

Introduction to Cryptography

Prepared by Dr. Mohammad Alkhatib

Associate Professor

Major: Information Security

جامعة الجميع الذكية

تسعى لتكون الكيان الأول الذي يتبنى مبدأ المجتمع التعليمي
التعاوني ويقدم العلم مقابل العمل لتنمية وبناء المجتمعات.

Lecture #4: AES Algorithm



جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

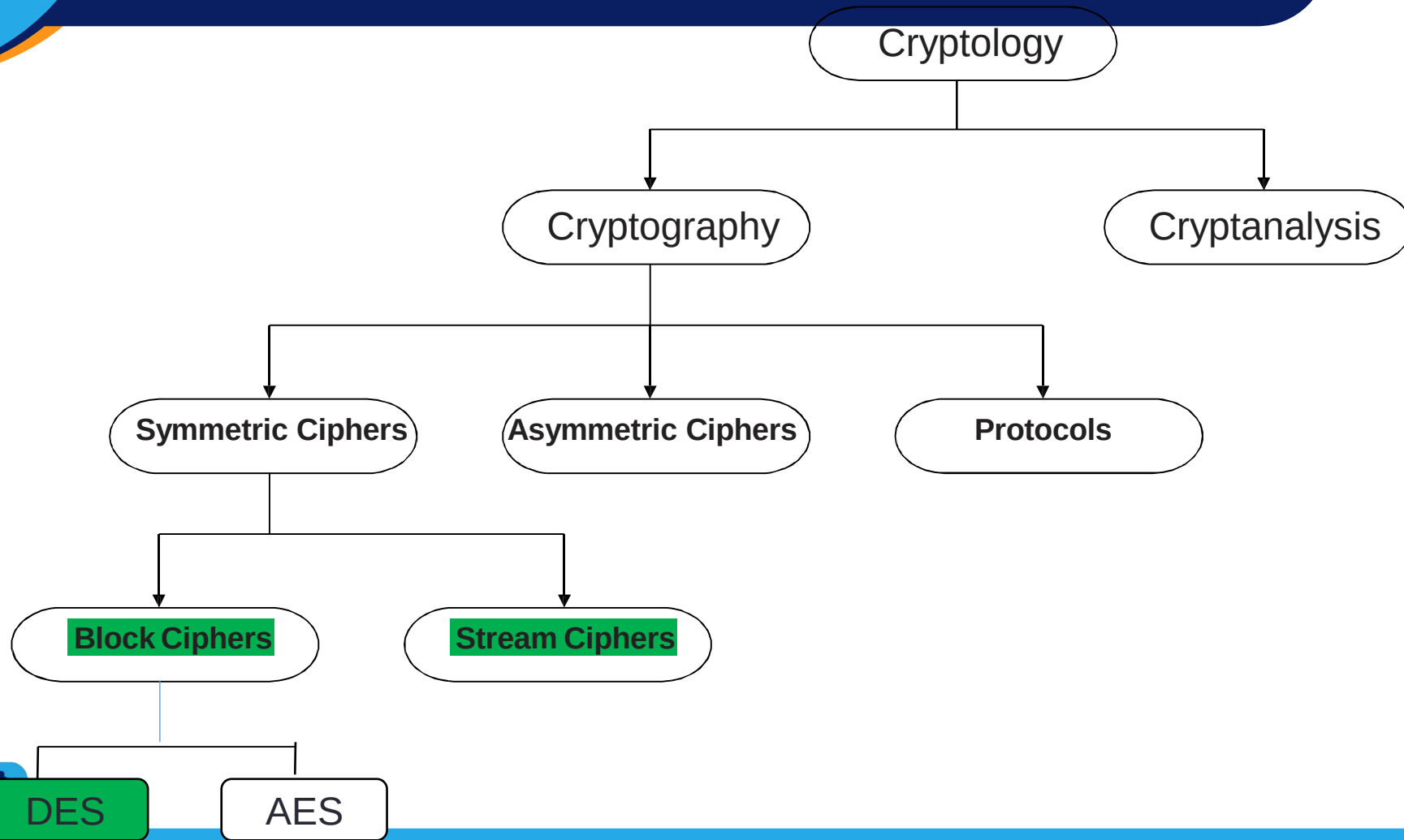
Cryptography Algorithms

AES Algorithm

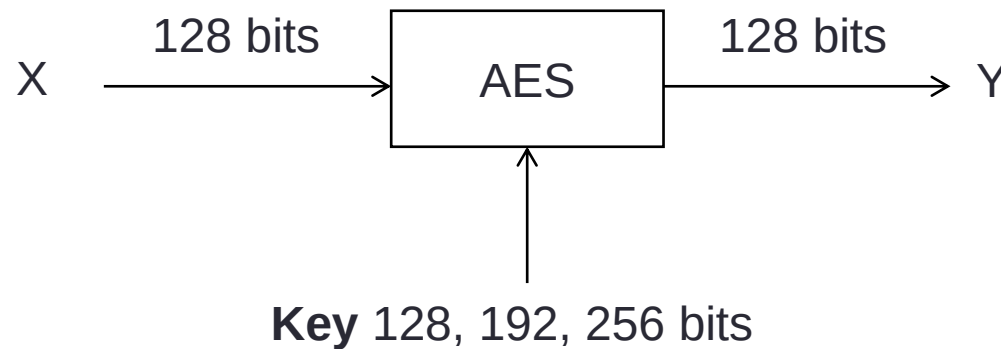
- Introduction to AES
- High-level description
- Confusion & Diffusion for secure block cipher
- Feistel Network
- DES Encryption Process
- Internal Structure of DES Algorithm



Classification of the field of Cryptology



- ❑ AES (Rijndael) uses (encrypts/decrypts) a block length of 128 bits, and support key length of 128, 192, or 256 bits
- ❑ Therefore, three key lengths must be supported by Rijndael as this was a NIST design requirement



- ❑ The number of internal rounds of the AES cipher depends on the key length.

Key Length of AES	Number of Rounds (nr)
128 bits	10
192 bits	12
256 bits	14

For almost all Commercial applications



- ❑ AES does not have a Feistel structure. It is worth mentioning that the Feistel networks do not encrypt an entire block per iteration, while AES encrypts all 128 bits in one iteration.

- ❑ AES contains small number of encryption rounds, compared to DES



❑ Motivations:

1. AES is currently the most important symmetric key algorithm in the world. Researchers estimate that more than 50% of data encrypted in the last decade use AES algorithm.

There are enormous amount of applications that use AES algorithm, such as wifi connection, vlan, etc..

2. NSA allows to encrypt classified data (up to Top Secret) using AES algorithm with key length = 192 bits or 256 bits.

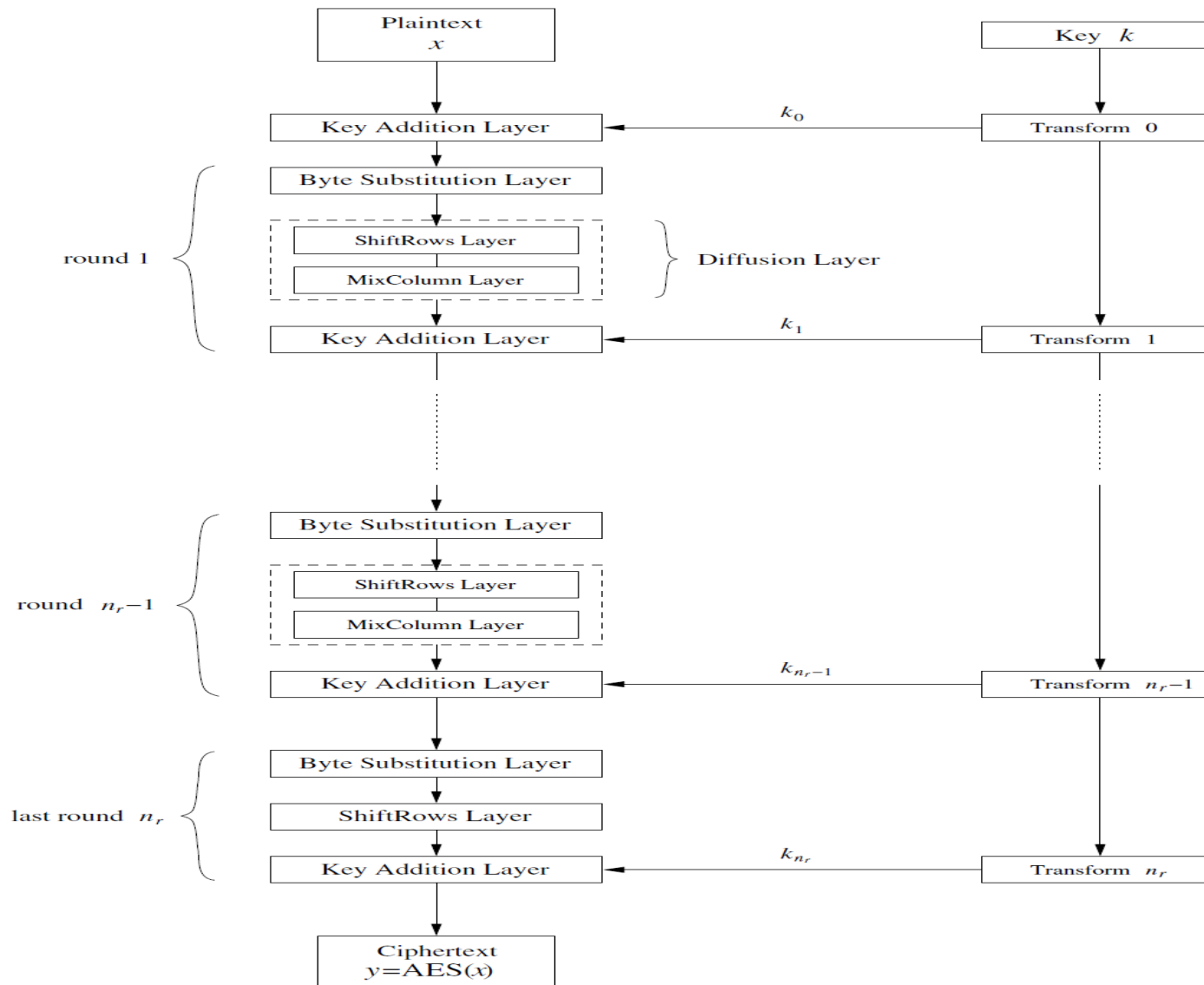
- ❑ AES consists of so-called layers. Each layer manipulates all 128 bits of the data path (also called the state of the algorithm).
- ❑ In AES, there are three different types of layers, as follows:
 1. *Key Addition layer*
 2. *Byte Substitution layer (S-Box)*
 3. *Diffusion layer*
- ❑ Each round, except the first, consists of all three layers
- ❑ The last round n_r does not make use of the MixColumn transformation,

1. **Key Addition layer:** A 128-bit round key (also called subkey), which has been derived from the main key in the key schedule, is XORed to the state (data path to be encrypted (128 bits)).
2. **Byte Substitution layer (S-Box):** Each element of the state is nonlinearly transformed using lookup tables with special mathematical properties, which provides confusion to the data.

3. Diffusion layer: It provides diffusion over all state bits, and consists of two sublayers, as follows:

- i. The **ShiftRows** layer permutes the data on a byte level.
- ii. The **MixColumn** layer is a matrix operation which combines (mixes) blocks of four bytes.

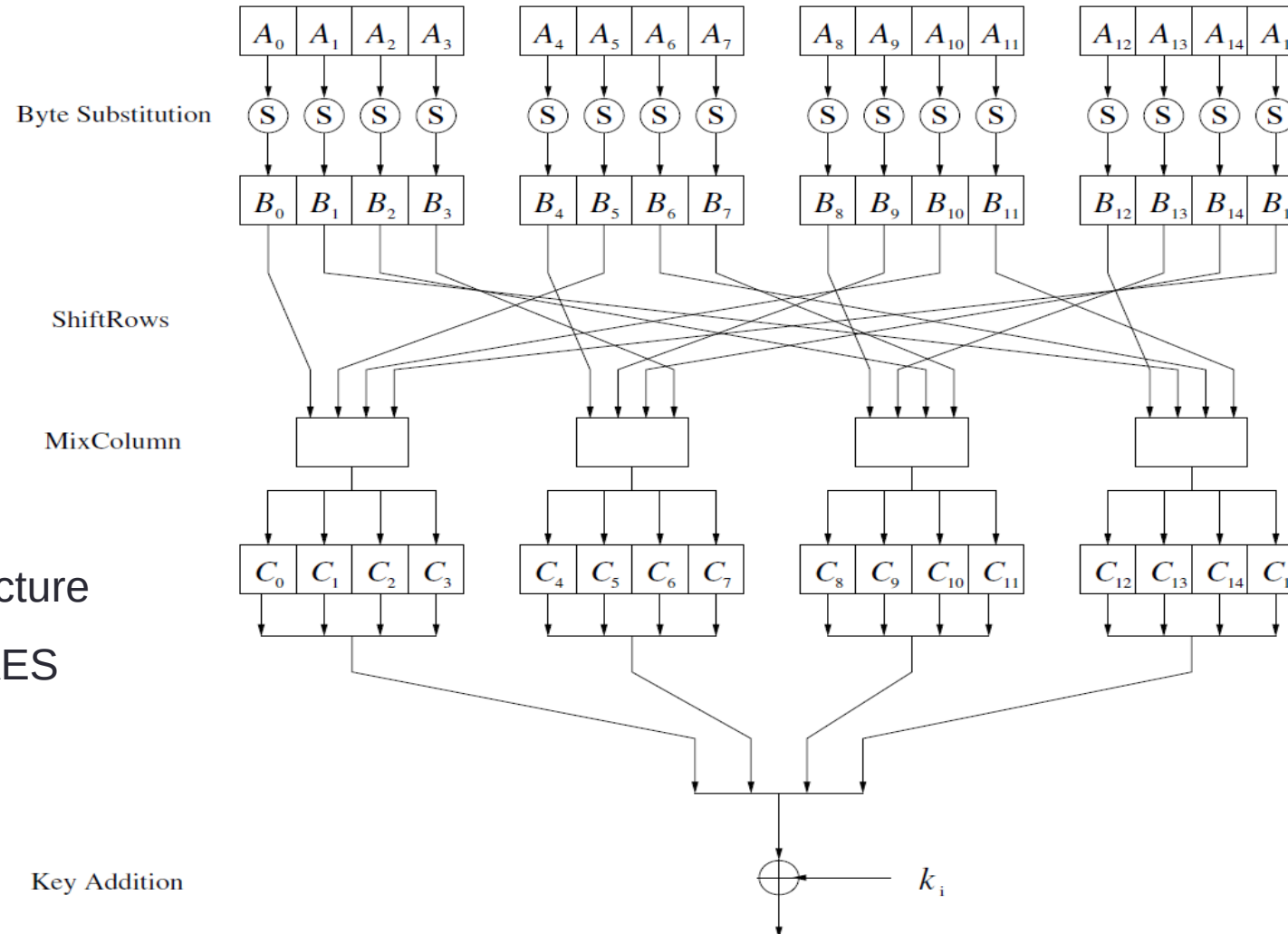
- ❑ Since it is a symmetric key algorithm, it has one main key shared between sender and recipient.
- ❑ The key schedule computes round keys, or subkeys, $(k_0, k_1, \dots, k_{nr})$ from the original AES key.
- ❑ It should be mentioned at this stage that all operations within the AES layers use *Galois field* computations
- ❑ Galois field is a new mathematical concept, which is widely used in cryptographic operations for both symmetric and public-key algorithms



➤ Question: What does happen inside the layers??



Internal structure
of a single AES
round



- Note that AES is not a feistel network algorithm
- Each encryption round of AES can be described as follows:
 - 1) The 16-byte input $A_0, \dots, A_{15}$ is fed byte-wise into the S-Box
 - 2) The 16-byte output $B_0, \dots, B_{15}$ is permuted byte-wise in the ShiftRows layer
 - 3) Then, the output is mixed by the MixColumn transformation $c(x)$.
 - 4) Finally, the 128-bit subkey k_i is XORed with the intermediate result

Note: Unlike DES, the AES is a byte-oriented cipher

Note that at the beginning of AES and the very end, a sub-key is added (key whitening)

In order to understand how the data moves through AES, we first imagine that the state A (i.e., the 128-bit data path) consisting of 16 bytes $A_0, A_1, \dots, A_{15}$ is arranged in a four-by-four byte matrix:

A_0	A_4	A_8	A_{12}
A_1	A_5	A_9	A_{13}
A_2	A_6	A_{10}	A_{14}
A_3	A_7	A_{11}	A_{15}

AES operates on elements, columns or rows of the current state matrix. Similarly, the key bytes are arranged into a matrix with four rows and four (128-bit key), six (192-bit key) or eight (256-bit key) columns. Here is, as an example, the state matrix of a 192-bit key:

k_0	k_4	k_8	k_{12}	k_{16}	k_{20}
k_1	k_5	k_9	k_{13}	k_{17}	k_{21}
k_2	k_6	k_{10}	k_{14}	k_{18}	k_{22}
k_3	k_7	k_{11}	k_{15}	k_{19}	k_{23}

- The Byte Substitution layer is first layer in each round of AES encryption process.
- The Byte Substitution layer can be viewed as a row of 16 parallel S-Boxes, each with 8 input and output bits.
- All 16 S-Boxes are identical in AES, where, in DES, eight different S-Boxes are used
- Each state byte A_i is *replaced* (substituted), by another byte B_i , such that:

$$S(A_i) = B_i.$$

- The S-Box is the only nonlinear element of AES.
- The S-Box substitution is a bijective mapping, which means that each of the $2^8 = 256$ possible input elements is one-to-one mapped to one output element.
- To perform decryption process, we need to reverse the S-Box uniquely. This is allowed by the bijective mapping property.



- The Byte Substitution layer is first layer in each round of AES encryption process.
- The Byte Substitution layer can be viewed as a row of 16 parallel S-Boxes, each with 8 input and output bits.
- All 16 S-Boxes are identical in AES, where, in DES, eight different S-Boxes are used
- Each state byte A_i is *replaced* (substituted), by another byte B_i , such that:

$$S(A_i) = B_i.$$

- The S-Box is the only nonlinear element of AES.
- The S-Box substitution is a bijective mapping, which means that each of the $2^8 = 256$ possible input elements is one-to-one mapped to one output element.
- To perform decryption process, we need to reverse the S-Box uniquely. This is allowed by the bijective mapping property.



- The Substitution values, in hexadecimal notation, for input byte (xy) in the AES S-Box are shown in the following figure.

		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

Example: Assume that the input byte to the S-Box is $A_i = (C2)_{\text{hex}}$, then the substituted value is:

$$S((C2)_{\text{hex}}) = (25)_{\text{hex}}.$$

On a bit, this substitution can be described as:

$$S(11000010) = (00100101).$$

Even though the S-Box is bijective, it does not have any fixed points. Therefore, there aren't any input values A_i such that $S(A_i) = A_i$. Even the zero-input is not a fixed point →

$$S(00000000) = (01100011).$$



Example: Assume the input to the Byte Substitution layer is

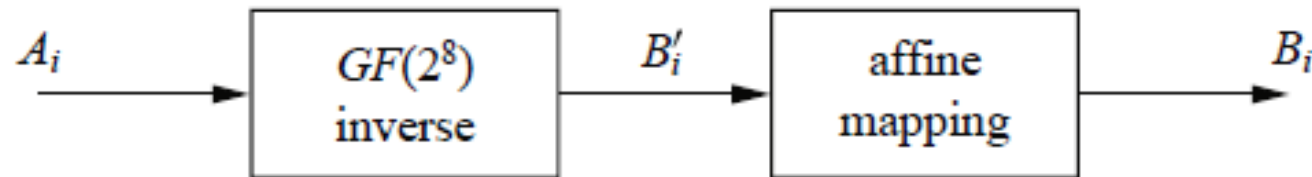
$(C2, C2, \dots, C2)$ in hexadecimal notation.

The output state is then $(25, 25, \dots, 25)$

Note: The DES S-boxes are random tables that fulfill certain properties.

The AES S-Boxes have a strong algebraic structure.

An AES S-Box can be represented by two steps of mathematical transformation, such that:



1. Galois field inversion:

- For each input element A_i , the inverse is computed: $B_i = A_i^{-1}$, where both A_i and B_i are considered elements in the field $GF(2^8)$ with the fixed irreducible polynomial

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

- A lookup table with all inverses is shown in the following table. Note that the inverse of the zero element does not exist. The AES assumes that the zero element $A_i = 0$ is mapped to itself.

		Y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
X	0	00	01	8D	F6	CB	52	7B	D1	E8	4F	29	C0	B0	E1	E5	C7
	1	74	B4	AA	4B	99	2B	60	5F	58	3F	FD	CC	FF	40	EE	B2
	2	3A	6E	5A	F1	55	4D	A8	C9	C1	0A	98	15	30	44	A2	C2
	3	2C	45	92	6C	F3	39	66	42	F2	35	20	6F	77	BB	59	19
	4	1D	FE	37	67	2D	31	F5	69	A7	64	AB	13	54	25	E9	09
	5	ED	5C	05	CA	4C	24	87	BF	18	3E	22	F0	51	EC	61	17
	6	16	5E	AF	D3	49	A6	36	43	F4	47	91	DF	33	93	21	3B
	7	79	B7	97	85	10	B5	BA	3C	B6	70	D0	06	A1	FA	81	82
	8	83	7E	7F	80	96	73	BE	56	9B	9E	95	D9	F7	02	B9	A4
	9	DE	6A	32	6D	D8	8A	84	72	2A	14	9F	88	F9	DC	89	9A
	A	FB	7C	2E	C3	8F	B8	65	48	26	C8	12	4A	CE	E7	D2	62
	B	0C	E0	1F	EF	11	75	78	71	A5	8E	76	3D	BD	BC	86	57
	C	0B	28	2F	A3	DA	D4	E4	0F	A9	27	53	04	1B	FC	AC	E6
	D	7A	07	AE	63	C5	DB	E2	EA	94	8B	C4	D5	9D	F8	90	6B
	E	B1	0D	D6	EB	C6	0E	CF	AD	08	4E	D7	E3	5D	50	1E	B3
	F	5B	23	38	34	68	46	03	8C	DD	9C	7D	A0	CD	1A	41	1C

1. Affine Mapping:

□ Each byte B_i is multiplied by a constant bitmatrix followed by the addition of a constant 8-bit vector. The operation can be described as follows:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} \equiv \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \pmod{2}.$$

□ Note that $B = (b_7, \dots, b_0)$ is the bitwise vector representation of $B_i(x) = A^{-1}i(x)$.

□ This step is referred to as affine mapping.

Example: Assume that the S-Box input $A_i = (11000010)_2 = (C2)_{\text{hex}}$.

❑ From S-Box inputs table we can see that the inverse is:

$$A^{-1}_i = B_i = (2F)_{\text{hex}} = (00101111)_2.$$

❑ We now apply the B_i bit vector as input to the affine transformation.

❑ Note that the least significant bit (lsb) b_0 of B_i is at the rightmost position.

$$B_i = (00100101) = (25)_{\text{hex}}$$

❑ Thus, $S((C2)_{\text{hex}}) = (25)_{\text{hex}}$, which is exactly the result that is also given in the S-Box table



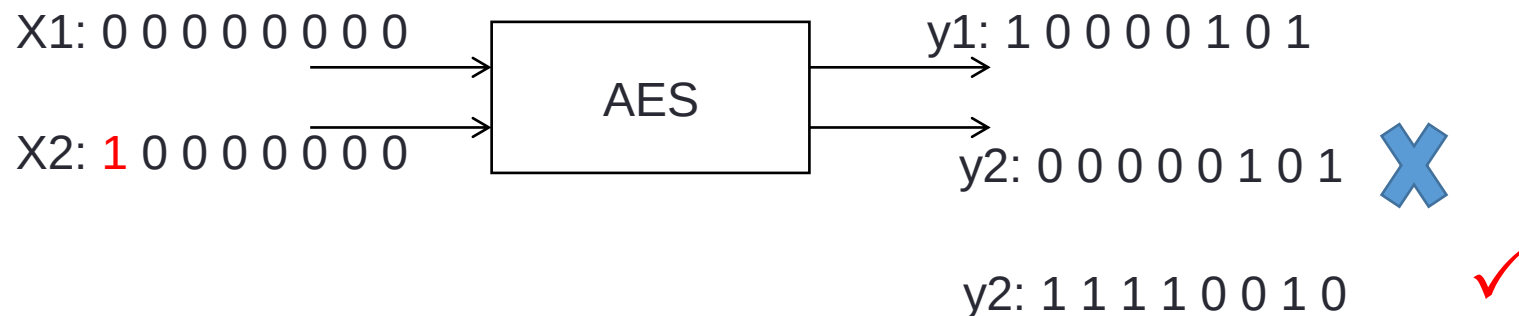
- The advantage of using **inversion** in $GF(2^8)$ as the core function of the Byte Substitution layer is that it provides a high degree of nonlinearity, which in turn provides optimum protection against some of the strongest known analytical attacks.
- Furthermore, the **affine step** “destroys” the algebraic structure of the Galois field, which in turn is needed to prevent attacks that would exploit the finite field inversion



❑ In AES, the Diffusion layer consists of two sublayers, as follows:

1. The *ShiftRows* transformation
2. The *MixColumn* transformation.

❑ The diffusion is the spreading of the influence of individual bits over the entire state. Unlike the nonlinear S-Box, the diffusion layer performs a linear operation on state matrices A,B.



Diffusion Layer (ShiftRows Sublayer)

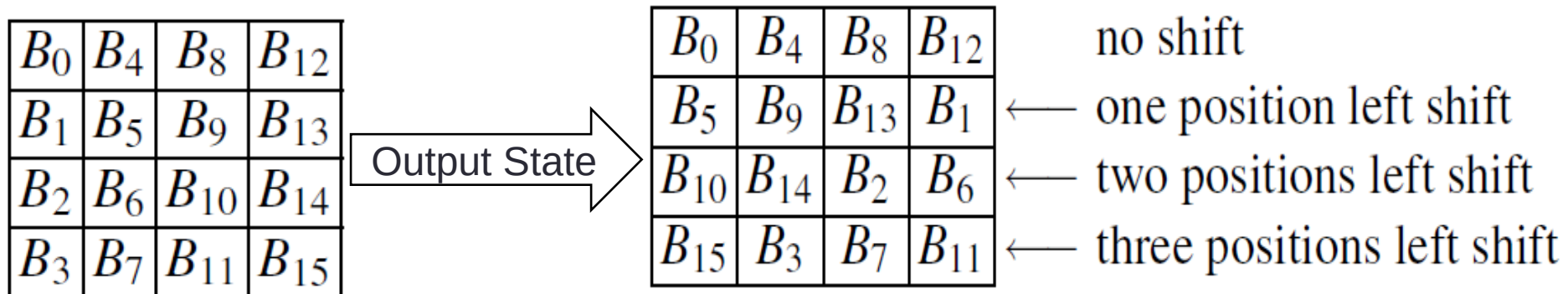
The ShiftRows transformation cyclically shifts the second row of the state matrix by three bytes to the right, the third row by two bytes to the right and the fourth row by one byte to the right.

The first row is not changed by the ShiftRows transformation.

The purpose of the ShiftRows transformation is to increase the diffusion properties of AES.

Example: Assume that the input of the ShiftRows sublayer is given as a state matrix

$B = (B_0, B_1, \dots, B_{15})$:



Diffusion Layer (MixColumn Sublayer)

- ❑ The MixColumn step is a linear transformation which mixes each column of the state matrix.
- ❑ The MixColumn operation is the major diffusion element in AES because every input byte influences four output bytes.
- ❑ Let us denote the 16-byte input state by B and the 16-byte output state by C , such that: $MixColumn(B) = C$, where B is the state after the *ShiftRows* operation.
- ❑ Each 4-byte column is considered as a vector and multiplied by a fixed 4×4 matrix.
- ❑ The matrix contains constant entries. Multiplication and addition of the coefficients is done in $GF(2^8)$. The following example shows how the first four output bytes are computed:

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix}$$

❑ The second column of output bytes (C_4, C_5, C_6, C_7) is computed by multiplying the four input bytes (B_4, B_9, B_{14}, B_3) by the same constant matrix, and so....

- ❑ Each state byte C_i and B_i is an 8-bit value representing an element from $GF(2^8)$. All arithmetic involving the coefficients is done in this Galois field.
- ❑ For the constants in the matrix a hexadecimal notation is used. For example: “01” refers to the $GF(2^8)$ polynomial with the coefficients (00000001), which means it is the element 1 of the Galois field, where “02” refers to the polynomial with the bit vector (00000010), which means the polynomial x , and “03” refers to the polynomial with the bit vector (00000011), which means the Galois field element $x+1$.
- ❑ The additions in the vector–matrix multiplication are $GF(2^8)$ additions, that is simple bitwise XORs of the respective bytes



- ❑ For the multiplication of the constants, we have to realize multiplications with the constants 01, 02 and 03.
- ❑ Multiplication by 01 is multiplication by the identity and does not involve any explicit operation. Multiplication by 02 can be implemented as a multiplication by x , and a modular reduction with $P(x) = x^8 + x^4 + x^3 + x + 1$. Similarly, multiplication by 03, which represents the polynomial $(x+1)$, and should be followed by a modular reduction with $P(x)$.

Example: Assume that the input state to the MixColumn layer is $B = (25, 25, \dots, 25)$.

In this special case, only two multiplications in $GF(2^8)$ have to be done. These are $02 \cdot 25$ and $03 \cdot 25$, which can be computed in polynomial notation:

$$02 \cdot 25 = x \cdot (x^5 + x^2 + 1)$$

$$= x^6 + x^3 + x,$$

$$03 \cdot 25 = (x+1) \cdot (x^5 + x^2 + 1)$$

$$= (x^6 + x^3 + x) + (x^5 + x^2 + 1)$$

$$= x^6 + x^5 + x^3 + x^2 + x + 1$$

Since both intermediate values have a degree smaller than 8, no modular reduction with $P(x)$ is necessary.



The output bytes of C result from the following addition in $GF(2^8)$:

$$01 \cdot 25 = x^5 + x^2 + 1$$

$$01 \cdot 25 = x^5 + x^2 + 1$$

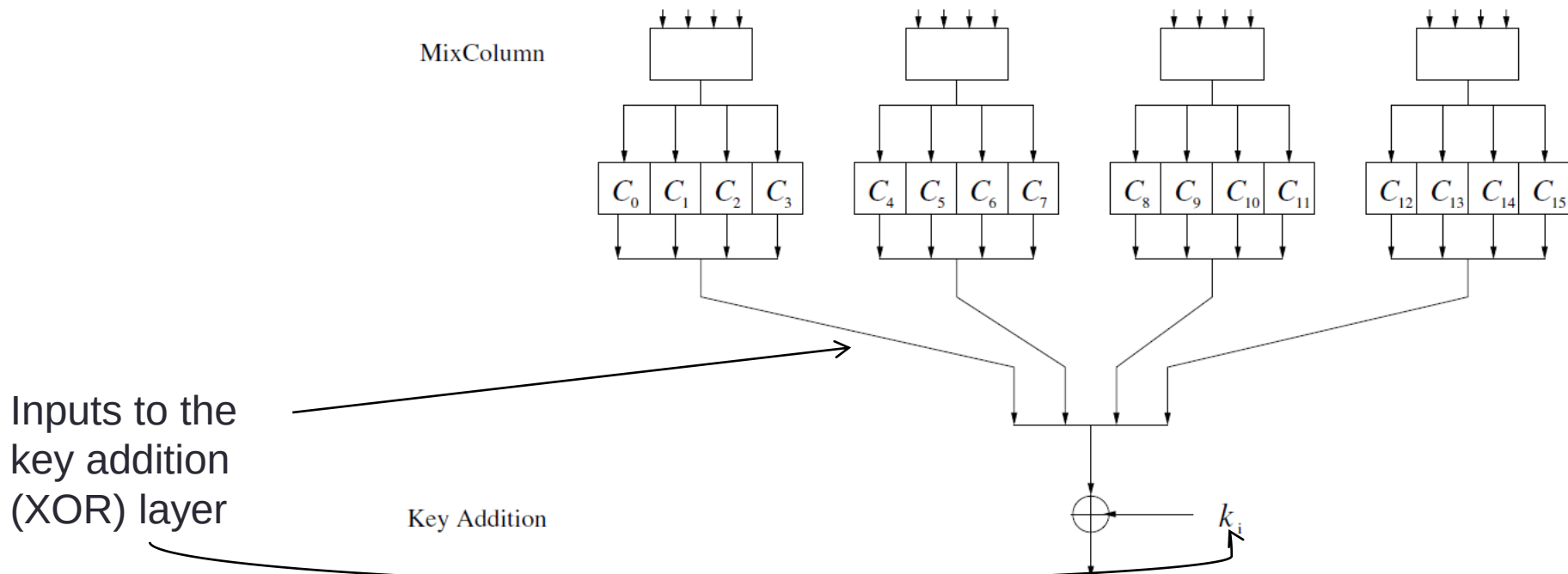
$$02 \cdot 25 = x^6 + x^3 + x$$

$$03 \cdot 25 = x^6 + x^5 + x^3 + x^2 + x + 1$$

$$C_i = x^5 + x^2 + 1,$$

where $i = 0, \dots, 15$. This leads to the output state $C = (25, 25, \dots, 25)$

- ❑ The two inputs to the Key Addition layer are the current 16-byte state matrix and a subkey which also consists of 16 bytes (128 bits).
- ❑ The two inputs are combined through a bitwise XOR operation, which is equal to addition in the Galois field $GF(2)$



- ❑ The key schedule takes the original input key (of length 128, 192 or 256 bit) and derives the subkeys used in AES.
- ❑ Note that an XOR addition of a subkey is used both at the input and output of AES. This process is sometimes referred to as key whitening.
- ❑ Therefore, the number of subkeys is equal to the number of rounds plus one.
- ❑ Thus, for the key length of 128 bits, the number of rounds is $n_r = 10$, and there are 11 subkeys, each of 128 bits. The AES with a 192-bit key requires 13 subkeys of length 128 bits, and AES with a 256-bit key has 15 subkeys.
- ❑ The AES subkeys are computed recursively, i.e., in order to derive subkey k_i , subkey k_{i-1} must be known
- ❑ The AES key schedule is word-oriented, where 1 word = 32 bits. Subkeys are stored in a key expansion array W that consists of words.

Key Addition Layer (Key Schedule)

- ❑ The 11 subkeys are stored in a key expansion array with the elements $W[0], \dots, W[43]$.
- ❑ The elements $K_0, \dots, K_{15}$ denote the bytes of the original AES key.
- ❑ Note that the first subkey k_0 is the original AES key, i.e., the key is copied into the first four elements of the key array W . The other array elements are computed as follows.

❑ The leftmost word of a subkey $W[4i]$, where $i = 1, \dots, 10$, is computed as:

$$W[4i] = W[4(i-1)] + g(W[4i-1]).$$

Here $g()$ is a nonlinear function with a four-byte input and output. The remaining three words of a subkey are computed recursively as:

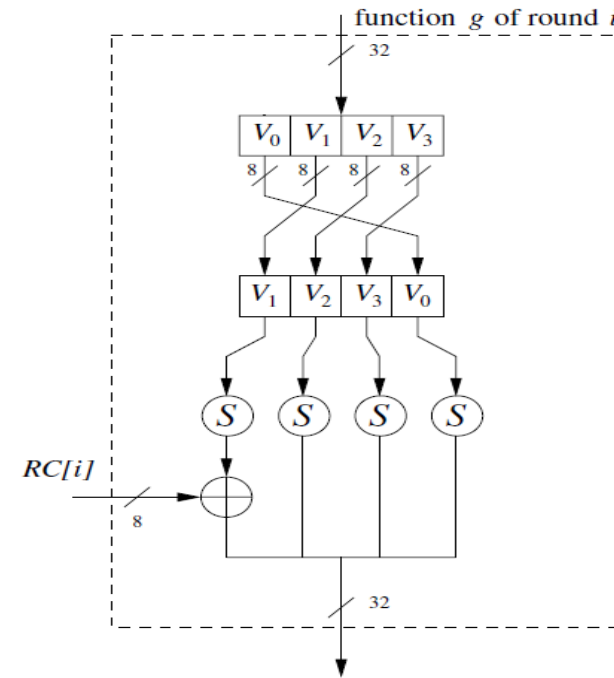
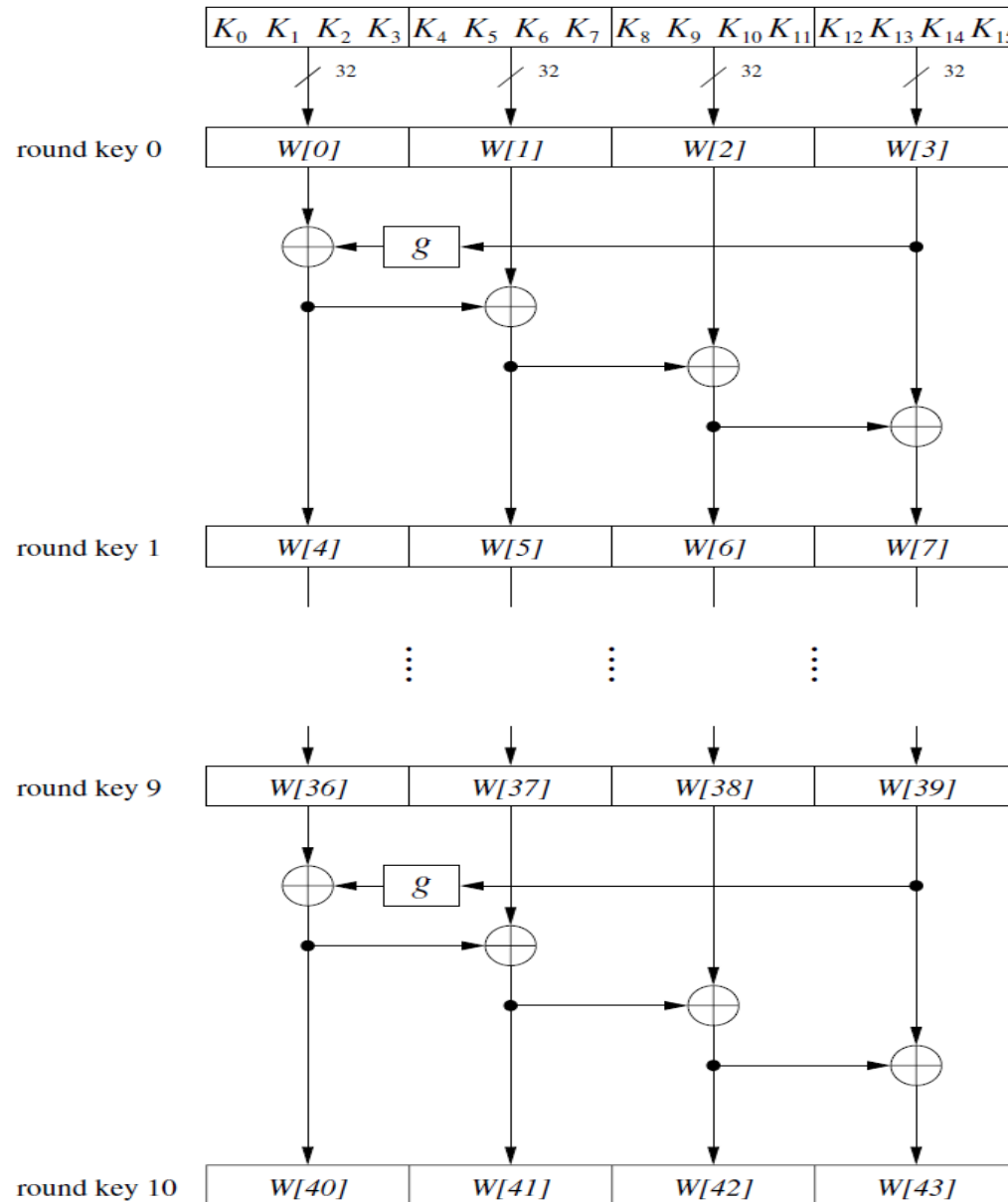
$$W[4i + j] = W[4i + j - 1] + W[4(i-1) + j], \text{ where } i = 1, \dots, 10 \text{ and } j = 1, 2, 3.$$

The computation of the subkeys are depicted in the following figure (next slide)



Module 2

Key Addition Layer (Key Schedule)



- ❑ The function $g()$ rotates its four input bytes, performs a byte-wise S-Box substitution, and adds a round coefficient RC to it.
- ❑ The round coefficient is an element of the Galois field $GF(2^8)$, i.e, an 8-bit value. It is only added to the leftmost byte in the function $g()$.
- ❑ The round coefficients vary from round to round according to the following rule:

$$RC[1] = x^0 = (00000001)_2,$$

$$RC[2] = x^1 = (00000010)_2,$$

$$RC[3] = x^2 = (00000100)_2,$$

...

$$RC[10] = x^9 = (00110110)_2.$$

- ❑ The function $g()$ has two purposes. First, it adds nonlinearity to the key schedule. Second, it removes symmetry in AES. Both properties are necessary to thwart certain block cipher attacks.

Each round coefficient (RC) is multiplied by 2. Both of them are elements in the field 2^8

Example: how can we calculate RC # 9

$$RC\#9 = 2 * RC\#8 \rightarrow$$

$$(0000\ 0010) * (10000000) \rightarrow$$

$$X * X^7 = X^8$$

Now, use the irreducible polynomial: $P(x) = x^8 + x^4 + x^3 + x + 1$

The result: $x^4 + x^3 + x + 1 \rightarrow 0001\ 1011 \rightarrow 1B$

Round j	Round coefficient j
1	01
2	02
3	04
4	08
5	10
6	20
7	40
8	80
9	1B
10	36



Each round coefficient (RC) is multiplied by 2. Both of them are elements in the field 2^8

Example: how can we calculate RC # 10

$$\text{RC\#10} = 2 * \text{RC\#9 (1B)} \rightarrow$$

$$(0000\ 0010) * (00011011) \rightarrow$$

$$X * X^8 = X^9$$

Note that X^8 was computed as: $x^4 + x^3 + x + 1$

The result: $x^5 + x^4 + x^2 + x \rightarrow 0011\ 0110 \rightarrow 36$

Round j	Round coefficient j
1	01
2	02
3	04
4	08
5	10
6	20
7	40
8	80
9	1B
10	36



Key Addition Layer (Key Schedule)

Example, suppose that the round key for round 8 is
EA D2 73 21 B5 8D BA D2 31 2B F5 60 7F 8D 29 2F

Then, the first 4 bytes (first column) of the round key for round 9 are calculated as follows:

Round i	i = 36
Word [35]	7F8D292F
After RotWord	8D292F7F
After SubWord	5DA515D2
Rcon (9)	1B000000
After XOR with Rcon	46A515D2
$w[i-4]$	EAD27321
$w[i] = g$ XOR $w[i-4]$	AC7766F3

➤ There are two different approaches when implementing any of the previous key schedules:

Precomputation: All subkeys are expanded first into the array W . The encryption (decryption) of a plaintext (ciphertext) is executed afterwards. This approach is often taken in PC and server implementations of AES, where large pieces of data are encrypted under one key. Please note that this approach requires $(nr + 1) \cdot 16$ bytes of memory, e.g., $11 \cdot 16 = 176$ bytes if the key size is 128 bits. This is the reason why such an implementation on a device with limited memory resources, such as a smart card, is sometimes not desirable.

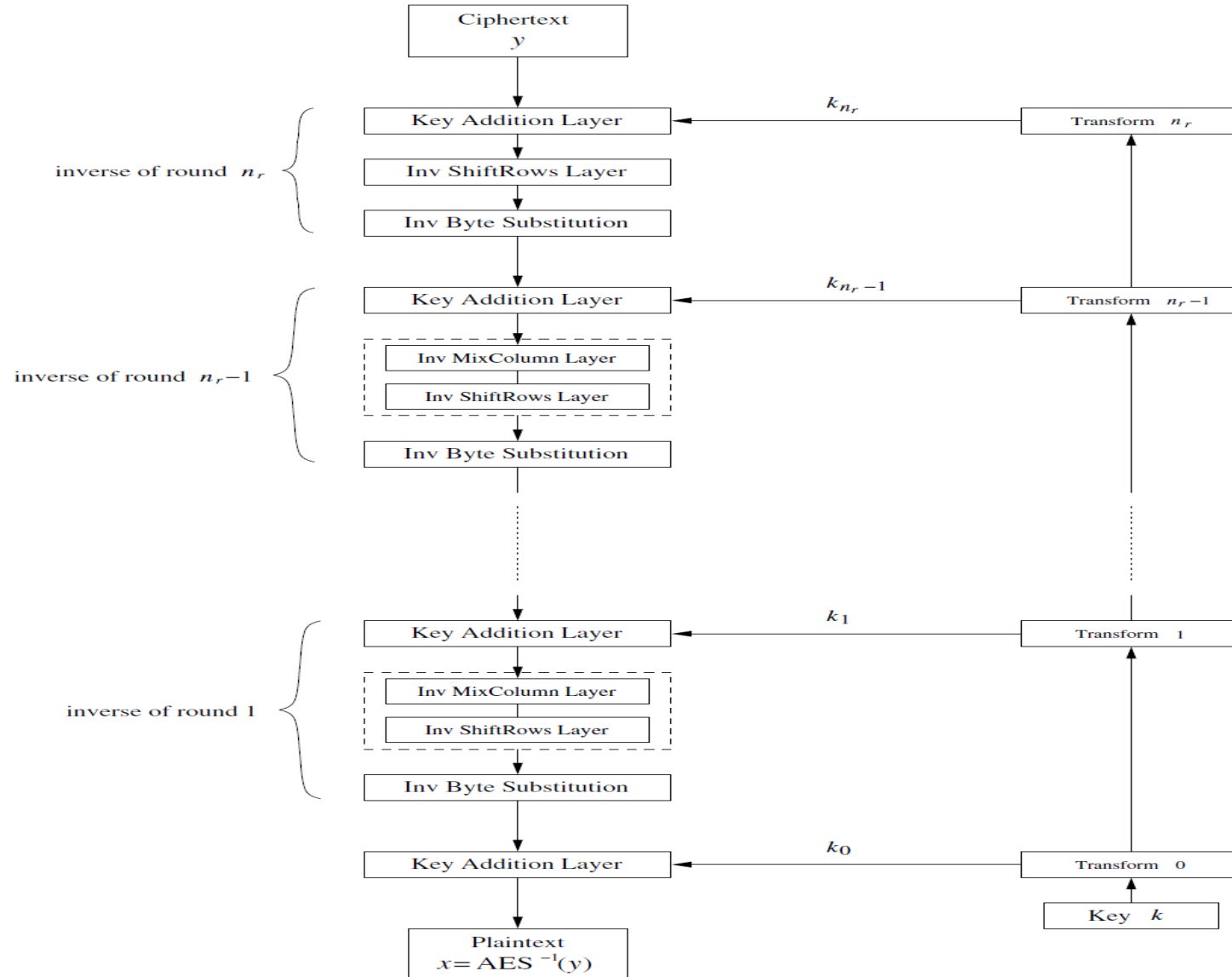
On-the-fly A new subkey is derived for every new round during the encryption (decryption) of a plaintext (ciphertext). Please note that when decrypting ciphertexts, the last subkey is XORed first with the ciphertext. Therefore, it is required to recursively derive all subkeys first and then start with the decryption of a ciphertext and the on-the-fly generation of subkeys. As a result of this overhead, the decryption of a ciphertext is always slightly slower than the encryption of a plaintext when the on-the-fly generation of subkeys is used.



Because AES is not based on a Feistel network, all layers must actually be inverted, i.e., the Byte Substitution layer becomes the Inv Byte Substitution layer, the ShiftRows layer becomes the Inv ShiftRows layer, and the MixColumn layer becomes Inv MixColumn layer. However, as we will see, it turns out that the inverse layer operations are fairly similar to the layer operations used for encryption. In addition, the order of the subkeys is reversed, i.e., we need a reversed key schedule. A block diagram of the decryption function is shown in the figure in next slide.



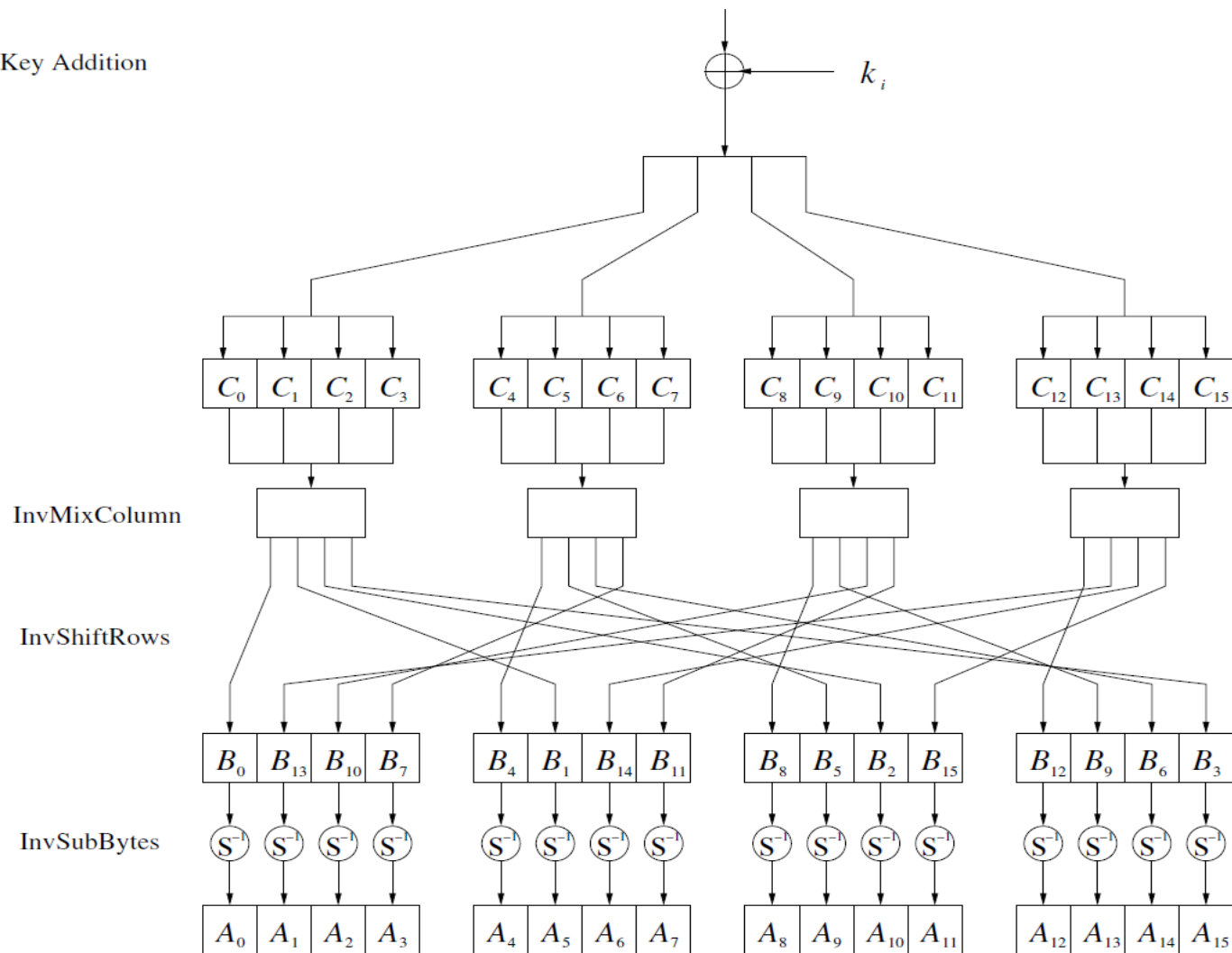
Decryption (Overview)



- ❑ Since the last encryption round does not perform the MixColumn operation, the first decryption round also does not contain the corresponding inverse layer.
- ❑ All other decryption rounds, however, contain all AES layers. Since XOR operation is its own inverse, the key addition layer in the decryption mode is the same as in the encryption mode: it consists of a row of plain XOR gates



Key Addition



- ❑ After the addition of the subkey, the inverse MixColumn step is applied to the state (again, the exception is the first decryption round).
 - ❑ In order to reverse the MixColumn operation, the inverse of its matrix must be used. The input is a 4-byte column of the State C which is multiplied by the inverse 4×4 matrix.
 - ❑ The matrix contains constant entries. Multiplication and addition of the coefficients is done in $GF(2^8)$.
 - ❑ The second column of output bytes (B_4, B_5, B_6, B_7) is computed by multiplying the four input bytes (C_4, C_5, C_6, C_7) by the same constant matrix, and so on.
 - ❑ Each value B_i and C_i is an element from $GF(2^8)$. Also, the constants are elements from $GF(2^8)$.
 - ❑ The notation for the constants is hexadecimal and is the same as was used for the MixColumn layer, for example: $0B = (0B)_{\text{hex}} = (00001011)_2 = x^3 + x + 1$.
- Additions in the vector–matrix multiplication are bitwise XORs



Example for MixColumn Multiplication

$$\begin{pmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix}$$



Decryption (Inverse ShiftRows)

In order to reverse the ShiftRows operation of the encryption algorithm, we must shift the rows of the state matrix in the opposite direction. The first row is not changed by the inverse ShiftRows transformation. If the input of the ShiftRows sublayer is given as a state matrix $B = (B_0, B_1, \dots, B_{15})$:

$$\begin{pmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{pmatrix} = \begin{pmatrix} B_0 & B_4 & B_8 & B_{12} \\ B_1 & B_5 & B_9 & B_{13} \\ B_2 & B_6 & B_{10} & B_{14} \\ B_3 & B_7 & B_{11} & B_{15} \end{pmatrix} \begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix}$$

The inverse ShiftRows sublayer yields the output:

B_0	B_4	B_8	B_{12}	no shift
B_{13}	B_1	B_5	B_9	→ one position right shift
B_{10}	B_{14}	B_2	B_6	→ two positions right shift
B_7	B_{11}	B_{15}	B_3	→ three positions right shift

Decryption (Inverse ShiftRows)

K0	K4	K8	K12	————→ No shift
K13	K1	K5	K9	————→ One position right shift
K10	K14	K2	K6	————→ Two positions right shift
K7	K11	K15	K3	————→ Three positions right shift

The inverse ShiftRows sublayer yields the output:

B_0	B_4	B_8	B_{12}	no shift
B_{13}	B_1	B_5	B_9	————→ one position right shift
B_{10}	B_{14}	B_2	B_6	————→ two positions right shift
B_7	B_{11}	B_{15}	B_3	————→ three positions right shift

The inverse S-Box is used when decrypting a ciphertext. Since the AES S-Box is a bijective, i.e., a one-to-one mapping, it is possible to construct an inverse S-Box such that:

$$A_i = S^{-1}(B_i) = S^{-1}(S(A_i)),$$

where A_i and B_i are elements of the state matrix. The entries of the inverse S-Box are given in the table in next slide.



Module 2

Decryption (Inverse Byte Substitution)

	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
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
x 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



Decryption (Inverse Byte Substitution)

In order to reverse the Sbox substitution, we first have to compute the inverse of the affine transformation. For this, each input byte B_i is considered an element of $GF(2^8)$.

The inverse affine transformation on each byte B_i is defined by:

$$\begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} \equiv \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \pmod{2},$$

where $(b_7, \dots, b_0)$ is the bitwise vector representation of $B_i(x)$, and $(b'_7, \dots, b'_0)$ the result after the inverse affine transformation.



In the second step of the inverse S-Box operation, the Galois field inverse has to be reversed. For this, note that $A_i = (A^{-1}_i)^{-1}$. This means that the inverse operation is reversed by computing the inverse again.

In our notation we thus have to compute $A_i = (B'_i)^{-1} \in GF(2^8)$ with the fixed reduction polynomial $P(x) = x^8 + x^4 + x^3 + x + 1$.

Again, the zero element

is mapped to itself. The vector $A_i = (a_7, \dots, a_0)$ (representing the field element $a_7x^7 + \dots + a_1x + a_0$)

is the result of the substitution: $A_i = S^{-1}(B_i)$.



Since decryption round one needs the last subkey, the second decryption round needs the second-to-last subkey and so on, we need the subkey in reversed order.

In practice this is mainly achieved by computing the entire key schedule first and storing all 11, 13 or 15 subkeys, depending on the number of rounds AES is using (which in turn depends on the three key lengths supported by AES).

This precomputation adds usually a small latency to the decryption operation relative to encryption.



Note: The main source for the majority of contents in these slides is:

C. Paar and J. Pelzl. Understanding Cryptography. A textbook for Student and Practitioners. Springer 2010