AES(Advanced Encryption Standard)

- Symmatric cipher, block cipher, 128 bits in with a 128/192/256 bits key, 128 bits out.
- And the **number of rounds** depend on the **key length**

key length	rounds
128	10
192	12
256	14

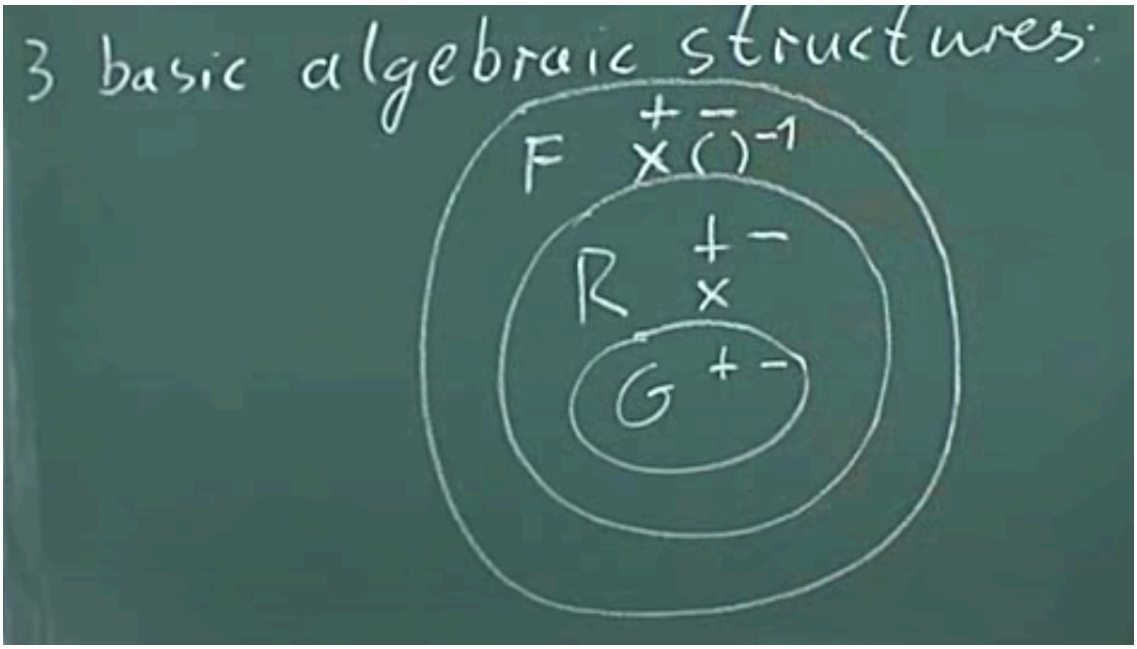
- the most used encryption method nowadays
  - AES is embeded in your web browser😆
- all internal operation of AES are based on **Finite Fields**

> In crypto we almost always want **finite sets**

Introduction to Finite Fields(aka Galois Fields)

Three basic algebraic structures

Group, Ring and Field



- a very informal definition "a field is a set of numbers in which we can +,-,*,/"

Theorem : A Finite Field only exist when there are p^m elements in the set, when p is a prime number and m is a positive integer. We denote this by $GF(p^m)$

- There are two types of FF(stand for Finit Field)
  - m is equal to 1 ,$GF(p)$, we call it Prime Field
  - m is larger than 1, $GF(p^m)$, we call it extension field
- and $GF(2^m)$ is quite important in cryptography

So for AES, $2^8 = 256$ elements are chosen, that is to say, we have $GF(2^8)$

- and the elements in the field are polynomials represented by 8 bits

$$10010100 \rightarrow 1 \times x^7 + 0 \times x^6 + \ldots + 0 \times x^0$$

- here we comes to polynomial arithmetic
  - addition and subtraction

    just do normal polynomial arithmetic and then reduce the coefficient to $GF(2)$

$$\begin{aligned} 1+1 &= 0 \\ 1+0 &= 1 \\ 0+1 &= 1 \\ 0+0 &= 0 \end{aligned}$$

$$\begin{aligned} 1-1 &= 0 \\ 1-0 &= 1 \\ 0-1 &= 1 \\ 0-0 &= 0 \end{aligned}$$

$$\text{note that} : a - b \equiv a + b$$

  - Multiplication

just do normal polynomial multiplication, but we should ensure that the result is in the field, that is to say, our output is the result module an **irreducible polynomial(不可约多项式)**

we must reduce our result to $GF(2^8)$

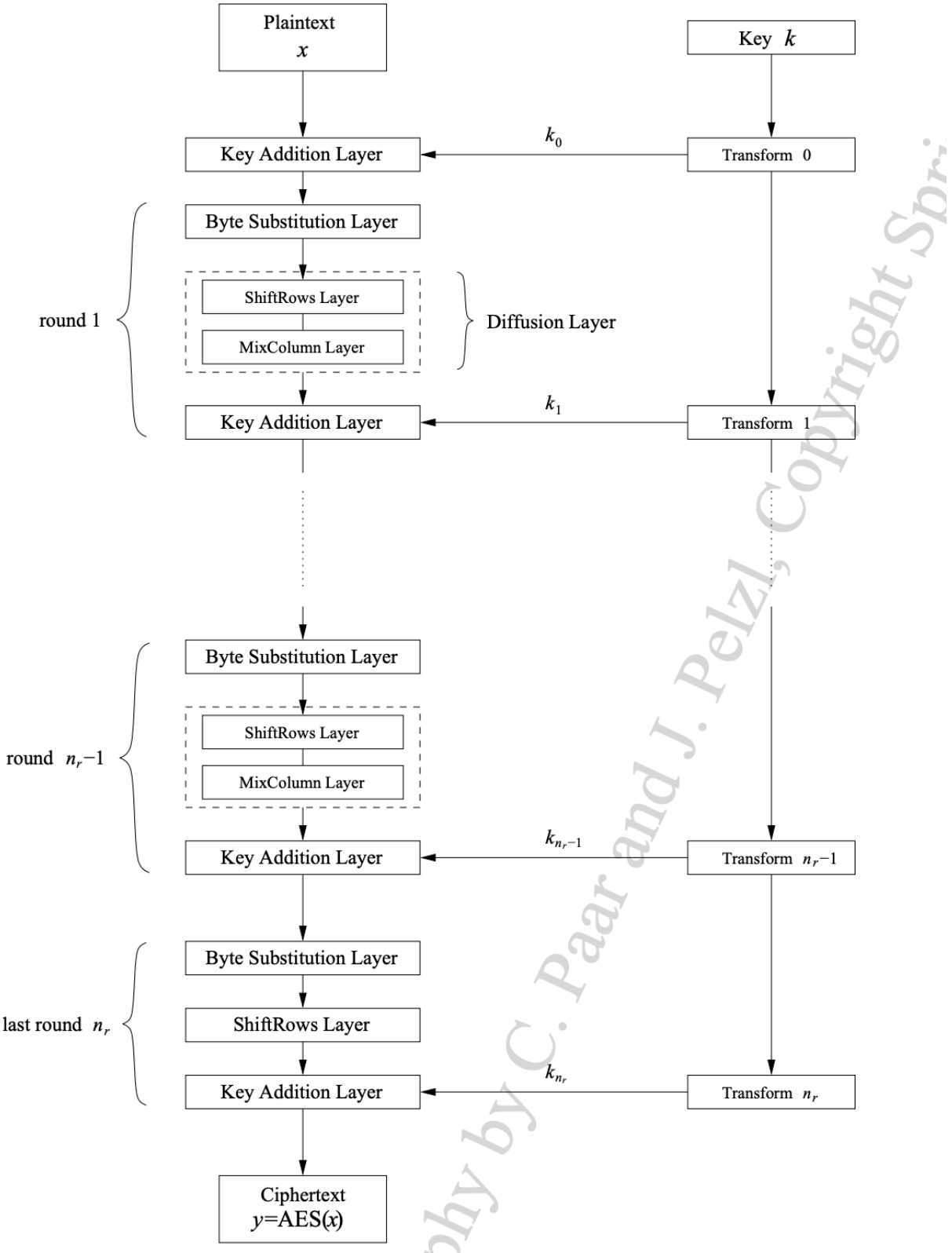
Here the irreducible polynomial we use is
$P(x) = x^8 + x^4 + x^3 + x + 1$

- Inversion

  just like normal integer inversion, we use extended Eucilean algorithm.

Note that there may be several different irreducible polynomials in a given GF, and the using of different irreducible polynomial will lead to different result. The polynomial given above is part of the AES standard

Structure of AES



- AES does not have a feistal scheme
- The last round does not have a misColumn layer
- ByteSub provide **Confusion**, just like S-box in DES
- ShiftRows and Mixcolumn together provide **Diffusion**
- at the most beginning and the most ending, subkey is added.

- Not like DES, which is bit-oriented, AES is byte-oriented

Each round consists of four layers

Step 1 : Byte Subtitution

- the input 16 bytes go into 16 **identical!!!** S-box respectively
- the S-boxes have 8 bits in, 8 bits out

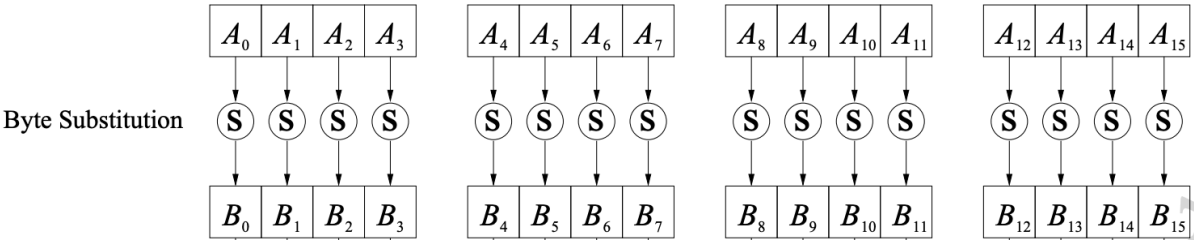


Table 4.3 AES S-Box: Substitution values in hexadecimal notation for input byte (xy)

	y															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
x 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

- how to substitute ?
  - 8 bits in : $(xxxxxxxx)_2$
  - convert to hexadecimal $(ab)_{16}$
  - read the value with (x,y) = (a,b)
  - 11000010 -> 25, for instance
- How was the table constructed ?

- 8 bits in, convert it into a polynomial in $GF(2^8)$
- calculate its multiply inverse
- Then apply an affine mapping to the inverse, we get the result

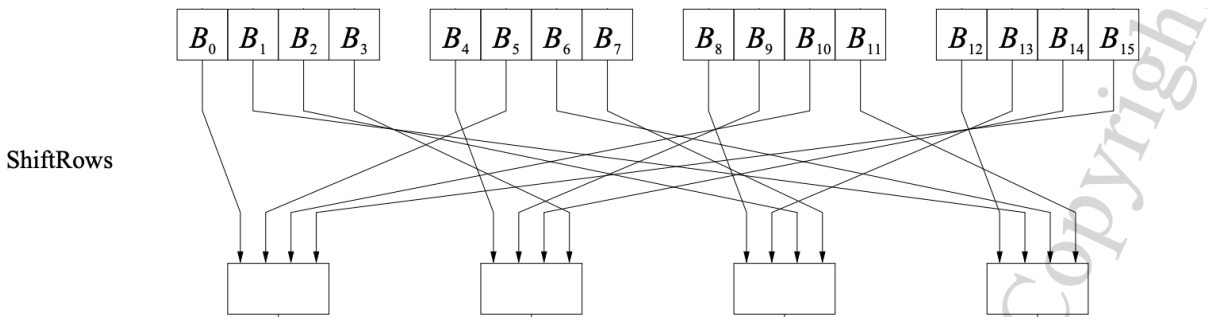
  Note that the affine table and the vector to be add is part of the AES standard.

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

- That is to say we can obtain the substitute result by calculation instead of substitution. However, we make the substitution table in order to be efficient.

Step 2 : ShiftRows

- this step is just permutation at byte level



- In fact, the permutation is not done by a permutation table, it is down by shifting rows(in order to increase Diffusion, note that each

adjacent bytes are spreaded)

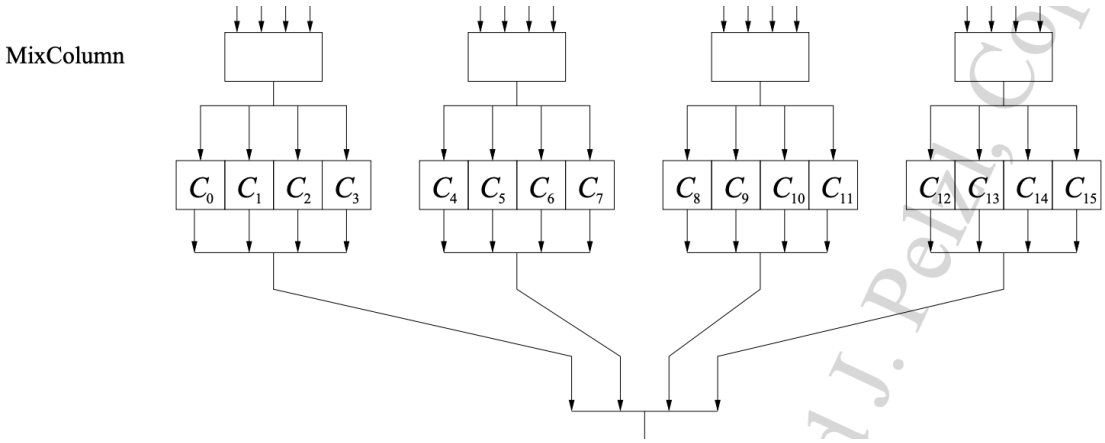
B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}
B_2	B_6	B_{10}	B_{14}
B_3	B_7	B_{11}	B_{15}

the output is the new state:

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

Step 3 : MixColumn

- this step provide strong Diffusion, all 32 bits are affected.



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

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 on. Figure 4.3 shows which input bytes are used in each of the four MixColumn operations.

- Note that the matrix is fixed, and there is a shift relationship between rows

  Moreover, all multiplication and addition is **polynomial arithmetic !!!**

Step 4 : Key Addition

- Just a simple XOR, but note that XOR is equal to addition in $GF(2)$

Key Schedule

- The number of subkeys is equal to **the number of rounds plus one**, due to the key needed for key whitening in the first key addition layer
- 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
- There are **different key schedules for the three different AES key sizes** of 128, 192 and 256 bit, which are all fairly similar

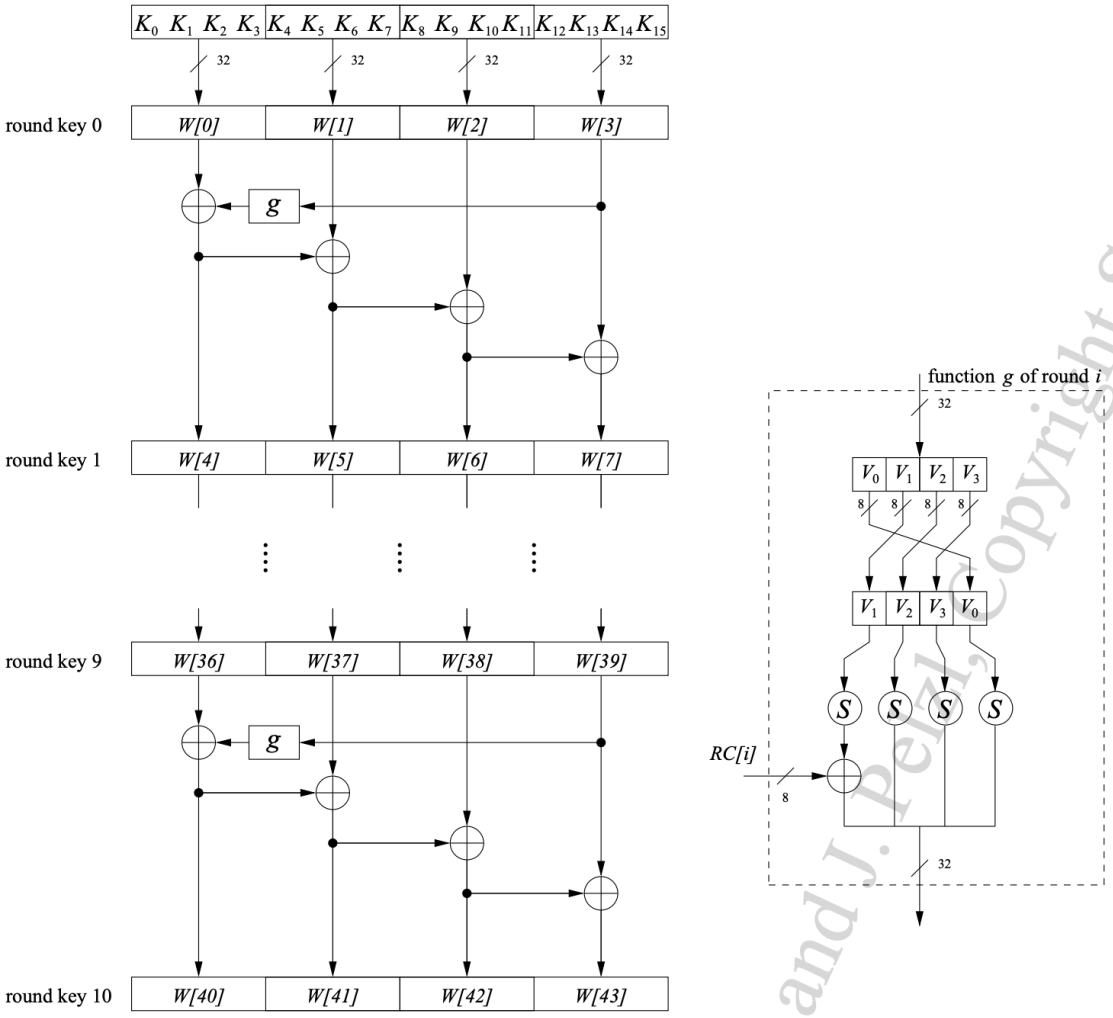


Fig. 4.5 AES key schedule for 128-bit key size

- The round key 0 is just the original input key
- As can be seen in the figure, the leftmost word of a subkey W[4i], where$i = 1, \ldots, 10$, 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 function g() **rotates** its four input bytes, performs a byte-wise S-Box substitution, and adds a round coefficient RC_i 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().
- RC_i vary according to the following rule:

$$\begin{aligned} RC[1] &= x^0 = (00000001)_2, \\ RC[2] &= x^1 = (00000010)_2, \\ RC[3] &= x^2 = (00000100)_2, \\ &\vdots \\ RC[10] &= x^9 = (00110110)_2. \end{aligned}$$

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.

---

for Decryption see at **crypto-textbook**

the **video**

Python code from **Github**