Advanced Encryption Standard
AES

AES Requirements

- private key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- stronger & faster than Triple-DES
- active life of 20-30 years (+ archival use)
- provide full specification & design details
- both C & Java implementations
- NIST have released all submissions & unclassified analyses

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)

15 Original Candidates

Algorithm	Submitter
CAST-256	Entrust Technologies Inc.
CRYPTON	Future Systems, Inc.
DEAL	Richard Outerbridge, Lars Knudsen
DFC	CNRS – Centre National pour la Recherche Scientifique – Ecole Normale Superieure
E2	NTT – Nippon Telegraph and Telephone
FROG	TecApro Internacional S.A.
HPC	Rich Schroeppel

15 Original Candidates

Algorithm	Submitter
LOK197	Lawrie Brown, Josef Pieprzyk, Jennifer Seberry
MAGENTA	Deutsche Telekom AG
MARS	IIBM
RC6	RSA Laboratories
RIJNDAEL	Joan Daemen, Vincent Rijmen
SAFER+	Cylink Corporation
SERPENT	Ross Anderson, Eli Biham, Lars Knudsen
TWOFISH	Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Neils Ferguson

AES Shortlist

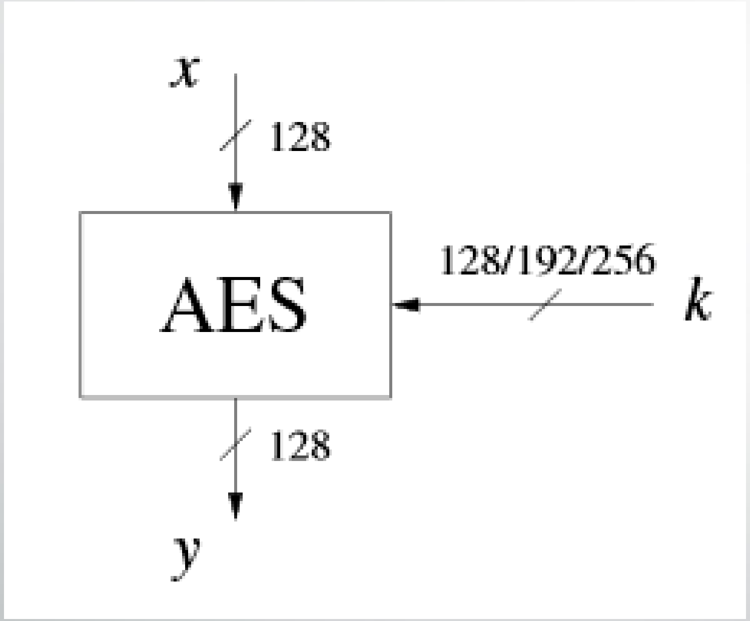
- after testing and evaluation, shortlist in Aug-99:
  - MARS (IBM) - complex, fast, high security margin
  - RC6 (USA) - v. simple, v. fast, low security margin
  - Rijndael (Belgium) - clean, fast, good security margin
  - Serpent (Euro) - slow, clean, v. high security margin
  - Twofish (USA) - complex, v. fast, high security margin
- then subject to further analysis & comment
- saw contrast between algorithms with
  - few complex rounds verses many simple rounds
  - which refined existing ciphers verses new proposals

The AES Cipher - Rijndael

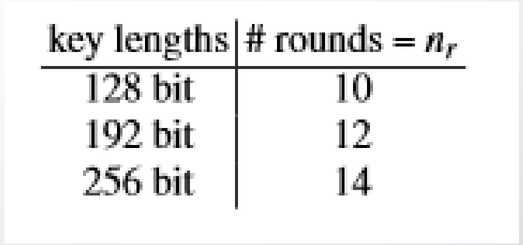
- designed by Rijmen-Daemen in Belgium
- has 128/192/256 bit keys, 128 bit data
- an **iterative** rather than **Feistel** cipher
    - processes data as block of 4 columns of 4 bytes
    - operates on entire data block in every round
- designed to be:
    - resistant against known attacks
    - speed and code compactness on many CPUs
    - design simplicity

The AES Cipher - Rijndael

AES input/output parameters



Key lengths and number of rounds for AES



key lengths	# rounds = n_r
128 bit	10
192 bit	12
256 bit	14

Fini

0	0	1	1	0	0	1	0
b7	b6	b5	b4	b3	b2	b1	b0

- In the Advanced Encryption Standard (AES) all operations are performed on 8-bit bytes

- AES uses the finite field GF(2^8)
  - $b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$

    $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$

32 = 0011 0010= $x^5 + x^4 + x$

$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$

$\{b_7,=0, b_6,=0 b_5 =1\ b_4=1\ b_3=0\ b_2=0\ b_1,=1\ b_0=0\}$

$x^5 + x^4 + x$

- The arithmetic operations of addition, multiplication, and division are performed over the finite field GF(2^8)

- A field is a set in which we can do addition, subtraction, multiplication, and division without leaving the set

- An example of a finite field (one with a finite number of elements) is the set $Z_p = \{0,1,\ldots,p-1\}$, where p is a prime number and in which arithmetic is carried out modulo p

Group

- a set of elements or “numbers”
  - A generalization of usual arithmetic
- obeys:
  - closure: a.b also in G
  - associative law: (a.b).c = a.(b.c)
  - has identity e: e.a = a.e = a
  - has inverses a^{-1}: $a.a^{-1} = e$
- if commutative a.b = b.a
  - then forms an **abelian group**

Cyclic Group

- define **exponentiation** as repeated application of operator
  - example: $a^3 = a.a.a$
- and let identity be: $e=a^0$
- a group is cyclic if every element is a power of some fixed element
  - i.e. $b = a^k$ for some **a** and every **b** in group
- a is said to be a generator of the group
- Example: positive numbers with addition

Ring

- a set of “numbers” with two operations (addition and multiplication) which are:
- an abelian group with addition operation
- multiplication:
  - has closure
  - is associative
  - distributive over addition: $a(b+c) = ab + ac$
- In essence, a ring is a set in which we can do addition, subtraction

  [$a - b = a + (-b)$], and multiplication without leaving the set.
- With respect to addition and multiplication, the set of all *n*-square matrices over the real numbers form a ring.

Ring

- if multiplication operation is commutative, it forms a **commutative ring**
- if multiplication operation has an identity element and no zero divisors (ab=0 means either a=0 or b=0), it forms an **integral domain**
- The set of Integers with usual + and x is an integral domain

Field

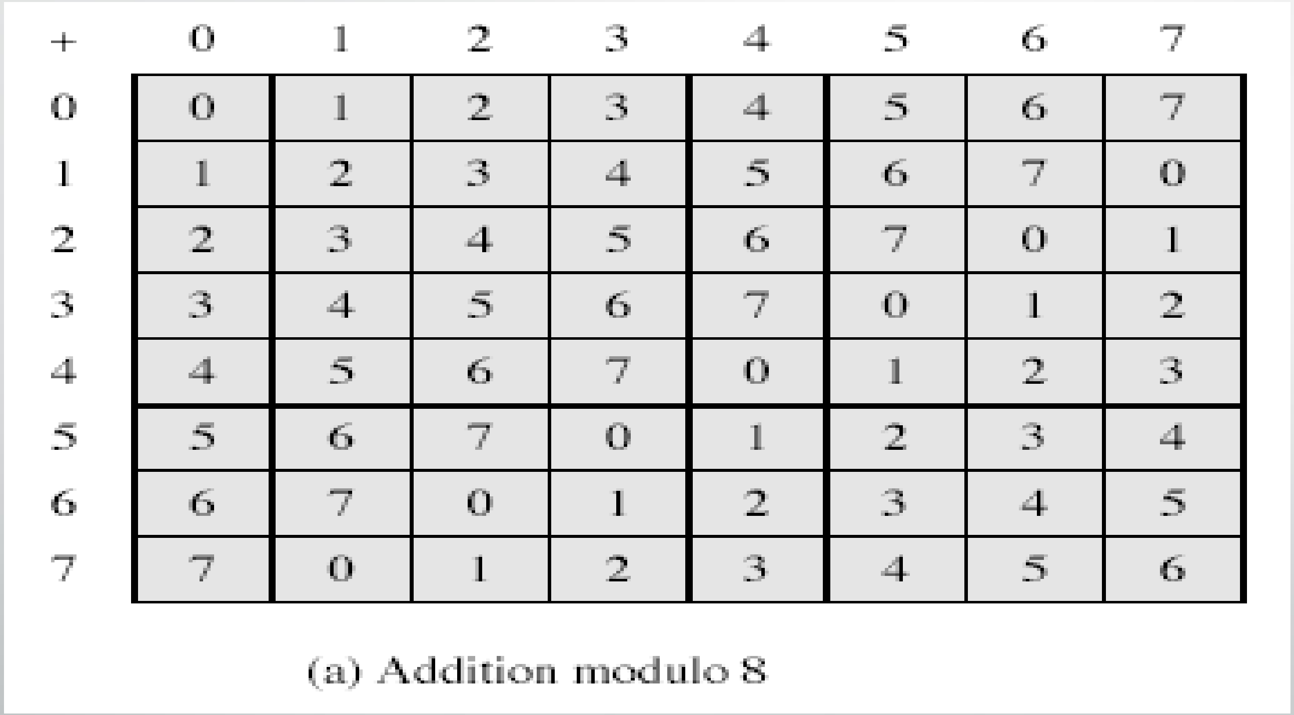
- a set of numbers with two operations:
  - Addition and multiplication
  - F is an integral domain
  - F has multiplicative reverse
    - For each a in F other than 0, there is an element b such that ab=ba=1
- In essence, a field is a set in which we can do addition, subtraction, multiplication, and division without leaving the set.
  - Division is defined with the following rule: $a/b = a\ (b^{-1})$
- Examples of fields: rational numbers, real numbers, complex numbers. Integers are NOT a field.

Modular Arithmetic

- can do modular arithmetic with any group of integers:

  $Z_n = \{0, 1, \ldots , n-1\}$
- form a commutative ring for addition
- with an additive identity (Table 4.2)
- some additional properties
  - if $(a+b)\equiv(a+c) \bmod n$ then $b\equiv c \bmod n$
  - but $(ab)\equiv(ac) \bmod n$ then $b\equiv c \bmod n$ only if a is relatively prime to n

Modulo 8 Example



+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

(a) Addition modulo 8

Galois Fields

- finite fields play a key role in many cryptography algorithms
- can show number of elements in any finite field **must** be a power of a prime number p^n
- known as Galois fields
- denoted $GF(p^n)$
- in particular often use the fields:
  - $GF(p)$
  - $GF(2^n)$

Galois Fields GF(p)

- GF(p) is the set of integers {0,1, … , p-1} with arithmetic operations modulo prime p
- these form a finite field
  - since have multiplicative inverses
- hence arithmetic is "well-behaved" and can do addition, subtraction, multiplication, and division without leaving the field GF(p)
  - Division depends on the existence of multiplicative inverses.

  Why p has to be prime?

Polynomial Arithmetic

- can compute using polynomials
- several alternatives available
  - ordinary polynomial arithmetic
  - poly arithmetic with coefficients mod p
  - poly arithmetic with coefficients mod p and polynomials mod another polynomial M(x)
- Motivation: use polynomials to model Shift and XOR operations

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n} a_i x^i$$

Ordinary Polynomial Arithmetic

- add or subtract corresponding coefficients
- multiply all terms by each other
- eg
  - let $f(x) = x^3 + x^2 + 2$ and $g(x) = x^2 – x + 1$

$f(x) + g(x) = x^3 + 2x^2 – x + 3$

$f(x) – g(x) = x^3 + x + 1$

$f(x) \text{ x } g(x) = x^5 + 3x^2 – 2x + 2$

Polynomial Arithmetic with Modulo Coefficients

- when computing value of each coefficient, modulo some value
- could be modulo any prime
- but we are most interested in mod 2
  - ie all coefficients are 0 or 1
  - eg. let $f(x) = x^3 + x^2$ and $g(x) = x^2 + x + 1$

$f(x) + g(x) = x^3 + x^2 + x^2 + x + 1 = x^3 + 2*x^2 + x + 1= x^3 + 0*x^2 + x + 1= x^3 + x + 1$

$f(x) \text{ x } g(x) = x^5 + x^2$

Modular Polynomial Arithmetic

- Given any polynomials f,g, can write in the form:
  - $f(x) = q(x)\,g(x) + r(x)$
  - can interpret $r(x)$ as being a remainder
  - $r(x) = f(x) \bmod g(x)$
- if have no remainder say $g(x)$ divides $f(x)$
- if $g(x)$ has no divisors other than itself & 1 say it is **irreducible** (or **prime**) polynomial
- Modular polynomial arithmetic modulo an irreducible polynomial forms a field
  - Check the definition of a field

Polynomial GCD

- can find greatest common divisor for polys
- GCD: the one with the greatest degree
  - $c(x)$ = GCD($a(x)$, $b(x)$) if $c(x)$ is the poly of greatest degree which divides both $a(x)$, $b(x)$
  - can adapt Euclid’s Algorithm to find it:
  - EUCLID[$a(x)$, $b(x)$]

```
1. A(x) = a(x); B(x) = b(x)
2. if B(x) = 0 return A(x) = gcd[a(x), b(x)]
3. R(x) = A(x) mod B(x)
4. A(x) ¨ B(x)
5. B(x) ¨ R(x)
6. goto 2
```

Modular Polynomial Arithmetic

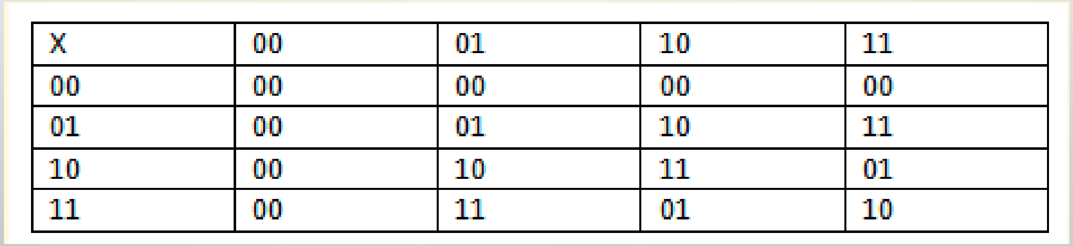
- can compute in field GF(2^n)
  - polynomials with coefficients modulo 2
  - whose degree is less than n
  - Coefficients always modulo 2 in an operation
  - hence must modulo an irreducible polynomial of degree n (for multiplication only)
- form a finite field
- can always find an inverse
  - can extend Euclid’s Inverse algorithm to find

Example GF(2^2)

Polynomial arithmetic Modulo m(x)= $x^2 + x + 1$

Addition(XOR)	00	01	10	11
p(x)=0= 00	00	01	10	11
p(x)=1= 01	01	00	11	10
p(x)=x = 10	10	11	00	01
p(x)=x+1=11	11	10	01	00

Multiplication



X	00	01	10	11
00	00	00	00	00
01	00	01	10	11
10	00	10	11	01
11	00	11	01	10

Example GF(2^3)

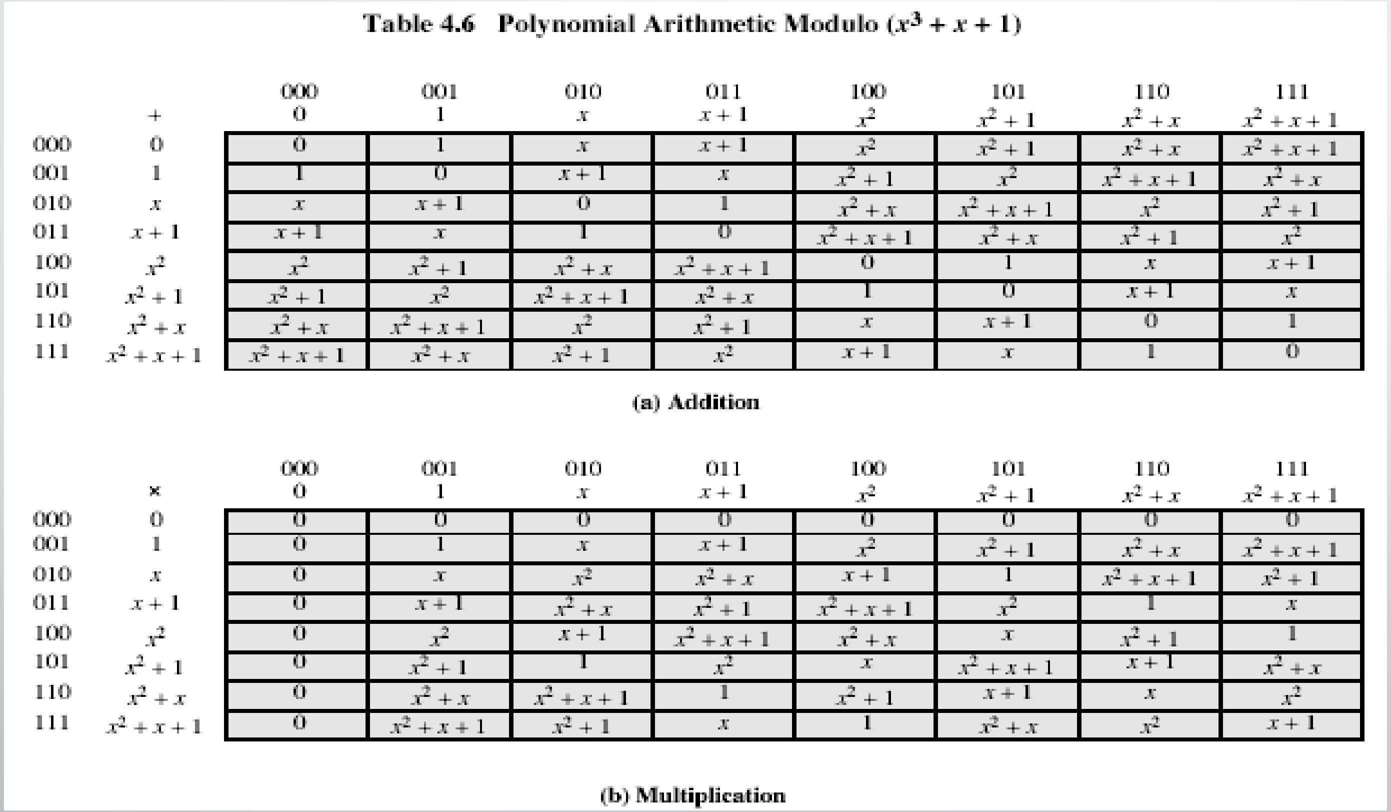


Table 4.6 Polynomial Arithmetic Modulo ($x^3 + x + 1$)

		000	001	010	011	100	101	110	111
	+	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
000	0	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
001	1	1	0	$x+1$	x	x^2+1	x^2	x^2+x+1	x^2+x
010	x	x	$x+1$	0	1	x^2+x	x^2+x+1	x^2	x^2+1
011	$x+1$	$x+1$	x	1	0	x^2+x+1	x^2+x	x^2+1	x^2
100	x^2	x^2	x^2+1	x^2+x	x^2+x+1	0	1	x	$x+1$
101	x^2+1	x^2+1	x^2	x^2+x+1	x^2+x	1	0	$x+1$	x
110	x^2+x	x^2+x	x^2+x+1	x^2	x^2+1	x	$x+1$	0	1
111	x^2+x+1	x^2+x+1	x^2+x	x^2+1	x^2	$x+1$	x	1	0

(a) Addition

		000	001	010	011	100	101	110	111
	×	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
000	0	0	0	0	0	0	0	0	0
001	1	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
010	x	0	x	x^2	x^2+x	$x+1$	1	x^2+x+1	x^2+1
011	$x+1$	0	$x+1$	x^2+x	x^2+1	x^2+x+1	x^2	1	x
100	x^2	0	x^2	$x+1$	x^2+x+1	x^2+x	x	x^2+1	1
101	x^2+1	0	x^2+1	1	x^2	x	x^2+x+1	$x+1$	x^2+x
110	x^2+x	0	x^2+x	x^2+x+1	1	x^2+1	$x+1$	x	x^2
111	x^2+x+1	0	x^2+x+1	x^2+1	x	1	x^2+x	x^2	$x+1$

(b) Multiplication

Computational Considerations

- since coefficients are 0 or 1, can represent any such polynomial as a bit string
- addition becomes XOR of these bit strings
- multiplication is shift & XOR
- modulo reduction done by repeatedly substituting highest power with remainder of irreducible poly (also shift & XOR)

Finite Fields

- AES uses the finite field $GF(2^8)$
  - $b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$
    - $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$
- Byte notation for the element: $x^6 + x^5 + x + 1$
  - {01100011} – binary
  - {63} – hex
- Has its own arithmetic operations
  - Addition
  - Multiplication

Finite Field Arithmetic

- Addition (XOR)
  - $(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$
  - $\{01010111\} \oplus \{10000011\} = \{11010100\}$
  - $\{57\} \oplus \{83\} = \{d4\}$
- Multiplication is tricky

Finite Field Multiplication (•)

$(x^6 + x^4 + x^2 + x + 1)\,(x^7 + x + 1) =$

$$x^{13} + x^{11} + x^9 + x^8 + x^7 + x^7 + x^5 + x^3 + x^2 + x + x^6 + x^4 + x^2 + x + 1$$

These cancel

$$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$$

and

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \text{ modulo } (x^8 + x^4 + x^3 + x + 1)$$

$$= x^7 + x^6 + 1.$$

Irreducible Polynomial

Efficient Finite field Multiply

- There’s a better way
  - xtime() – very efficiently multiplies its input by {02}
- Multiplication by higher powers can be accomplished through repeat application of xtime()

Efficient Finite field Multiply

Example 1: xtime({57}) = {57} • {02}

{57} • {02} = {0101 0111} • {0000 0010} =

= shift to left {0101 0111} = {1010 1110} = {ae}

Example 2: xtime({ae}) ={ae} • {02}

{ae} • {02} = {1010 1110} • {0000 0010} =

= shift to left {1010 1110} XOR {0001 1011} =

={0101 1100} XOR {0001 1011} = {0100 0111}={47}

Efficient Finite field Multiply

Example: $\{57\} \bullet \{13\} = \{57\} \bullet (\{01\} \oplus \{02\} \oplus \{10\})$

$\{57\} \bullet \{02\} = \text{xtime}(\{57\}) = \{ae\}$

$\{57\} \bullet \{04\} = \{ae\} \bullet \{02\} = \text{xtime}(\{ae\}) = \{47\}$

$\{57\} \bullet \{08\} = \{47\} \bullet \{02\} = \text{xtime}(\{47\}) = \{8e\}$

$\{57\} \bullet \{10\} = \{8e\} \bullet \{02\} = \text{xtime}(\{8e\}) = \{07\}$

$\{57\} \bullet \{13\} = \{57\} \bullet (\{01\} \oplus \{02\} \oplus \{10\}) =$

$= (\{57\} \bullet \{01\}) \oplus (\{57\} \bullet \{02\}) \oplus (\{57\} \bullet \{10\})$

$= \{57\} \oplus \{ae\} \oplus \{07\} = \{fe\}$

Rijndael

- data block of 4 columns of 4 bytes is state
- key is expanded to array of words
- has 9/11/13 rounds in which state undergoes:
    - byte substitution (1 S-box used on every byte)
    - shift rows (permute bytes between groups/columns)
    - mix columns (subs using matrix multipy of groups)
    - add round key (XOR state with key material)
    - view as alternating XOR key & scramble data bytes
- initial XOR key material & incomplete last round
- with fast XOR & table lookup implementation

The Layers

The 128 input bits are grouped into 16 bytes of 8 bits each, call them

$$a_{0,0}, a_{1,0}, a_{2,0}, a_{3,0}, a_{0,1}, a_{1,1}, \cdots, a_{3,3},$$

and are arranged int 4×4 matrix

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}.$$

In the following, we'll need to work with the finite field $GF(2^8)$. The model of $GF(2^8)$ depends on a choice of irreducibl e polynomial of degree 8. The choice for Rijndeal is $X^8 + X^4 + X^3 + X + 1$. The elements of $GF(2^8)$ can be represented by bytes. They can added by XOR. They also be multiplied in a certain way. Each element has a multiplica tive inverse.

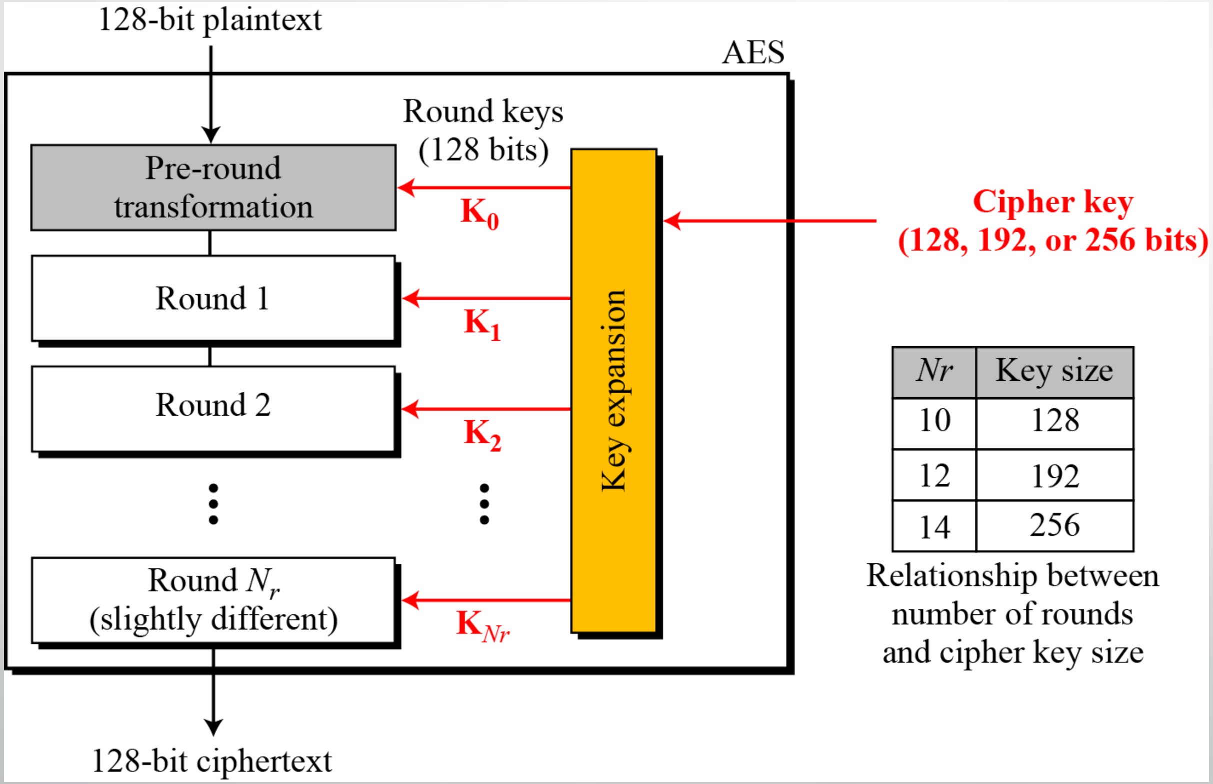
Rounds

AES is a non-Feistel cipher that encrypts and decrypts a data block of 128 bits. It uses 10, 12, or 14 rounds. The key size, which can be 128, 192, or 256 bits, depends on the number of rounds.

Note

AES has defined three versions, with 10, 12, and 14 rounds. Each version uses a different cipher key size (128, 192, or 256), but the round keys are always 128 bits.

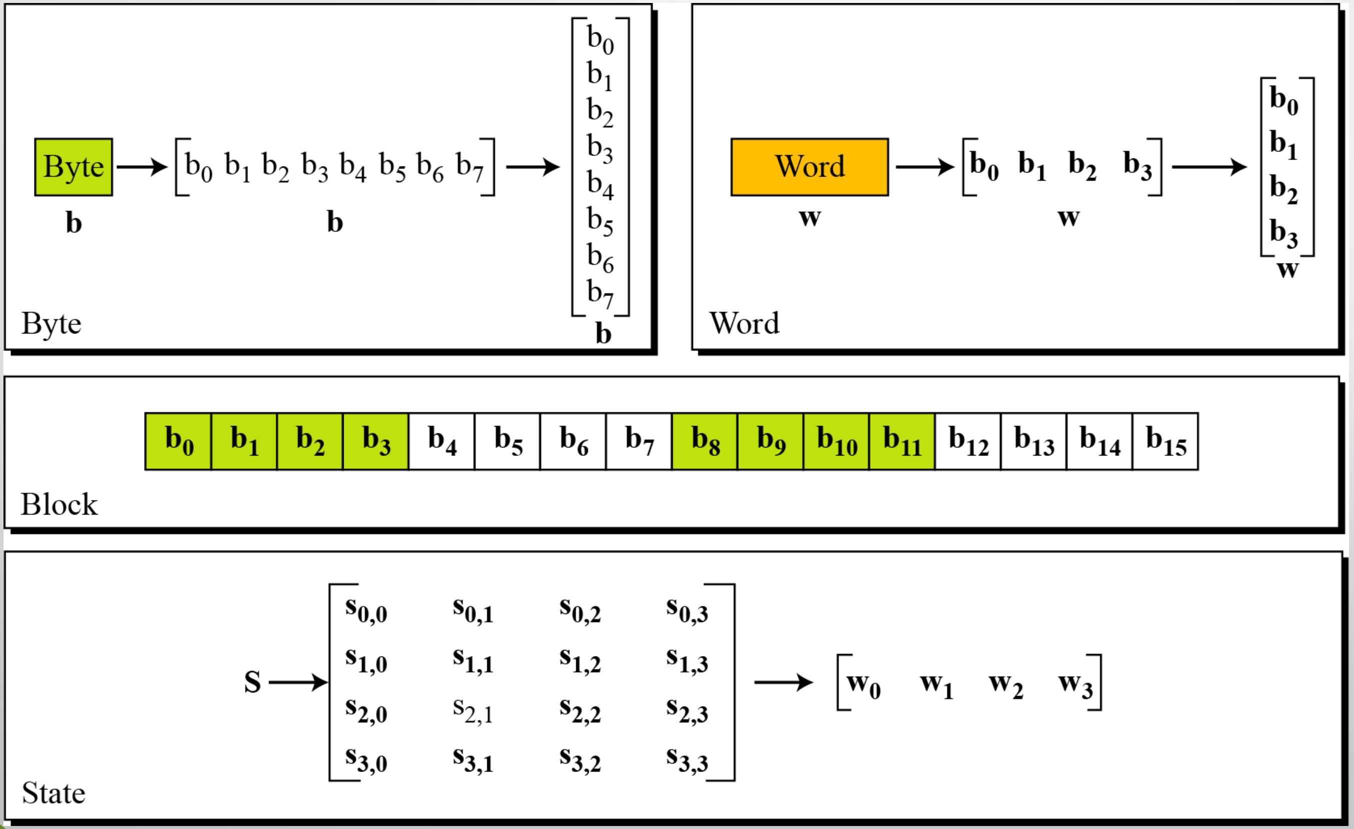
General design of AES encryption cipher



Nr	Key size
10	128
12	192
14	256

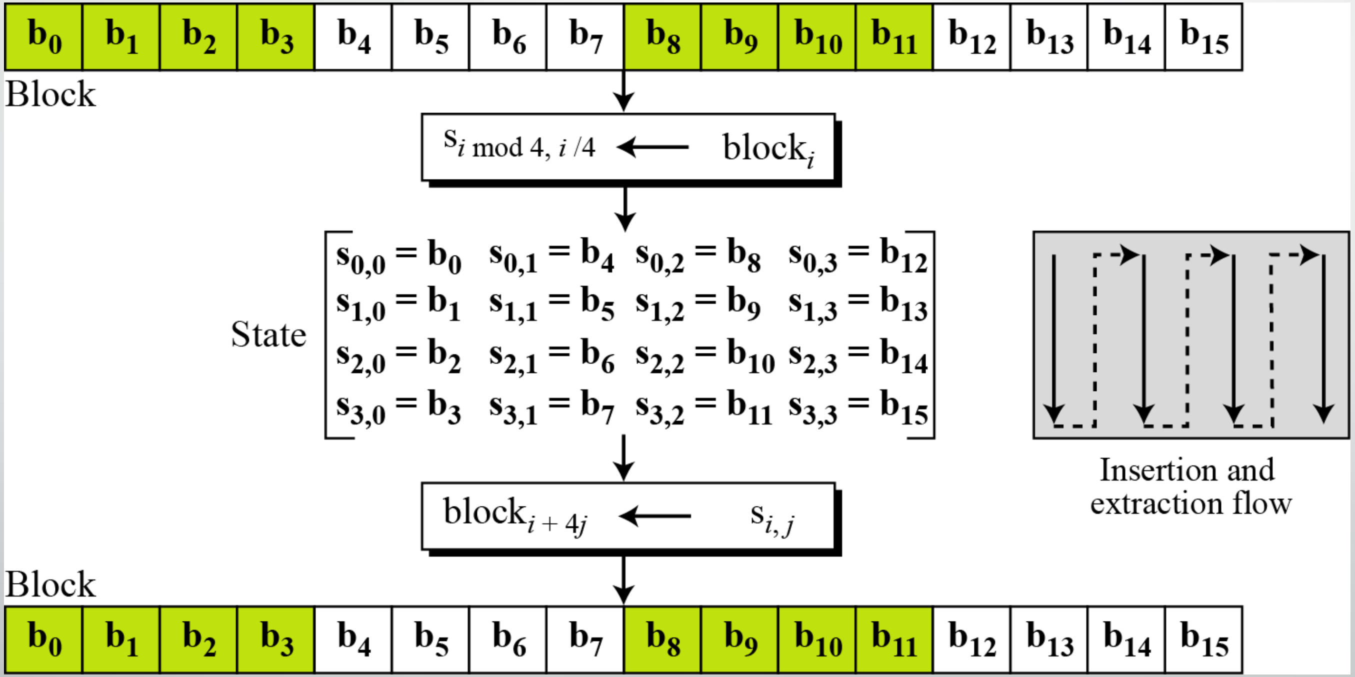
Relationship between number of rounds and cipher key size

Data units used in AES



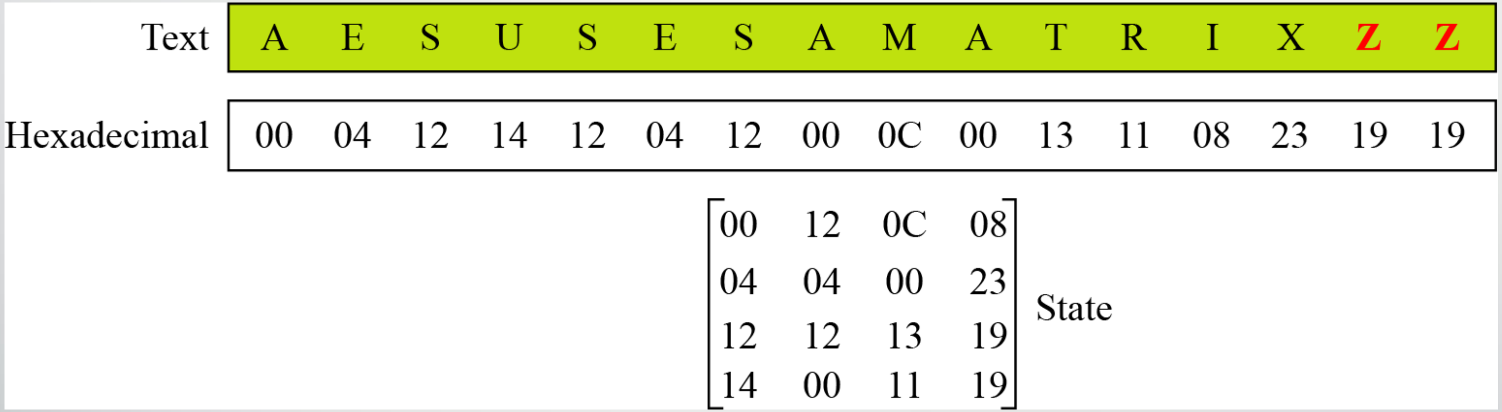
Byte
b
$[b_0\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7]$
b
Byte
Word
w
$[b_0\ b_1\ b_2\ b_3]$
w
Word
$b_0\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7\ b_8\ b_9\ b_{10}\ b_{11}\ b_{12}\ b_{13}\ b_{14}\ b_{15}$
Block
S
$s_{0,0}\ s_{0,1}\ s_{0,2}\ s_{0,3}$
$s_{1,0}\ s_{1,1}\ s_{1,2}\ s_{1,3}$
$s_{2,0}\ s_{2,1}\ s_{2,2}\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

Block-to-state and state-to-block transformation



Block
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}
$s_{i \bmod 4,\, i/4} \leftarrow \text{block}_i$
State
$s_{0,0} = b_0$ $s_{0,1} = b_4$ $s_{0,2} = b_8$ $s_{0,3} = b_{12}$
$s_{1,0} = b_1$ $s_{1,1} = b_5$ $s_{1,2} = b_9$ $s_{1,3} = b_{13}$
$s_{2,0} = b_2$ $s_{2,1} = b_6$ $s_{2,2} = b_{10}$ $s_{2,3} = b_{14}$
$s_{3,0} = b_3$ $s_{3,1} = b_7$ $s_{3,2} = b_{11}$ $s_{3,3} = b_{15}$
$\text{block}_{i+4j} \leftarrow s_{i,j}$
Block
Insertion and extraction flow

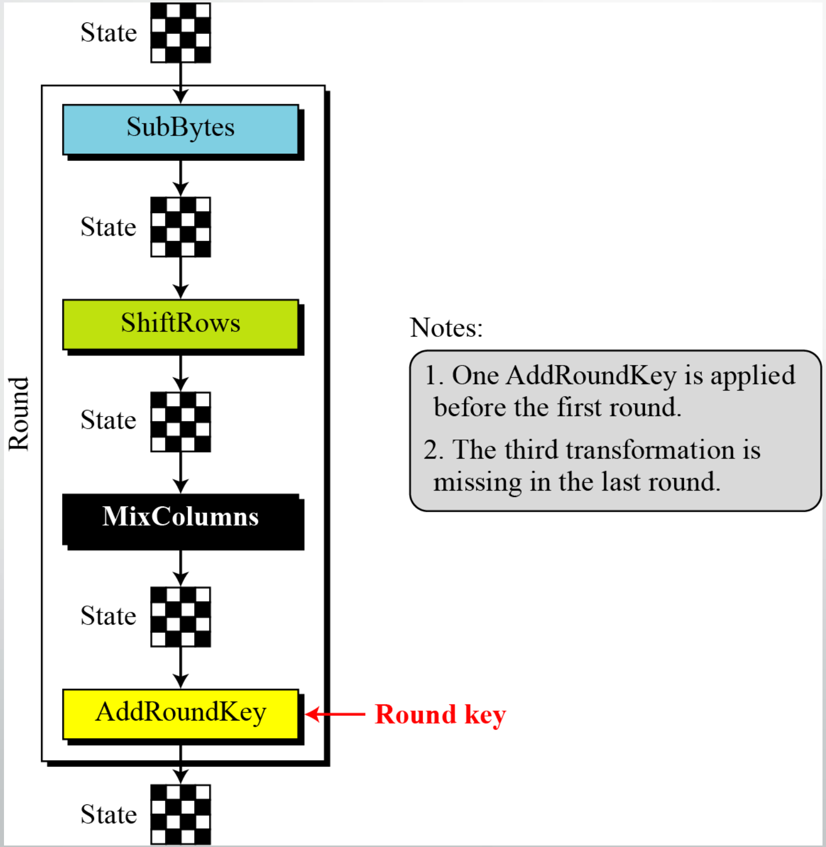
Changing plaintext to state



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

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

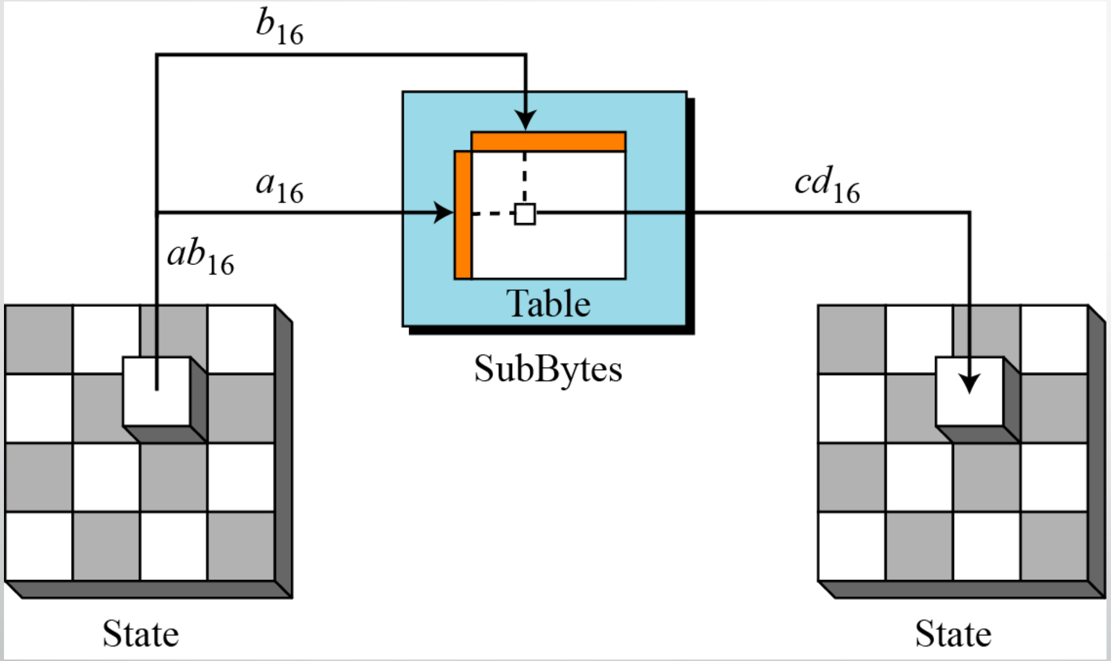
Structure of each round at the encryption



Notes:

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

S-BOX SubBytes transformation



S-BOX SubBytes transformation

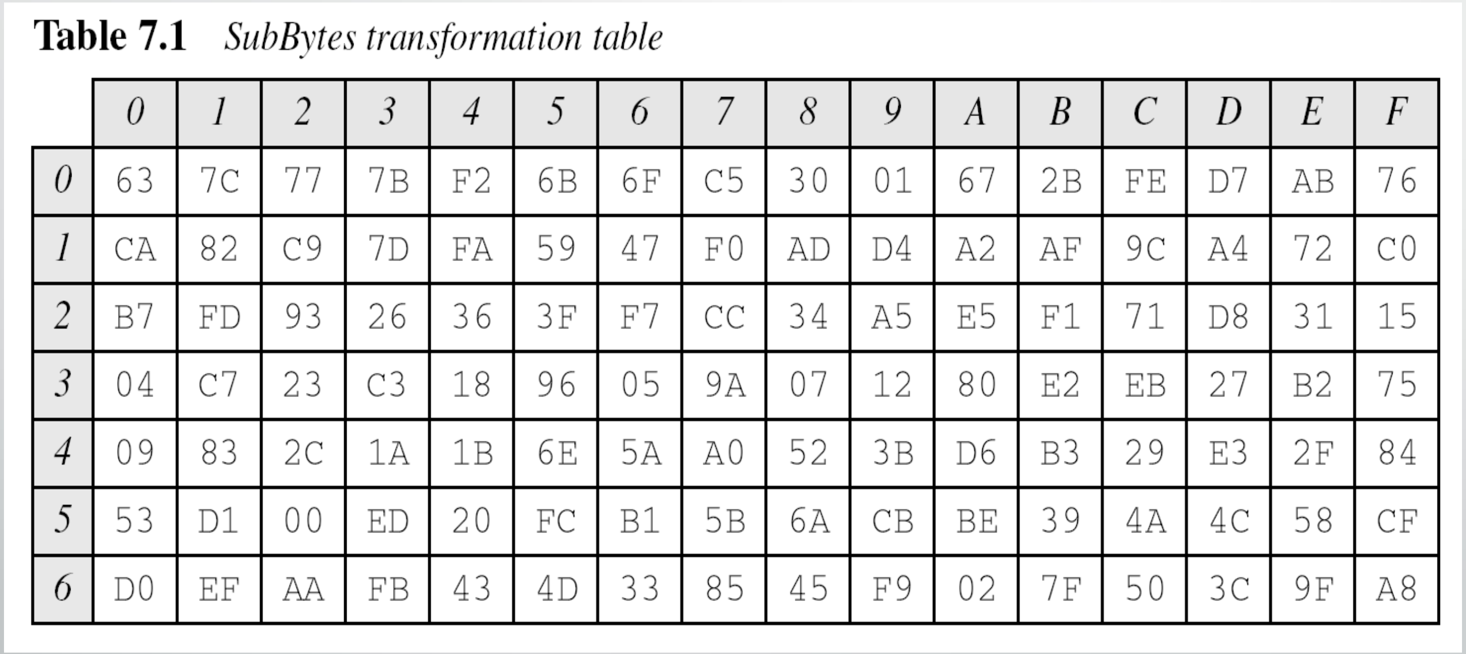


Table 7.1 *SubBytes transformation table*

	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

S-BOX SubBytes transformation

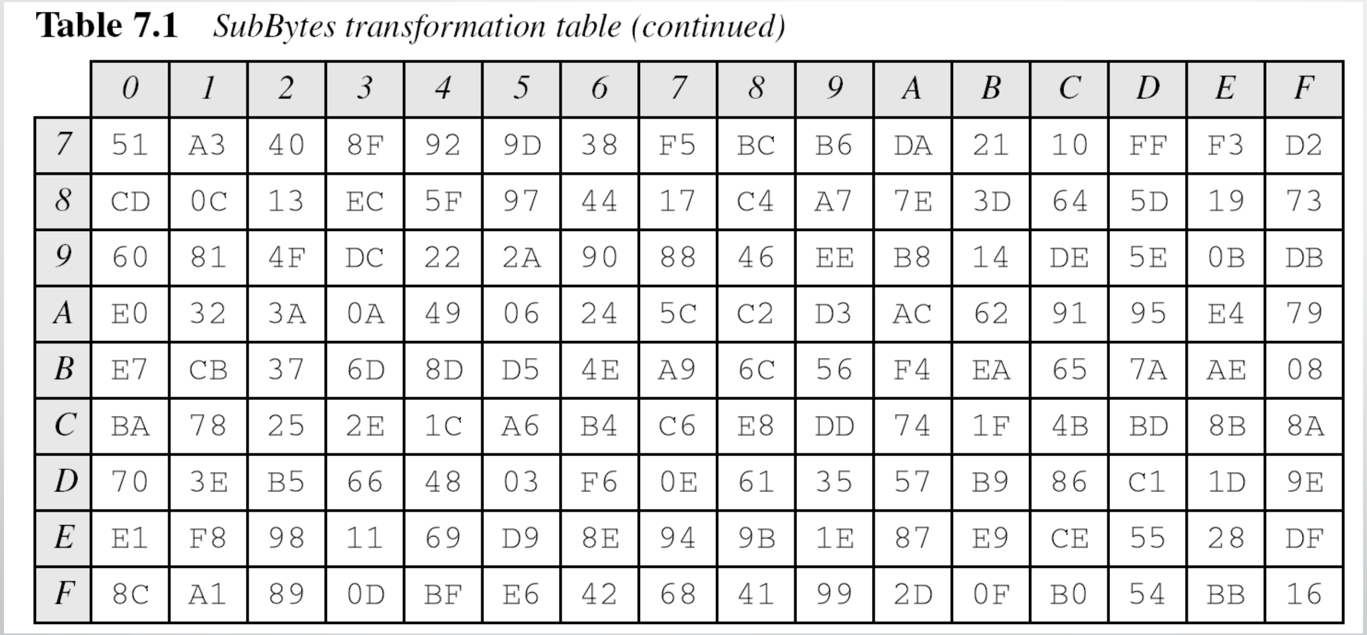


Table 7.1 *SubBytes transformation table (continued)*

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
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	CB	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 S-BOX SubBytes transformation

InvSubBytes

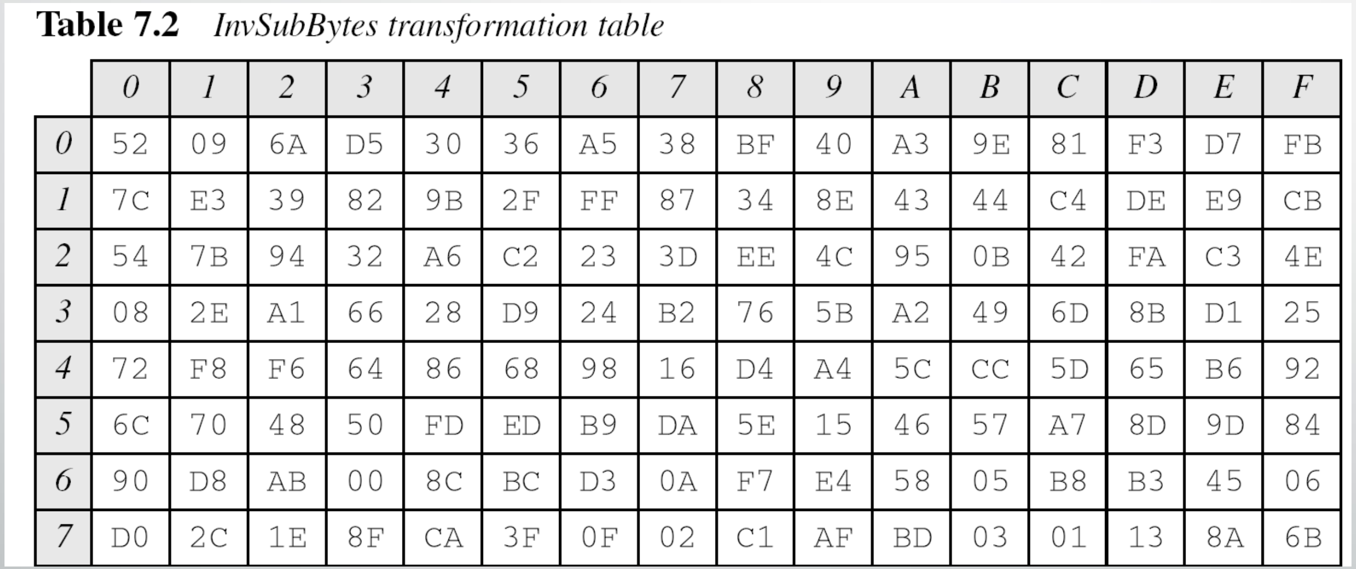
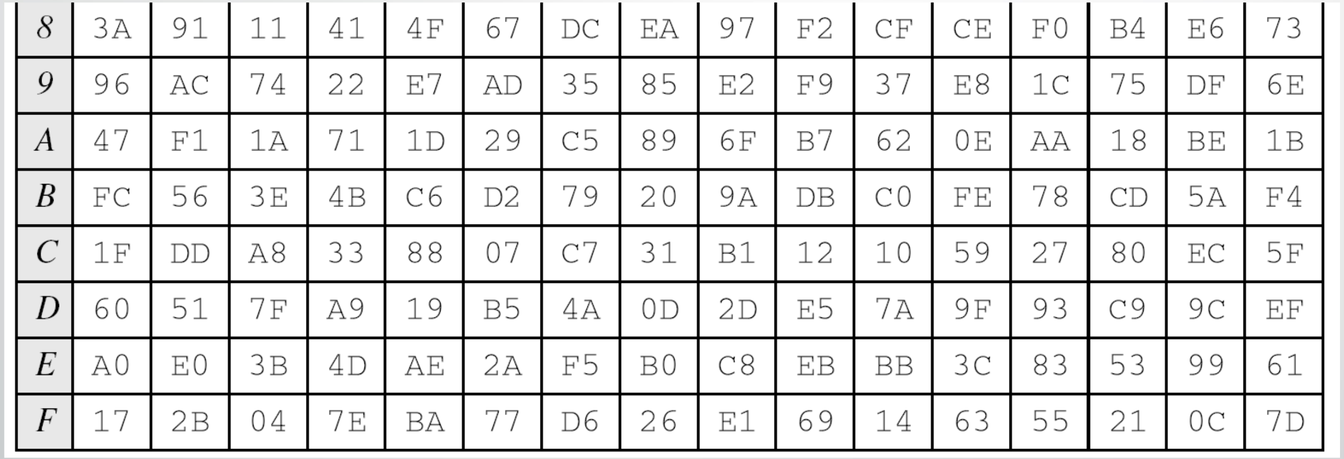


Table 7.2 *InvSubBytes transformation table*

	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

Inverse S-BOX SubBytes transformation

InvSubBytes



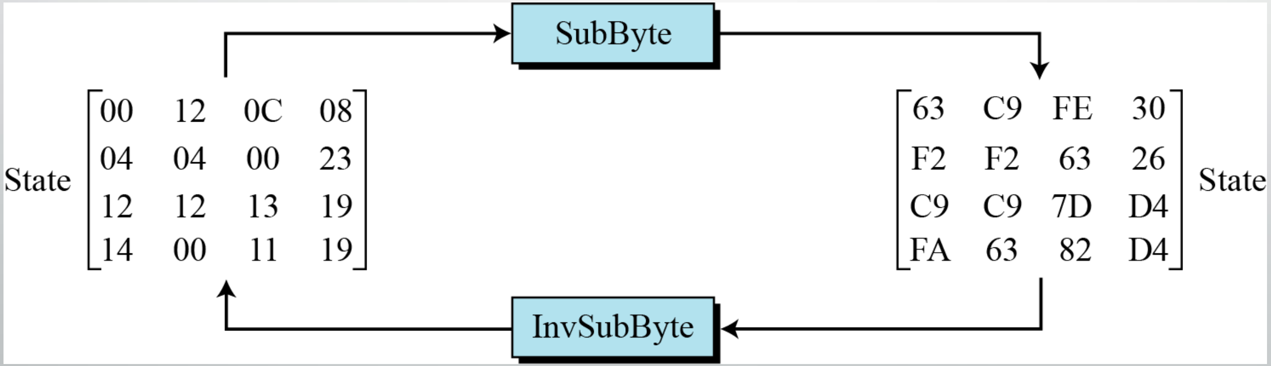
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	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

S-BOX SubBytes transformation

Example 2

Figure 7.7 shows how a state is transformed using the SubBytes transformation. The figure also shows that the InvSubBytes transformation creates the original one. Note that if the two bytes have the same values, their transformation is also the same.

SubBytes transformation for Example 2



$$\text{State}\begin{bmatrix} 00 & 12 & 0C & 08 \\ 04 & 04 & 00 & 23 \\ 12 & 12 & 13 & 19 \\ 14 & 00 & 11 & 19 \end{bmatrix} \quad \begin{bmatrix} 63 & C9 & FE & 30 \\ F2 & F2 & 63 & 26 \\ C9 & C9 & 7D & D4 \\ FA & 63 & 82 & D4 \end{bmatrix}\text{State}$$

S-BOX SubBytes transformation

Transformation Using the GF(2^8) Field
AES also defines the transformation algebraically using the GF(28) field with the irreducible polynomials ($x^8 + x^4 + x^3 + x + 1$), as shown in Figure 7.8.

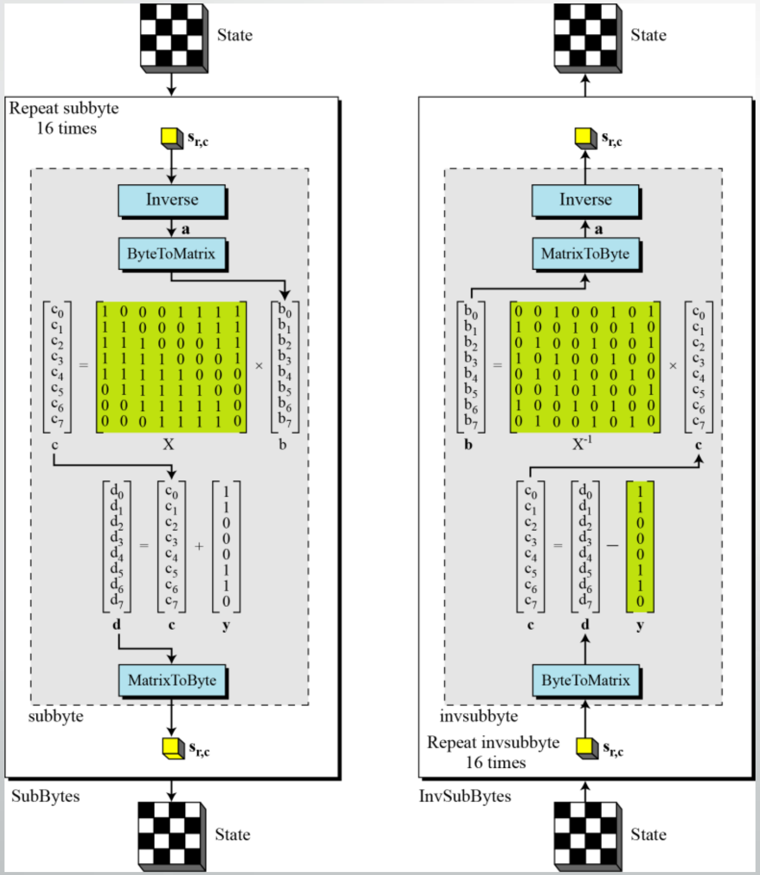
$$\text{subbyte:} \quad \rightarrow \quad \mathbf{d} = \mathbf{X}\,(s_{r,c})^{-1} \oplus \mathbf{y}$$

$$\text{invsubbyte:} \quad \rightarrow \quad [\mathbf{X}^{-1}(\mathbf{d} \oplus \mathbf{y})]^{-1} = [\mathbf{X}^{-1}(\mathbf{X}\,(s_{r,c})^{-1} \oplus \mathbf{y} \oplus \mathbf{y})]^{-1} = [(s_{r,c})^{-1}]^{-1} = s_{r,c}$$

Note

The SubBytes and InvSubBytes transformations are inverses of each other.

Figure 7.8 *SubBytes and InvSubBytes processes*



$$\begin{bmatrix} c_0\\c_1\\c_2\\c_3\\c_4\\c_5\\c_6\\c_7 \end{bmatrix} = \begin{bmatrix} 1&0&0&0&1&1&1&1\\1&1&0&0&0&1&1&1\\1&1&1&0&0&0&1&1\\1&1&1&1&0&0&0&1\\1&1&1&1&1&0&0&0\\0&1&1&1&1&1&0&0\\0&0&1&1&1&1&1&0\\0&0&0&1&1&1&1&0 \end{bmatrix} \times \begin{bmatrix} b_0\\b_1\\b_2\\b_3\\b_4\\b_5\\b_6\\b_7 \end{bmatrix}$$

$$\begin{bmatrix} d_0\\d_1\\d_2\\d_3\\d_4\\d_5\\d_6\\d_7 \end{bmatrix} = \begin{bmatrix} c_0\\c_1\\c_2\\c_3\\c_4\\c_5\\c_6\\c_7 \end{bmatrix} + \begin{bmatrix} 1\\1\\0\\0\\0\\1\\1\\0 \end{bmatrix}$$

$$\begin{bmatrix} b_0\\b_1\\b_2\\b_3\\b_4\\b_5\\b_6\\b_7 \end{bmatrix} = \begin{bmatrix} 0&0&1&0&0&1&0&1\\1&0&0&1&0&0&1&0\\0&1&0&0&1&0&0&1\\1&0&1&0&0&1&0&0\\0&1&0&1&0&0&1&0\\0&0&1&0&1&0&0&1\\1&0&0&1&0&1&0&0\\0&1&0&0&1&0&1&0 \end{bmatrix} \times \begin{bmatrix} c_0\\c_1\\c_2\\c_3\\c_4\\c_5\\c_6\\c_7 \end{bmatrix}$$

$$\begin{bmatrix} c_0\\c_1\\c_2\\c_3\\c_4\\c_5\\c_6\\c_7 \end{bmatrix} = \begin{bmatrix} d_0\\d_1\\d_2\\d_3\\d_4\\d_5\\d_6\\d_7 \end{bmatrix} - \begin{bmatrix} 1\\1\\0\\0\\0\\1\\1\\0 \end{bmatrix}$$

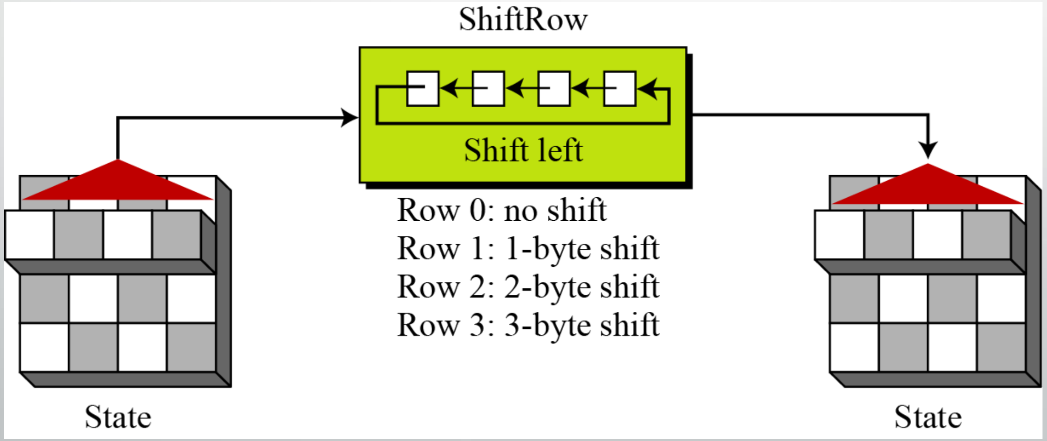
Shift Rows Permutation

Another transformation found in a round is shifting, which permutes the bytes.

ShiftRows

In the encryption, the transformation is called ShiftRows.

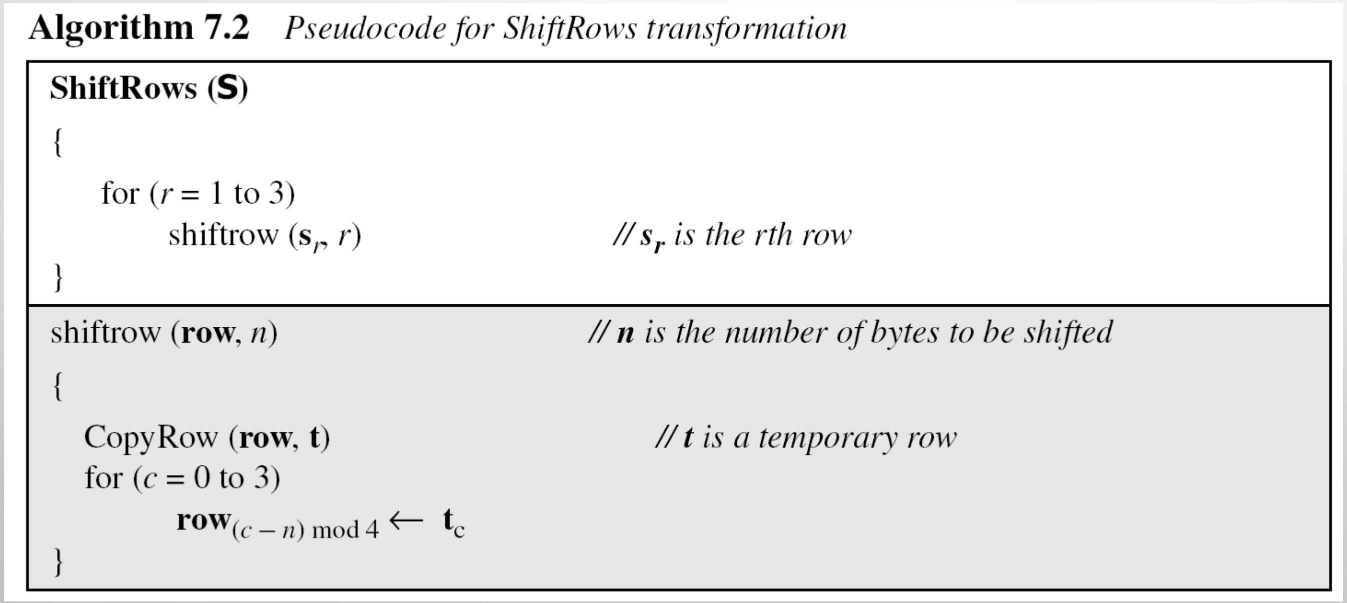
Figure 7.9 *ShiftRows transformation*



Permutation

InvShiftRows

In the decryption, the transformation is called InvShiftRows and the shifting is to the right.



Algorithm 7.2 *Pseudocode for ShiftRows transformation*

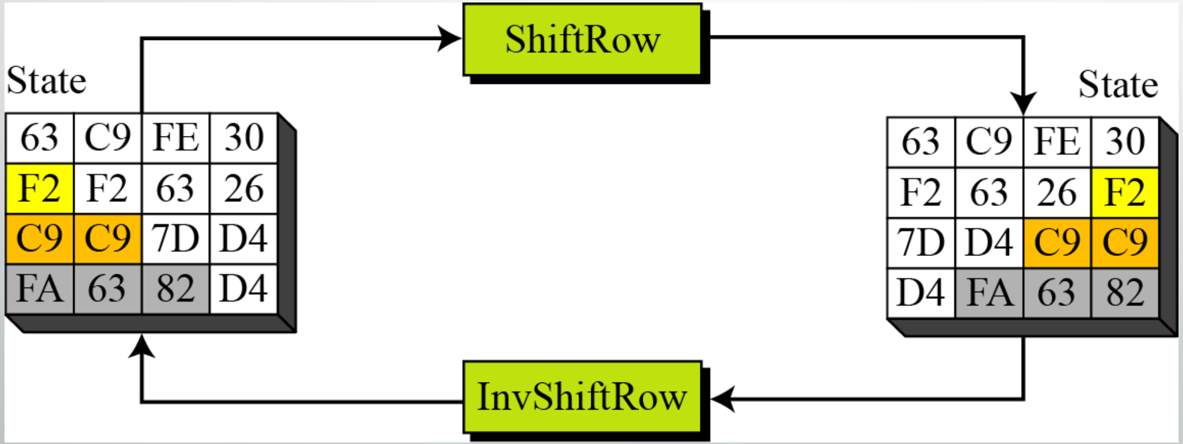
```
ShiftRows (S)
{
    for ($r$ = 1 to 3)
        shiftrow ($\mathbf{s}_r$, $r$)                // $\mathbf{s}_r$ is the rth row
}
shiftrow (row, $n$)                           // $n$ is the number of bytes to be shifted
{
    CopyRow (row, t)                          // t is a temporary row
    for ($c$ = 0 to 3)
        $\mathbf{row}_{(c - n) \bmod 4} \leftarrow \mathbf{t}_c$
}
```

Permutation

Example 4

Figure 7.10 shows how a state is transformed using ShiftRows transformation. The figure also shows that InvShiftRows transformation creates the original state.

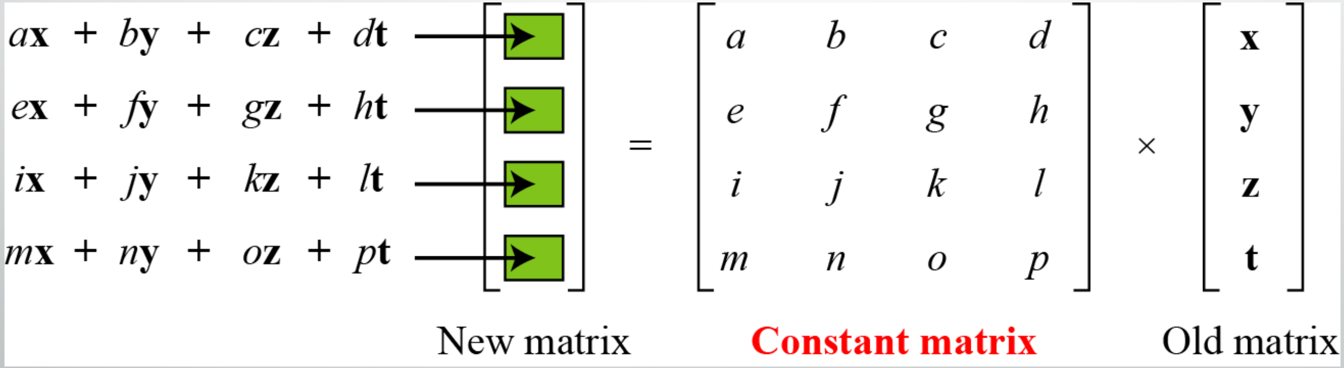
Figure 7.10 *ShiftRows transformation in Example 4*



Mixing

We need an interbyte transformation that changes the bits inside a byte, based on the bits inside the neighboring bytes. We need to mix bytes to provide diffusion at the bit level.

Figure 7.11 *Mixing bytes using matrix multiplication*



Mixing

Figure 7.12 *Constant matrices used by MixColumns and InvMixColumns*

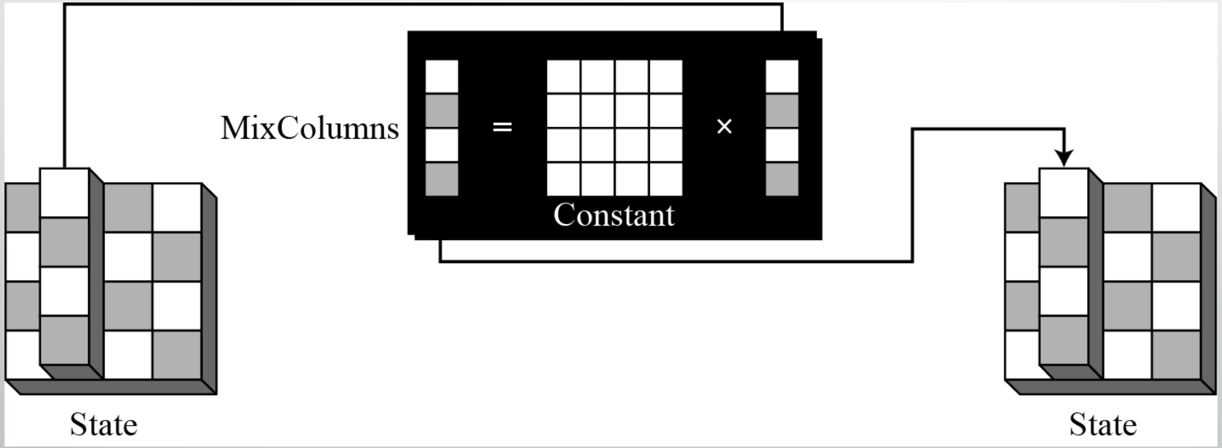
$$\underset{C}{\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}} \xleftrightarrow{\text{Inverse}} \underset{C^{-1}}{\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}}$$

Mixing

MixColumns

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

Figure 7.13 *MixColumns transformation*



Mixing

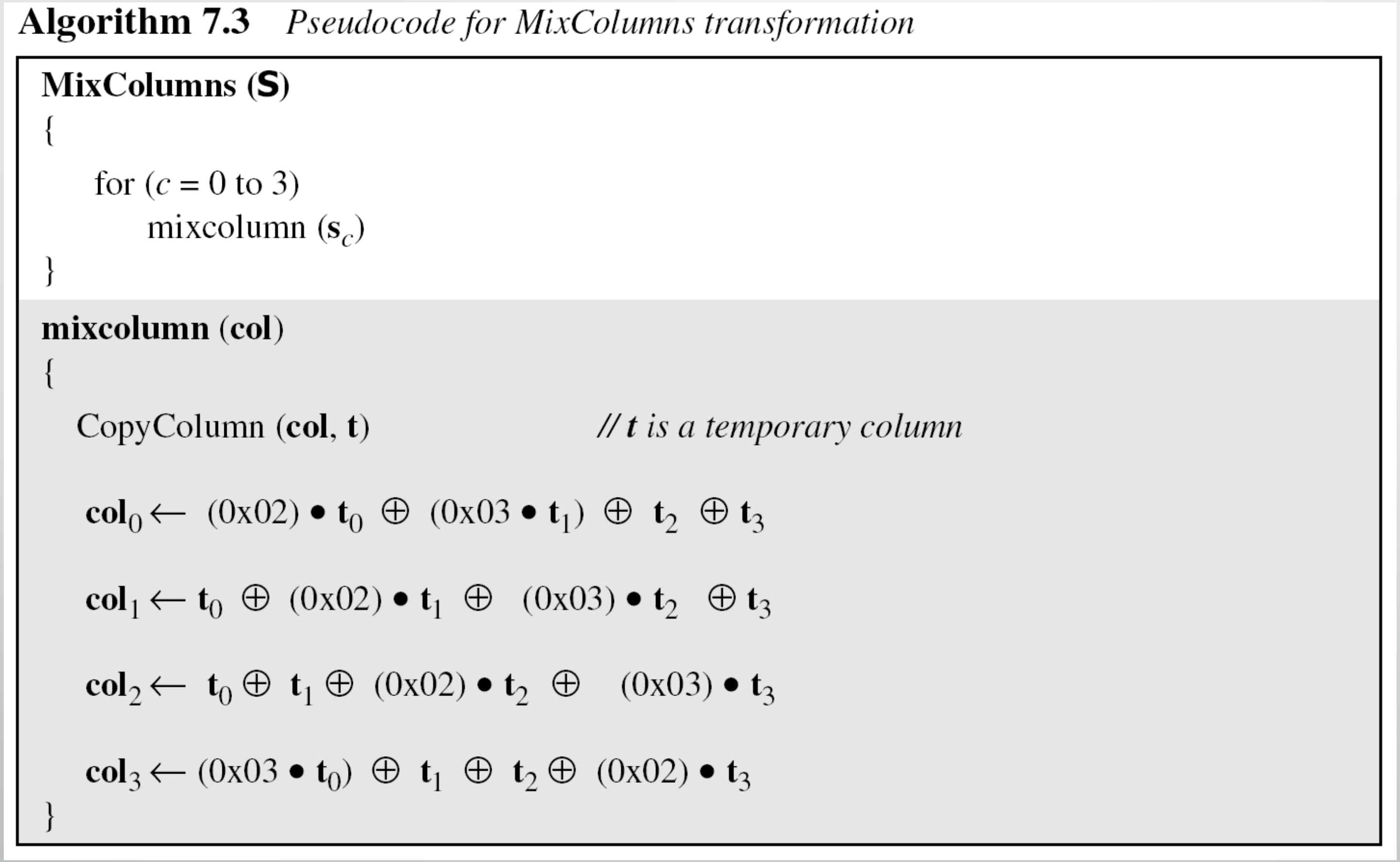
InvMixColumns

The InvMixColumns transformation is basically the same a s the MixColumns transformation.

Note

The MixColumns and InvMixColumns transformations are inverses of each other.

Mixing



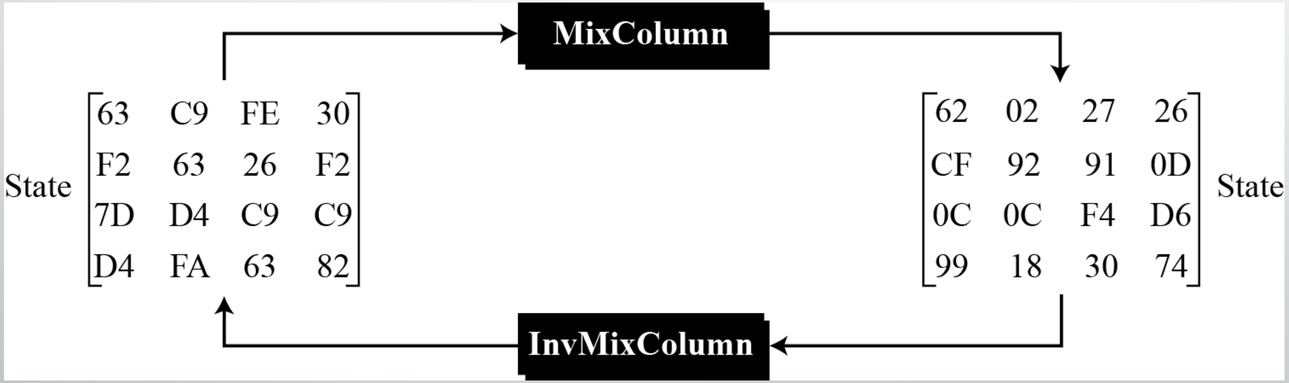
Algorithm 7.3 *Pseudocode for MixColumns transformation*

```
MixColumns (S)
{
    for ($c$ = 0 to 3)
        mixcolumn ($\mathbf{s}_c$)
}

mixcolumn (col)
{
    CopyColumn (col, t)                    // t is a temporary column

    $\mathbf{col}_0 \leftarrow (0x02) \bullet \mathbf{t}_0 \oplus (0x03 \bullet \mathbf{t}_1) \oplus \mathbf{t}_2 \oplus \mathbf{t}_3$

    $\mathbf{col}_1 \leftarrow \mathbf{t}_0 \oplus (0x02) \bullet \mathbf{t}_1 \oplus (0x03) \bullet \mathbf{t}_2 \oplus \mathbf{t}_3$

    $\mathbf{col}_2 \leftarrow \mathbf{t}_0 \oplus \mathbf{t}_1 \oplus (0x02) \bullet \mathbf{t}_2 \oplus (0x03) \bullet \mathbf{t}_3$

    $\mathbf{col}_3 \leftarrow (0x03 \bullet \mathbf{t}_0) \oplus \mathbf{t}_1 \oplus \mathbf{t}_2 \oplus (0x02) \bullet \mathbf{t}_3$
}
```

Mixing

Example 5

Figure 7.14 shows how a state is transformed using the MixColumns transformation. The figure also shows that the InvMixColumns transformation creates the original one.



$$\text{State}\begin{bmatrix}63 & C9 & FE & 30\\ F2 & 63 & 26 & F2\\ 7D & D4 & C9 & C9\\ D4 & FA & 63 & 82\end{bmatrix} \qquad \begin{bmatrix}62 & 02 & 27 & 26\\ CF & 92 & 91 & 0D\\ 0C & 0C & F4 & D6\\ 99 & 18 & 30 & 74\end{bmatrix}\text{State}$$

Figure 7.14 *The MixColumns transformation in Example 5*

Key Adding

AddRoundKey

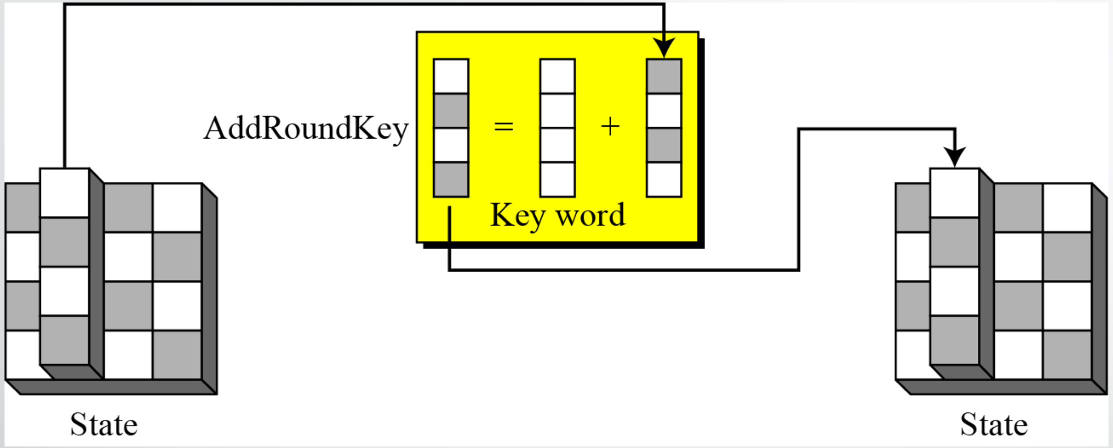
AddRoundKey proceeds one column at a time. AddRound Key adds a round key word with each state column matrix; the operation in AddRoundKey is matrix addition.

Note

The AddRoundKey transformation is the inverse of itself.

Key Adding

Figure 7.15 *AddRoundKey transformation*



Algorithm 7.4 *Pseudocode for AddRoundKey transformation*

```
AddRoundKey (S)
{
    for ($c$ = 0 to 3)
        $\mathbf{s}_c \leftarrow \mathbf{s}_c \oplus \mathbf{w}_{\text{round} + 4c}$
}
```

Key Adding

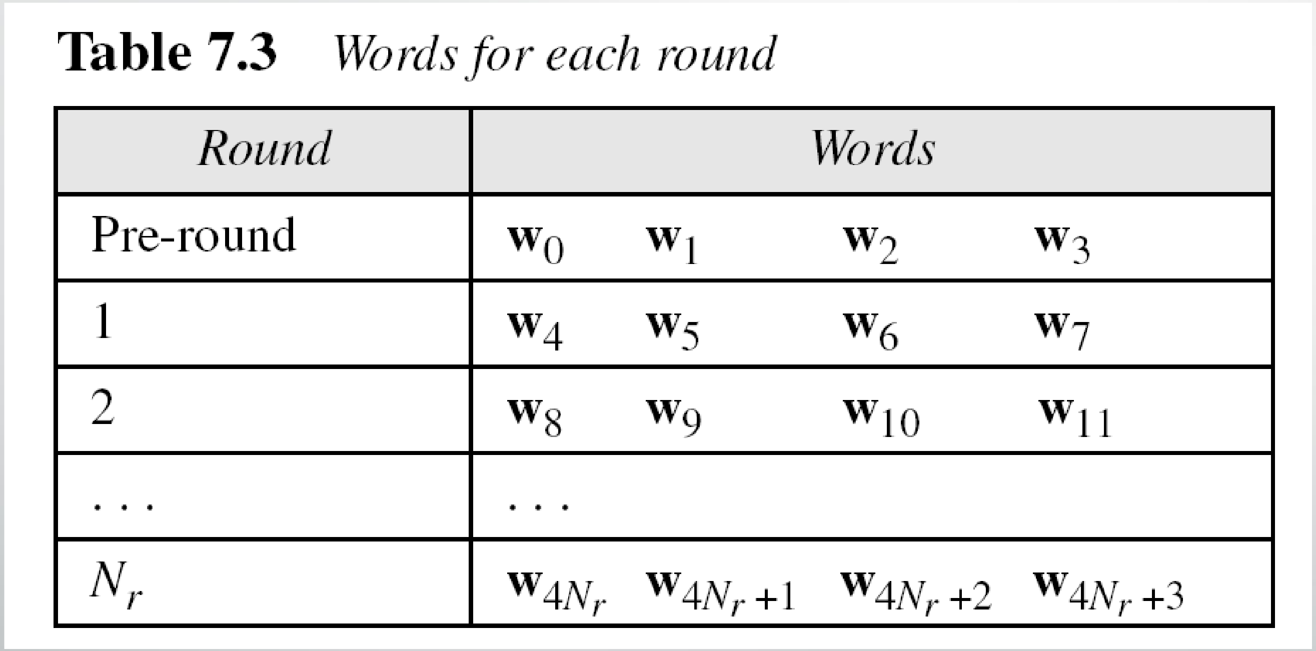
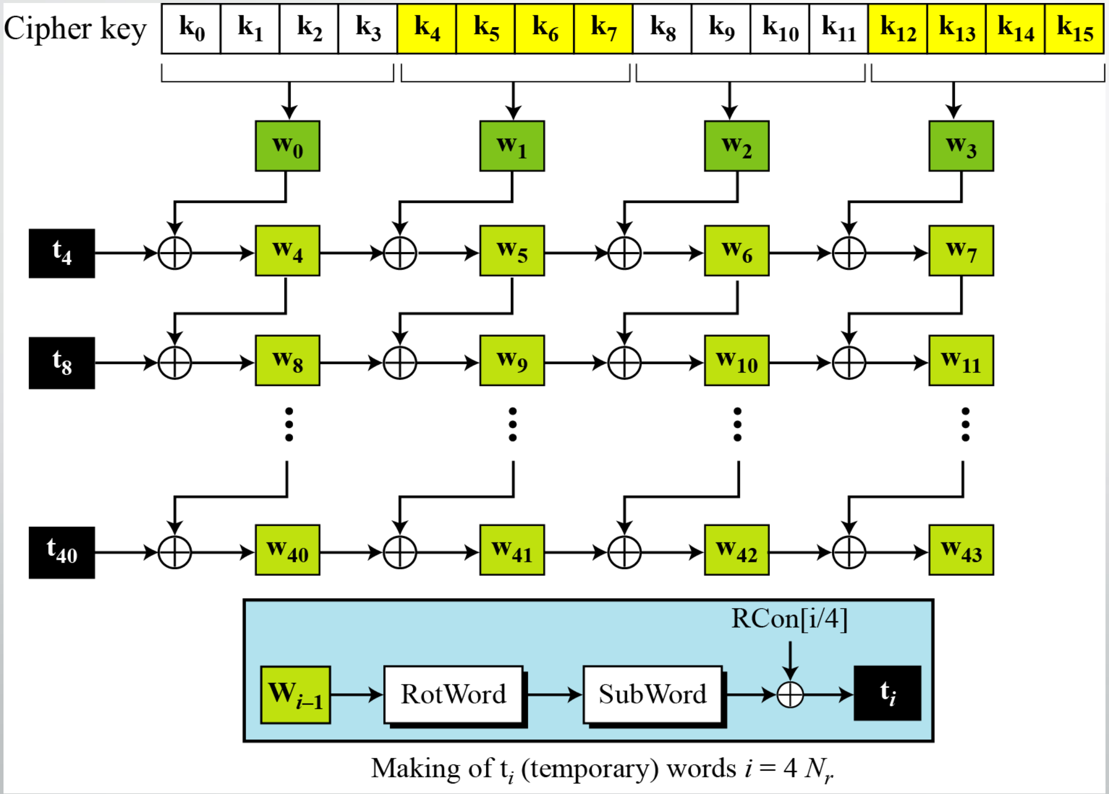


Table 7.3 *Words for each round*

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 in AES-128

Figure 7.16 *Key expansion in AES*


Cipher key
k_0 k_1 k_2 k_3 k_4 k_5 k_6 k_7 k_8 k_9 k_{10} k_{11} k_{12} k_{13} k_{14} k_{15}
w_0 w_1 w_2 w_3
t_4 w_4 w_5 w_6 w_7
t_8 w_8 w_9 w_{10} w_{11}
t_{40} w_{40} w_{41} w_{42} w_{43}
RCon[i/4]
w_{i-1}
RotWord
SubWord
t_i

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

Key Expansion in AES-128

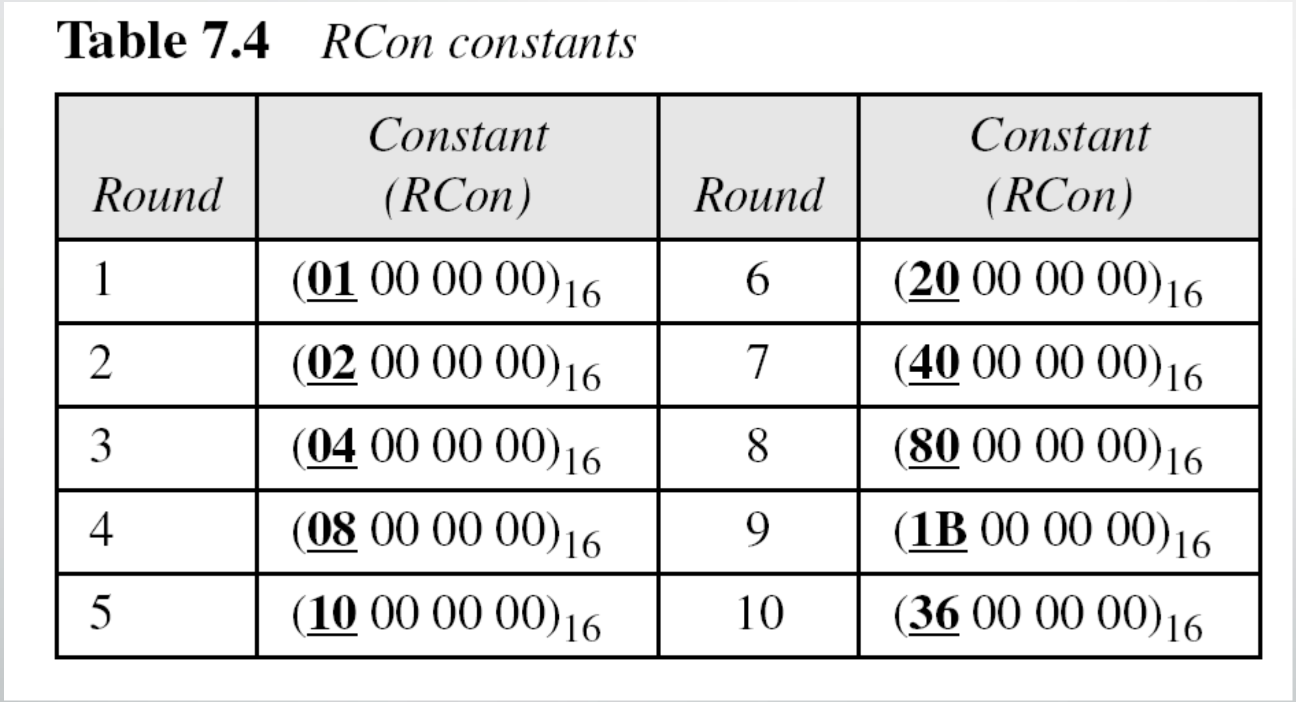
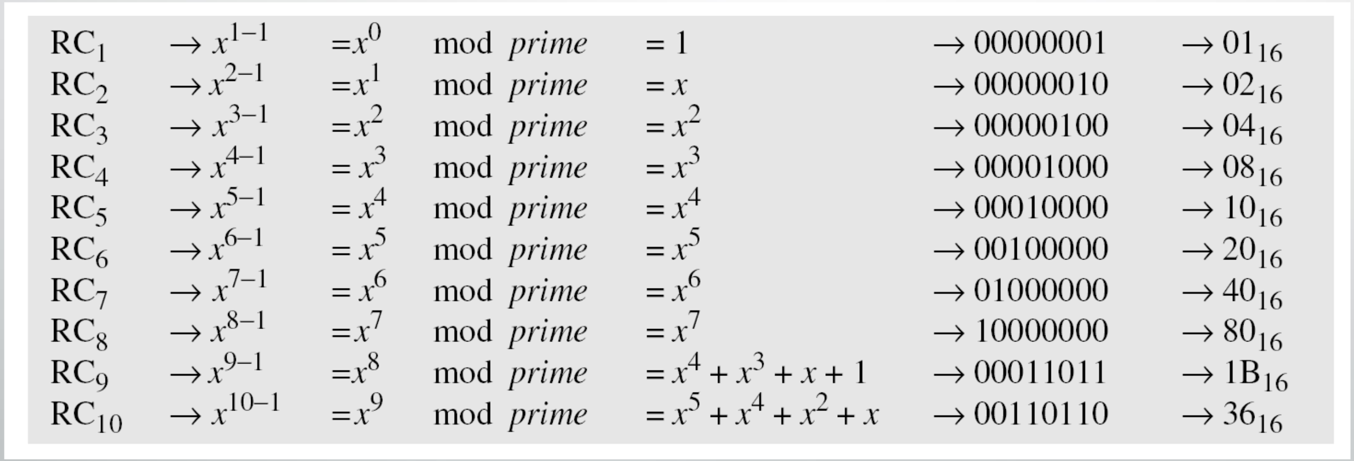


Table 7.4 *RCon constants*

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	$(\underline{\mathbf{36}}\ 00\ 00\ 00)_{16}$

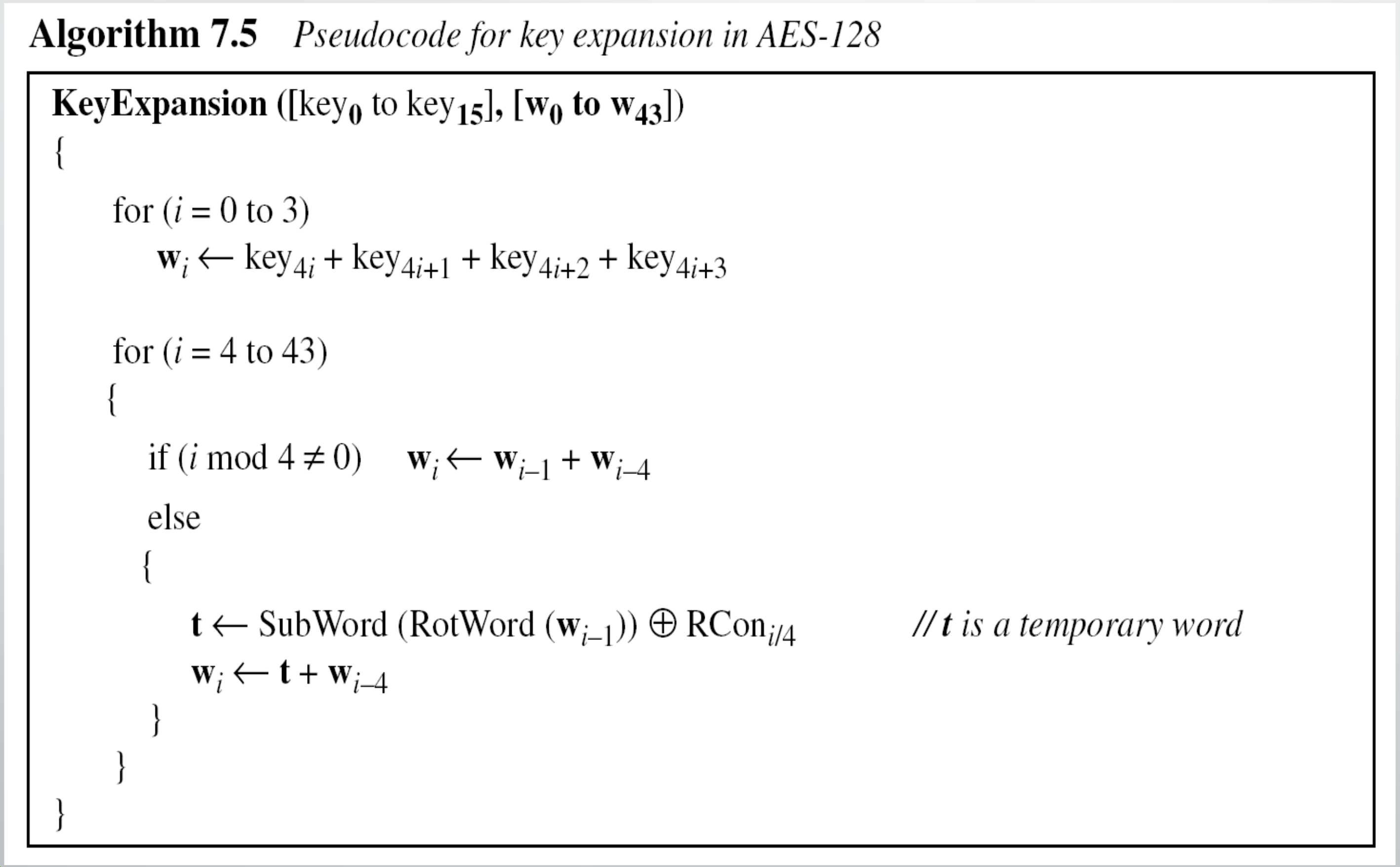
Key Expansion in AES-128

The key-expansion routine can either use the above tabl e when calculating the words or use the GF(2^8) field to calculate the leftmost byte dynamically, as shown below (prime is the irreducible polynomial):



RC_1	$\rightarrow x^{1-1}$	$=x^0$	mod *prime*	$=1$	$\rightarrow 00000001$	$\rightarrow 01_{16}$
RC_2	$\rightarrow x^{2-1}$	$=x^1$	mod *prime*	$=x$	$\rightarrow 00000010$	$\rightarrow 02_{16}$
RC_3	$\rightarrow x^{3-1}$	$=x^2$	mod *prime*	$=x^2$	$\rightarrow 00000100$	$\rightarrow 04_{16}$
RC_4	$\rightarrow x^{4-1}$	$=x^3$	mod *prime*	$=x^3$	$\rightarrow 00001000$	$\rightarrow 08_{16}$
RC_5	$\rightarrow x^{5-1}$	$=x^4$	mod *prime*	$=x^4$	$\rightarrow 00010000$	$\rightarrow 10_{16}$
RC_6	$\rightarrow x^{6-1}$	$=x^5$	mod *prime*	$=x^5$	$\rightarrow 00100000$	$\rightarrow 20_{16}$
RC_7	$\rightarrow x^{7-1}$	$=x^6$	mod *prime*	$=x^6$	$\rightarrow 01000000$	$\rightarrow 40_{16}$
RC_8	$\rightarrow x^{8-1}$	$=x^7$	mod *prime*	$=x^7$	$\rightarrow 10000000$	$\rightarrow 80_{16}$
RC_9	$\rightarrow x^{9-1}$	$=x^8$	mod *prime*	$=x^4+x^3+x+1$	$\rightarrow 00011011$	$\rightarrow 1B_{16}$
RC_{10}	$\rightarrow x^{10-1}$	$=x^9$	mod *prime*	$=x^5+x^4+x^2+x$	$\rightarrow 00110110$	$\rightarrow 36_{16}$

Key Expansion in AES-128



Algorithm 7.5 *Pseudocode for key expansion in AES-128*

```
KeyExpansion ([$key_0$ to $key_{15}$], [$w_0$ to $w_{43}$])
{
    for ($i$ = 0 to 3)
        $w_i \leftarrow key_{4i} + key_{4i+1} + key_{4i+2} + key_{4i+3}$

    for ($i$ = 4 to 43)
    {
        if ($i$ mod $4 \neq 0$)    $w_i \leftarrow w_{i-1} + w_{i-4}$
        else
        {
            $t \leftarrow$ SubWord (RotWord ($w_{i-1}$)) $\oplus$ $RCon_{i/4}$        // t is a temporary word
            $w_i \leftarrow t + w_{i-4}$
        }
    }
}
```

Key Expansion in AES-128

Example 6

Table 7.5 shows how the keys for each round are calculated assu ming that the 128-bit cipher key agreed upon by Alice and Bob is (24 75 A2 B3 34 75 56 88 31 E2 12 00 13 AA 54 87)$_{16}$.

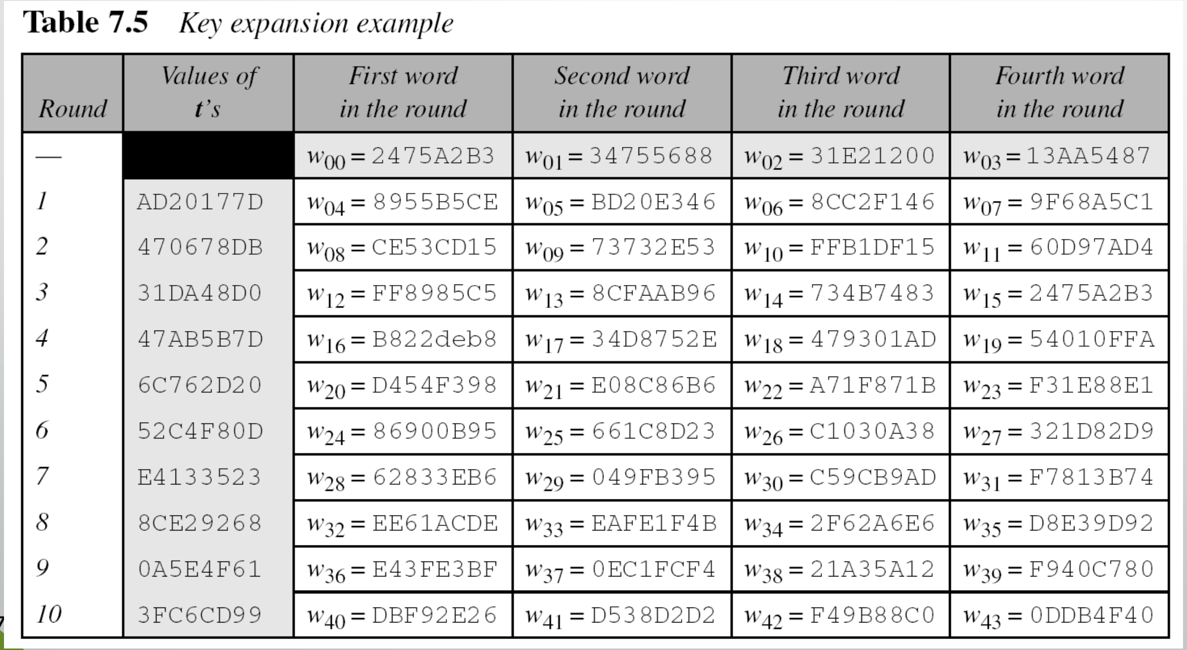


Table 7.5 *Key expansion example*

Round	*Values of t's*	*First word in the round*	*Second word in the round*	*Third word in the round*	*Fourth word in the round*
—		w_{00} = 2475A2B3	w_{01} = 34755688	w_{02} = 31E21200	w_{03} = 13AA5487
1	AD20177D	w_{04} = 8955B5CE	w_{05} = BD20E346	w_{06} = 8CC2F146	w_{07} = 9F68A5C1
2	470678DB	w_{08} = CE53CD15	w_{09} = 73732E53	w_{10} = FFB1DF15	w_{11} = 60D97AD4
3	31DA48D0	w_{12} = FF8985C5	w_{13} = 8CFAAB96	w_{14} = 734B7483	w_{15} = 2475A2B3
4	47AB5B7D	w_{16} = B822deb8	w_{17} = 34D8752E	w_{18} = 479301AD	w_{19} = 54010FFA
5	6C762D20	w_{20} = D454F398	w_{21} = E08C86B6	w_{22} = A71F871B	w_{23} = F31E88E1
6	52C4F80D	w_{24} = 86900B95	w_{25} = 661C8D23	w_{26} = C1030A38	w_{27} = 321D82D9
7	E4133523	w_{28} = 62833EB6	w_{29} = 049FB395	w_{30} = C59CB9AD	w_{31} = F7813B74
8	8CE29268	w_{32} = EE61ACDE	w_{33} = EAFE1F4B	w_{34} = 2F62A6E6	w_{35} = D8E39D92
9	0A5E4F61	w_{36} = E43FE3BF	w_{37} = 0EC1FCF4	w_{38} = 21A35A12	w_{39} = F940C780
10	3FC6CD99	w_{40} = DBF92E26	w_{41} = D538D2D2	w_{42} = F49B88C0	w_{43} = 0DDB4F40

Key Expansion in AES-128

Example 7

Each round key in AES depends on the previous round key. The dependency, however, is nonlinear because of SubWord transfor mation. The addition of the round constants also guarantees that each round key will be different from the previous one.

Key Expansion in AES-128

Example 8

The two sets of round keys can be created from two cipher keys that are different only in one bit.

Cipher Key 1: 12 45 A2 A1 23 31 A4 A3 B2 CC A**A** 34 C2 BB 77 23
Cipher Key 2: 12 45 A2 A1 23 31 A4 A3 B2 CC A**B** 34 C2 BB 77 23

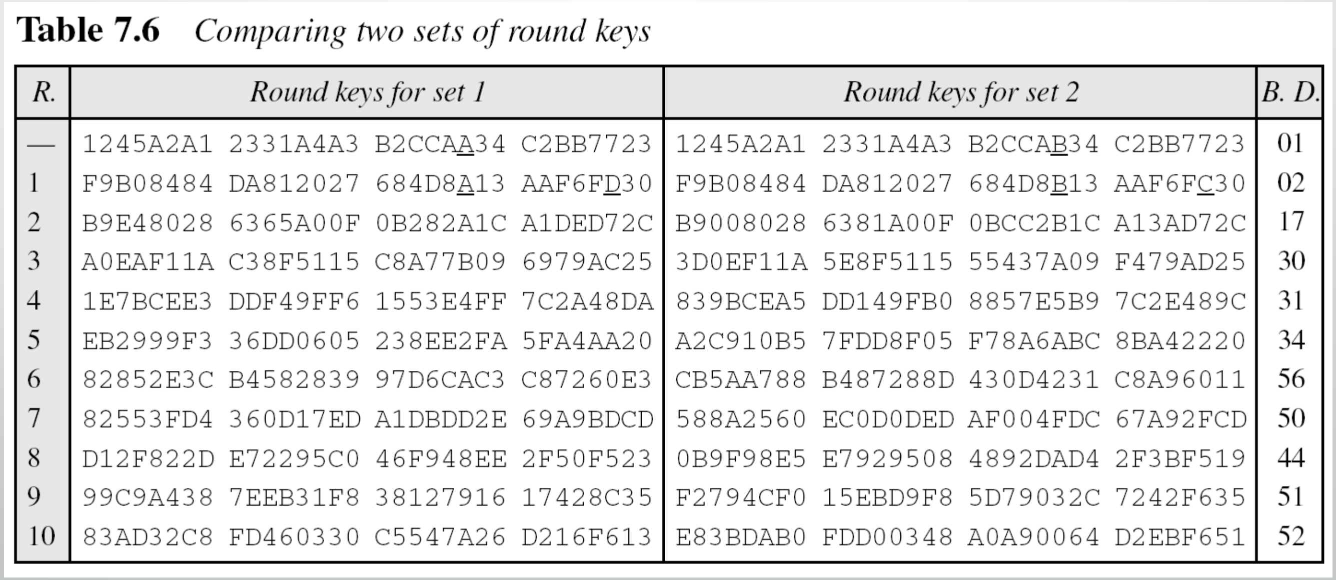


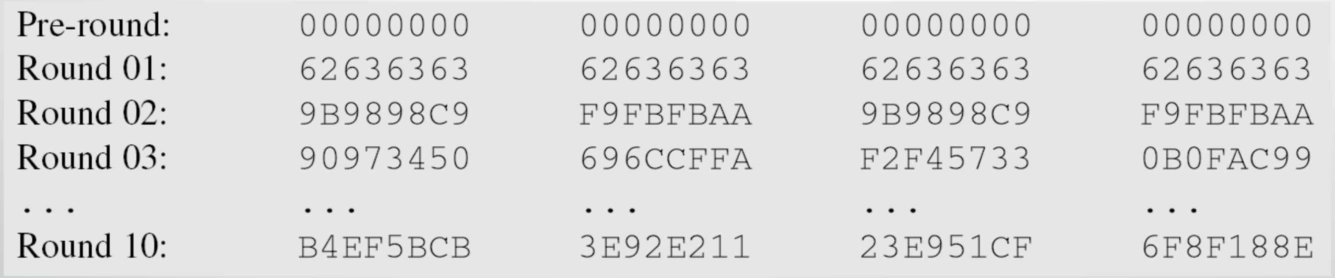
Table 7.6 *Comparing two sets of round keys*

R.	*Round keys for set 1*	*Round keys for set 2*	*B. D.*
—	1245A2A1 2331A4A3 B2CCAA34 C2BB7723	1245A2A1 2331A4A3 B2CCAB34 C2BB7723	01
1	F9B08484 DA812027 684D8A13 AAF6FD30	F9B08484 DA812027 684D8B13 AAF6FC30	02
2	B9E48028 6365A00F 0B282A1C A1DED72C	B9008028 6381A00F 0BCC2B1C A13AD72C	17
3	A0EAF11A C38F5115 C8A77B09 6979AC25	3D0EF11A 5E8F5115 55437A09 F479AD25	30
4	1E7BCEE3 DDF49FF6 1553E4FF 7C2A48DA	839BCEA5 DD149FB0 8857E5B9 7C2E489C	31
5	EB2999F3 36DD0605 238EE2FA 5FA4AA20	A2C910B5 7FDD8F05 F78A6ABC 8BA42220	34
6	82852E3C B4582839 97D6CAC3 C87260E3	CB5AA788 B487288D 430D4231 C8A96011	56
7	82553FD4 360D17ED A1DBDD2E 69A9BDCD	588A2560 EC0D0DED AF004FDC 67A92FCD	50
8	D12F822D E72295C0 46F948EE 2F50F523	0B9F98E5 E7929508 4892DAD4 2F3BF519	44
9	99C9A438 7EEB31F8 38127916 17428C35	F2794CF0 15EBD9F8 5D79032C 7242F635	51
10	83AD32C8 FD460330 C5547A26 D216F613	E83BDAB0 FDD00348 A0A90064 D2EBF651	52

Key Expansion in AES-128

Example 9

The concept of weak keys, as we discussed for DES, does not apply to AES. Assume that all bits in the cipher key are 0s. The following shows the words for some rounds:



Pre-round:	00000000	00000000	00000000	00000000
Round 01:	62636363	62636363	62636363	62636363
Round 02:	9B9898C9	F9FBFBAA	9B9898C9	F9FBFBAA
Round 03:	90973450	696CCFFA	F2F45733	0B0FAC99
. . .	. . .	. . .	. . .	. . .
Round 10:	B4EF5BCB	3E92E211	23E951CF	6F8F188E

The words in the pre-round and the first round are all the same. In the second round, the first word matches with the third; the second word matches with the fourth. However, after the second round the pattern disappears; every word is different.

Key Expansion in AES-192 and AES-256

Key-expansion algorithms in the AES-192 and AES-256 versions are very similar to the key expansion algorithm in AES-128.

The key-expansion mechanism in AES has been designed to provide several features that thwart the cryptanalyst.