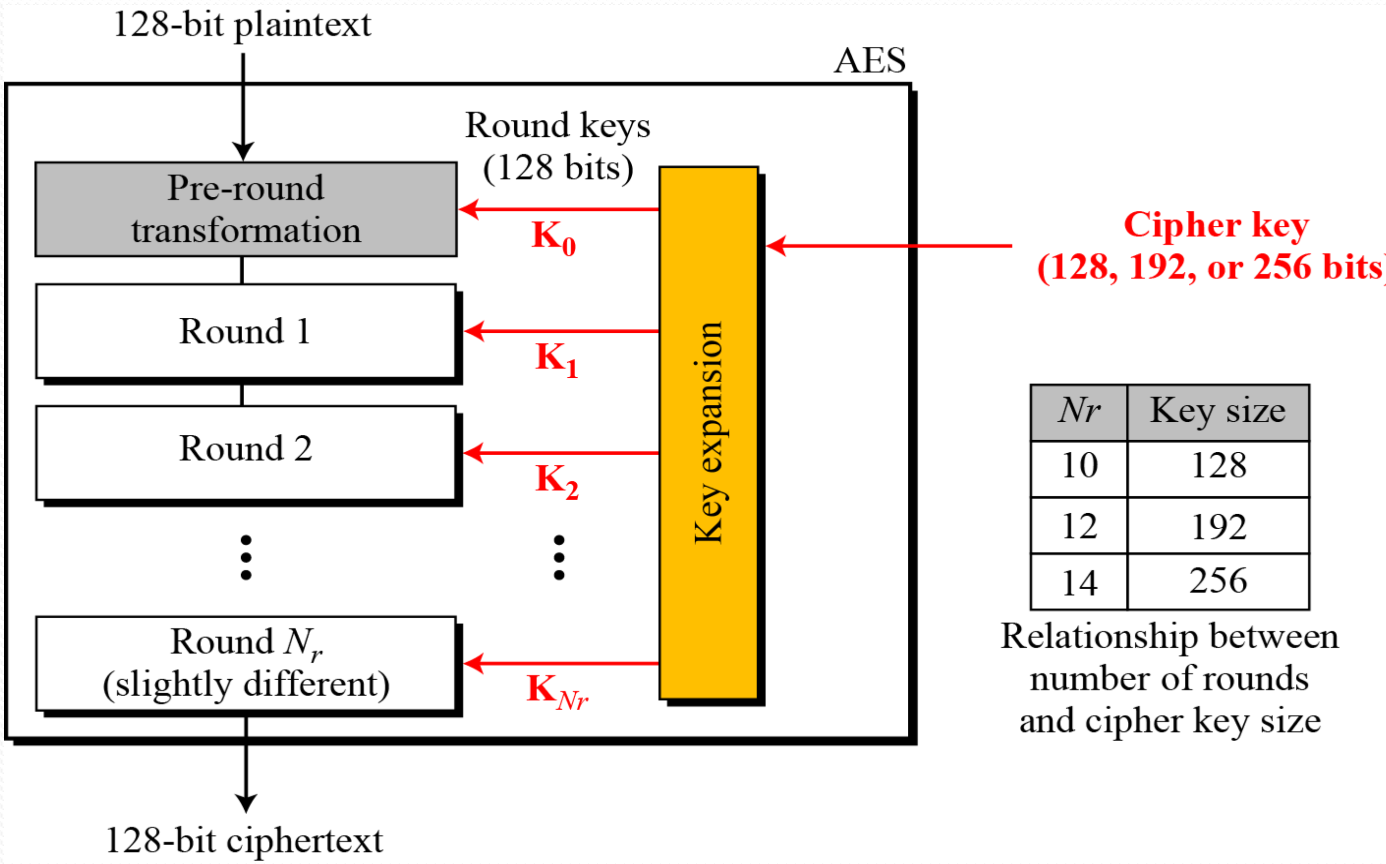
- Rijndael design:
  - 128 bits plain text and 128/192/256 bit keys,
  - resistant against known attacks
  - Both software and hardware friendly

AES Structure



AES Rounds

- First 9 rounds are identical
- last round are a little different



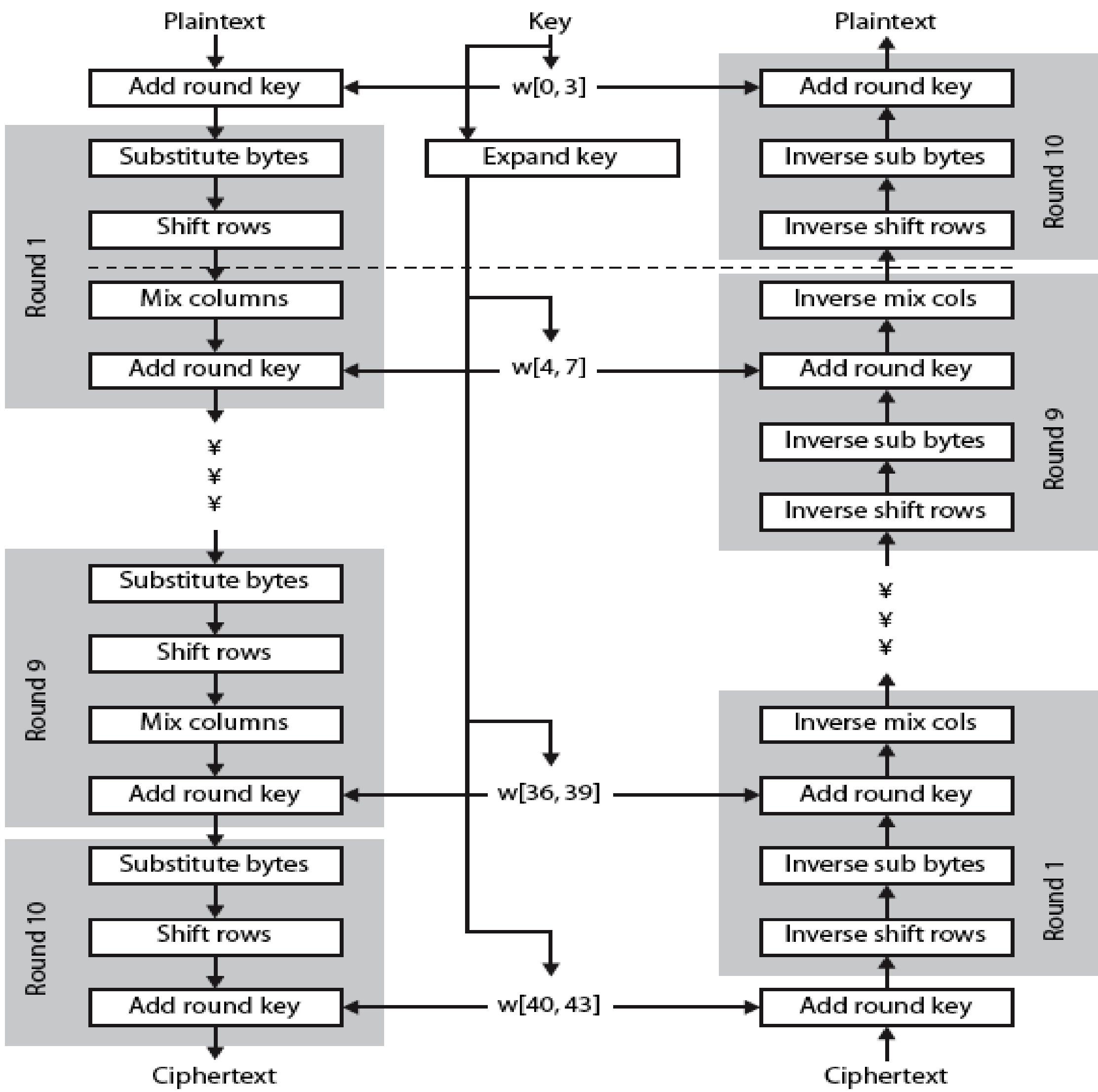
Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size

AES Overview

Key Expansion	• Round keys are derived from the cipher key using Rijndael's key schedule
Initial Round	• AddRoundKey : Each byte of the state is combined with the round key using bitwise xor
Rounds	• SubBytes : non-linear substitution step • ShiftRows : transposition step • MixColumns : mixing operation of each column. • AddRoundKey
Final Round	• SubBytes • ShiftRows • AddRoundKey No MixColumns

Rijndael

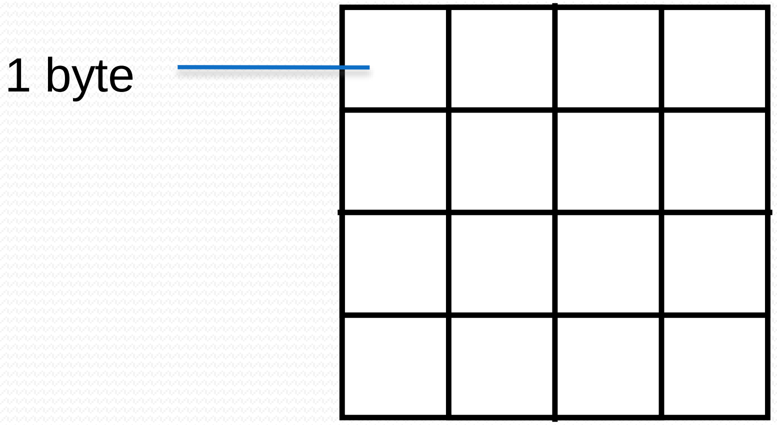


(a) Encryption

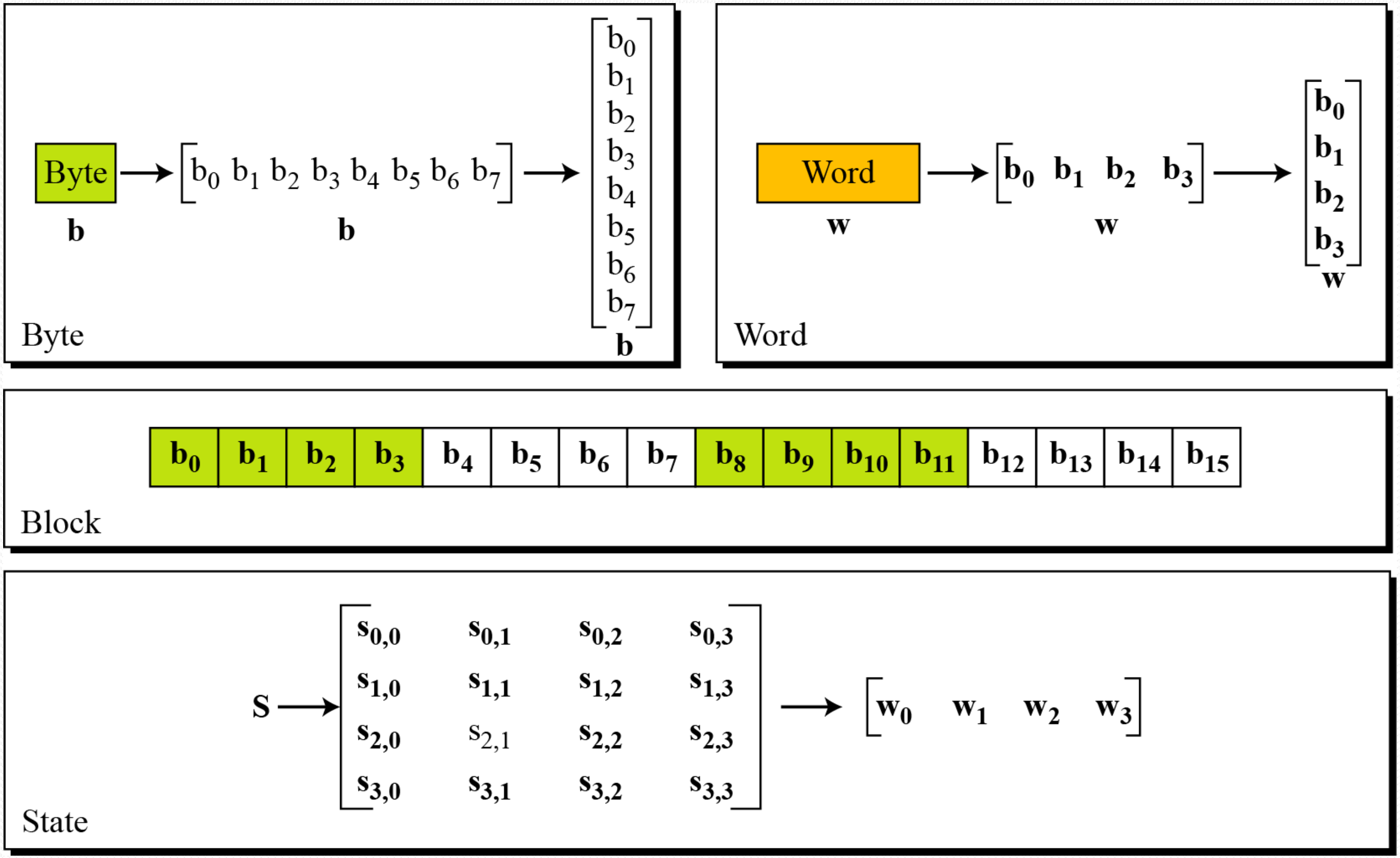
(b) Decryption

AES-128

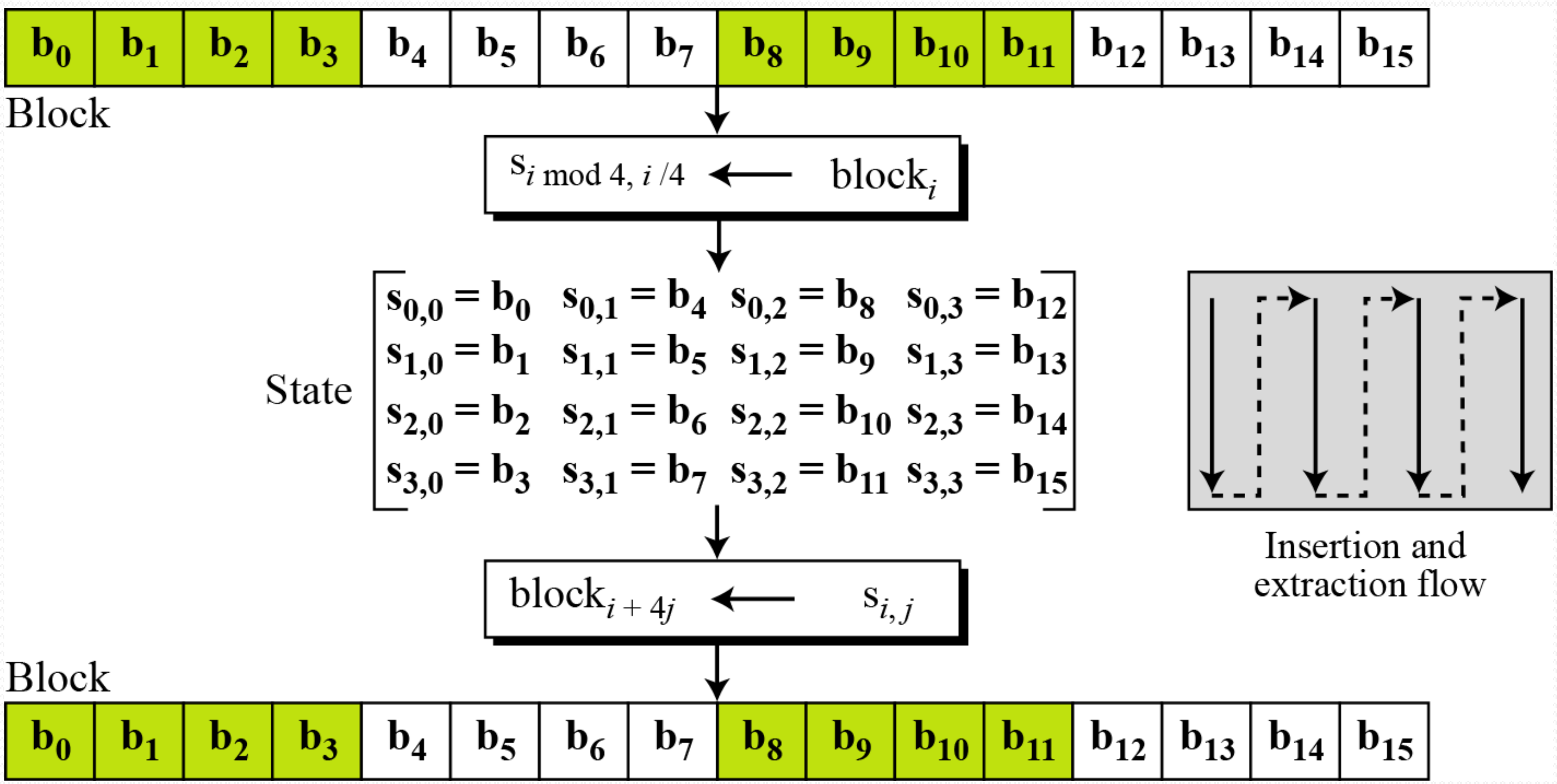
- Data block viewed as 4-by-4 table of bytes
- Represented as 4 by 4 matrix of 8-bit bytes.
- Key is expanded to array of 32 bits words



AES Data Structure



Unit Transformation



Changing Plaintext to State

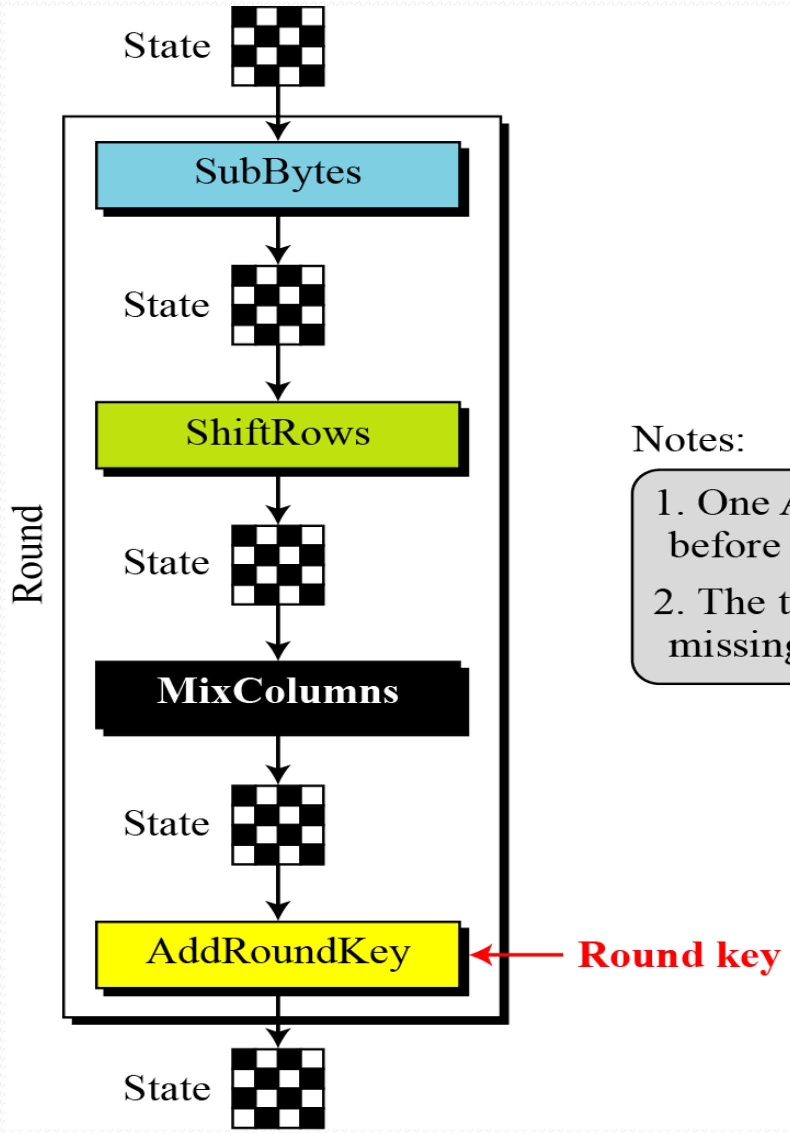
- Text: CRYPTOCLASSIITCS
- Hexadecimal Data:

  02 11 18 0F 13 0D 02 0B 00 12 12 08 08 13 02 12

State:

02	13	00	08
11	0D	12	13
18	02	12	02
0F	0B	08	12

AES Round



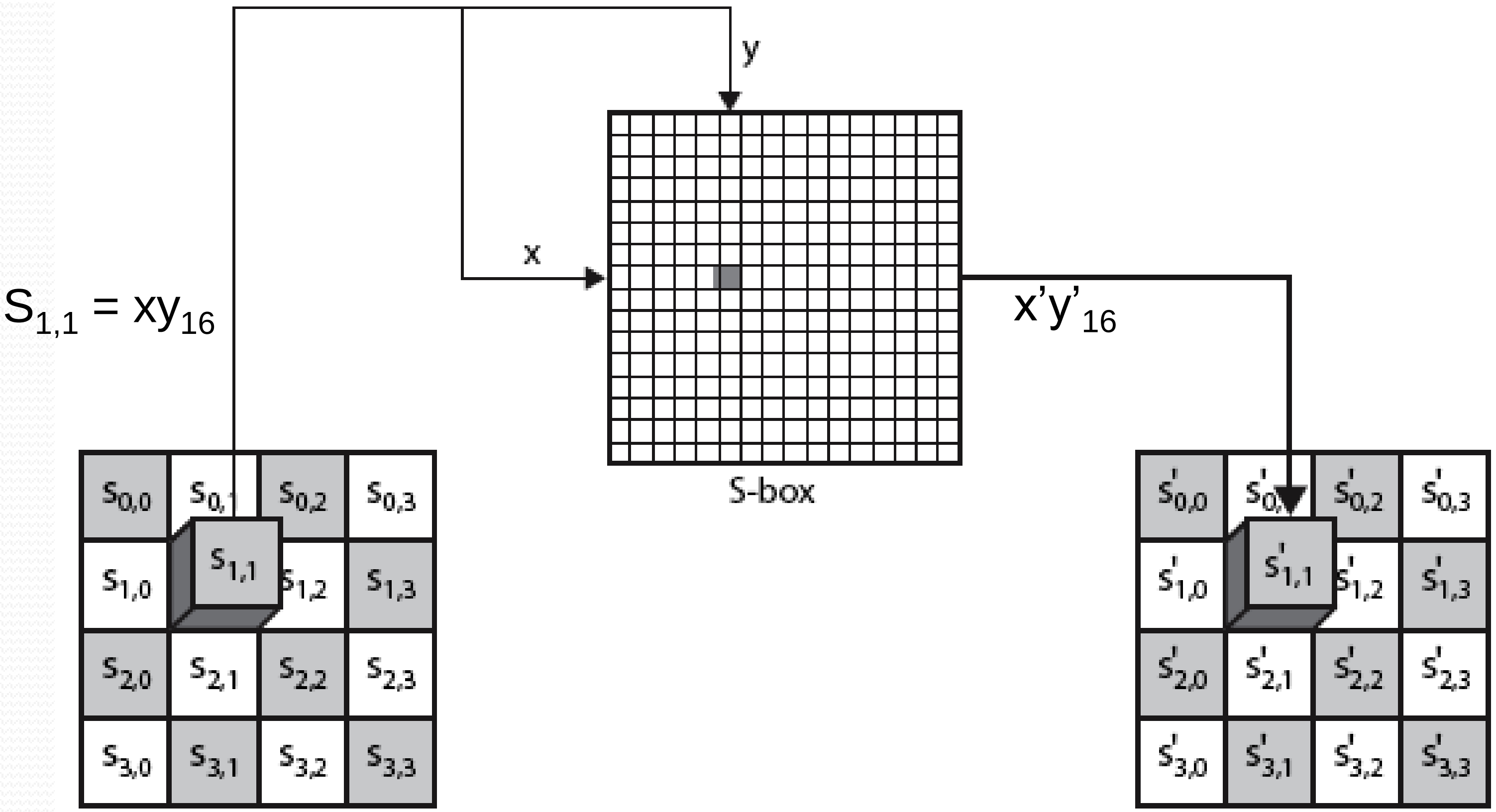
Notes:

1. One AddRoundKey is applied before the first round.
2. The third transformation is missing in the last round.

Substitute Byte Transformation

- A simple substitution of each byte
- Uses one S-box 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)
  - e.g. byte {95} is replaced by byte in row 9 column 5
  - which has value {2A}
- S-box constructed using defined transformation of values in Galois Field- GF(2^8)

SubBytes Operation



Sbox Construction

- Initialize the Sbox with byte values

  1st row: 00 01 02 … 0F

  2nd row: 10 11 12 … 1F

- Map each byte to its multiplicative inverse in the finite field GF(2^8); byte 00 is mapped to itself
- Consider each byte as (b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0). Apply the transformation to each bit of each byte

$$b_i' = b_i + b_{(i+4)\bmod 8} + b_{(i+5)\bmod 8} + b_{(i+6)\bmod 8} + b_{(i+7)\bmod 8} + c_i$$

+ is XOR and c_i is the ith bit of byte C with the value (63) (say)

Substitute Byte Transformation

- Byte transformation:

$$
\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix}
=
\begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}
\times
\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix}
+
\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}
$$

Example: Input 95, multiplicative inverse of 95 in $GF(2^8)$ = 8A.

After bit transformation the result is 2A (appears in row 9, column 5 of the Sbox)

Inverse Byte Transformation

- The inverse transformation to each bit of each byte

$$b_i' = b_{(i+2)\text{mod}8} + b_{(i+5)\text{mod}8} + b_{(i+7)\text{mod}8} + d_i$$

+ is XOR and d_i is the ith bit of byte D with the value.

D = 05 for C = (63)

B' = XB + C

Y(XB + c) + D = B

YXB + YC + D = B

YX = I, YC = D , YC + D is null vector

AES SBox

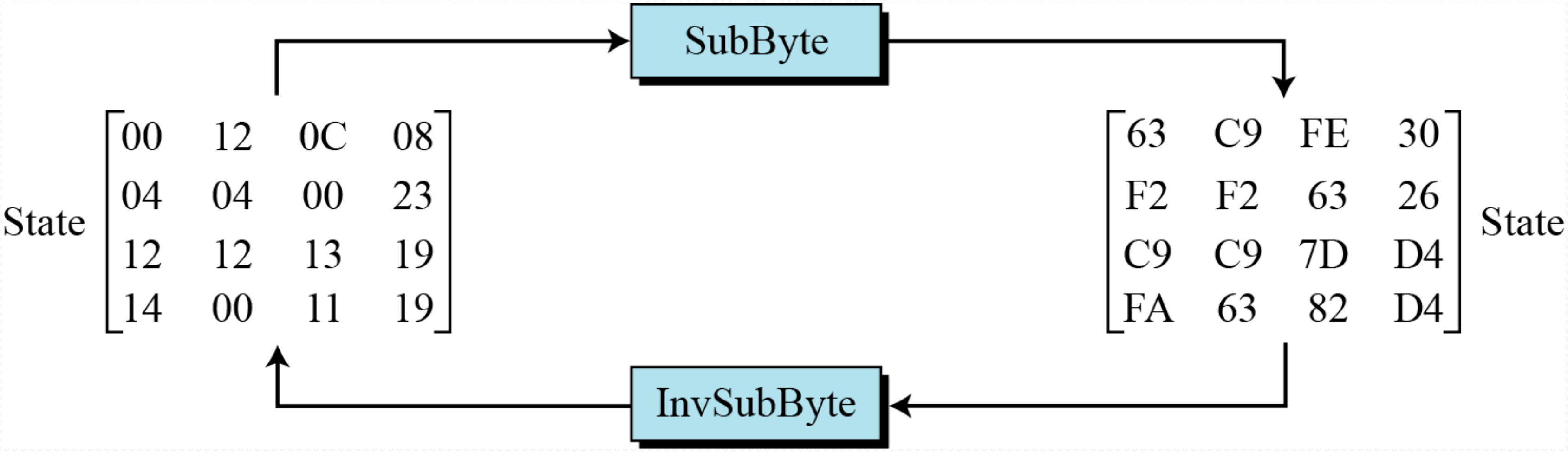
		y															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2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

Inverse SBox

		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

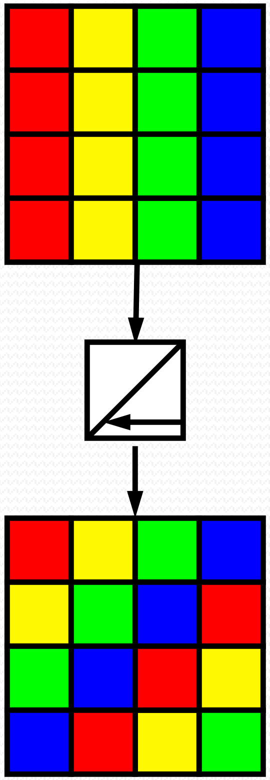
Sample SubByte Transformation

- The SubBytes and InvSubBytes transformations are inverses of each other.

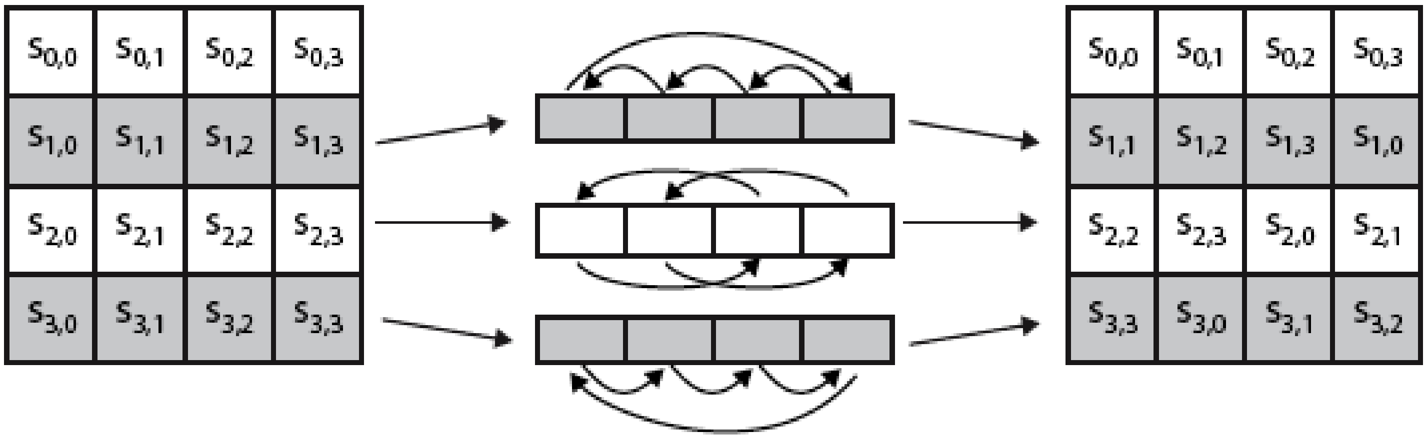


ShiftRows

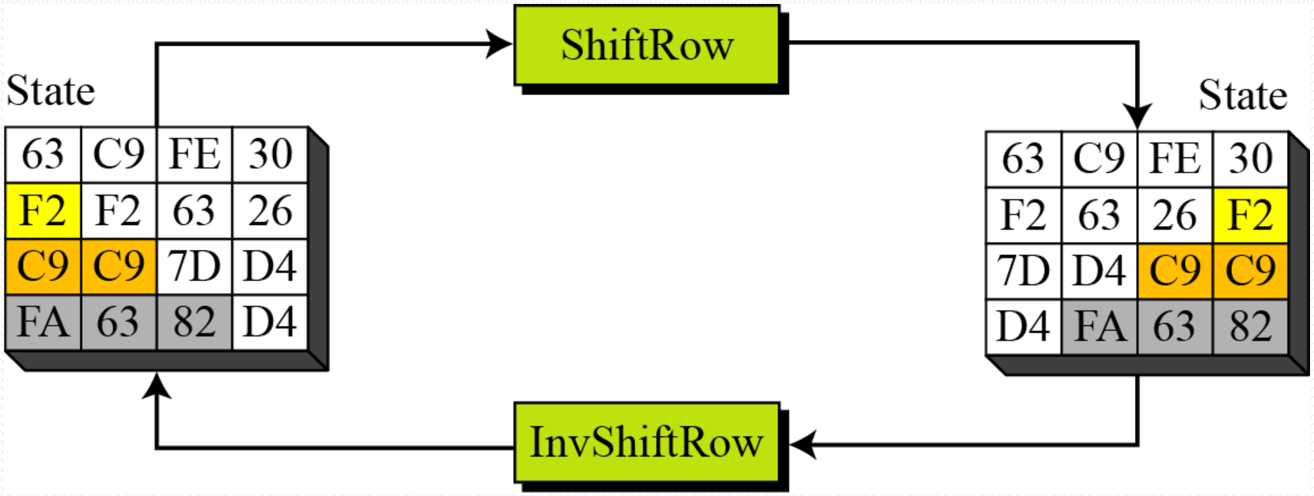
- Shifting, which permutes the bytes.
- A circular byte shift in each each
  - 1st row is unchanged
  - 2nd row does 1 byte circular shift to left
  - 3rd row does 2 byte circular shift to left
  - 4th row does 3 byte circular shift to left
- In the encryption, the transformation is called ShiftRows
- In the decryption, the transformation is called InvShiftRows and the shifting is to the right



ShiftRows Scheme



ShiftRows and InvShiftRows

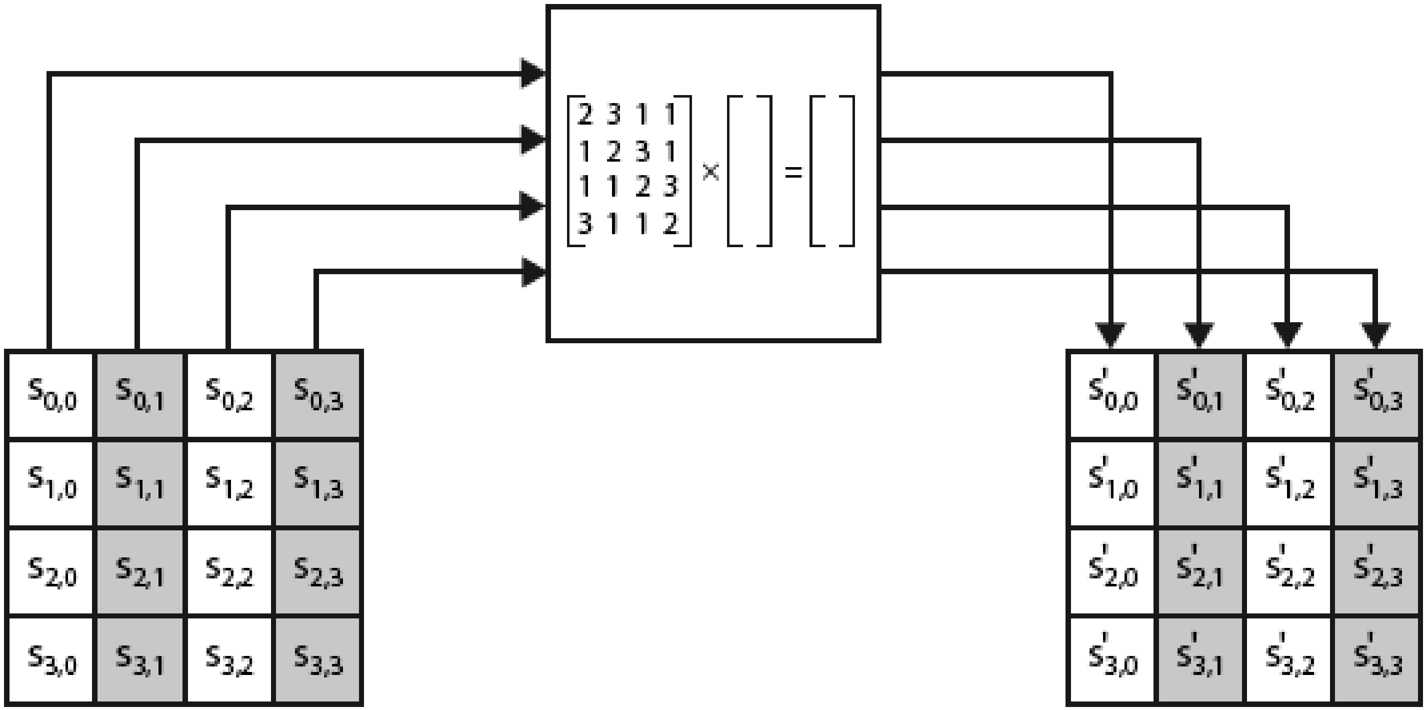


MixColumns

- ShiftRows and MixColumns provide diffusion to the cipher
- Each column is processed separately
- Each byte is replaced by a value dependent on all 4 bytes in the column
- Effectively a matrix multiplication in $GF(2^8)$ using prime poly $m(x) = x^8+x^4+x^3+x+1$

$$\begin{vmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{vmatrix} \times [S] = [S']$$

MixClumns Scheme



The MixColumns transformation operates at the column level; it transforms each column of the state to a new column.

MixColumn and InvMixColumn

$$\underbrace{\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}}_{\mathrm{C}} \xleftrightarrow{\text{Inverse}} \underbrace{\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}}_{\mathrm{C}^{-1}}$$

Mix Columns Transformation

- Another way
- Each column of state is treated as 4-term polynomial with coefficiens in $GF(2^8)$. Each column is multiplied modulo $(1+x^4)$ by the fixed polynomial $a(x)$

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

Similarly the inverse Mix Column transformation cabe performed by multiplying each column by $b(x)$ where

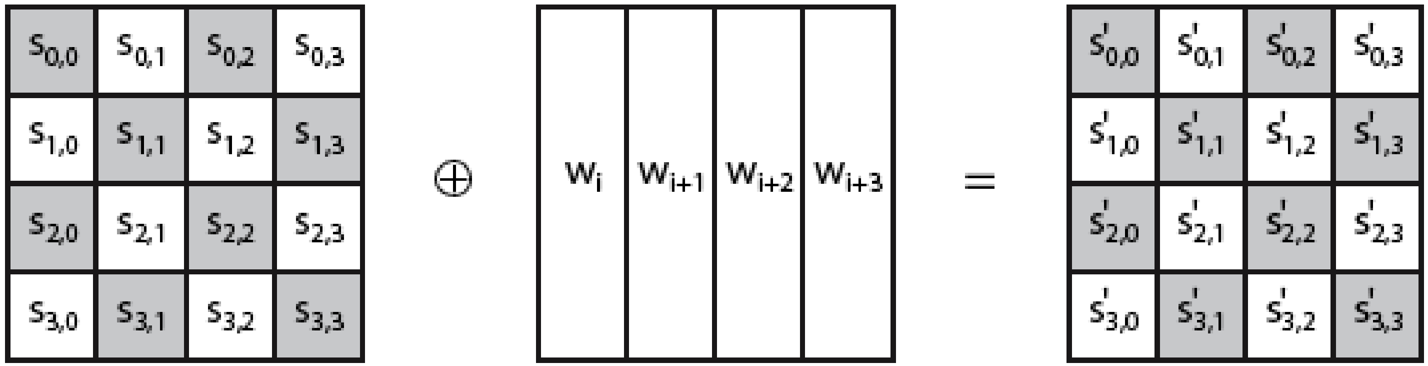
$$b(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$$

Here, $b(x) = a^{-1}(x) \bmod (1 + x^4)$

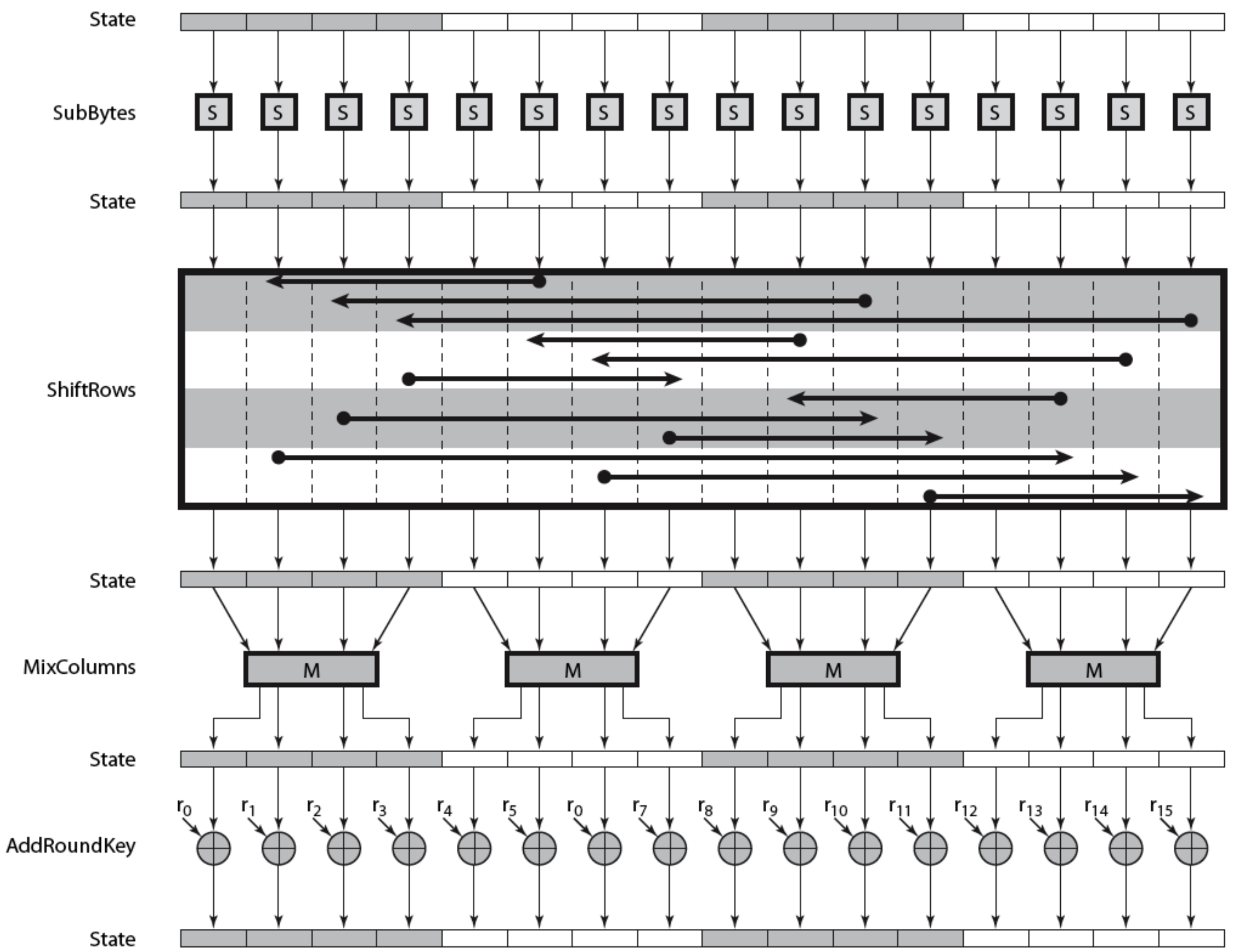
AddRoundKey

- XOR state with 128-bits of the round key
- AddRoundKey proceeds one column at a time.
  - adds a round key word with each state column matrix
  - the operation is matrix addition
- Inverse for decryption identical
  - since XOR own inverse, with reversed keys

AddRoundKey Scheme



AES Round


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

AES Key Scheduling

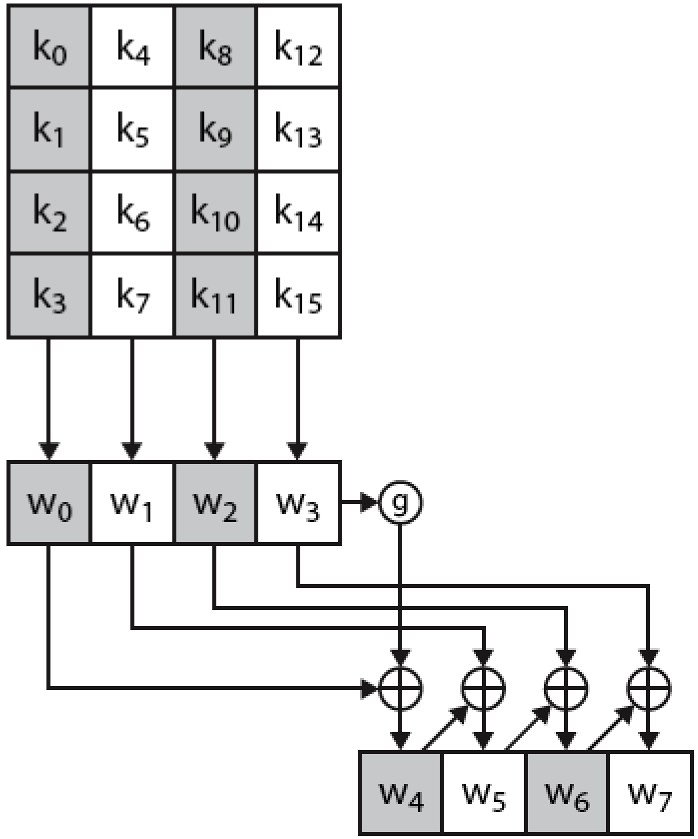
- takes 128-bits (16-bytes) key and expands into array of 44 32-bit words [44 words = 4 words + 40 words = 128 bit key + 10 128 bit round keys]

Round	*Words*			
Pre-round	$\mathbf{w}_0$	$\mathbf{w}_1$	$\mathbf{w}_2$	$\mathbf{w}_3$
1	$\mathbf{w}_4$	$\mathbf{w}_5$	$\mathbf{w}_6$	$\mathbf{w}_7$
2	$\mathbf{w}_8$	$\mathbf{w}_9$	$\mathbf{w}_{10}$	$\mathbf{w}_{11}$
. . .	. . .			
N_r	$\mathbf{w}_{4N_r}$	$\mathbf{w}_{4N_r+1}$	$\mathbf{w}_{4N_r+2}$	$\mathbf{w}_{4N_r+3}$

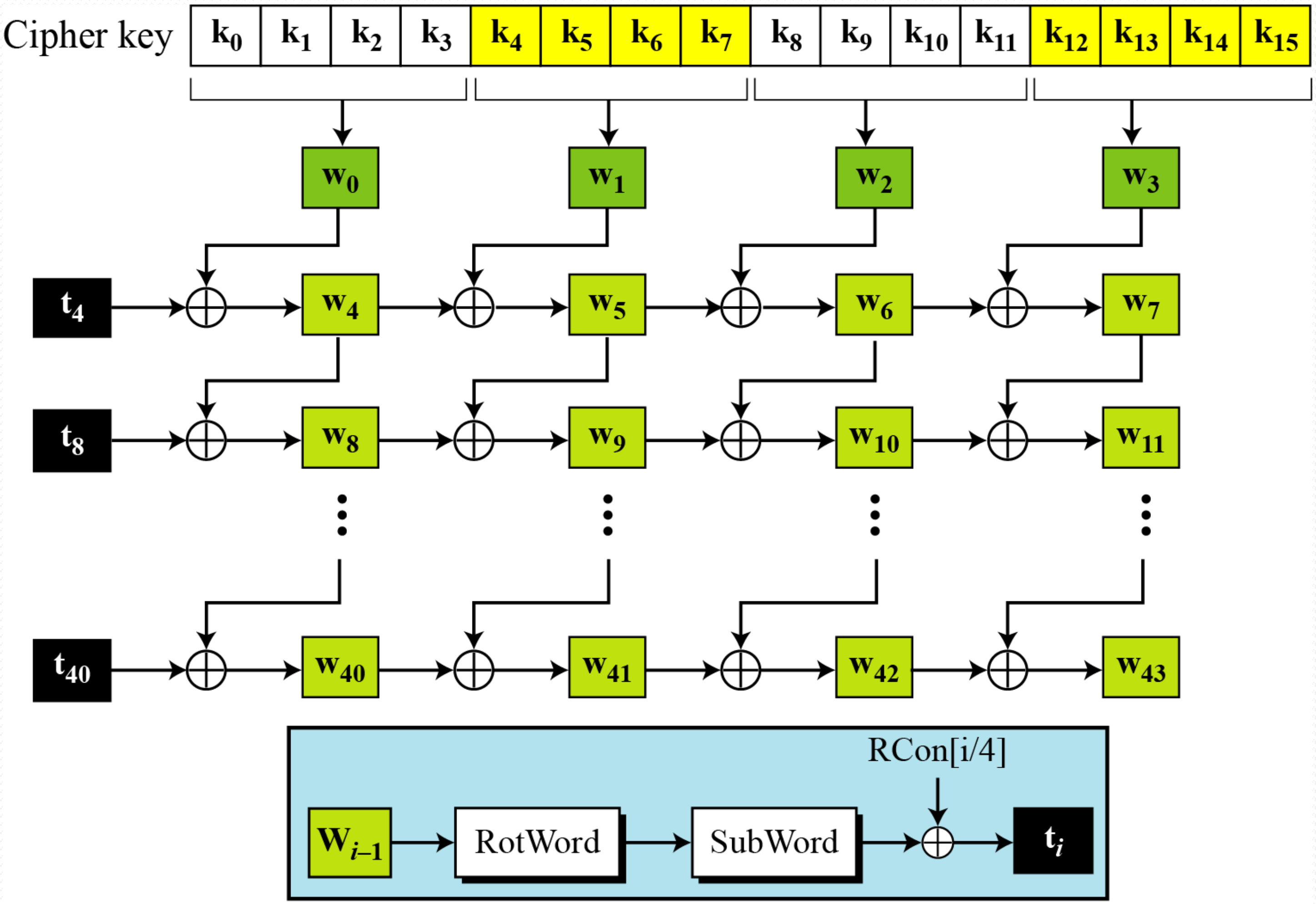
AES Key Expansion

- takes 128-bit (16-byte) key and expands into array of 44/52/60 32-bit words
- start by copying key into first 4 words
- then loop creating words that depend on values in previous & 4 places back
  - in 3 of 4 cases just XOR these together
  - 1st word in 4 has rotate + S-box + XOR round constant on previous, before XOR 4th back

AES Key Expansion



Key Expansion Scheme



Making of t_i (temporary) words $i = 4\ N_r$

Key Expansion submodule

- **RotWord** performs a one byte circular left shift on a word
  For example:

  RotWord[b0,b1,b2,b3] = [b1,b2,b3,b0]

- **SubWord** performs a byte substitution on each byte of input word using the S-box

- **SubWord(RotWord(temp))** is XORed with RCon[j] – the round constant

Round Constant (RCon)

- RCON is a word in which the three rightmost bytes are zero
- It is different for each round and defined as:

  RCon[j] = (RCon[j],0,0,0)

  where RCon[1] =1 , RCon[j] = 2 * RCon[j-1]
- Multiplication is defined over GF(2^8) but can be implement in Table Lookup

Round	*Constant (RCon)*	*Round*	*Constant (RCon)*
1	$(\underline{\mathbf{01}}\ 00\ 00\ 00)_{16}$	6	$(\underline{\mathbf{20}}\ 00\ 00\ 00)_{16}$
2	$(\underline{\mathbf{02}}\ 00\ 00\ 00)_{16}$	7	$(\underline{\mathbf{40}}\ 00\ 00\ 00)_{16}$
3	$(\underline{\mathbf{04}}\ 00\ 00\ 00)_{16}$	8	$(\underline{\mathbf{80}}\ 00\ 00\ 00)_{16}$
4	$(\underline{\mathbf{08}}\ 00\ 00\ 00)_{16}$	9	$(\underline{\mathbf{1B}}\ 00\ 00\ 00)_{16}$
5	$(\underline{\mathbf{10}}\ 00\ 00\ 00)_{16}$	10	$(\underline{\mathbf{36}}\ 00\ 00\ 00)_{16}$

Key Expansion Example (1st Round)

- Example of expansion of a 128-bit cipher key

`Cipher key = 2b7e151628aed2a6abf7158809cf4f3c`

w0=2b7e1516 **w1**=28aed2a6 **w2**=abf71588 **w3**=09cf4f3c

i	w_{i-1}	RotWord	SubWord	Rcon[i/4]	t_i	w[i-4]	w_i
4	09cf4f3c	cf4f3c09	8a84eb01	01000000	8b84eb01	2b7e1516	a0fafe17
5	a0fafe17	-	-	-	-	28aed2a6	88542cb1
6	88542cb1	-	-	-	-	Abf71588	23a33939
7	23a33939	-	-	-	-	09cf4f3c	2a6c7605

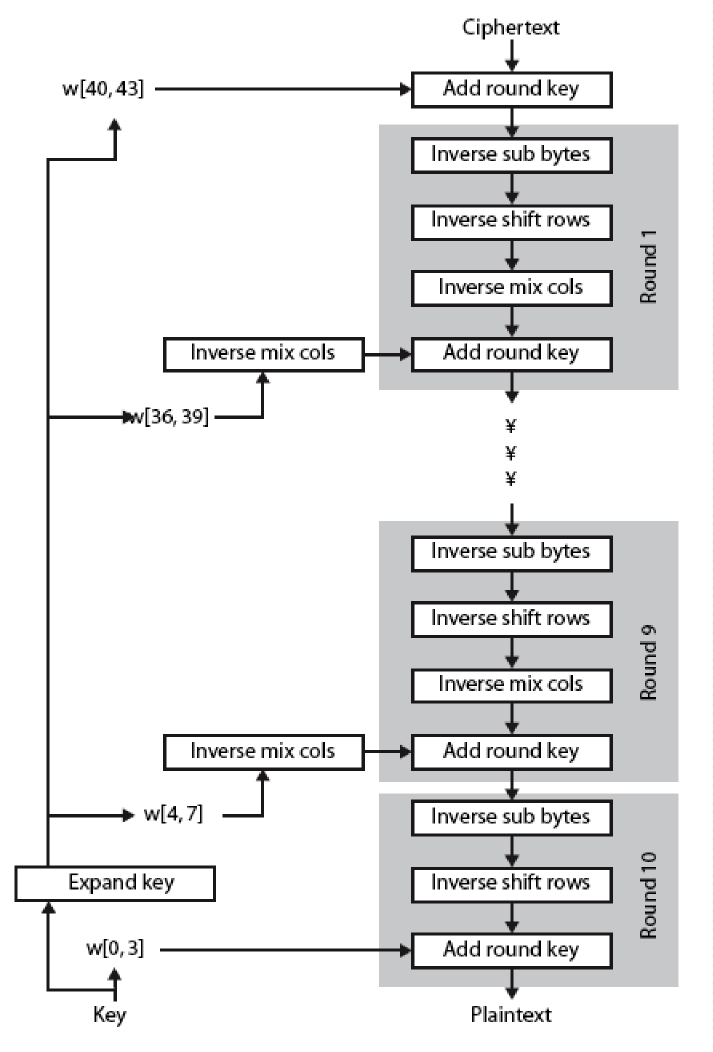
Key Expansion Rationale

- designed to resist known attacks
- design criteria included
  - knowing part key insufficient to find many more
  - invertible transformation
  - fast on wide range of CPU's
  - use round constants to break symmetry
  - diffuse key bits into round keys
  - enough non-linearity to hinder analysis
  - simplicity of description

AES Decryption

- AES decryption is not identical to encryption since steps done in reverse
- but can define an equivalent inverse cipher with steps as for encryption
  - but using inverses of each step
  - with a different key schedule
- works since result is unchanged when
  - swap byte substitution & shift rows
  - swap mix columns & add (tweaked) round key

AES Decryption



Implementation Aspects

- can efficiently implement on 8-bit CPU
  - byte substitution works on bytes using a table of 256 entries
  - shift rows is simple byte shift
  - add round key works on byte XOR’s
  - mix columns requires matrix multiply in GF(2^8) which works on byte values, can be simplified to use table lookups & byte XOR’s

Implementation Aspects

- can efficiently implement on 32-bit CPU
  - redefine steps to use 32-bit words
  - can precompute 4 tables of 256-words
  - then each column in each round can be computed using 4 table lookups + 4 XORs
  - at a cost of 4Kb to store tables
- designers believe this very efficient implementation was a key factor in its selection as the AES cipher

AES Security

- AES was designed after DES.
- Most of the known attacks on DES were already tested on AES.
- Brute-Force Attack
  - AES is definitely more secure than DES due to the larger-size key.
- Statistical Attacks
  - Numerous tests have failed to do statistical analysis of the ciphertext
- Differential and Linear Attacks
  - There are no differential and linear attacks on AES as yet.

Implementation Aspects

- The algorithms used in AES are simple and can be easily implemented using cheap processors and a minimum amount of memory.
- Very efficient
- Implementation was a key factor in its selection as the AES cipher