Rijndael

Приет за стандарт от NIST: 2.10.2000г.
Joan Daemen & Vincent Rijmen

Дължина на блока и ключа: 256, 192, 128 бита
Принцип на изграждане: SP-мрежа
(Substitution-permutation network)

Слоеве:

1) Нелинеен субституционен слой: ByteSub
2) Линеен разбъркващ слой: ShiftRow, MixColumn
3) Смесващ слой: AddRoundKey

В последния рунд – линейният разбъркващ слой е различен

Всеки байт се разглежда като полином

$$\underline{b} = b_7b_6\dots b_1b_0 \rightarrow b_7x^7+b_6x^6+\dots+b_1x+b_0$$

Пример: $01111001 \rightarrow '79' \rightarrow x^6+x^5+x^4+x^3+1$

Дефинира се събиране и умножение на байтове:
събиране и умножение на елементи в $\mathbb{F}_{2^8}$

$$\mathbb{F}_{2^8} = \mathbb{F}_2[x]/(m(x))$$

$$m(x) = x^8+x^4+x^3+x+1 \rightarrow '11B'$$

Полиномът $m(x)$ не е примитивен :

$$\operatorname{ord} m(x) = 51.$$

Това е първият в списъка на неразложимите полиноми от степен 8 в Lidl & Niederreiter.

Пример.

- събиране

'79' + '35' = '4C'

01111001 + 00110101 = 01001100

$$(x^6+x^5+x^4+x^3+1)+(x^5+x^4+x^2+1) = x^6+x^3+x^2$$

- умножение

'57' • '83' = 'C1'

01010111 • 10000011 = 11000001

$$(x^6+x^4+x^2+x+1)(x^7+x+1) = x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1 \pmod{m(x)}$$

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

Някои операции са дефинират на ниво дума.

Дума (4 байта) → полиноми от степен $\leqq 3$ над $\mathbb{F}_{2^8}$.

$a(x) = a_3x^3+a_2x^2+a_1x+a_0$

a_3	a_2	a_1	a_0

$b(x) = b_3x^3+a_2x^2+b_1x+b_0$

b_3	b_2	b_1	b_0

$a(x)\, b(x)$ произведение във $\mathbb{F}_{2^8}[x]/(x^4+1)$.

Ако

$$d(x) = a(x) . b(x) = d_3 x^3 + d_2 x^2 + d_1 x + d_0$$

$$d_0 = a_0 \cdot b_0 \oplus a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3$$
$$d_1 = a_1 \cdot b_0 \oplus a_0 \cdot b_1 \oplus a_3 \cdot b_2 \oplus a_2 \cdot b_3$$
$$d_2 = a_2 \cdot b_0 \oplus a_1 \ b_1 \oplus a_0 \cdot b_2 \oplus a_3 \cdot b_3$$
$$d_3 = a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3$$

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Блок (вектор на състоянието)

State :

a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}

$a_{00} a_{10} a_{20} a_{30} \ldots \quad a_{35}$

(192 бита)

Ключ

Key :

k_{00}	k_{01}	k_{02}	k_{03}
k_{10}	k_{11}	k_{12}	k_{13}
k_{20}	k_{21}	k_{22}	k_{23}
k_{30}	k_{31}	k_{32}	k_{33}

$k_{00} k_{10} k_{20} k_{30} \ldots \quad k_{33}$

(128 бита)

$$Nb = \frac{\text{дължина на блока (в битове)}}{32} = \text{брой думи в блока}$$

$$Nk = \frac{\text{дължина на ключа (в битове)}}{32} = \text{брой думи в ключа}$$

Nr = брой рундове

Nr	Nb=4	Nb=6	Nb=8
Nk=4	10	12	14
Nk=6	12	12	14
Nk=8	14	14	14

Рунд различен от последния:

```
Round (State, Round Key)
{
    ByteSub (State)
    ShiftRow (State)
    MixColumn (State)
    AddRoundKey (State, RoundKey)
}
```

Последен рунд:

```
FinalRound (State, RoundKey)
{
    ByteSub (State)
    Shift Row (State)
    Add Round Key (State, RoundKey)
}
```

1. ByteSub

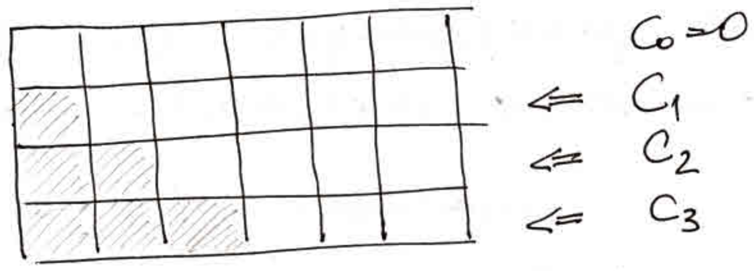
(а) всеки байт b се замества с байта b^{-1} като b и b^{-1} се разглеждат като елементи на $\mathbb{F}_{2^8}$.

(б) към всеки байт се прилага следната афинна трансформация

$$\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Прилагане ByteSub върху всички байтове.

2. ShiftRow



i-тия ред се измества C_i байта вляво
Обратна трансформация : шифт на $Nb - C_i$ байта вляво

Nb	C_1	C_2	C_3
4	1	2	3
6	1	2	3
8	1	3	4

3. Mix Column

Стълбовете на State се разглеждат като полиноми над $\mathbb{F}_{2^8}$.

В MixColumn всеки стълб на стейт се умножава по фиксиран полином

$$c(x) = '03'x^3 + '01'x^2 + '01'x + '02'$$

$$\left(c(x)\ ,\ x^4+1\right) = 1.$$

Ако

$$b(x) = c(x) \otimes a(x)$$

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

Обратната трансформация се състои в умножаване с $d(x)$, който се задава чрез

$$c(x)\, d(x) = 1 \quad (\mathrm{mod}\ x^4+1)$$

Оказва се, че

$$d(x) = '0B'x^3 + '0D'x^2 + '09'x + '0E'$$

4. AddRoundKey

- Побитов XOR на State и RoundKey
- AddRoundKey и обратната ѝ съвпадат.

Генериране на подключове

Това се извършва чрез специална трансформация, която се състои от

(а) разширяване на ключа до т.нар.
разширен ключ

(б) избиране на подключове

- Общият брой на битовете (от разширения ключ), които са необходими е равен на (дължината на блока) × (броя на стъпките +1)

  (192 бита) × (12 стъпки +1) = 2496 бита за разширения ключ

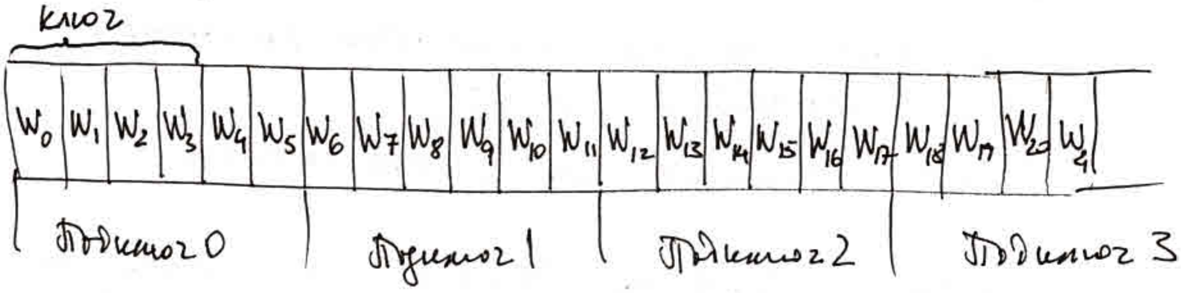
- Първите Nb думи от разширения ключ образуват първия подключ, вторите Nb думи втория подключ и т.н.

- Разширен ключ:
  масив от 4 байтови думи
  $W[Nb * (Nr+1)]$

- Първите Nk думи са оригиналния ключ

- Има различни функции за разширяване в зависимост от това дали
  $Nk \leqq 6$ или $Nk > 6$

Разширяване на ключа и избор на подключовете за отделните рундове за Nb= 6, Nk=4.



Key Expansion for $N_k \leq 6$

```
KeyExpansion (byte Key[4*Nk], word W[Nb*(Nr+1)])
{ for (i=0; i<Nk; i++)
        W[i] = (Key[4*i], Key[4*i+1], Key[4*i+2], Key[4*i+3]);

  for (i=Nk; i<Nb*(Nr+1); i++)
  {
        temp = W[i-1];
        if ((i%Nk)==0)
            temp = SubByte(RotByte(temp)) ⊕ Rcon[i/Nk];
        W[i] = W[i-Nk] ⊕ temp;
  }
}
```

Rcon[i] = (RC[i], '00', '00', '00')

RC[1] = '01'
RC[2] = '02'
RC[i] = '02'•RC[i-1]

RotByte : $(a,b,c,d) \rightarrow (b,c,d,a)$

Шифърът Rijndael

Шифърът Rijndael се състои от

- първоначално добавяне на подключ
- Nr-1 рунда
- последен рунд

```
Rijndael ( State, CipherKey)
{
        KeyExpansion (CipherKey, Expanded key);
        AddRoundKey (State, Expanded key);
        for (i=1; i<Nr; i++)
              Round (State, Expanded Key + Nb*i);
         Final Round (State, ExpandedKey + Nb*Nr);
}
```