AES

- operates on bytes: $\mathbb{F}_{256} \equiv \mathbb{Z}_2[X]/(X^8 + X^4 + X^3 + X + 1)$.
- matrix representation: $32 \times k$ bits = $4 \times k$ bytes.

$$k = 4 : [a_0, a_1, a_2, \ldots, a_{15}] \rightarrow \begin{bmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{bmatrix} \ldots$$

where $a_0, a_1, \ldots$ are bytes

Example:

- 128 bits key $\Rightarrow$matrix 4×4,
- 192 bits key $\Rightarrow$matrix 4×6,
- 256 bits key $\Rightarrow$matrix 4×8,

AES

	AES-128	AES-192	AES-256
N_b: columns in a block matrix	4	4	4
N_k: columns in the key matrix	4	6	8
N_r: # of rounds	10	12	14

AES

	AES-128	AES-192	AES-256
N_b: columns in a block matrix	4	4	4
N_k: columns in the key matrix	4	6	8
N_r: # of rounds	10	12	14

Operations in a round:

`SubBytes`: non linear substitution (SBox)

`ShiftRows`: Rotation of each L_i

`MixColumns`: left Multiplication by a matrix

`AddRoundKey`: bitwise Addition with a subkey K_i
(via diversification of the key)

- Initialization: `AddRoundKey`
- Final round: similar except for `Mixcolumns`

AES

Function: `SubBytes`

- For each element of the matrix: $a_{i,j} \leftarrow S(a_{i,j})$
- S non linear: $S(0) = B$ and $S(x) = Ax^{-1} + B \mod X^8 + 1$, with $A = X^4 + X^3 + X^2 + X^1 + 1, B = X^6 + X^5 + X + 1$:

$$S(x) = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} x^{-1} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \mod 2$$

Function: `ShiftRows`

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \longrightarrow \begin{bmatrix} a & b & c & d \\ f & g & h & e \\ k & l & i & j \\ p & m & n & o \end{bmatrix}$$

AES

Function: `MixColumns`

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \leftarrow \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Algebraically:

- Each column is a degree-4 polynomial in $\mathbb{F}_{256}[Y]$
- Multiplication by [03]$Y^3 + Y^2 + Y +$[02] $\mod Y^4 + 1$

AES

`AddRoundKey`

- bitwise Addition with the subkey K_i
- Obtained via diversification:

Key schedule

K: $4N_k$ bytes $\rightarrow W$:$4N_b(N_r + 1)$ bytes
$\Rightarrow N_r + 1$ round keys.

- $W_{1\ldots N_k} = K$ (N_k columns of K are duplicated in W)
- W_i computed from W_{i-1} via shifts, SBox, and $\oplus$ with a constant

Key schedule (end)

Each W_i on 32 bits:

begin
- $K_0 = [W_0|W_1|W_2|W_3] = K$;
- **for** *i=1...10* **do**
  - $T = W_{4i-1} <<< 8$;
  - `/* 8 bits left cyclic shift */`
  - $T = \texttt{SubBytes}(T)$;
  - `/* 4 applications of the SBox */`
  - $T = T \oplus (X^i \mod P_8)$;
  - $W_{4i} = W_{4i-4} \oplus T$;
  - $W_{4i+1} = W_{4i-3} \oplus W_{4i}$;
  - $W_{4i+2} = W_{4i-2} \oplus W_{4i+1}$;
  - $W_{4i+3} = W_{4i-1} \oplus W_{4i+2}$;
  - $K_i = [W_{4i}|W_{4i+1}|W_{4i+2}|W_{4i+3}]$;

Implementation: `SubBytes`

$S(x) \leftarrow Ax^{-1} + B \mod X^8 + 1$, with
$A = X^4 + X^3 + X^2 + X^1 + 1, B = X^6 + X^5 + X + 1$.

63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

- Example: $63 = [01100011]_2 = A \cdot 0 + B = B = X^6 + X^5 + X + 1$

AES software implementation example: tables

```
 UINT32    m_uRounds;
 UINT8     m_expandedKey[_MAX_ROUNDS+1][4][4];
static UINT8 T1[256][4]={ 0xc6,0x63,0x63,0xa5, 0xf8,0x7c,0x7c,0x84, ...}
static UINT8 T2[256][4]={ 0xa5,0xc6,0x63,0x63, 0x84,0xf8,0x7c,0x7c, ...}
static UINT8 T3[256][4]={ 0x63,0xa5,0xc6,0x63, 0x7c,0x84,0xf8,0x7c, ...}
static UINT8 T4[256][4]={ 0x63,0x63,0xa5,0xc6, 0x7c,0x7c,0x84,0xf8, ...}


void Rijndael::encrypt(const UINT8 a[16], UINT8 b[16]) {
       int r;
       UINT8 temp[4][4];

    // XOR with key
    *((UINT32*)temp[0]) = *((UINT32*)(a   )) ^ *((UINT32*)m_expandedKey[0][0]);
    *((UINT32*)temp[1]) = *((UINT32*)(a+ 4)) ^ *((UINT32*)m_expandedKey[0][1]);
    *((UINT32*)temp[2]) = *((UINT32*)(a+ 8)) ^ *((UINT32*)m_expandedKey[0][2]);
    *((UINT32*)temp[3]) = *((UINT32*)(a+12)) ^ *((UINT32*)m_expandedKey[0][3]);

       // First round
    *((UINT32*)(b    )) = *((UINT32*)T1[temp[0][0]])
                                              ^ *((UINT32*)T2[temp[1][1]])
                                              ^ *((UINT32*)T3[temp[2][2]])
                                              ^ *((UINT32*)T4[temp[3][3]]);
    *((UINT32*)(b + 4)) = *((UINT32*)T1[temp[1][0]])
                                              ^ *((UINT32*)T2[temp[2][1]])
                                              ^ *((UINT32*)T3[temp[3][2]])
                                              ^ *((UINT32*)T4[temp[0][3]]);
    *((UINT32*)(b + 8)) = *((UINT32*)T1[temp[2][0]])
                                              ^ *((UINT32*)T2[temp[3][1]])
                                              ^ *((UINT32*)T3[temp[0][2]])
                                              ^ *((UINT32*)T4[temp[1][3]]);
    *((UINT32*)(b +12)) = *((UINT32*)T1[temp[3][0]])
                                              ^ *((UINT32*)T2[temp[0][1]])
                                              ^ *((UINT32*)T3[temp[1][2]])
                                              ^ *((UINT32*)T4[temp[2][3]]);
```

AES: summary

Encryption:

begin
- A=M;
- A=`AddRoundKey`(A,K_0);
- **for** *i=1...9* **do**
  - A=`SubBytes`(A);
  - A=`ShiftRows`(A);
  - A=`MixColumns`(A);
  - A=`AddRoundKey`(A,K_i);
- A=`SubBytes`(A);
- A=`ShiftRows`(A);
- A=`AddRoundKey`(A,K_{10});
- **return** $C = A$

Decryption:

begin
- A=C;
- A=`AddRoundKey`(A,K_{10});
- A=`InverseShiftRows`(A);
- A=`InverseSubBytes`(A);
- **for** *i=1...9* **do**
  - A=`AddRoundKey`(A,K_i);
  - A=`InverseMixColumns`(A);
  - A=`InverseShiftRows`(A);
  - A=`InverseSubBytes`(A);
- A=`AddRoundKey`(A,K_0);
- **return** $M = A$

AES security

- Cryptanalysis:
  - SBox: no fix point, no opposite nor reverse fix point
  - `ShiftRows` diffusion of consecutive inputs
  - `MixColumns` potential dependency of each output bit to each input bit
- Fast: FPGA implementation ⇒throughput 21.54 Go/s [Hodjat-Verbauwhede 2004]
- Security:
  - No significant attack found
  - Theoretically: O (2^{100}) on AES-128 [Courtois,Pieprzyk, 2002, 05]
  - 2^{45} for AES-256, on 10 rounds (simplified AES-256) [Schneier 09]
  - ...

Outline