密码学与信息安全

深圳大学

第8讲

内容回顾

$GF(2)$ 上的多项式

高级数据加密标准

分组密码运行模式

扩散

- 扩散, 使明文的统计特征消散在密文中, 让每个明文数字尽可能地影响多个密文数字.
- Diffusion hides the relationship between the ciphertext and the plaintext, it will frustrate the adversary who uses ciphertext statistics to find the plaintext. If a single symbol in the plaintext is changed, several or all symbols in the ciphertext will also changed.
- The avalanche effect is evident if, when an input is changed slightly (e.g., flipping a single bit) the output changes significantly (e.g., half the output bits flip).
- Diffusion can be accomplished using permutations on data.

混淆

- 混淆是尽可能使作用于明文的密钥和密文的关系复杂化, 使得明文和密文之间, 密文和密钥之间的统计相关性极小化, 以阻止攻击者发现密钥.
- Confusion hides the relationship between the ciphertext and the key, it will frustrate the adversary who tries to use the ciphertext to find the key.
- Confusion can be accomplished by using substitutions or nonlinear operations on data.

Diffusion and confusion can be achieved using iterated product ciphers where each iteration is a combination of S-boxes, P-boxes, and other components.

内容回顾

什么是群?

内容回顾

什么是群?

- 一个非空集合 **G**
- 集合上定义了一个二元运算 $+$
- 二元运算满足封闭性, 结合律
- 有单位元
- 有逆元

内容回顾

什么是群?

- 一个非空集合 **G**
- 集合上定义了一个二元运算 $+$
- 二元运算满足封闭性, 结合律
- 有单位元
- 有逆元
- 若满足交换律, 则称为交换群

内容回顾

什么是环?

内容回顾

什么是环?

- 一个非空集合 $\mathbf{R}$
- 集合上定义了加法 $+$ 和乘法 $\times$ 两个二元运算,
- $\mathbf{R}$ 关于加法是一个交换群
- $\mathbf{R}$ 关于乘法满足结合律
- 加法和乘法之间满足分配律

内容回顾

什么是环?

- 一个非空集合 $\mathbf{R}$
- 集合上定义了加法 $+$ 和乘法 $\times$ 两个二元运算,
- $\mathbf{R}$ 关于加法是一个交换群
- $\mathbf{R}$ 关于乘法满足结合律
- 加法和乘法之间满足分配律
- $\mathbf{R}$ 关于乘法不一定有单位元, 也不一定要满足交换律
- 满足乘法交换律的环称为交换环

内容回顾

什么是域?

内容回顾

什么是域?

- 一个非空集合 $\mathbf{F}$
- 集合上定义了加法 $+$ 和乘法 $\times$ 两个二元运算
- $\mathbf{F}$ 关于加法是一个交换群, 加法单位元记为 0
- $\mathbf{F}$ 中非 0 元素关于乘法是一个交换群, 乘法单位元记为 1
- 加法和乘法之间满足分配律

内容回顾

有限域的加法特性: 特征

- 有理数域, 实数域和复数域中, 任意多个 1 相加都不等于 0.
- 有限域因为元素个数有限, 若干个 1 相加中不可能没有相同的元素, 即存在 $1 \leq i < j$,

$$i \times 1 = j \times 1 \Longrightarrow (j - i) \times 1 = 0$$

- 如果对任意正整数 m, 都有 $m \times 1 \neq 0$, 则称域 $\mathbf{F}$ 的特征是 0; 否则, 称满足条件的最小正整数 m 为域 $\mathbf{F}$ 的特征 (m 一定是素数, 记为 p).
- $\mathbf{F}$ 是有限域, 其阶为 q, 则 $q = p^n$, 其中 p 为素数, n 为一正整数; 对于每一个素数 p, 任何一个正整数 n, 一定存在阶为 p^n的有限域, 且在同构意义下, 这样的域是唯一的.

Polynomials Over $GF(2)$

- There exist only two irreducible polynomials of degree 3 over $GF(2)$. One is $f(x) = x^3 + x + 1$, the other is $x^3 + x^2 + 1$.
- We will consider all polynomials defined over $GF(2)$ modulo the irreducible polynomial $x^3 + x + 1$.
- For example

$$\begin{aligned}(x^2 + x + 1) \times (x^2 + 1) \mod (x^3 + x + 1) \\ = x^4 + x^3 + x + 1 \mod (x^3 + x + 1) \\ = x^2 + x \mod (x^3 + x + 1)\end{aligned}$$

Polynomials Over $GF(2)$

- With multiplications modulo $x^3 + x + 1$, we have only the following eight polynomials in the set of polynomials over $GF(2)$
  - 0, 1
  - x, $x + 1$
  - x^2, $x^2 + 1$, $x^2 + x$, $x^2 + x + 1$
- We will refer to this set as $GF(2^3)$.
- Compare $\mathbb{Z}_8$ with $GF(2^3)$.

Polynomials Over $GF(2)$

- To find all the polynomials in $GF(2^n)$, we obviously need an irreducible polynomial of degree n.
- AES arithmetic is based on $GF(2^8)$, which uses the following irreducible polynomial

$$x^8 + x^4 + x^3 + x + 1$$

- The finite field $GF(2^8)$ used by AES obviously contains 256 distinct polynomials over $GF(2)$.

Polynomials Over $GF(2)$

- A key motivation for using polynomial arithmetic in $GF(2^n)$ is that the polynomials can be represented as a bit string, and the calculations only use simple common machine instructions - addition is just XOR, and multiplication is shifts and XOR.
- For example, consider $GF(2^3)$, we can denote (x^2+1) by 101_2, and (x^2+x+1) by 111_2. Thus

$$
\begin{aligned}
& (x^2+1)+(x^2+x+1)=x \\
\Leftrightarrow \quad & 101_2 \oplus 111_2 = 010_2 \\
& (x+1)\cdot(x^2+1)=x^3+x^2+x+1 \\
\Leftrightarrow \quad & 011_2 \cdot 101_2 = 1111_2
\end{aligned}
$$

Multiplication operation in $GF(2^8)$

- Let's consider the finite field $GF(2^8)$ that is used in AES, which is derived using the following irreducible polynomial of degree 8:

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

- Now let's see how we can carry out multiplications with direct bitwise operations in this $GF(2^8)$.

- We first take note of the following equality in $GF(2^8)$:

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

Multiplication operation in $GF(2^8)$

- Let's represent $f(x)$ by

$$f(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$$

i.e., $f(x)$ stands for the bit pattern $b_7b_6b_5b_4b_3b_2b_1b_0$, and

$$xf(x) = b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x$$

- But now recall that we must take the modulo of this polynomial with respect to

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

Multiplication operation in $GF(2^8)$

- If the bit b_7 of $f(x)$ is equals 0, then the output bit pattern is $b_6b_5b_4b_3b_2b_1b_00$.
- If b_7 equals 1, we need to divide the polynomial we have for $xf(x)$ by the modulus polynomial $m(x)$ and keep just the remainder.

$$
\begin{aligned}
& xf(x) \mod m(x) \\
= \; & b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 \\
& + b_2x^3 + b_1x^2 + b_0x \mod m(x) \\
= \; & (b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x) \\
& + x^8 \mod m(x) \\
= \; & (b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x) \\
& + (x^4 + x^3 + x + 1) \\
= \; & (b_6b_5b_4b_3b_2b_1b_00) \oplus (00011011)
\end{aligned}
$$

Multiplication operation in $GF(2^8)$

- Let's say you want to multiply two bit patterns B_1 and B_2, each 8 bits long.
- If B_2 is the bit pattern 00000001, then the result is B_1 itself.
- If B_2 is the bit pattern 00000010, then we are multiplying B_1 by x. If B_1's MSB (the most significant bit) is 0, the result is obtained by shifting the B_1 bit pattern to the left by one bit and inserting a 0 bit from the right. If B_1's MSB is 1, first we again shift the B_1 bit pattern to the left as above. Next, we take the XOR of the shifted pattern with the bit pattern 00011011 for the final answer.

Multiplication operation in $GF(2^8)$

- If B_2 is the bit pattern 00000100, then we are multiplying B_1 by x^2. This amounts to first multiplying B_1 by x, and then multiplying the result again by x. So it amounts to two applications of the logic in the previous two steps.
- In general, if B_2 consists of a single bit in the j-th position from the right (using the 0 index for the right-most position), we need j applications of the logic laid out above for multiplying with x.
- Even more generally, when B_2 consists of an arbitrary bit pattern, we consider the bit pattern to be a sum of bit patterns each containing only single bit.

Finding Multiplicative Inverses in $GF(2^n)$

- In general, you can use the Extended Euclid's Algorithm for finding the multiplicative inverse of a bit pattern in $GF(2^n)$ provided you carry out all the arithmetic in that algorithm according to the rules appropriate for $GF(2^n)$.
- The table below shows the multiplicative inverses for the bit patterns of $GF(2^3)$.

	Additive Inverse	Multiplicative Inverse
000	000	—
001	001	001
010	010	101
011	011	110
100	100	111
101	101	010
110	110	011
111	111	100

Represent Elements of $GF(2^n)$ Using a Generator

- It is particularly convenient to represent the elements of a Galois Field $GF(2^n)$ with the help of a generator element.
- If g is a generator element, then every element of $GF(2^n)$, except for the 0 element, can be expressed as some power of g.
- Consider a finite field of order q. If g is the generator of this finite field, then the finite field can be expressed by the set

$$\{0, g^0, g^1, g^2, \ldots, g^{q-2}\}$$

- How does one specify a generator?

Represent Elements of $GF(2^n)$ Using a Generator

- A generator is obtained from the irreducible polynomial that went into the creation of the finite field. If $f(x)$ is the irreducible polynomial used, then g is that element which symbolically satisfies the equation $f(g) = 0$. You do not actually solve this equation for its roots since an irreducible polynomial cannot have actual roots in the underlying number system used, but only use this equation for the relationship it gives between the different powers of g.

- Consider the case of $GF(2^3)$ defined with the irreducible polynomial $x^3 + x + 1$. The generator g is that element which symbolically satisfies $g^3 + g + 1 = 0$, implying that such an element will obey

$$g^3 = g + 1$$

Represent Elements of $GF(2^n)$ Using a Generator

- Now we can show that every power of g will correspond to some element of $GF(2^3)$.

0	$g^0 = 1$	$g^1 = g$
$g^2 = g^2$	$g^3 = g + 1$	$g^4 = g^2 + g$
$g^5 = g^2 + g + 1$	$g^6 = g^2 + 1$	$g^7 = 1$
$\ldots$	$\ldots$	$\ldots$
$\ldots$	$g^k = g^{k \mod 7}$	$\ldots$

- Since every polynomial in $GF(2^n)$ is represented by a power of g, multiplying any two polynomials in $GF(2^n)$ becomes trivial: we just have to add the exponents of g modulo $(2^n - 1)$. That is, using the generator notation allows the multiplications of the elements of the finite field to be carried out without reference to the irreducible polynomial.

Represent Elements of $GF(p^n)$ Using a Generator

- For an irreducible polynomial $f(x)$ of degree n over a field $\mathbb{F}_p$, let α be a root of $f(x)$. Then the field $\mathbb{F}_p[x]/(f(x))$ can be represented as

$$\mathbb{F}_p[\alpha] = \{a_0 + a_1\alpha + \cdots + a_{n-1}\alpha^{n-1} : a_i \in \mathbb{F}_p\}$$

- If α is a root of an irreducible polynomial of degree n over $\mathbb{F}_p$, and it is also a primitive element of $\mathbb{F}_{p^n} = \mathbb{F}_p[\alpha]$.

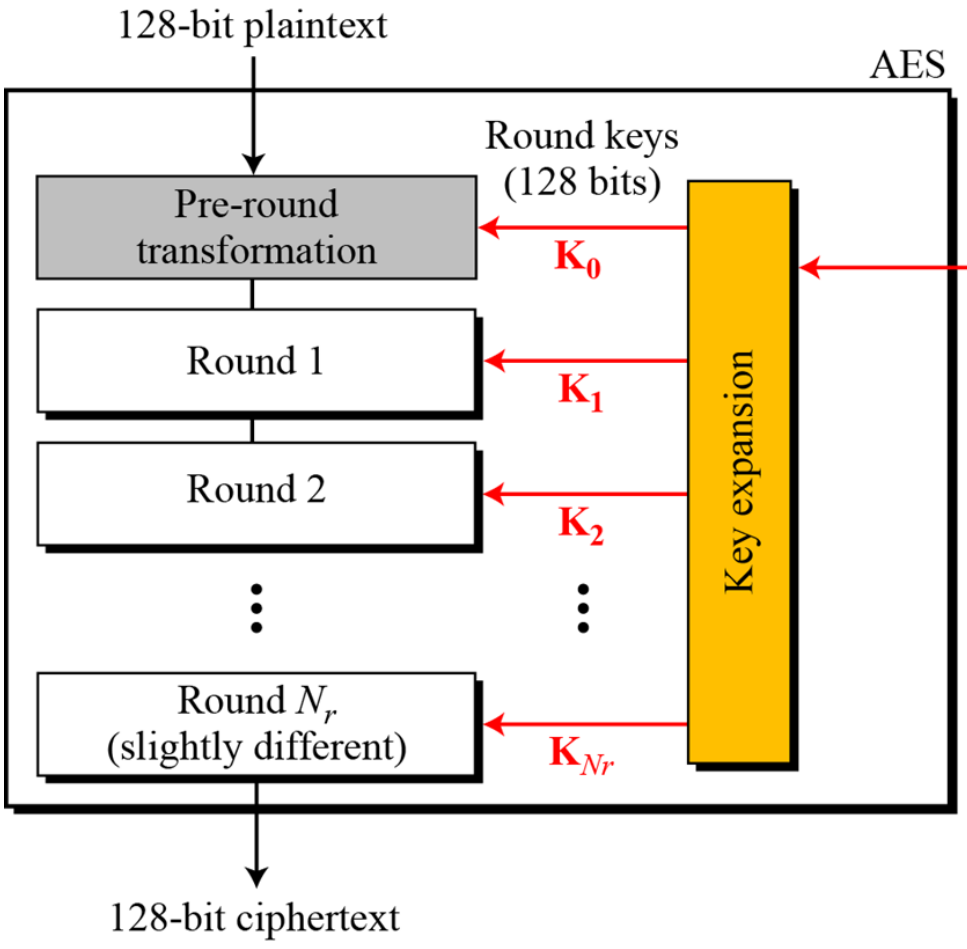
$$\begin{aligned}\mathbb{F}_{p^n} &= \{a_0 + a_1\alpha + \cdots + a_{n-1}\alpha^{n-1} : a_i \in \mathbb{F}_p\} \\ &= \{0, \alpha, \alpha^2, \cdots, \alpha^{p^n-1}\}\end{aligned}$$

高级数据加密标准

- AES, notified by NIST as a standard in 2001, is a slight variation of the Rijndael cipher invented by two Belgian cryptographers Joan Daemen and Vicent Rijmen. `http://en.wikipedia.org/wiki/Advanced_Encryption_Standard`.
- The criteria defined by NIST for selecting AES fall into three areas: Security, Cost and Implementation. `http://www.nist.gov/aes/`
- AES is a non-Feistel cipher that encrypts and decrypts a data block of 128 bits. 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.

AES Structure

- General design of AES encryption is as follows.

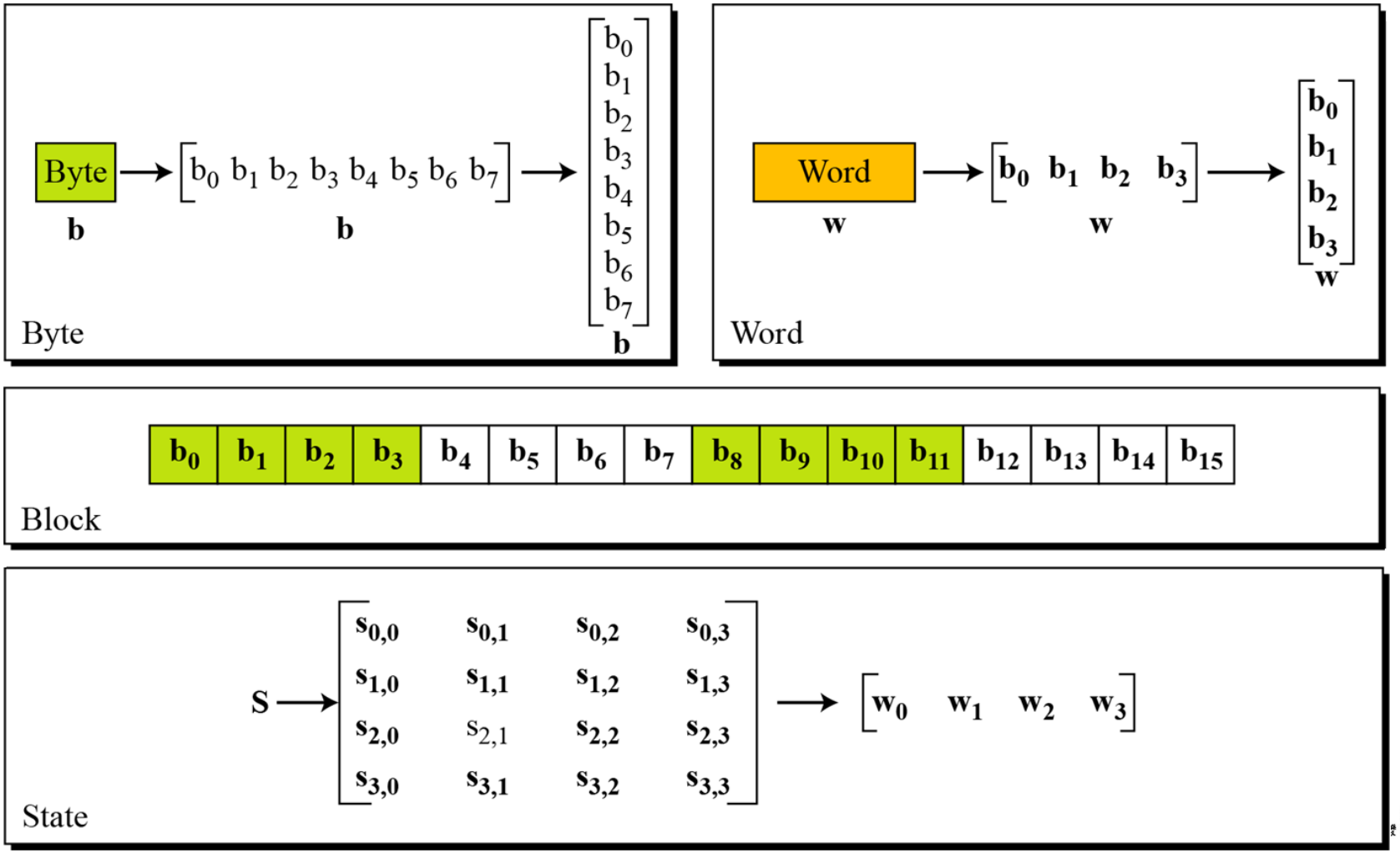


Cipher key
(128, 192, or 256 bits)

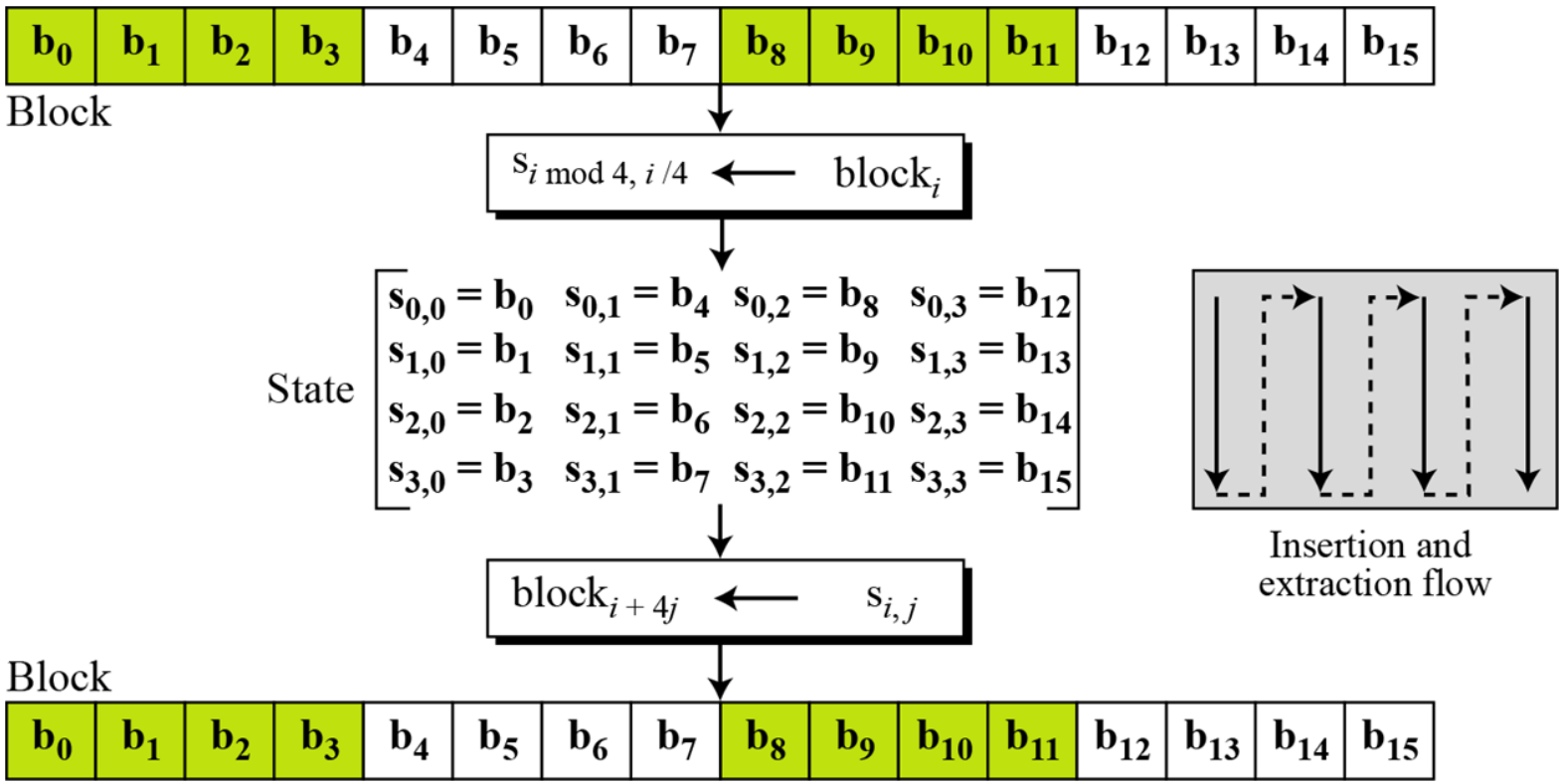
Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size

Data Unit in AES



Data Unit in AES



Data Unit in AES

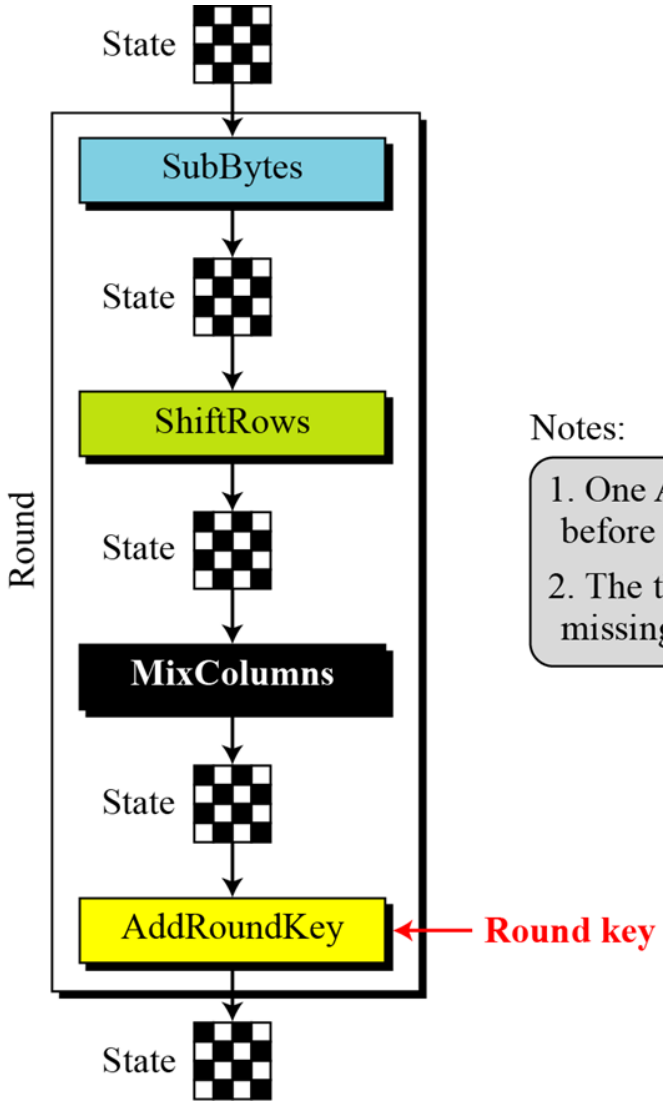
- A 128-bit block consists of a 4×4 matrix of bytes, which is referred to as the **state array**.

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

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

- Each round of processing works on the **input state array** and produces an **output state array**.

Structure of Each Round



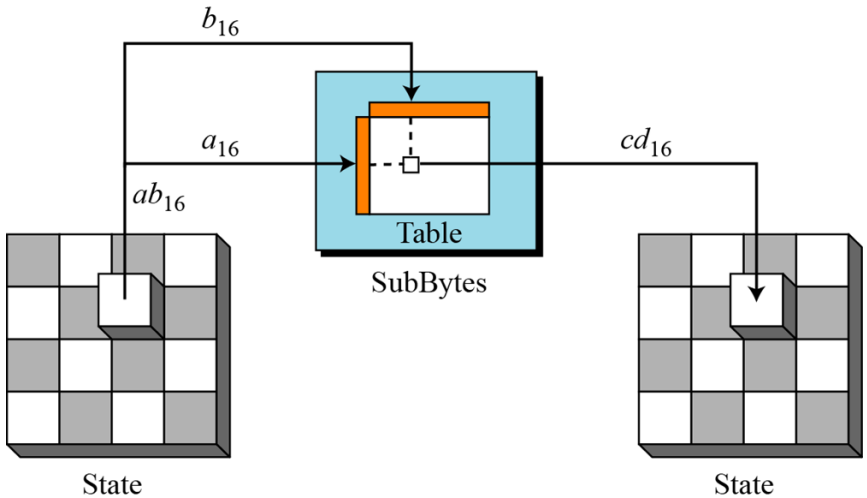
Notes:

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

SubBytes Step

- SubBytes for byte-by-byte substitution, the corresponding substitution step used during decryption is called InvSubBytes.

- This step consists of using a 16×16 lookup table to find a replacement byte for a given byte in the input state array.



SubBytes Step

	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
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

SubBytes Step

- Entries in the lookup table are created by using the notions of multiplicative inverses in $GF(2^8)$ and bit scrambling to destroy the bit-level correlations inside each byte.
- First fill each cell of the 16×16 table with the byte obtained by joining together its row index and the column index. For example, for the cell located at row 3 and column A, we place $3A$ in the cell.
- Replace the value in each cell by its multiplicative inverse in $GF(2^8)$ based on the irreducible polynomial

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

The value 00 is replaced by itself.

SubBytes Step

- Let's represent a byte stored in each cell of the table by $\mathbf{b} = (b_7 b_6 b_5 b_4\, b_3 b_2 b_1 b_0)$. For example, the byte stored in the cell $(9, 5)$ of the above table is the multiplicative inverse of $0x95$, which is $0x8A = (1000\,1010)$.
- For bit mangling, we apply the following transformation to each bit b_i of the byte stored in a cell of the lookup table:

$$\begin{aligned} b'_i &= b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \\ &\quad \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \end{aligned}$$

where $\mathbf{c} = (c_7 c_6 c_5 c_4\, c_3 c_2 c_1 c_0) = (0110\,0011)$.

SubBytes Step

- 上面的变换用矩阵描述

$$\mathbf{b}' = \mathbf{X}(s_{i,j})^{-1} \oplus \mathbf{c} = \mathbf{X}\mathbf{b} \oplus \mathbf{c}$$

$$\begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} \oplus \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

InvSubBytes Step

$$\mathbf{b}' = \mathbf{X}\mathbf{b} \oplus \mathbf{c} \implies \mathbf{b} = \mathbf{X}^{-1}(\mathbf{b}' \oplus \mathbf{c})$$
$$\mathbf{X}^{-1} \triangleq \mathbf{Y}, \mathbf{X}^{-1}\mathbf{c} = \mathbf{d} \implies \mathbf{b} = \mathbf{Y}\mathbf{b}' \oplus \mathbf{d}$$

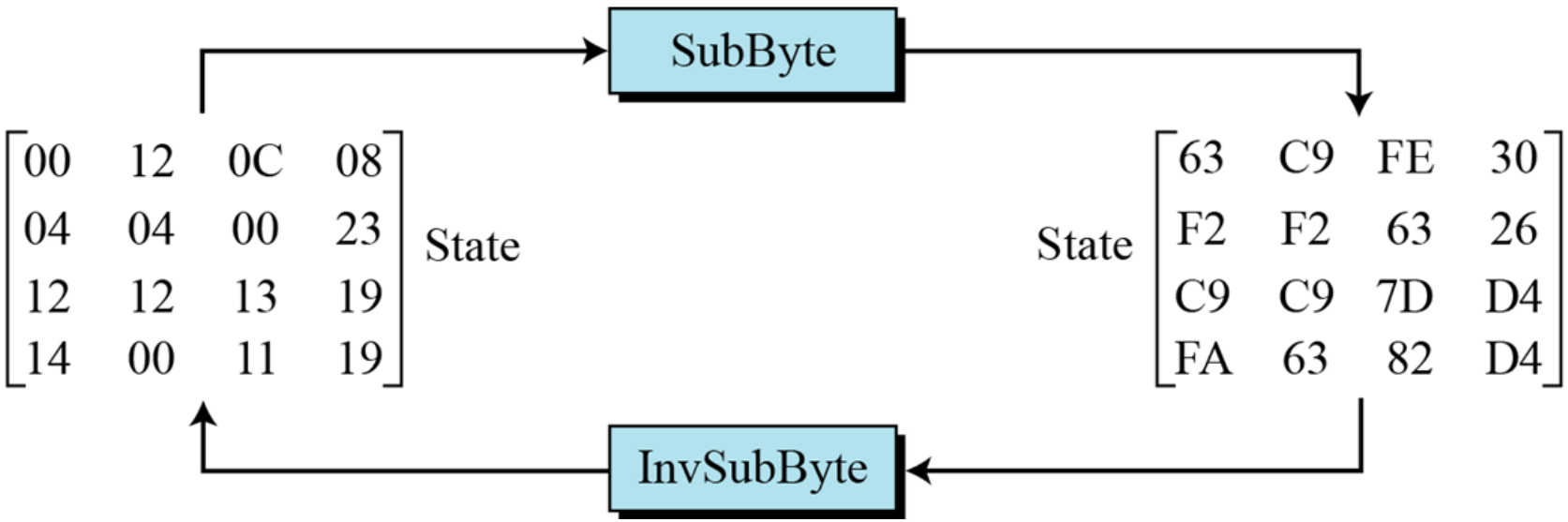
$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} \oplus \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

InvSubBytes Step

	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
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

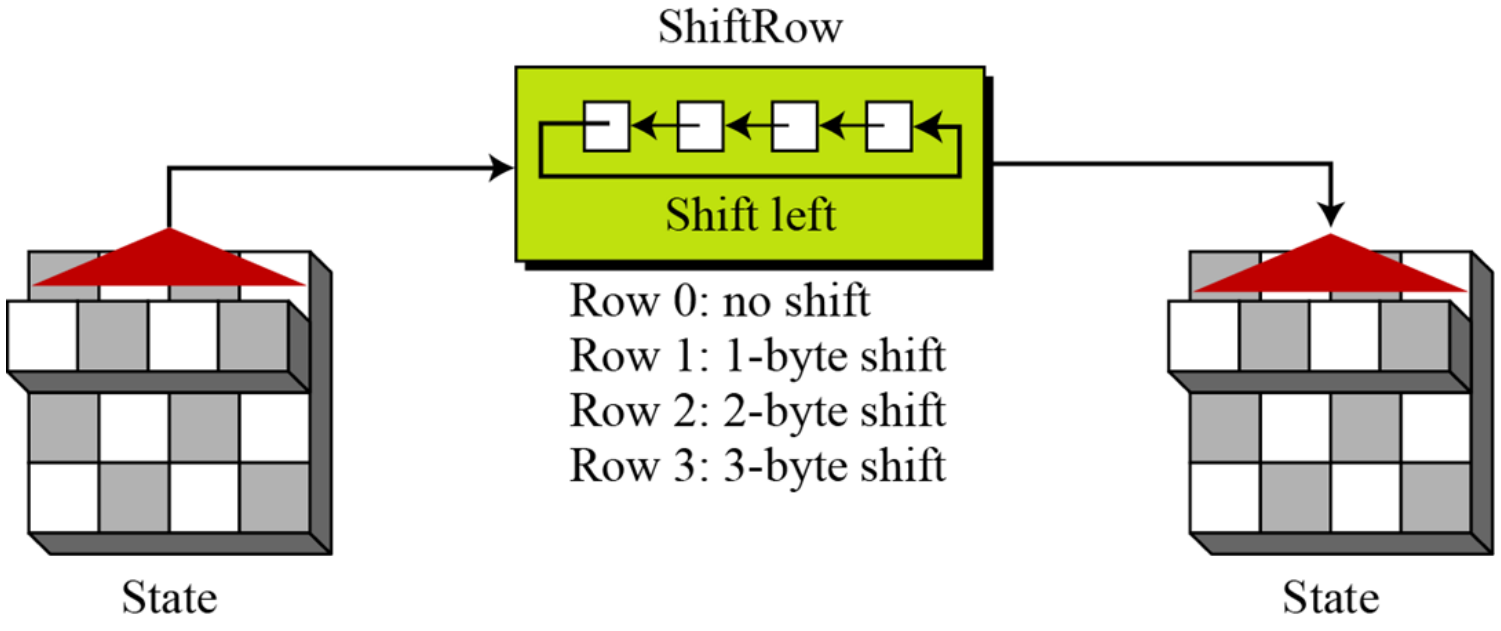
Example for SubBytes and InvSubBytes

- The following figure shows between SubBytes transformation and InvSubBytes transformation. Note that if the two bytes have the same values, their transformation is also the same.



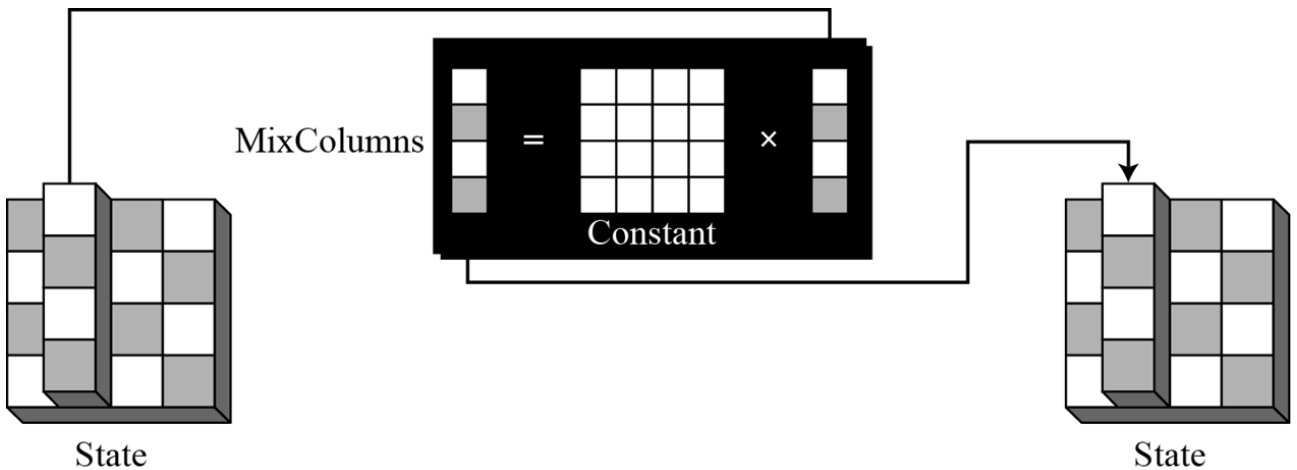
ShiftRows Step

- Rows shifted over different offsets. Purpose: diffusion over the columns by permuting the bytes.
- In the encryption, the transformation is called ShiftRows.
- In the decryption, the transformation is called InvShiftRows and the shifting is to the right.



MixColumns Step

- The MixColumns transformation operates over finite field $GF(2^8)$, bytes in columns are combined linearly.
- Good diffusion properties over rows.
- The InvMixColumns transformation is basically the same as the MixColumns transformation.



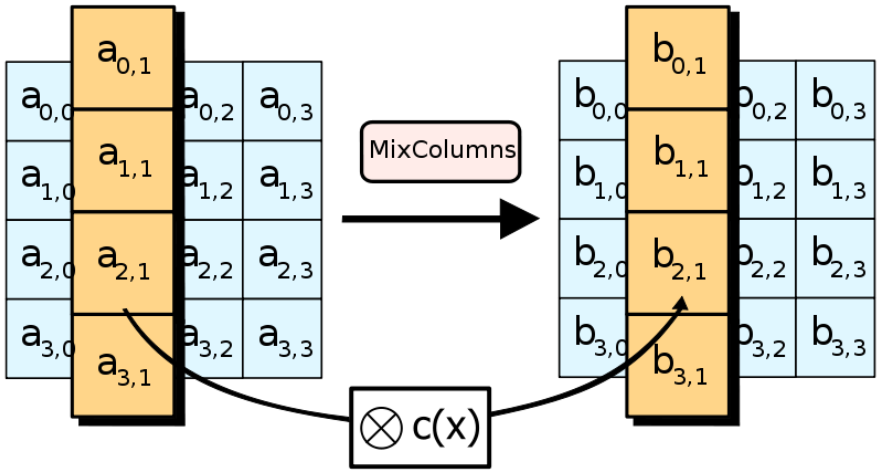
MixColumns Step

- 矩阵的系数是基于在码字间的最大距离的线性编码, 这使得在每列的所有字节中有良好的混淆性.
- 列混淆变换的系数是基于算法执行效率考虑.

$$\mathbf{C} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \leftrightarrow \mathbf{S}' = \mathbf{CS}$$

MixColumns Step

- 列混淆变换也可以通过将**S**的每列看作是一个系数在$GF(2^8)$上的4次多项式, 每列乘上一个固定的多项式$c(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$, 再模$x^4 + 1$.



InvMixColumns Step

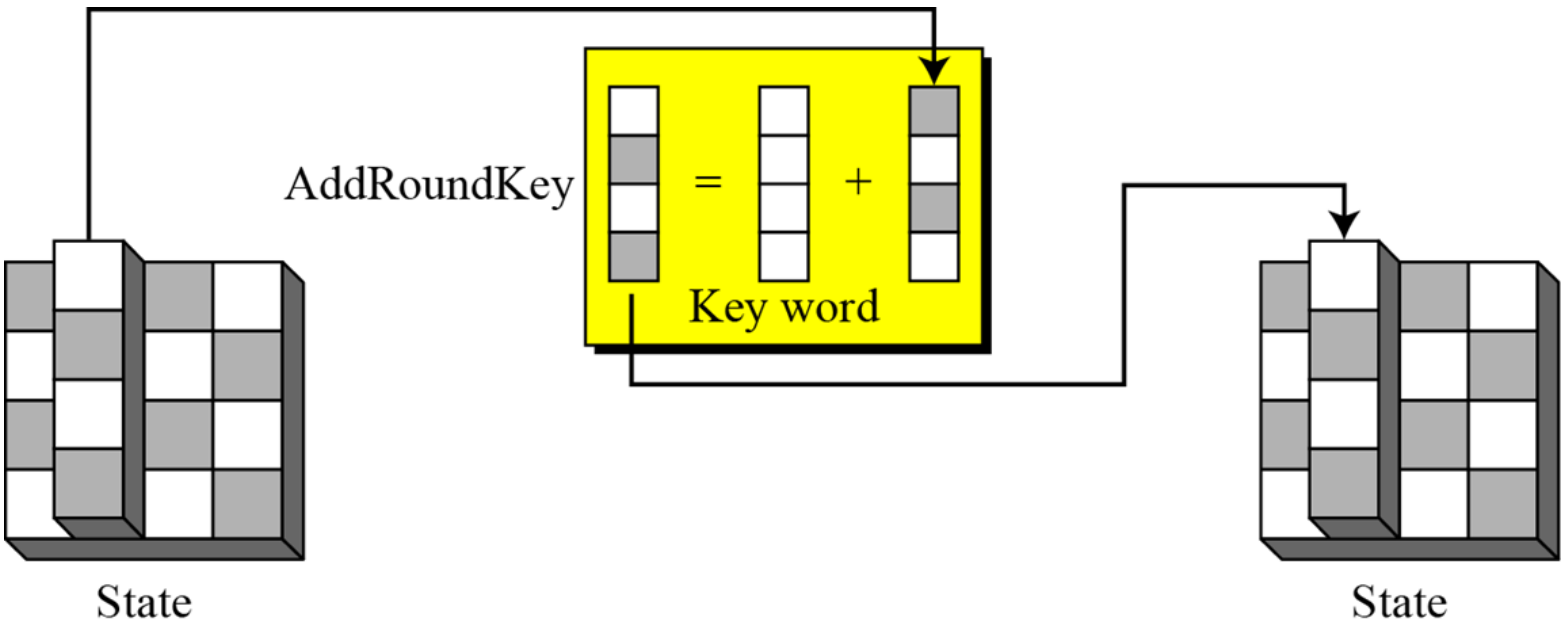
- 逆向列混淆变换可表示为

$$\mathbf{C}^{-1} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \leftrightarrow \mathbf{S} = \mathbf{C}^{-1}\mathbf{S}'$$

- 类似地, 逆向列混淆变换也可以通过将$\mathbf{S}'$的每列看作是一个系数在 $GF(2^8)$上的4次多项式, 每列乘上一个固定的多项式 $d(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$, 再模$x^4 + 1$.
- 容易发现 $d(x) = c^{-1}(x) \bmod x^4 + 1$.

AddRoundKey Step

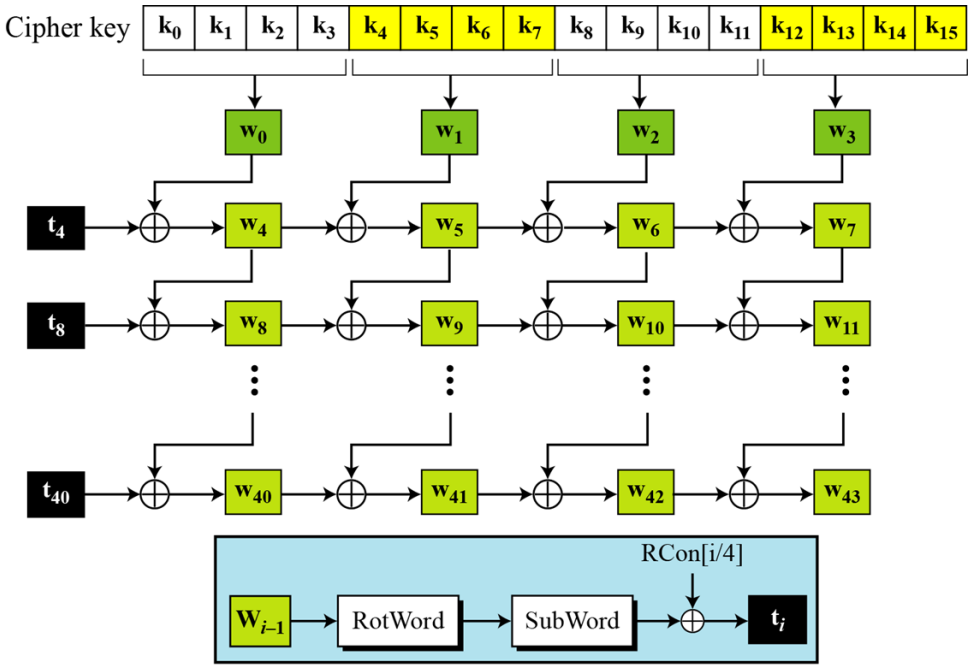
- AddRoundKey proceeds one column at a time. AddRoundKey adds a round key word with each state column matrix.



The AddRoundKey transformation is the inverse of itself.

AES Key Expansion

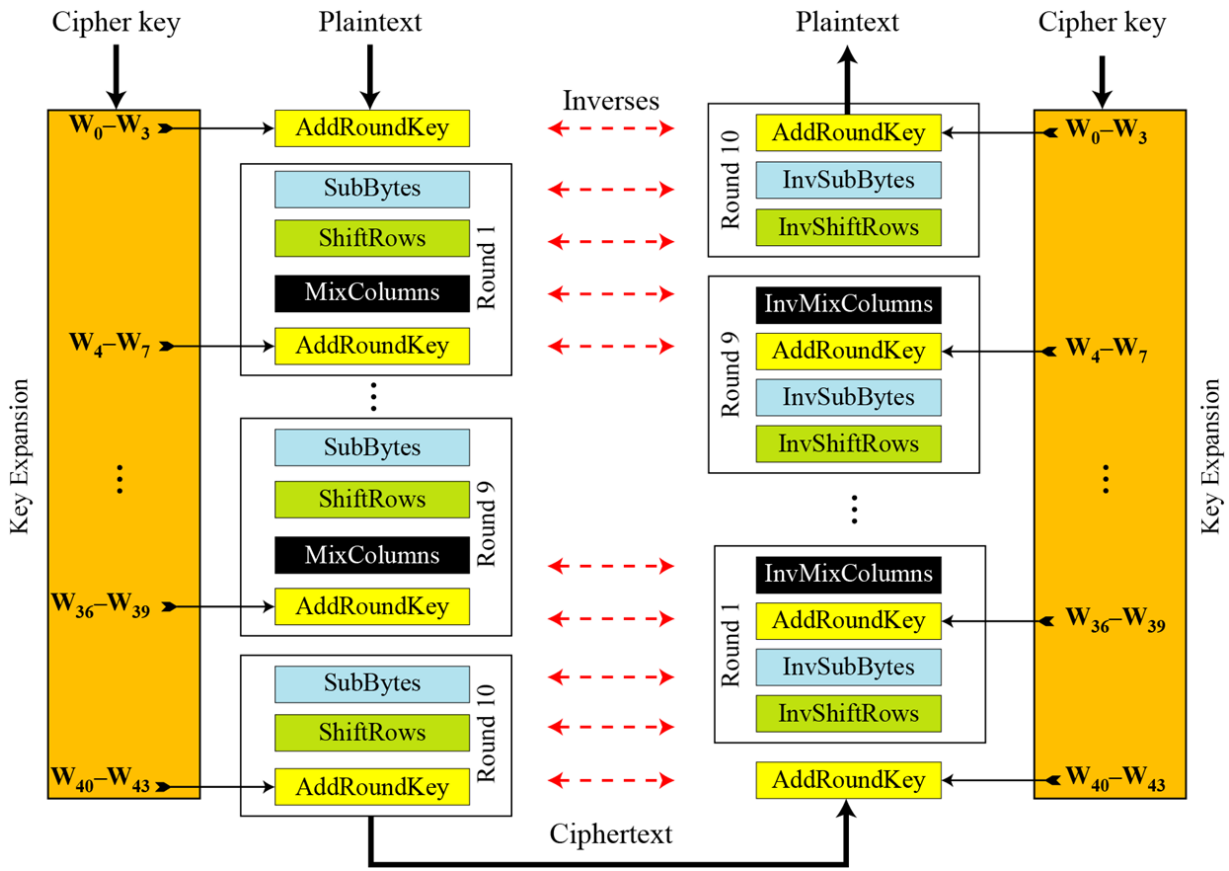
- To create round keys for each round, AES uses a key-expansion process. If the number of rounds is N_r , the key-expansion routine creates $N_r + 1$ 128-bit round keys from one single 128-bit cipher key.
- Key Expansion in AES-128



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

AES

- Although, overall, the same steps are used in encryption and decryption, the order in which the steps are carried out is different.



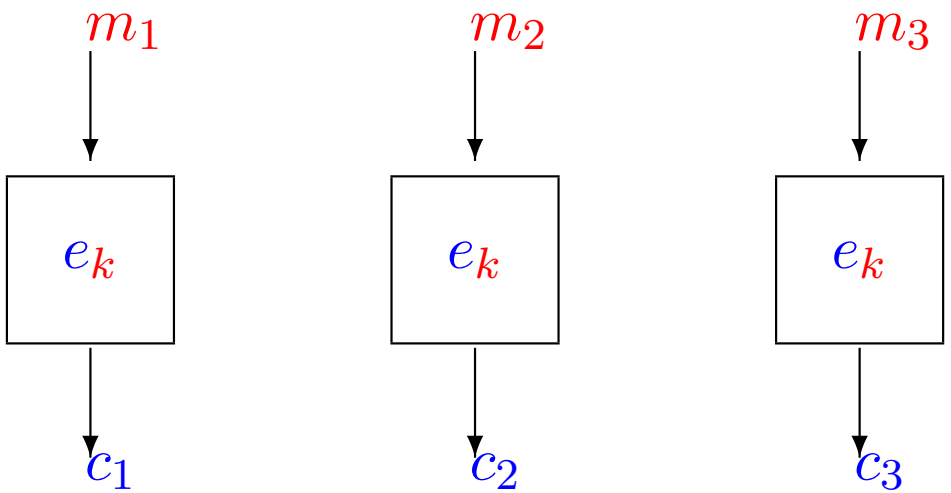
分组密码运行模式

Modes of operation have been devised to encipher text of any size employing block ciphers, `http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation`.

- 电码本模式 (Electronic Code Book, ECB)
- 密文分组链接模式 (Cipher Block Chaining Mode, CBC)
- 密文反馈模式 (Cipher Feedback Mode, CFB)
- 输出反馈模式 (Output Feedback Mode, OFB)
- 计数器模式 (Counter Mode, CTR)

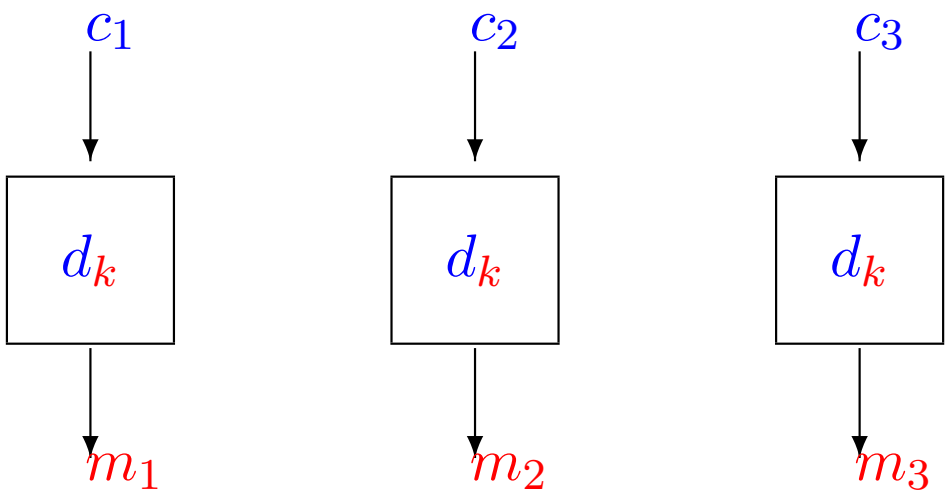
ECB Mode

- ECB is the native mode and one message block is encrypted independently of the encryptions of other blocks.
- Plaintext m is divided into t blocks of n bits $m_1, \ldots, m_t$, the last block is padded if necessary.
- The ciphertext blocks $c_1, \ldots, c_t$ are defined as $c_i = e_k(m_i)$.



ECB Mode

- The decryption process is described as $m_i = d_k(c_i)$.



ECB Mode

- The encryption process can be processed in parallel, and it offers the property of random access since any block can be decrypted independently of any other.
- The encryption of any other block would be unaffected, error propagation is contained within the block.
- ECB mode is not well suited to the encryption of longer messages and there is always the serious risk of information leakage since equal message blocks are encrypted into identical ciphertext blocks under the same key. ECB mode might be used for the encryption of keys of fixed length that fit within a single block.
- Attacker can initiate active attacks on plaintext.

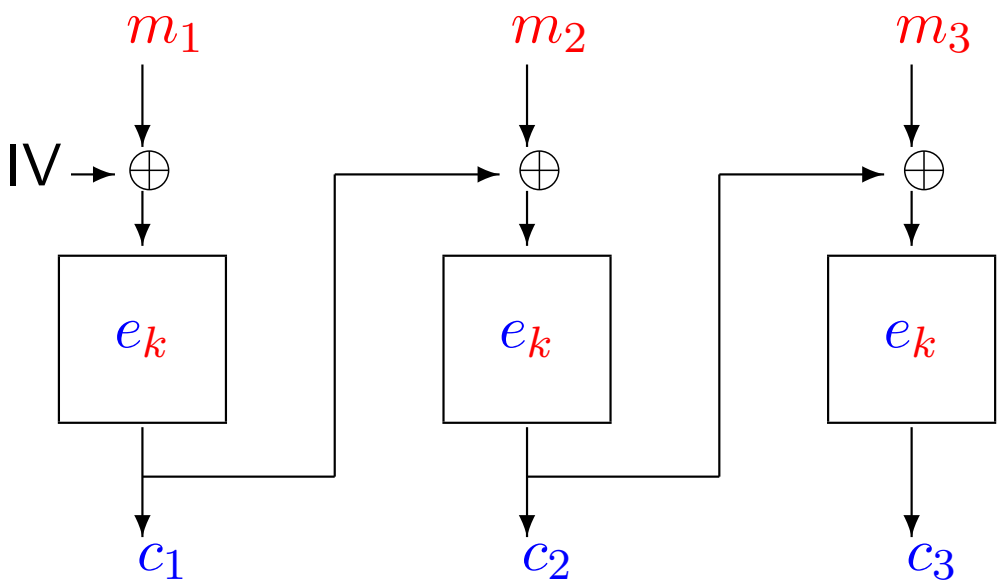
An Active Attack Example on ECB Mode

- Assume that Eve works in a company a few hours per month. She knows that the company uses several blocks of information for each employee in which the seventh block is the amount of money to be deposited in the employee's account.
- Eve can intercept the cipher-text sent to the bank at the end of the month, replace the block with the information about her payment with a copy of the block with the information about the payment of a full-time colleague.
- Each month Eve can receive more money than she deserves.

CBC Mode

- Plaintext m is divided into t blocks of n bits $m_1, \ldots, m_t$. The last block is padded if necessary.
- The encryption process is described as

$$\begin{aligned} c_1 &= e_k(m_1 \oplus \mathrm{IV}) \\ c_i &= e_k(m_i \oplus c_{i-1}) \text{ for } 2 \le i \le t. \end{aligned}$$

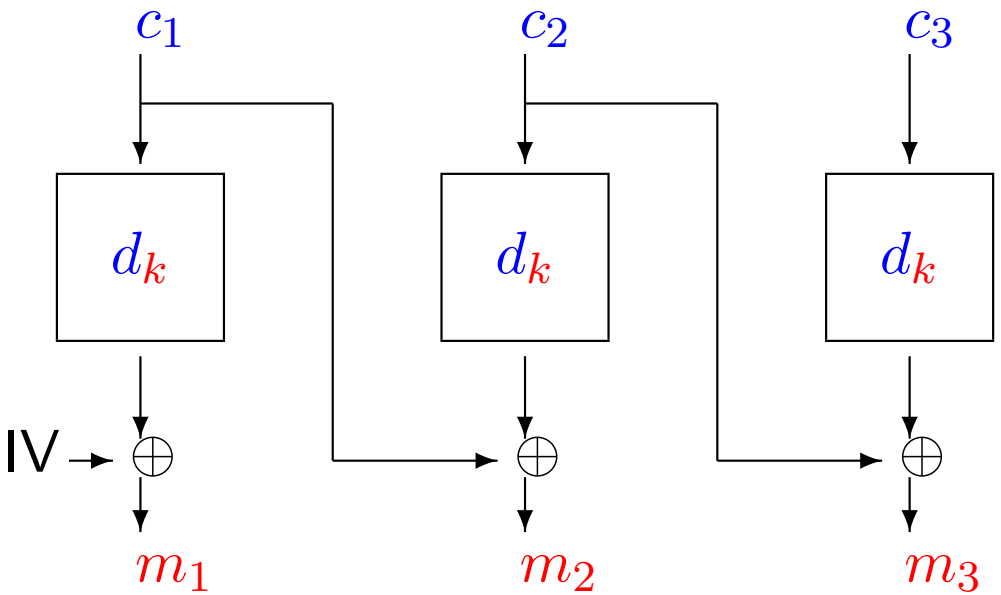


CBC Mode

- The decryption process is described as

$$\begin{aligned} m_1 &= d_k(c_1) \oplus \mathrm{IV} \\ m_i &= d_k(c_i) \oplus c_{i-1} \text{ for } 2 \le i \le t. \end{aligned}$$

- IV is the same initial value that was used by the sender.



CBC Mode

- The initialization vector (IV) is random, but need not be secret. Randomized encryption.
- Note that the CBC encryption of any single plaintext block depends on all previous ciphertext blocks and so the encryption process cannot be parallelised.
- For error propagation, an error in the single ciphertext block c_i will ensure that the recovered value of m_i is garbage and after that the recovered value of m_{i+1} will be in error. However, the error pattern in m_{i+1} will be identical to the error pattern in c_i. After this, provided two consecutive ciphertext blocks are received without error and correctly aligned, the second ciphertext block can be correctly decrypted. Thus CBC mode has a self-synchronising property.

CBC Mode

- CBC mode exhibits a limited form of information leakage.
- Suppose that we have $c_i = c_j$ for some $1 \leq i, j \leq n$ with $i \neq j$. Then we know that

$$\begin{aligned} c_i = c_j &\Rightarrow e_k(m_i \oplus c_{i-1}) = e_k(m_j \oplus c_{j-1}) \\ &\Rightarrow m_i \oplus c_{i-1} = m_j \oplus c_{j-1} \\ &\Rightarrow m_i \oplus m_j = c_{i-1} \oplus c_{j-1}. \end{aligned}$$

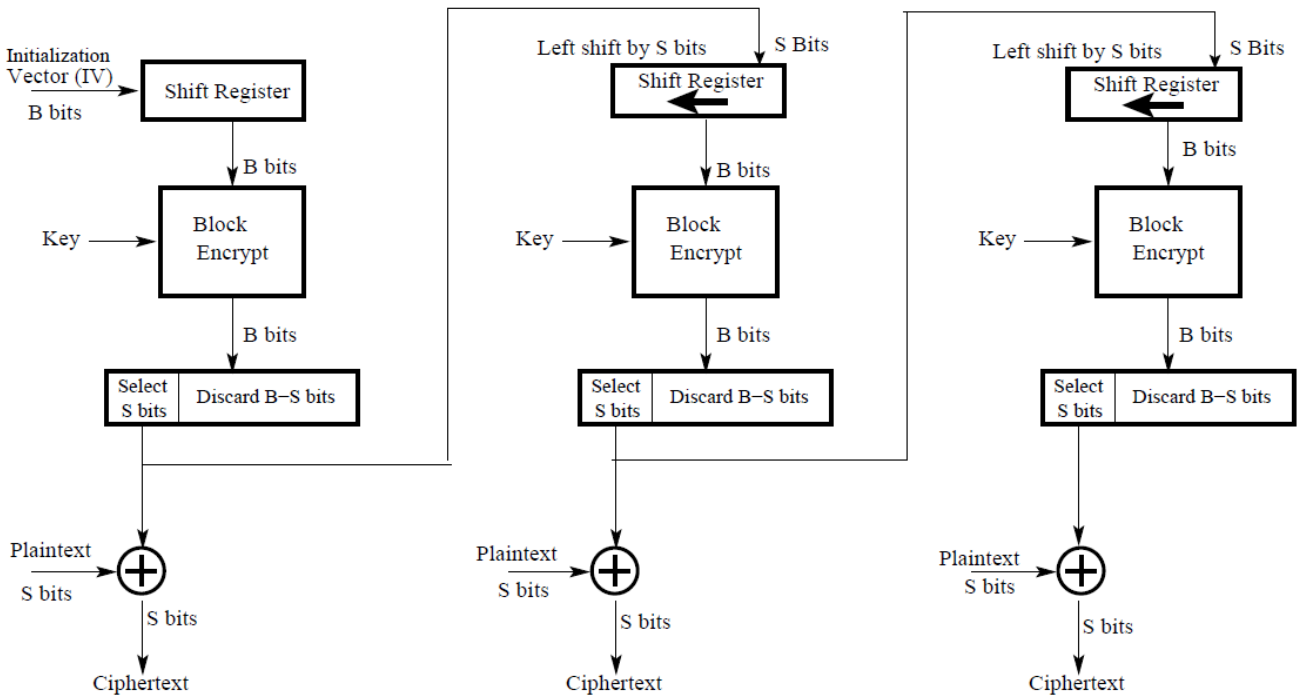
- Thus, when two blocks of ciphertext are observed to have the same value, we can recover the value of the exclusive-or of two unknown plaintext blocks and information is leaked.

输出反馈模式

OFB 将一个分组密码转换为一个密钥序列产生器, 从而可以实现用分组密码按流密码的方式进行加解密.

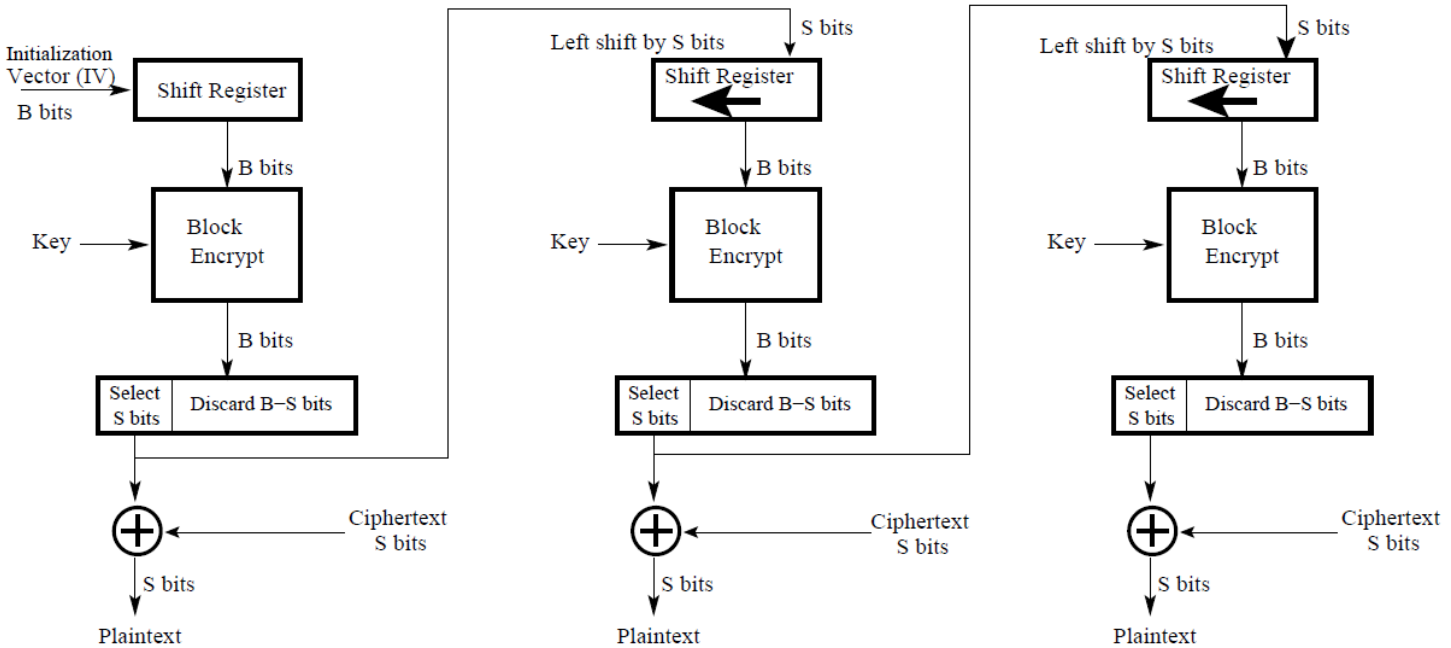
- Block length of block cipher is n, one can choose to use keystream blocks of $j \leq n$ bits only.
- Divide plaintext into a series of j-bit blocks $m_1, \ldots, m_t$.
- Encryption: $c_i = m_i \oplus e_i$, where e_i is the selection of j bits of ciphertext generated by block cipher, starting with IV in shift register.

OFB Mode



OFB Encryption

OFB Mode



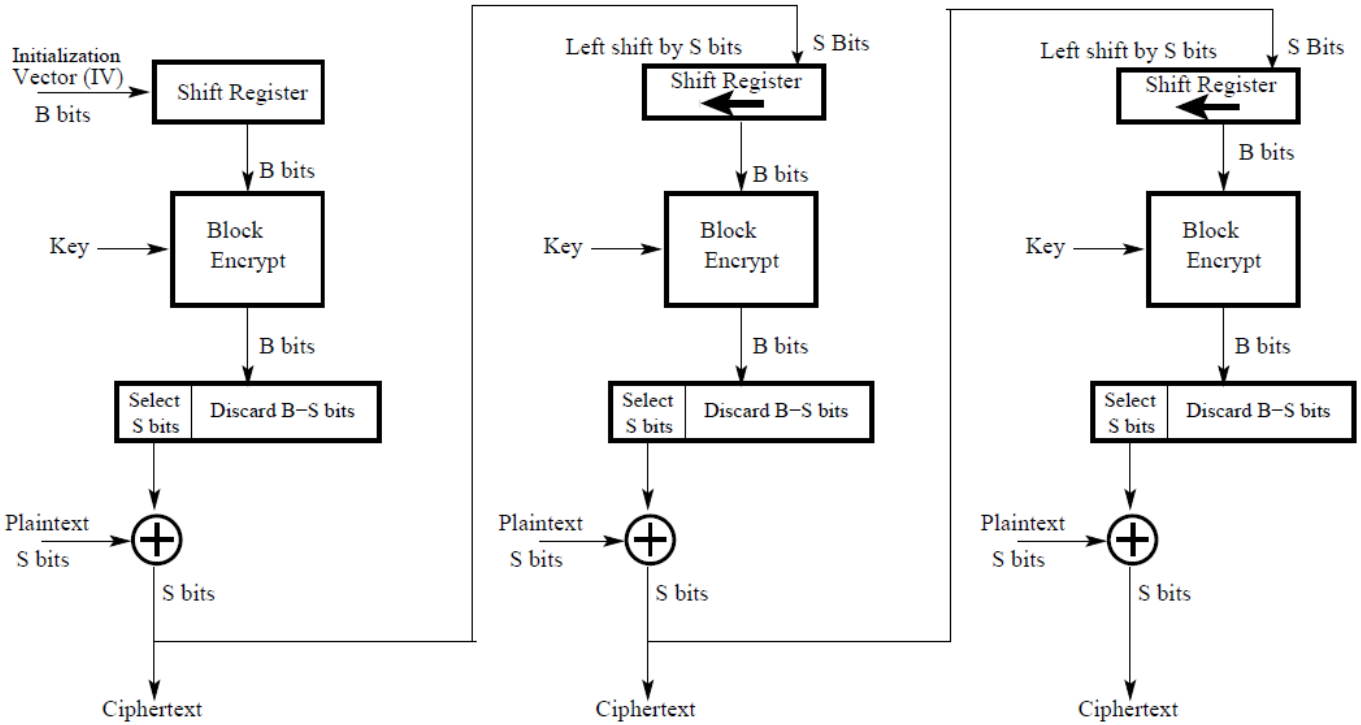
OFB Decryption

OFB Mode

- Key stream independent of plaintext: can be precomputed.
- Synchronous stream cipher.
- Different IV necessary; otherwise insecure.
- Only uses encryption (no decryption algorithm necessary).
- No error propagation: errors are only copied.
- 如果分组密码是安全的, 则产生的密钥序列也是安全的.
- 加密和解密都没有错误传播 (No error propagation), 难以检测密文的篡改.

密文反馈模式

不同于OFB模式, 把密文反馈到移位寄存器.



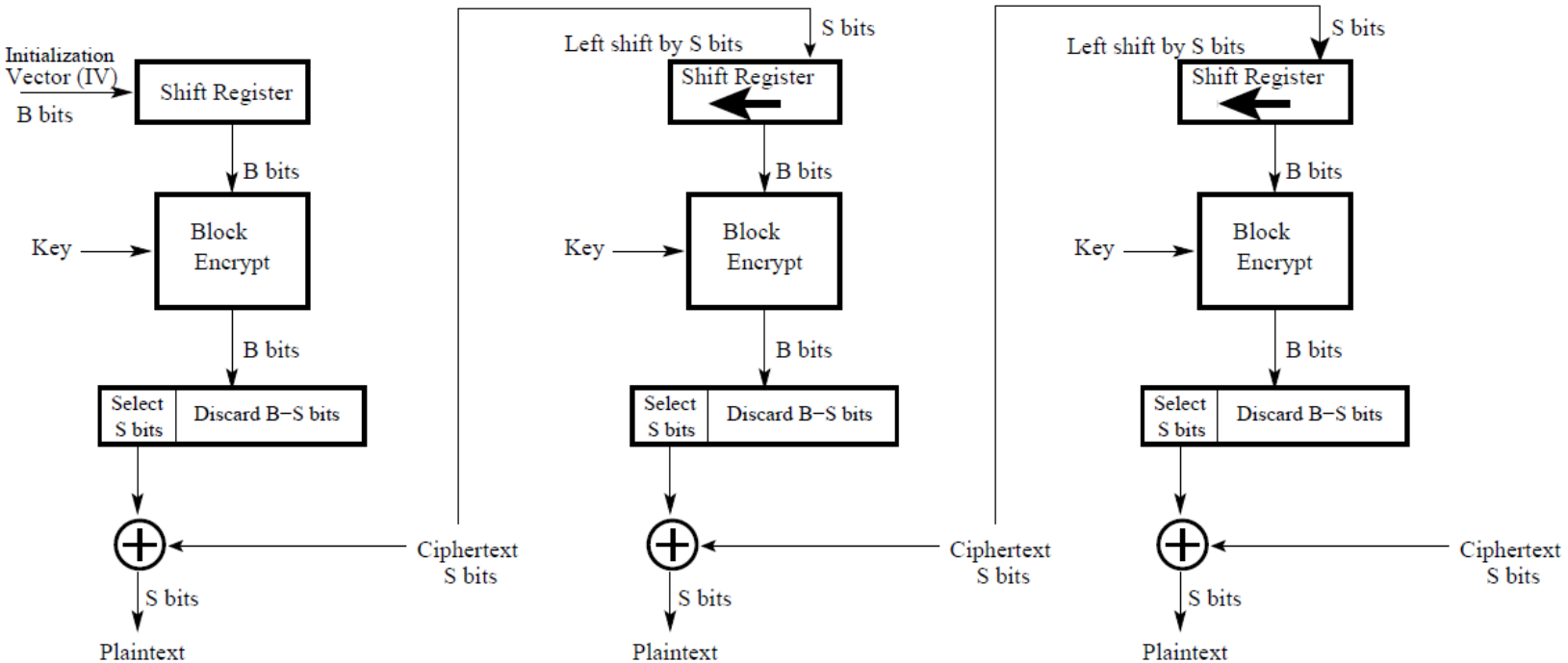
CFB Encryption

- Note that the ciphertext byte produced for any plaintext byte depends on all the previous plaintext bytes.

密文反馈模式

- Start with an Initialization Vector IV of the same size as the block size expected by the block cipher. The IV is stored in shift register.
- Encrypt the IV with the block cipher encryption algorithm.
- Retain only s bits from the output of the encryption algorithm, discard the rest of the output.
- XOR the s bits retained with the s bits of the plaintext that needs to be transmitted.
- Shift the IV s bits to the left (discarding the leftmost bits) and insert the ciphertext byte produced by the previous step as the rightmost byte.
- Go back to the step "Encrypt the IV with the block cipher encryption algorithm".

CFB Mode



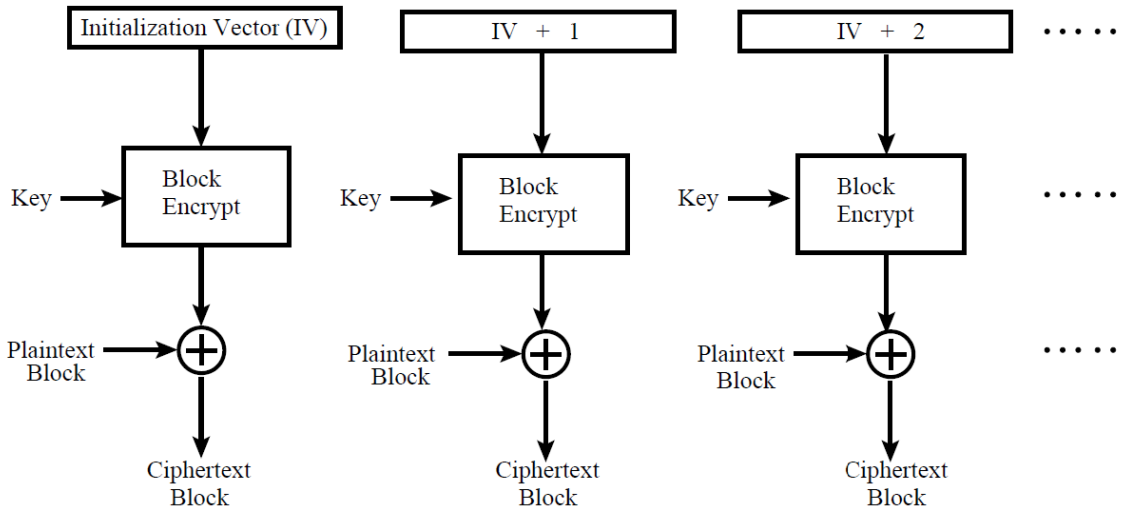
CFB Decryption

CFB Mode

- Self-synchronizing stream cipher. No synchronization needed between sender and receiver, synchronization if n bits have been received correctly.
- Different IV hides patterns and repetitions. Different IV necessary; otherwise insecure.
- Only uses encryption (no decryption algorithm necessary).
- 加密时若明文错了一位, 则影响相应的密文错, 这一错误反馈到移位寄存器后将影响到后续的密钥序列错, 导致后续的密文都错.
- 解密时若密文错了一位, 则影响相应的明文错, 密文的这一错误反馈到移位寄存器后将影响到后续的密钥序列错, 导致后续的明文都错.
- 因错误传播无界, 可用于检查发现对明文或密文的篡改.

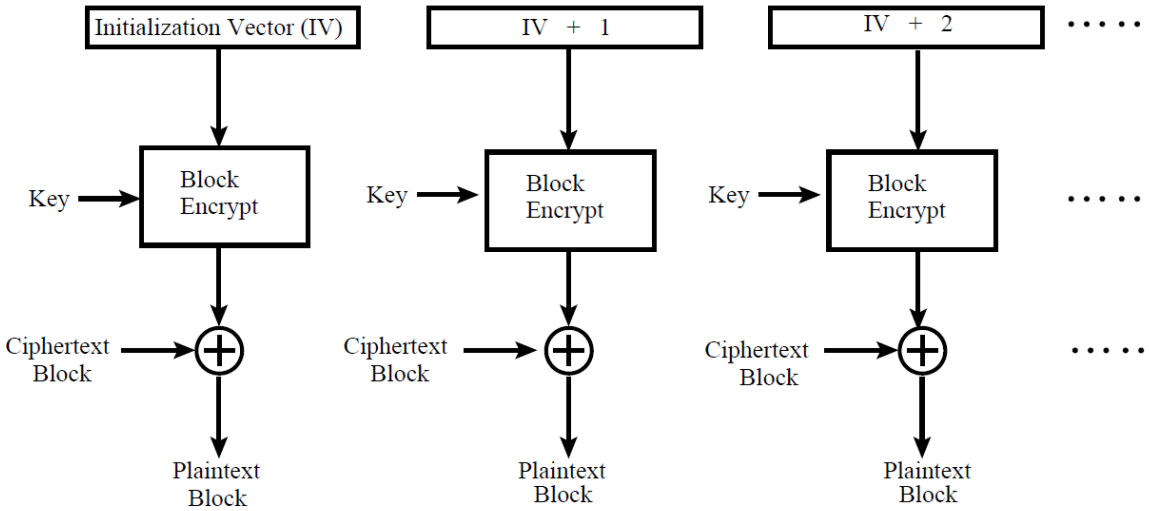
CTR Mode

- In CTR mode, there is no feedback. The pseudo randomness in the key stream is achieved using a counter.



CTR Encryption

CTR Mode



CTR Decryption

- Fast encryption and decryption, we can precompute the encryptions for as many counter values as needed. The same applies to fast decryption.
- Because there is no block-to-block feedback, the algorithm can be implemented on parallel machines. For the same reason, any block can be decrypted with random access.

Choosing the IV

- In the case of the OFB and CTR modes, reuse of the same key and IV combination will lead to the same keystream being generated. This would be catastrophic for security.
- So the IV in the OFB and CTR modes should be used only once, such an IV is sometimes referred to as a nonce.
- By contrast, for the CBC and CFB modes we require the IV to be unpredictable. One way to achieve this would be to use a PRE-IV which can be a predictable nonce such as a counter. Then the IV that is used for the actual encryption operation can be generated as $\mathrm{IV} = e_k(\text{PRE-IV})$.

填充技术

- For OFB and CTR modes we don't need to think about padding. Therefore there is no message expansion when using these modes.
- For s-bit CFB, CBC, and ECB modes we might need to pad some input block to a multiple of s bits in the case of CFB mode and a multiple of n bits in the cases of CBC and ECB.
- 用无用的数据填充短块, 使之成为标准块.
- 为了确保加密强度, 填充数据应是随机的.
- 接收方如何知道哪些数字是填充的呢? 这就需要增加指示信息, 通常用最后 8 位作为填充指示符.

填充技术

- Suppose length of plaintext is not multiple of block length, the last block m_t only contains $l < n$ bits.
  - Append $n - l$ 0-bits zeros to last block. Problem is that trailing 0-bits of plaintext cannot be distinguished from padding.
  - Append 1 and $n - l - 1$ 0-bits. If $l = n$, then create extra block starting with 1 and remaining $n - 1$ bits 0.
  - As the one above, but add an extra block which contains the length of the message in bits.
- 填充法适于通信加密而不适于文件和数据库加密.

密文挪用技术

- One way to provide secure encryption in ECB and CBC modes without padding a message, therefore without any message expansion, is to use what is called ciphertext stealing.
- `http://en.wikipedia.org/wiki/Ciphertext_stealing`
- Ciphertext stealing requires that the message be greater than n bits in length, i.e., there must be at least two blocks in the encryption process.
- Consider ECB mode, $c_i = e_k(m_i)$, $m_i = d_k(c_i)$ for $1 \leq i \leq t$, and suppose $|m_t| < n$.
- We compute $c_i = e_k(m_i)$ for $1 \leq i \leq t-1$ as normal, but consider this block as $c_t \parallel J$ where $|c_t| = |m_t|$ and J is $n - |m_t|$ bits of some temporary quantity.

密文挪用技术

- One way to provide secure encryption in ECB and CBC modes without padding a message, therefore without any message expansion, is to use what is called ciphertext stealing (CTS).
- `http://en.wikipedia.org/wiki/Ciphertext_stealing`
- Ciphertext stealing requires that the message be greater than n bits in length, i.e., there must be at least two blocks in the encryption process.

密文挪用技术

- Consider ECB mode using CTS technique, the last two plaintext blocks, m_{t-1} and m_t, are encrypted differently and out of order, as shown below, assuming that m_{t-1} has n bits and m_t has s bits, where $s \leq n$.

$$\begin{aligned} x &= e_k(m_{t-1}) \rightarrow c_t = \mathrm{Head}_s(x) \\ y &= m_t \parallel \mathrm{Tail}_{n-s}(x) \rightarrow c_{t-1} = e_k(y) \end{aligned}$$

Merkle–Damgård construction