Advance Encryption Standard

Topics

- **Origin of AES**
- Basic AES
- Inside Algorithm
- Final Notes

Origins

- A replacement for DES was needed
  - Key size is too small
- Can use Triple-DES – but slow, small block
- US NIST issued call for ciphers in 1997
- 15 candidates accepted in Jun 98
- 5 were shortlisted in Aug 99

AES Competition Requirements

- Private key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- Stronger & faster than Triple-DES
- Provide full specification & design details
- Both C & Java implementations

AES Evaluation Criteria

- initial criteria:
  - security – effort for practical cryptanalysis
  - cost – in terms of computational efficiency
  - algorithm & implementation characteristics

- final criteria
  - general security
  - ease of software & hardware implementation
  - implementation attacks
  - flexibility (in en/decrypt, keying, other factors)

AES Shortlist

- After testing and evaluation, shortlist in Aug-99
  - MARS (IBM) - complex, fast, high security margin
  - RC6 (USA) - v. simple, v. fast, low security margin
  - Rijndael (Belgium) - clean, fast, good security margin
  - Serpent (Euro) - slow, clean, v. high security margin
  - Twofish (USA) - complex, v. fast, high security margin

- Found contrast between algorithms with
  - few complex rounds versus many simple rounds
  - Refined versions of existing ciphers versus new proposals

Rijndae: pronounce "Rain-Dahl"

The AES Cipher - Rijndael

- Rijndael was selected as the AES in Oct-2000
  - Designed by Vincent Rijmen and Joan Daemen in Belgium
  - Issued as FIPS PUB 197 standard in Nov-2001
- An **iterative** rather than **Feistel** cipher
  - processes data as block of 4 columns of 4 bytes (128 bits)
  - operates on entire data block in every round
- Rijndael design:
  - simplicity
  - has 128/192/256 bit keys, 128 bits data
  - resistant against known attacks
  - speed and code compactness on many CPUs

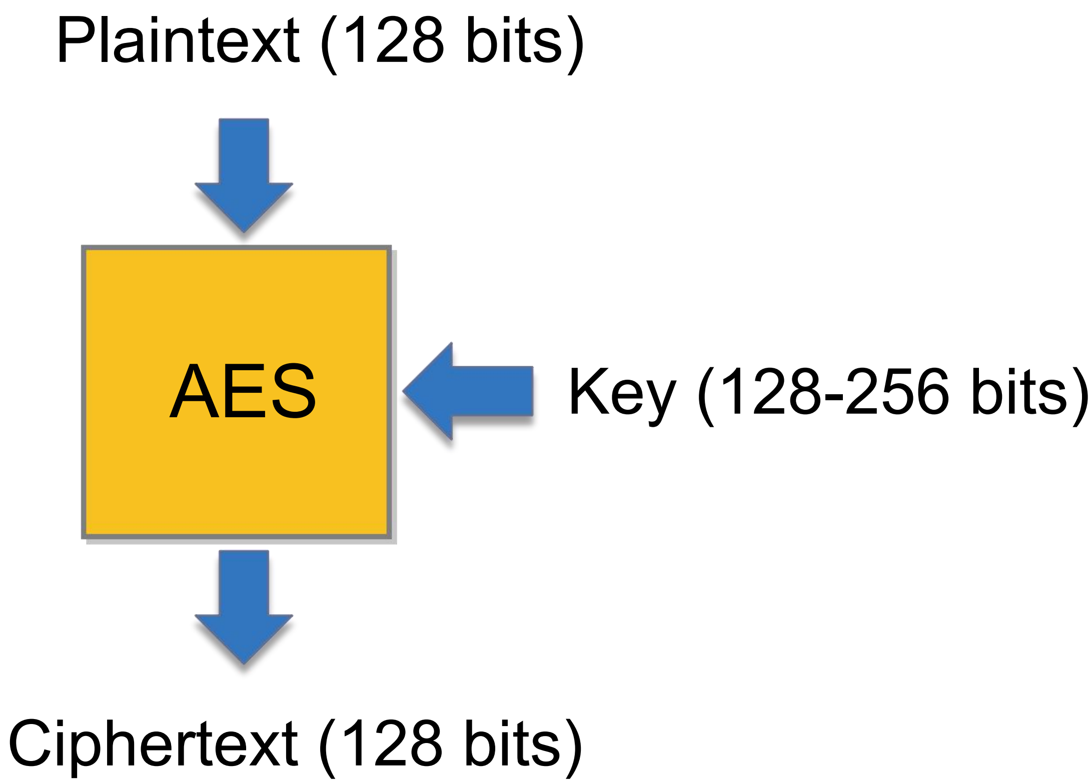
V. Rijmen

J. Daemen

Topics

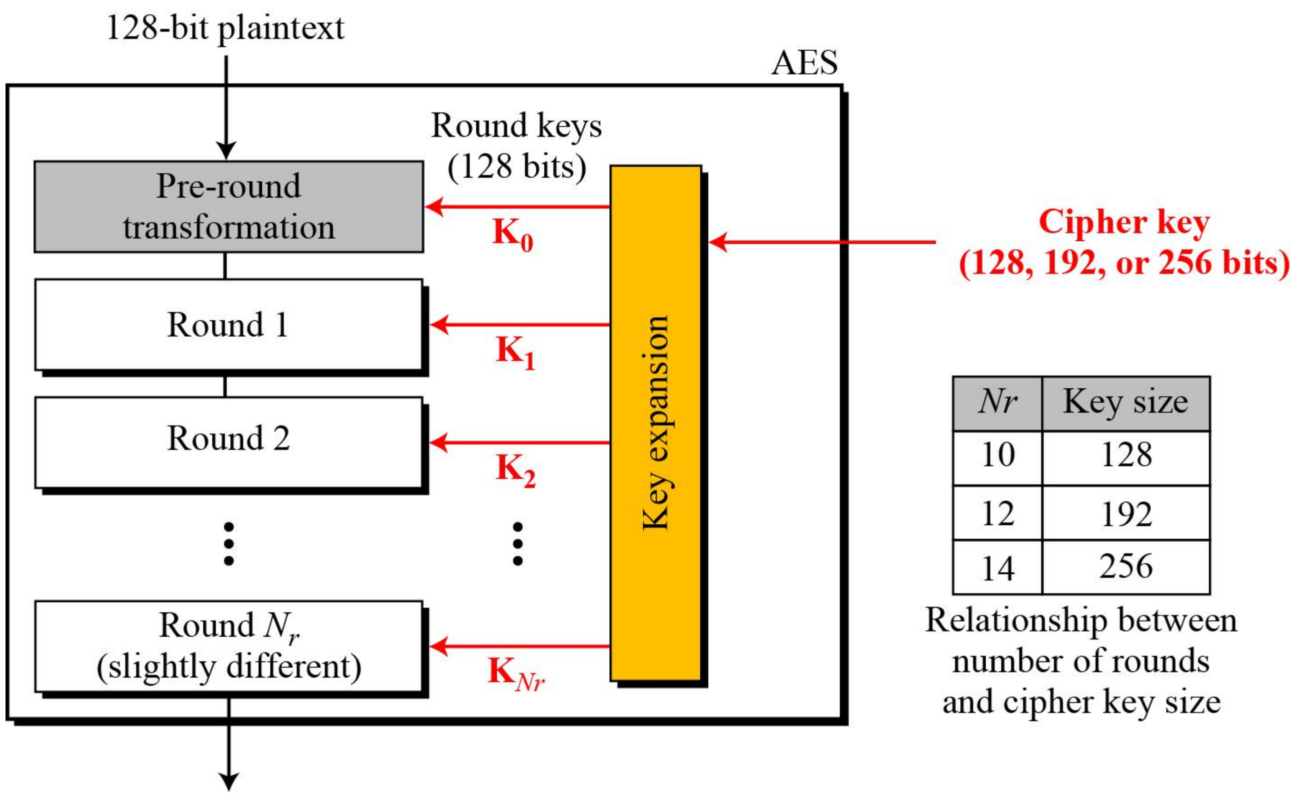
- Origin of AES
- **Basic AES**
- Inside Algorithm
- Final Notes

AES Conceptual Scheme



Multiple rounds

- Rounds are (almost) identical
  - First and last round are a little different



Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size

High Level Description

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

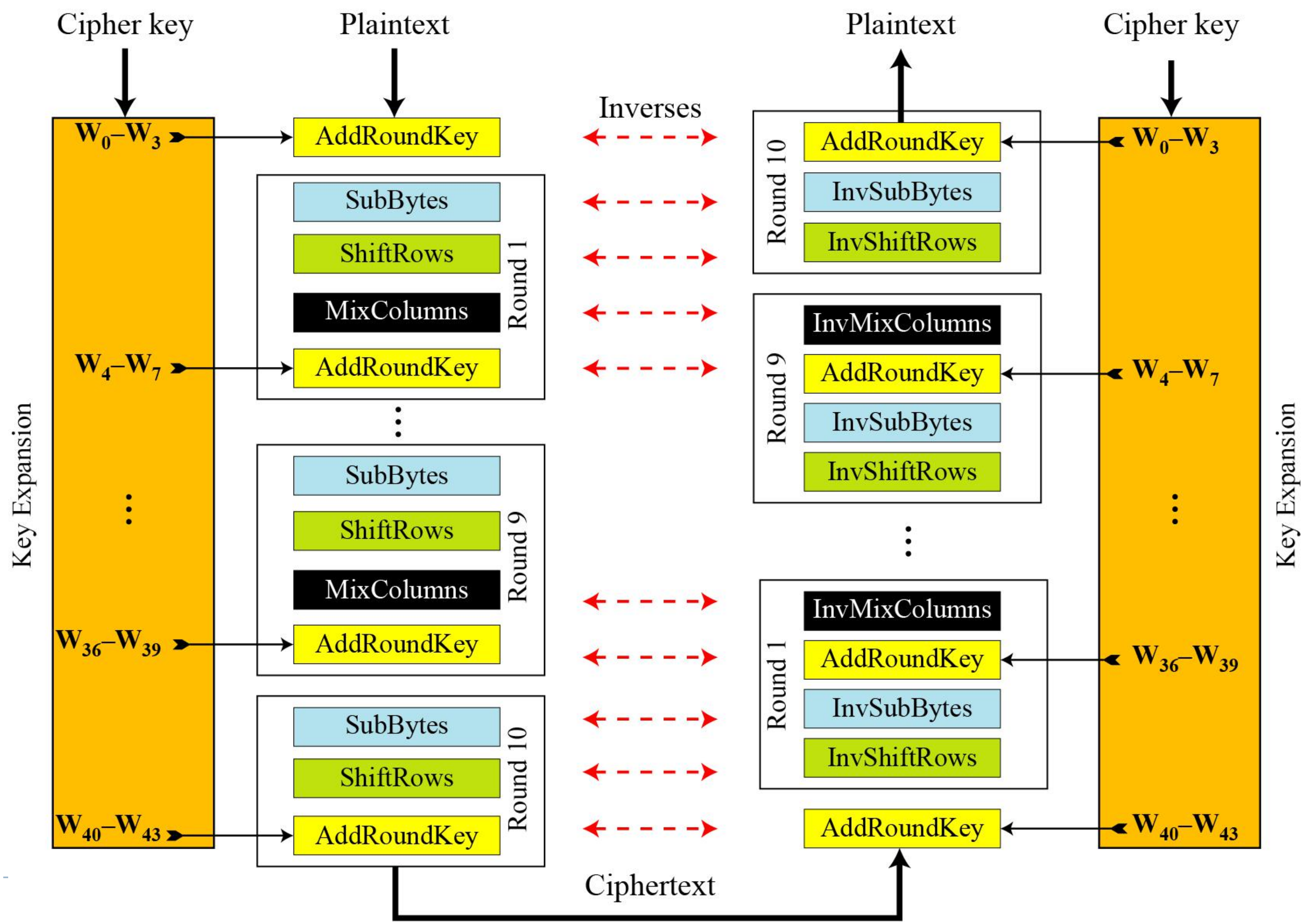
- SubBytes : non-linear substitution step
- ShiftRows : transposition step
- MixColumns : mixing operation of each column.
- AddRoundKey

Final Round

- SubBytes
- ShiftRows
- AddRoundKey

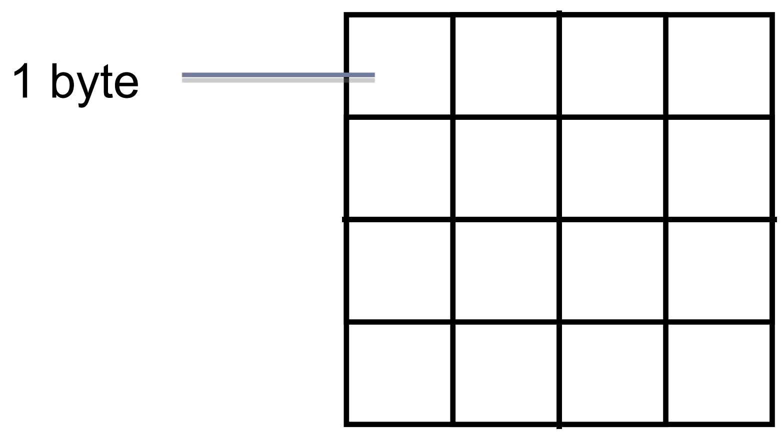
No MixColumns

Overall Structure

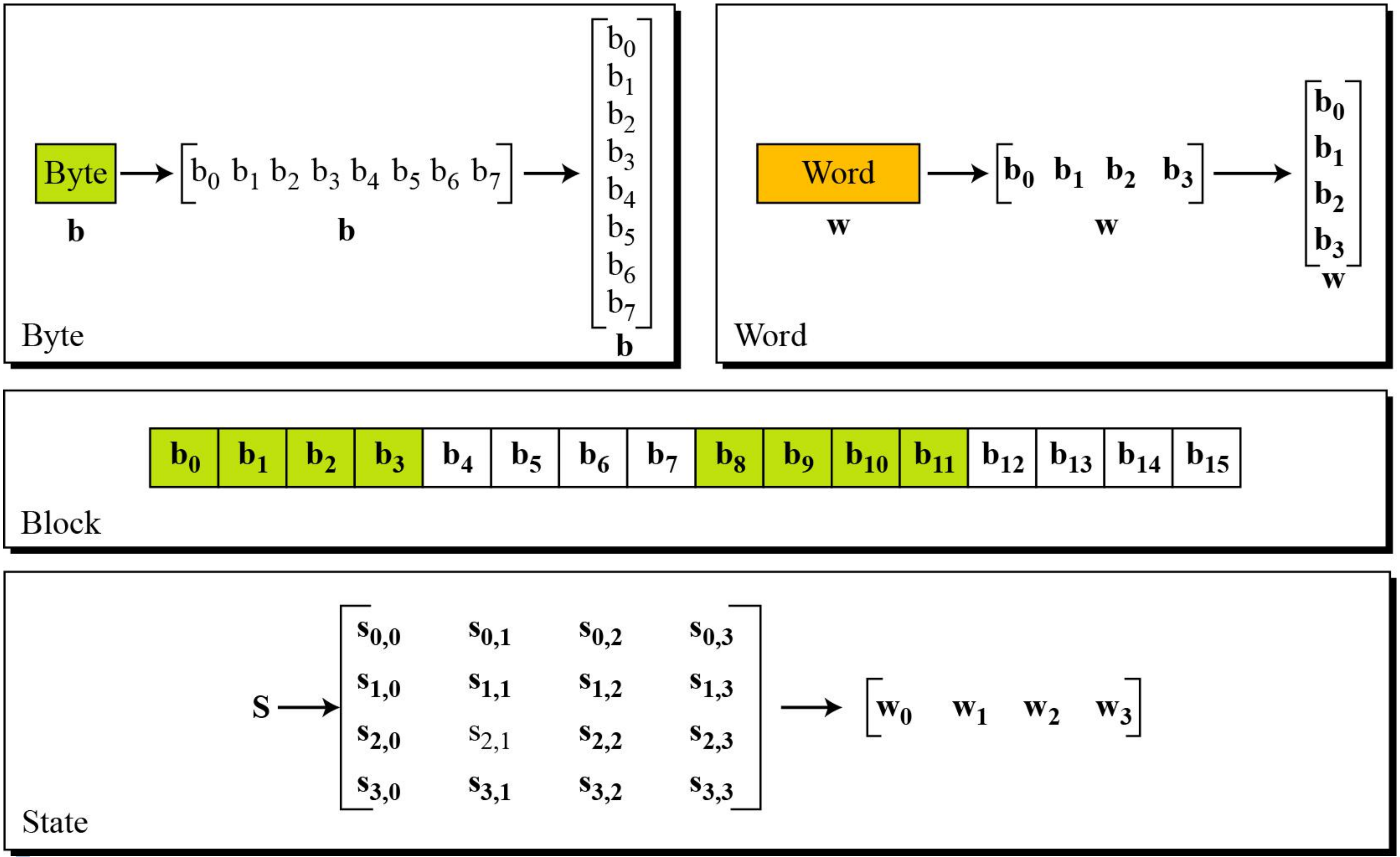


128-bit values

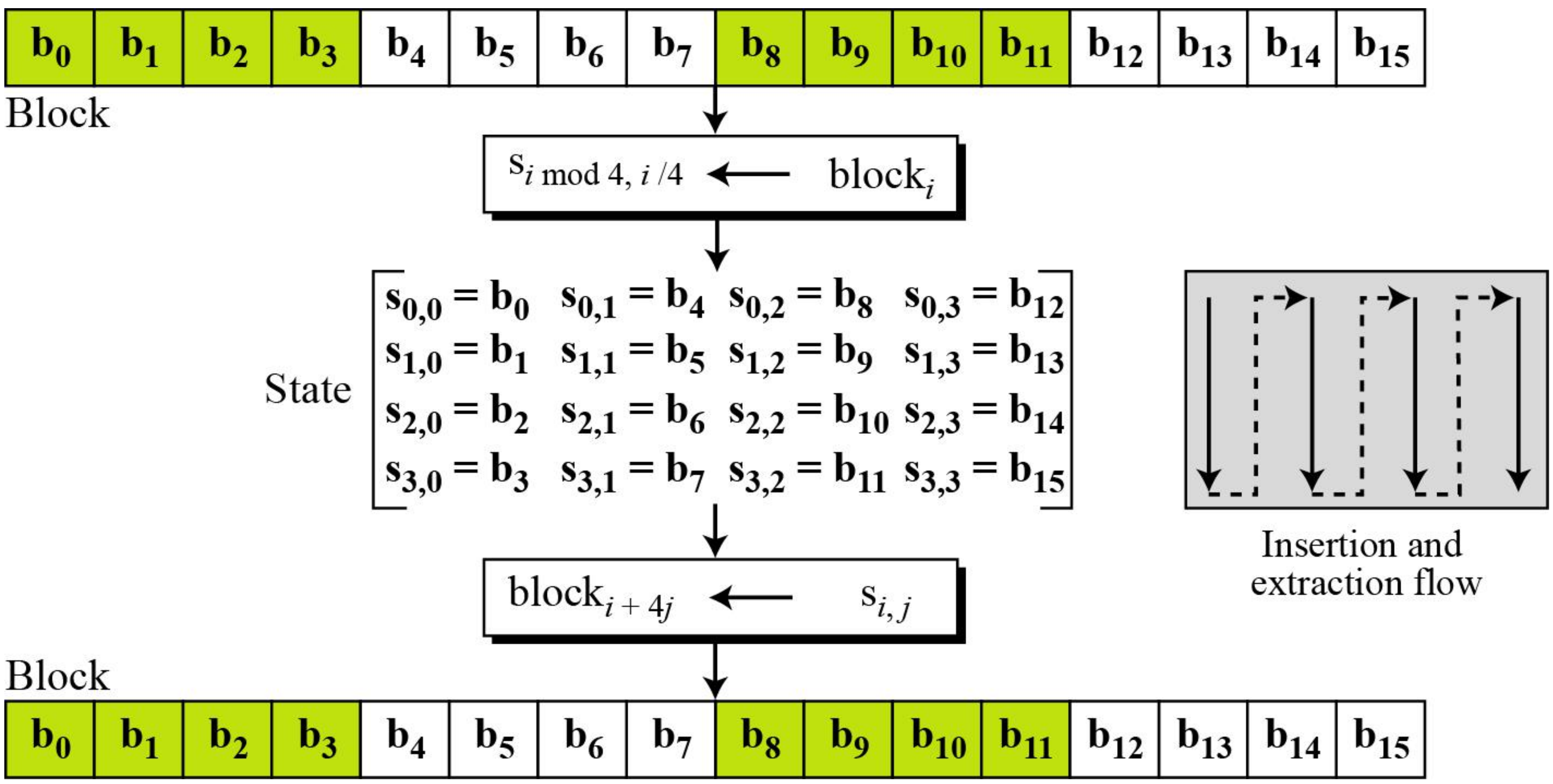
- Data block viewed as 4-by-4 table of bytes
- Represented as 4 by 4 matrix of 8-bit bytes.
- Key is expanded to array of 32 bits words



Data Unit


Byte
b
$[b_0\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7]$
b
Byte
Word
w
$[b_0\ b_1\ b_2\ b_3]$
w
Word
b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11} b_{12} b_{13} b_{14} b_{15}
Block
S
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$[w_0\ w_1\ w_2\ w_3]$
State

Unit Transformation



Changing Plaintext to State

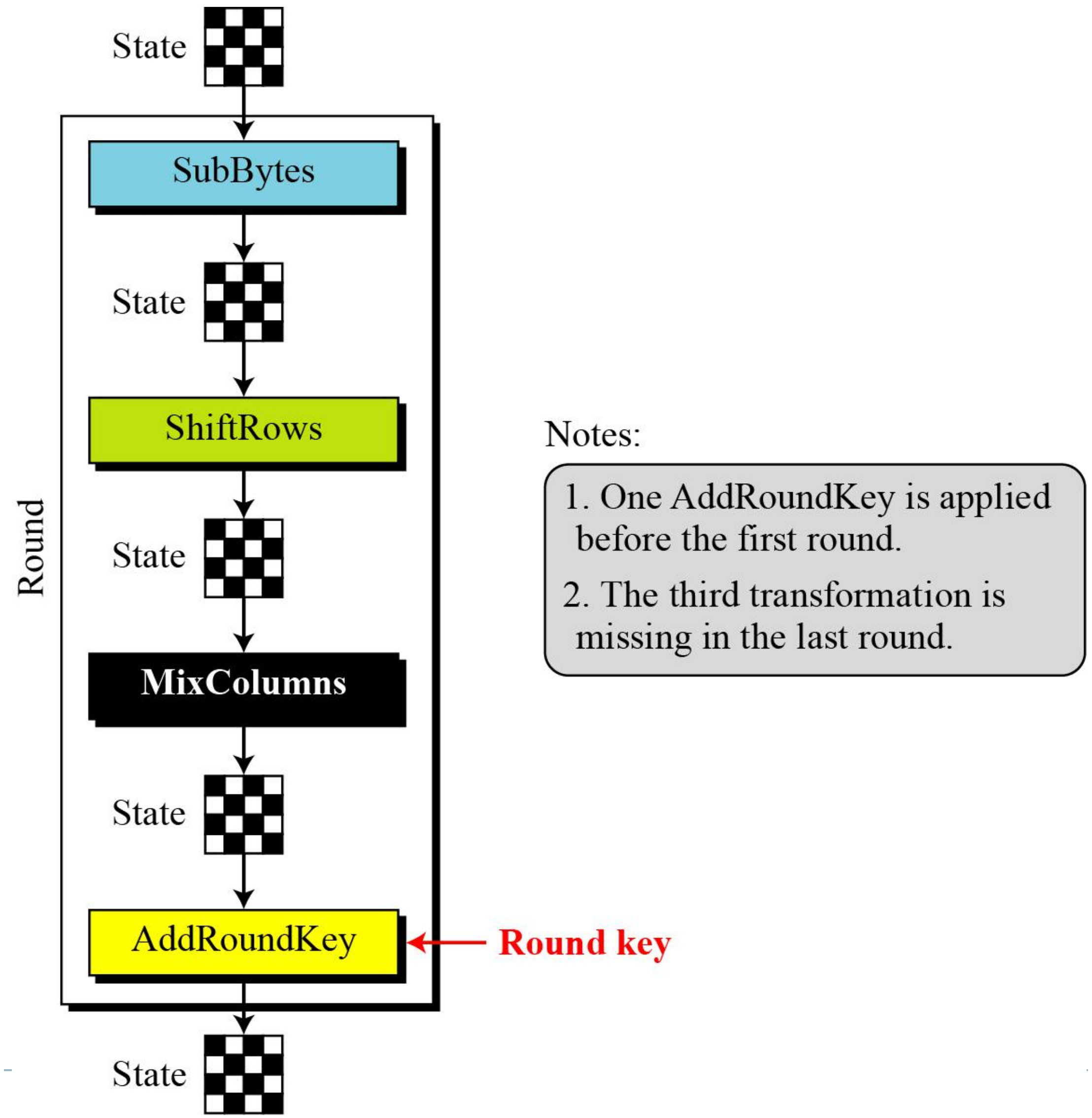
Text	A	E	S	U	S	E	S	A	M	A	T	R	I	X	Z	Z
Hexadecimal	00	04	12	14	12	04	12	00	0C	00	13	11	08	23	19	19

$$\begin{bmatrix} 00 & 12 & 0C & 08 \\ 04 & 04 & 00 & 23 \\ 12 & 12 & 13 & 19 \\ 14 & 00 & 11 & 19 \end{bmatrix} \text{State}$$

Topics

- Origin of AES
- Basic AES
- **Inside Algorithm**
- Final Notes

Details of Each Round



SubBytes: Byte Substitution

- A simple substitution of each byte
  - provide a confusion

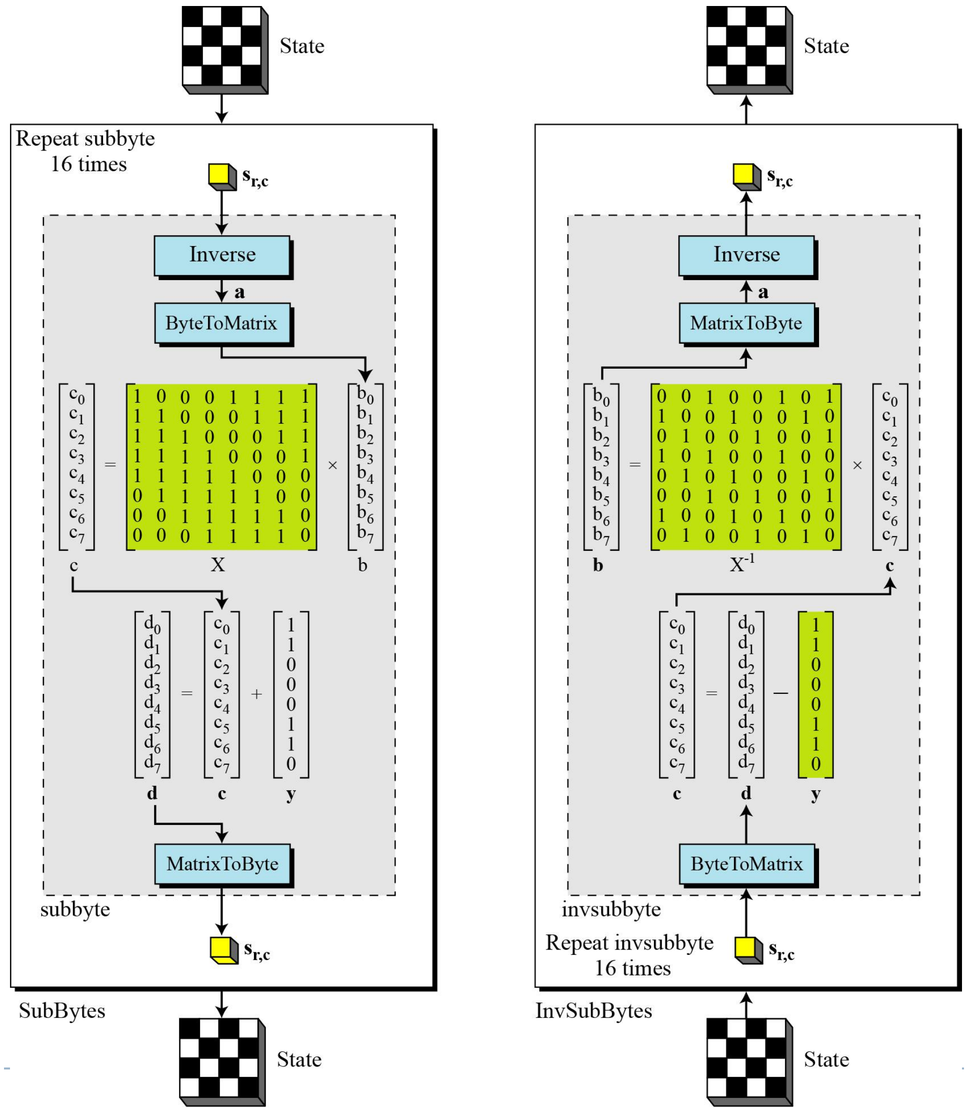
- Uses one S-box of 16x16 bytes containing a permutation of all 256 8-bit values

- Each byte of state is replaced by byte indexed by row (left 4-bits) & column (right 4-bits)
  - eg. 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)

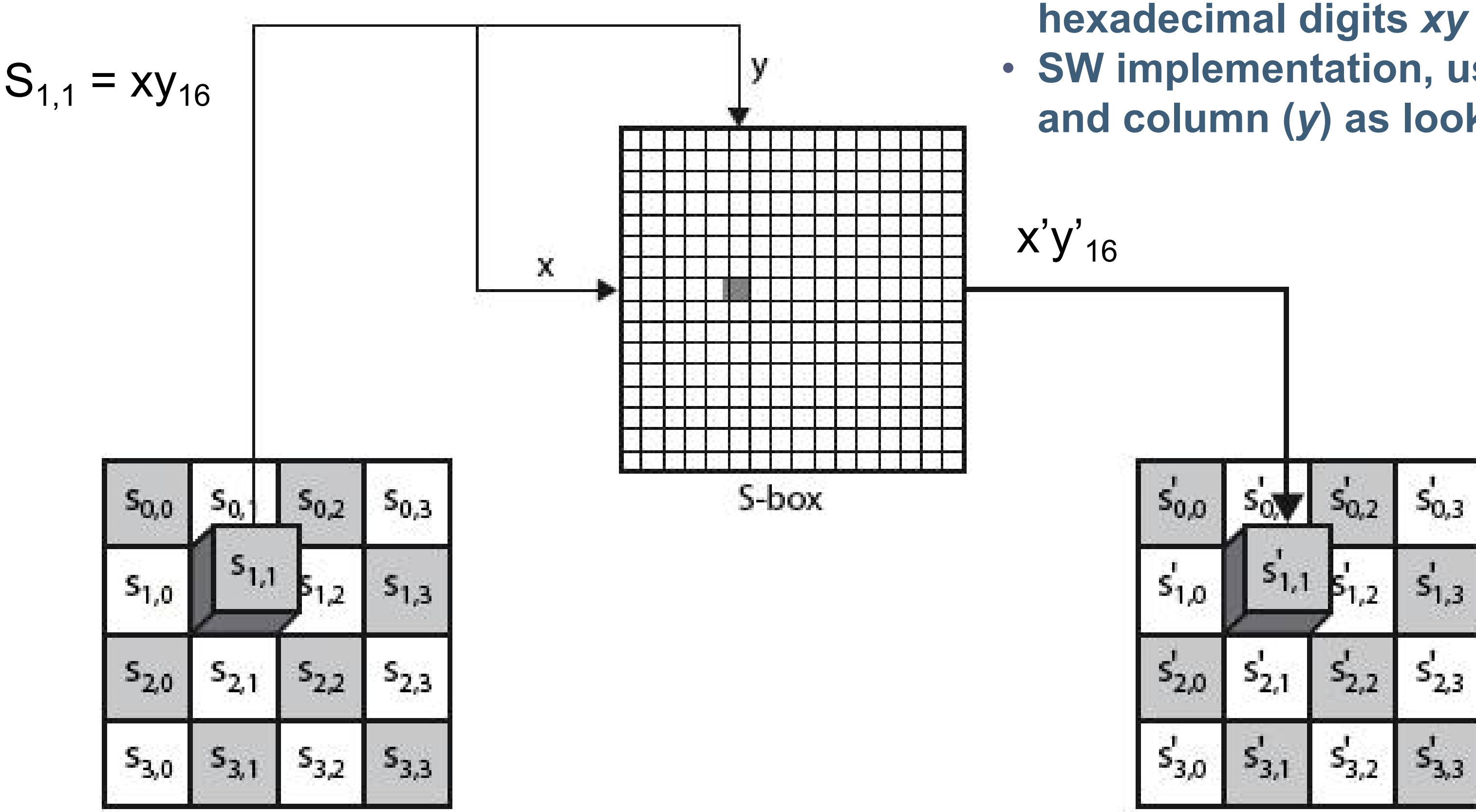
Galois : pronounce “Gal-Wa”

SubBytes and InvSubBytes



SubBytes Operation

- The SubBytes operation involves 16 independent byte-to-byte transformations.
  - Interpret the byte as two hexadecimal digits *xy*
  - SW implementation, use row (*x*) and column (*y*) as lookup pointer



SubBytes Table

- Implement by Table Lookup

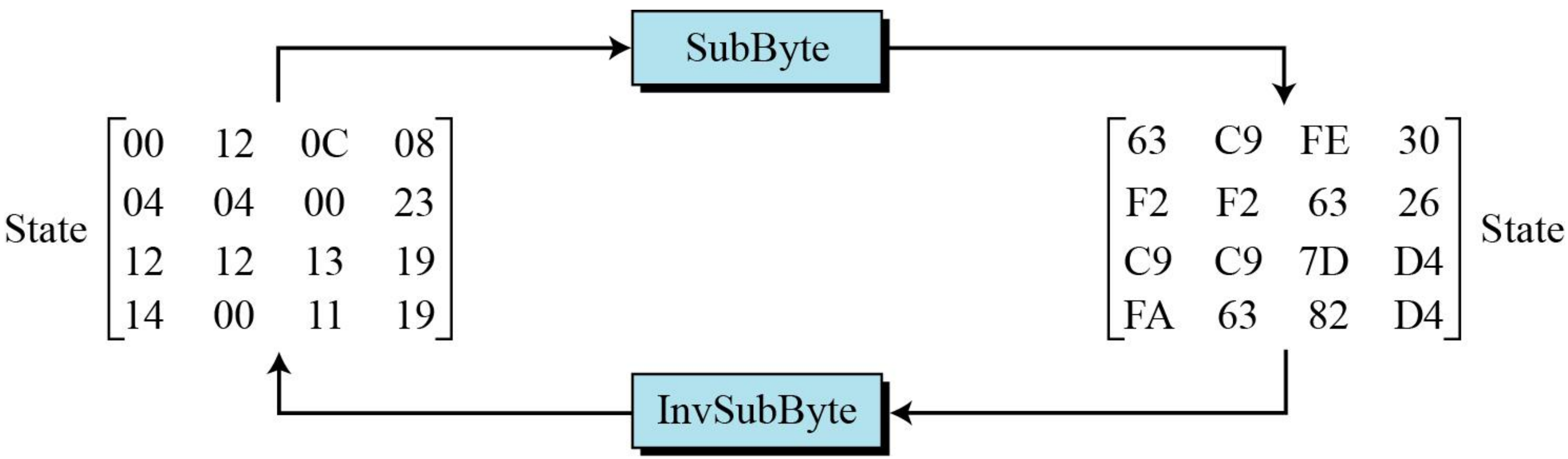
		y															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
x	**0**	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12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

InvSubBytes Table

		y															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
x	**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

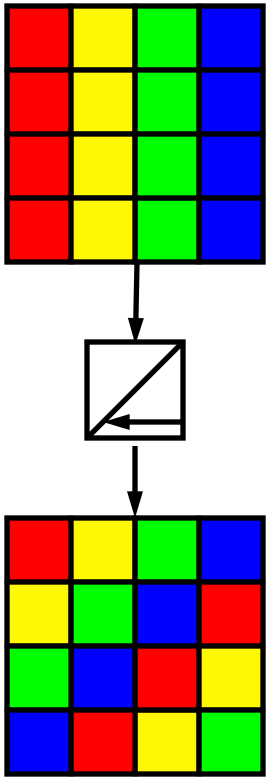
Sample SubByte Transformation

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

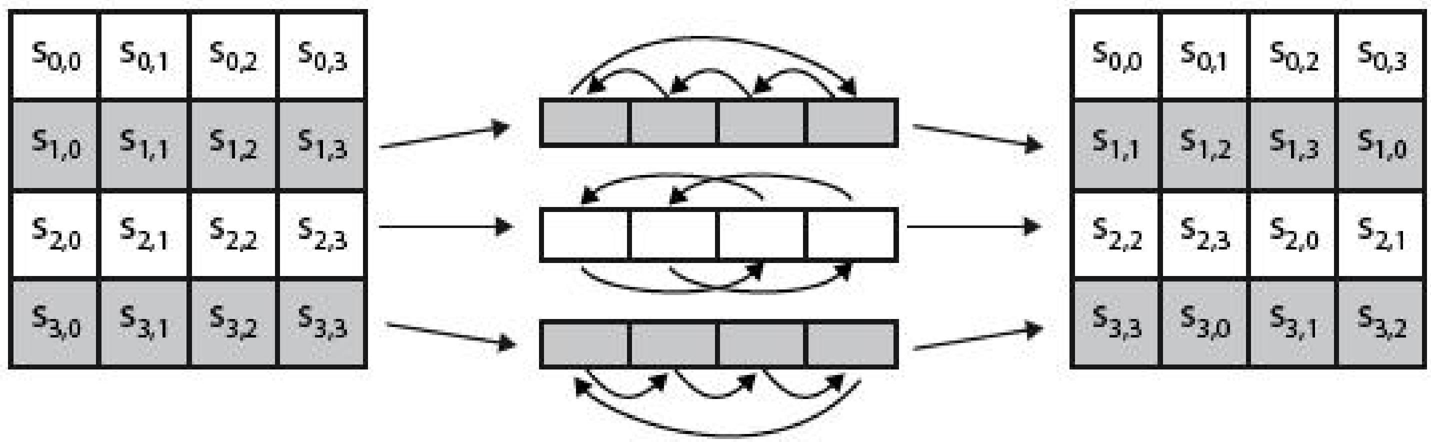


ShiftRows

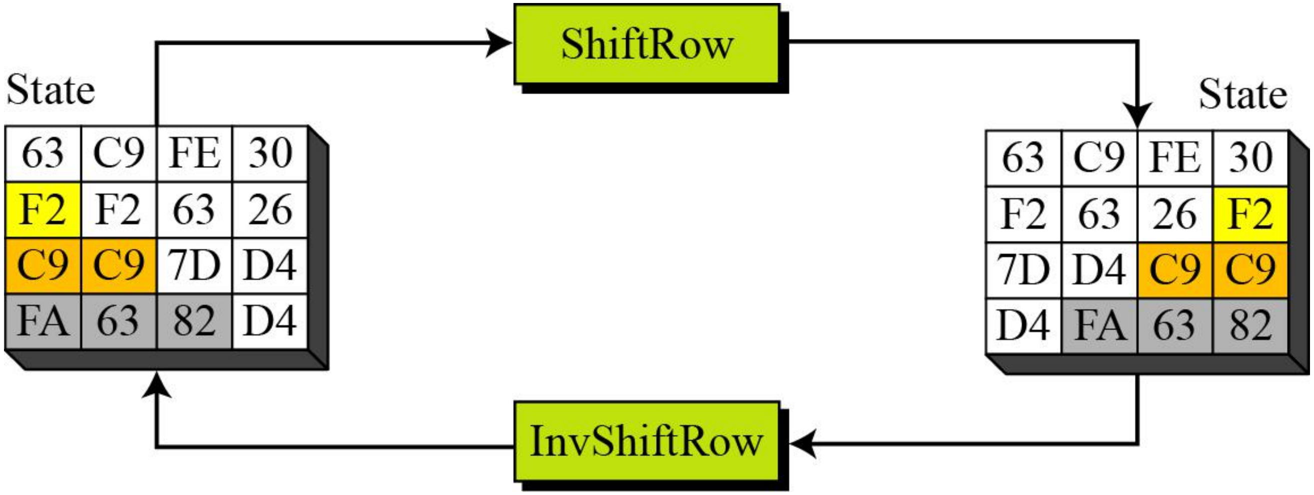
- Shifting, which permutes the bytes.
- A circular byte shift in each each
  - 1st row is unchanged
  - 2nd row does 1 byte circular shift to left
  - 3rd row does 2 byte circular shift to left
  - 4th row does 3 byte circular shift to left
- In the encryption, the transformation is called ShiftRows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



ShiftRows Scheme



ShiftRows and InvShiftRows



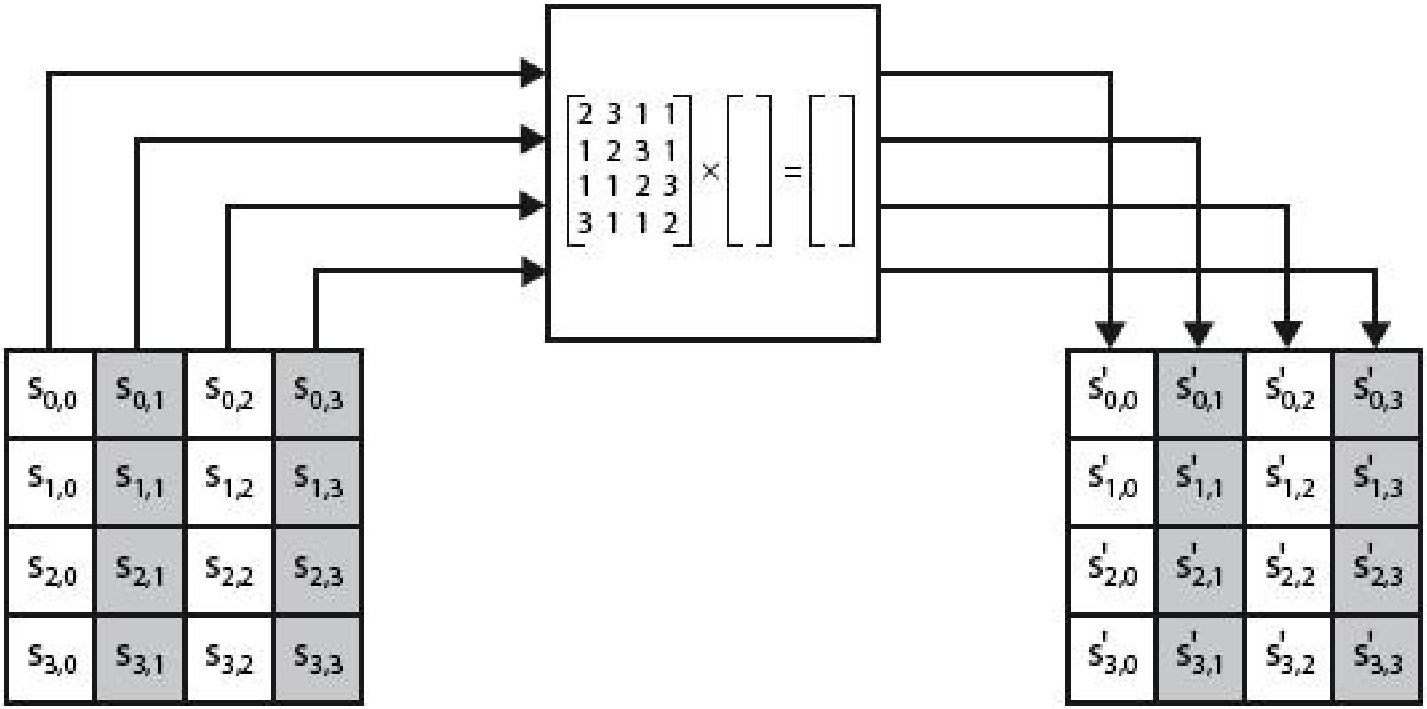
MixColumns

- 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 $m(x) = x^8+x^4+x^3+x+1$

$$\begin{matrix} a\mathbf{x} + b\mathbf{y} + c\mathbf{z} + d\mathbf{t} \\ e\mathbf{x} + f\mathbf{y} + g\mathbf{z} + h\mathbf{t} \\ i\mathbf{x} + j\mathbf{y} + k\mathbf{z} + l\mathbf{t} \\ m\mathbf{x} + n\mathbf{y} + o\mathbf{z} + p\mathbf{t} \end{matrix} \rightarrow \begin{bmatrix} \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \times \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \\ \mathbf{t} \end{bmatrix}$$

New matrix **Constant matrix** Old matrix

MixClumns Scheme



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

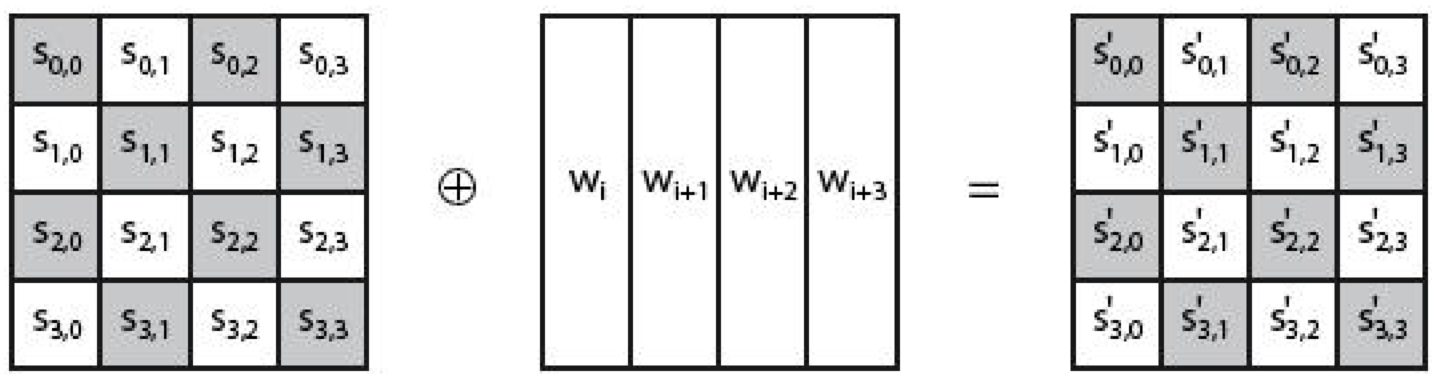
MixColumn and InvMixColumn

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

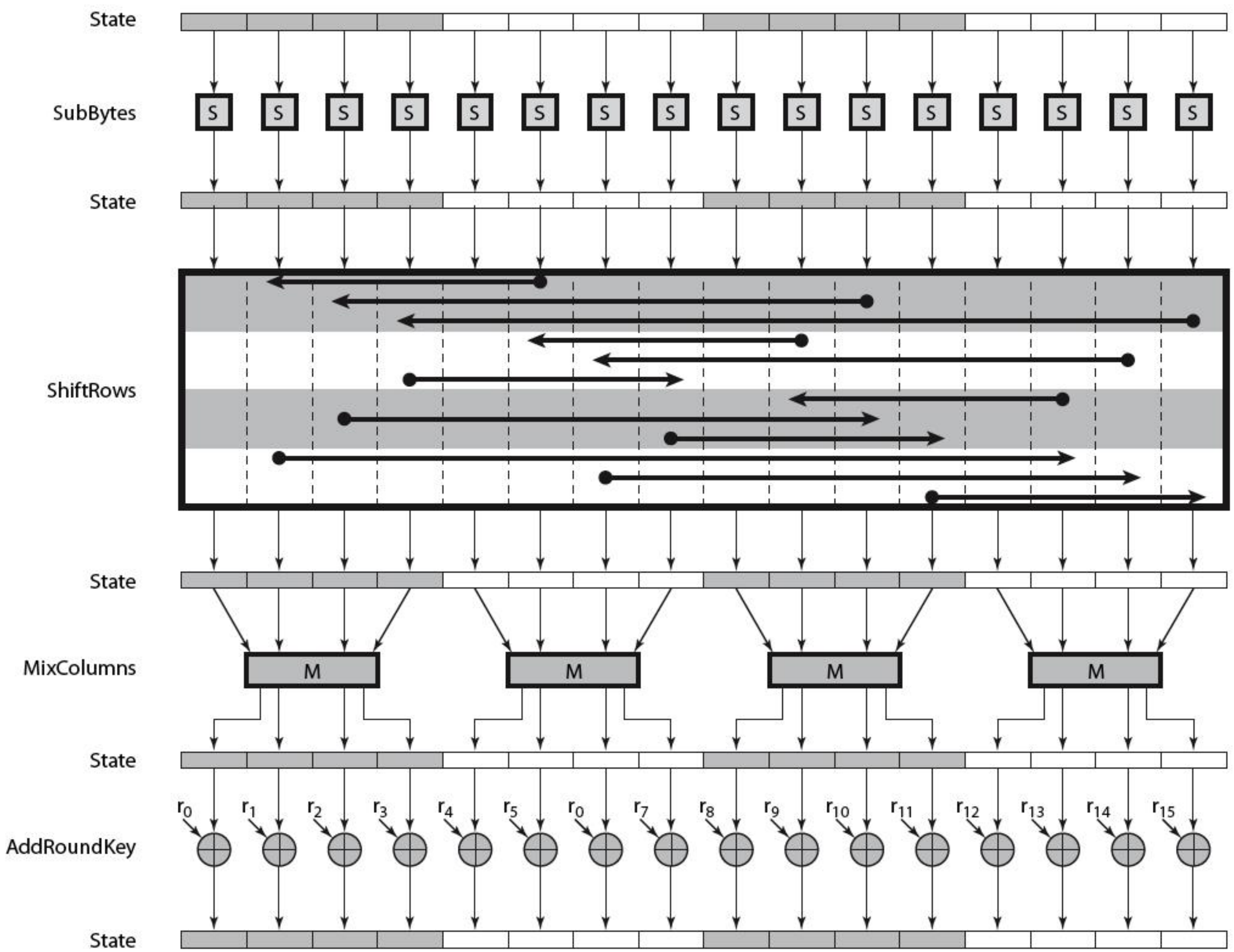
AddRoundKey

- 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
- Designed to be as simple as possible

AddRoundKey Scheme



AES Round



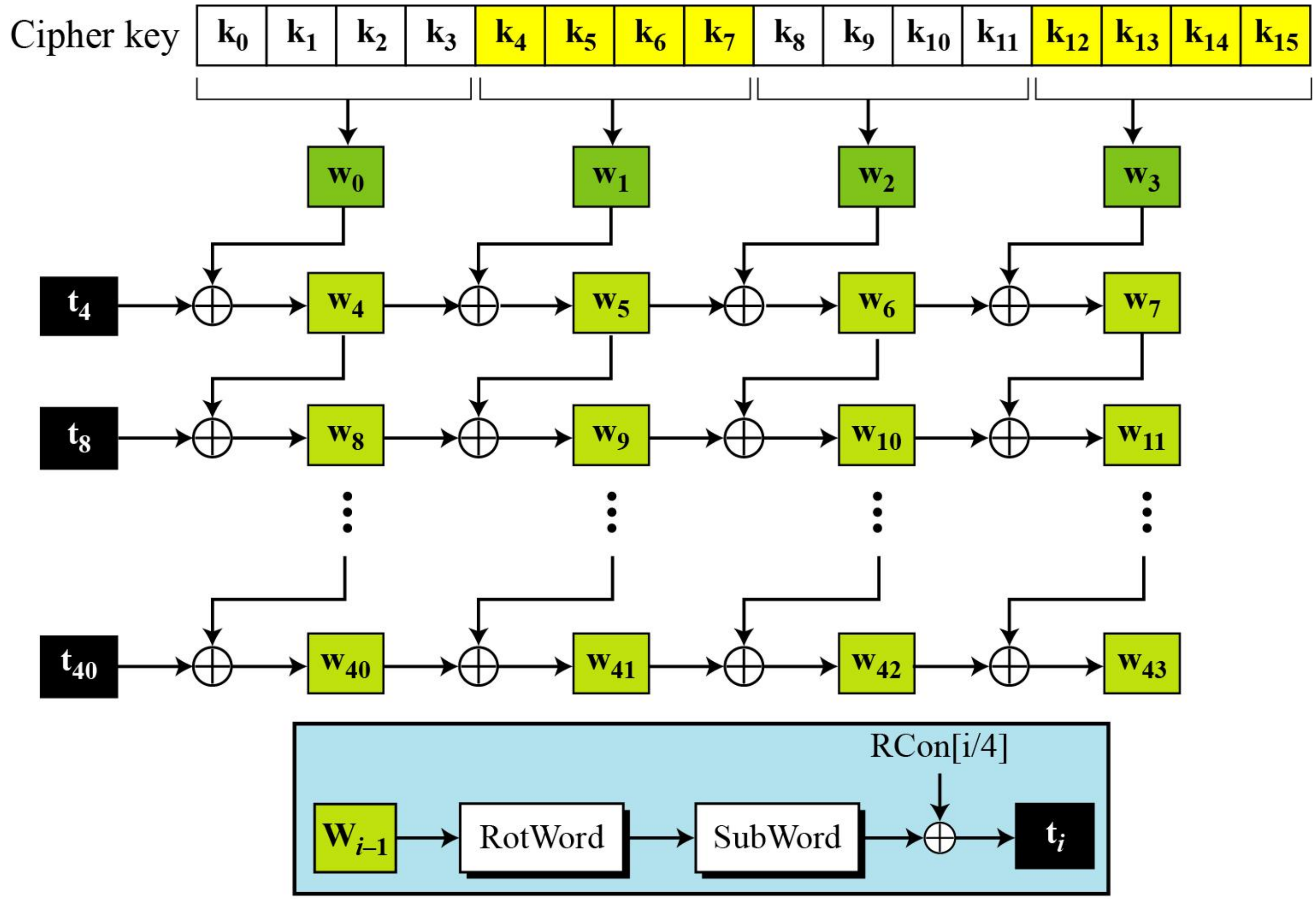
State
SubBytes
S S S S S S S S S S S S S S S S
State
ShiftRows
State
MixColumns
M M M M
State
AddRoundKey
r_0 r_1 r_2 r_3 r_4 r_5 r_0 r_7 r_8 r_9 r_{10} r_{11} r_{12} r_{13} r_{14} r_{15}
State

AES Key Scheduling

- takes 128-bits (16-bytes) key and expands into array of 44 32-bit words

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	$\mathbf{w}_{4N_r}$	$\mathbf{w}_{4N_r+1}$	$\mathbf{w}_{4N_r+2}$	$\mathbf{w}_{4N_r+3}$

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 (1^{st} Round)

- Example of expansion of a 128-bit cipher key

  Cipher key = 2b**7e**15**16**28**ae**d2**a6**ab**f7**15**88**09**cf**4f**3c**

  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

Topics

- Origin of AES
- Basic AES
- Inside Algorithm
- **Final Notes**

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 so simple that they 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

- AES animation:
  - http://www.cs.bc.edu/~straubin/cs381-05/blockciphers/rijndael_ingles2004.swf