AES - Noções Fortes de Segurança - InfoSec

31 de Maio de 2017

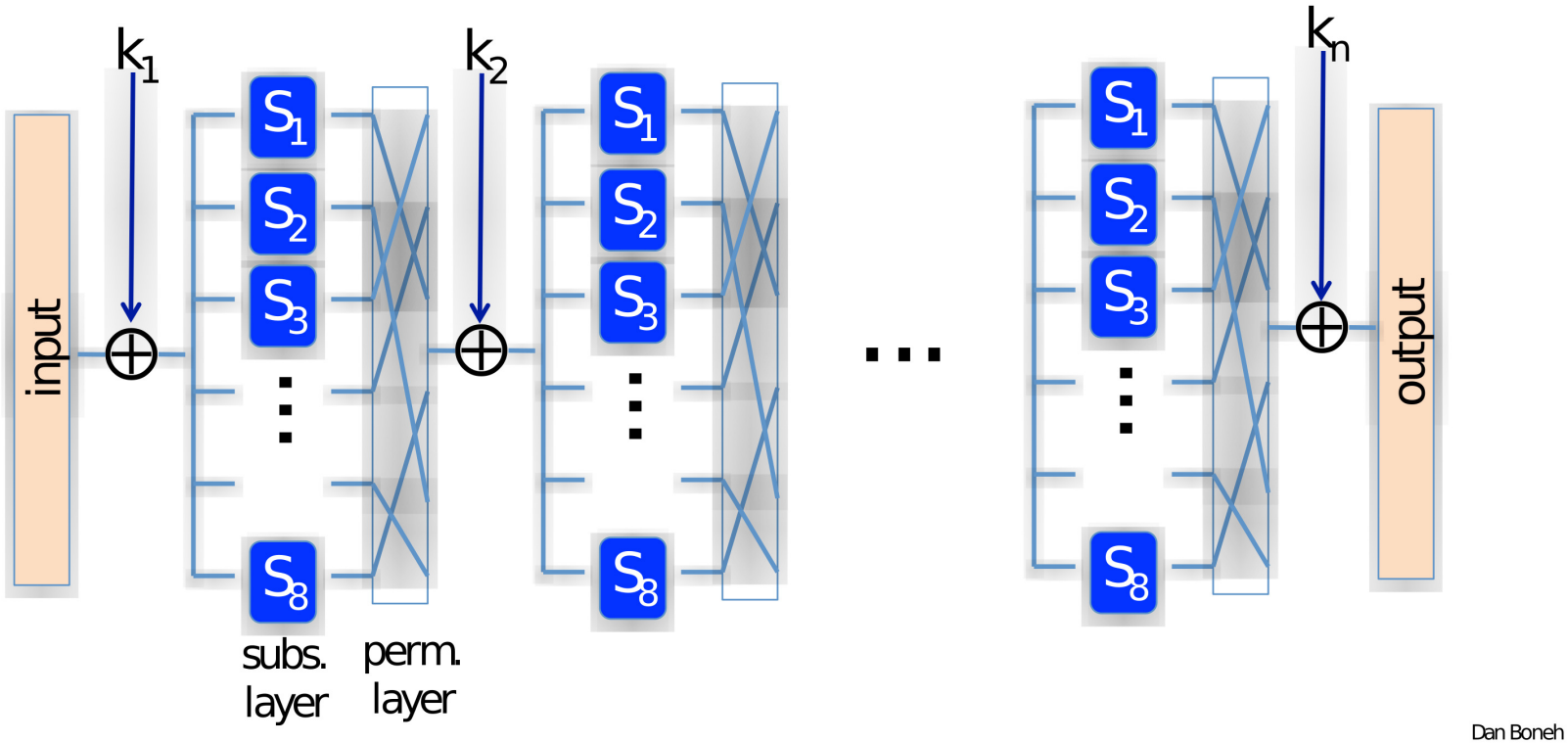
Processo NIST para AES

- 1997: pedido por propostas eficientes e seguras (blocos de 128,192 e 25 bits)
- 1998: 15 propostas
- 1999: finalistas: Rijndael, Serpent, Twofish, FC6, MARS
- 2000: Rijndael adaptado é escolhido como AES - bloco de 128 bits
  - Tem desempenho adaptativo com possibilidade de pré-computação de tabelas de substituição e permutação

Comparação DES com AES

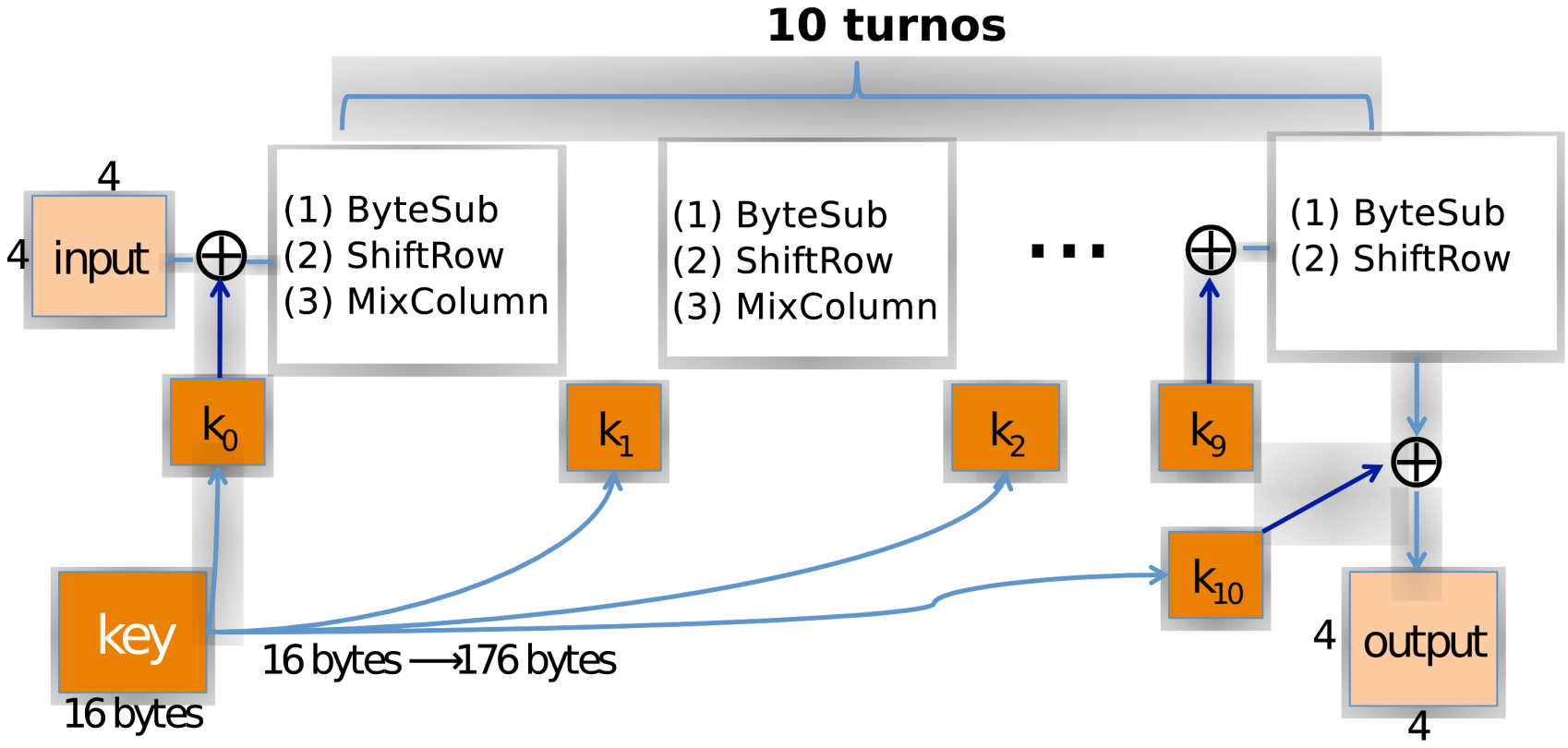
- Bloco: DES 64 bits, AES 128 bits
- Chave: DES 56 bits, AES 128, 192, 256 bits
- Turnos: DES usa 16, AES usa 10, 12 ou 14
- Estratégia geral: confundir e espalhar
- Estratégia específica: DES tipo Feistel, AES tipo Substituição - Permutação
  - Rede de Feistel pode ser considerado um tipo específico de rede de Substituição - Permutação em que as permutações são feitas em apenas metade do bloco

Rede AES: Substituição e Permutação



- Estratégia: confundir e espalhar

AES-128


10 turnos
4
4
input
(1) ByteSub
(2) ShiftRow
(3) MixColumn
(1) ByteSub
(2) ShiftRow
(3) MixColumn
(1) ByteSub
(2) ShiftRow
k0
k1
k2
k9
k10
key
16 bytes
16 bytes ⟶ 176 bytes
4
output
4

AES: estado

- Dados de entrada são organizados em matriz quadrada
- Operação em matrix "estado" com 4×4 bytes

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

- Operação por turnos com subchaves criadas pelo *Rijndael key schedule*
- `for (i in 1:4) for (j in 1:4)`
  `estado[j][i] = bloco[32*i+8*j ... 32*i + 8*(j+1)]`

Expansão de chave

- Temos $|k| = 128$ bits em matriz M_k de 4×4 bytes
- Expande-se M_k em 44 palavras de 4 bytes
- São 1 subchave inicial mais 10 subchaves de turnos
- As 11 subchaves são a key schedule (programação de chaves)

Expansão de chave

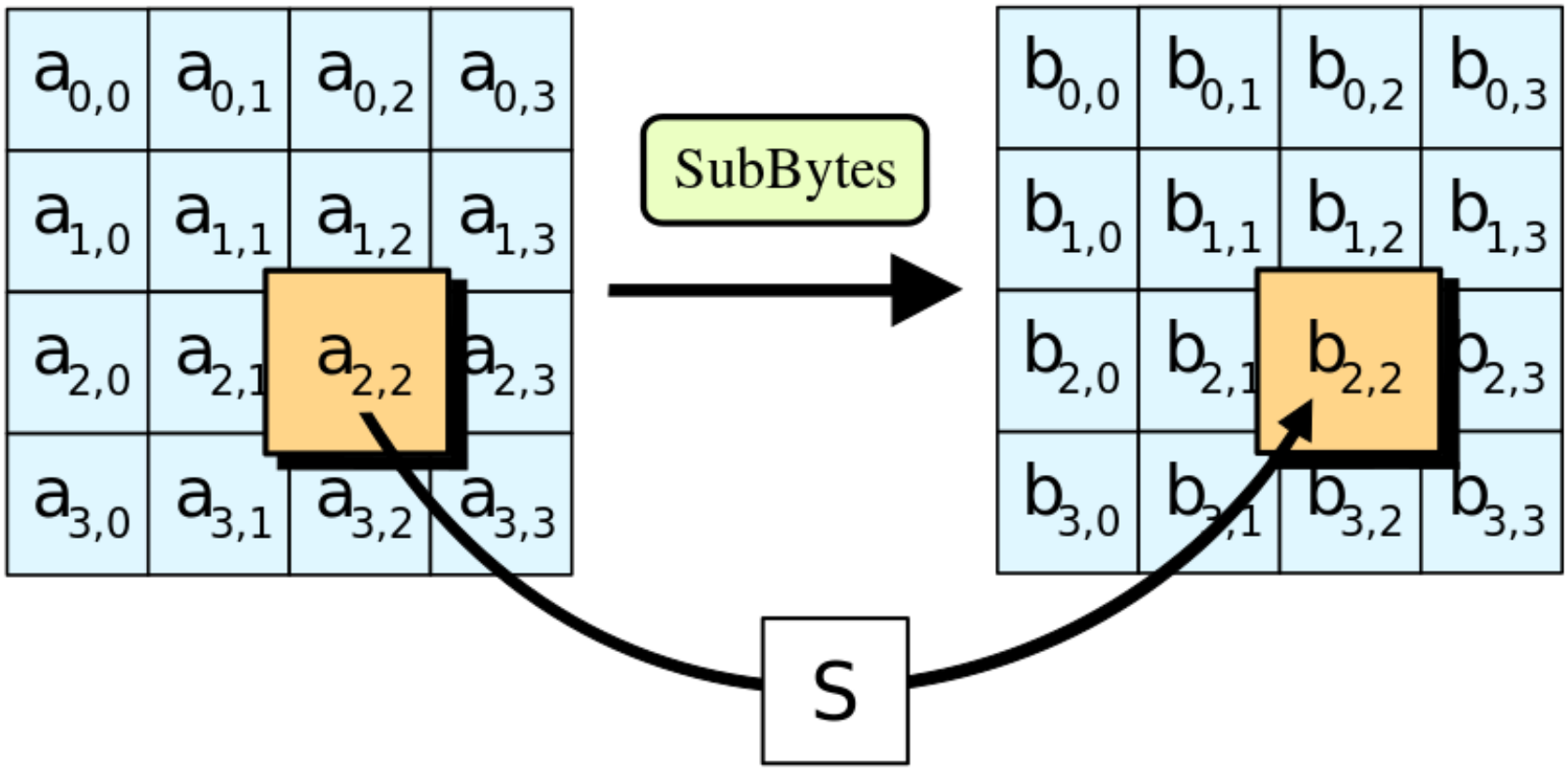
- Temos $|k| = 128$ bits em matriz M_k de 4×4 bytes
- A chave do turno seguinte é produzida da seguinte forma (simplificado):
  - Na coluna mais à direita do turno anterior, byte de cima "entra em baixo"
  - Usar uma caixa de substituição S
  - Aplica XOR com uma constante diferente para cada turno
  - Primeira coluna recebe XOR com a coluna modificada
  - As outras colunas recebem XOR com a primeira já modificada

Cada turno

1. **ByteSub**: função-tabela de substituição de 1 byte por 1 byte: S-box
2. **ShiftRows**: desloca linhas do estado de maneira cíclica
3. **MixColumns**: operação linear para misturar dados nas colunas
   - Esta operação não acontece no último turno
4. Aplicar subchave do turno (com 128 bits) com XOR

Passo 1: ByteSub

- Usa tabela de consulta de 4×4 posições para obter o byte a trocar por outro byte
- Tabela é criada usando inversa da multiplicação em um grupo finito $GF(2^8)$
- Objetivo: confundir por reduzir correlação entre bits de entrada e saída em cada byte

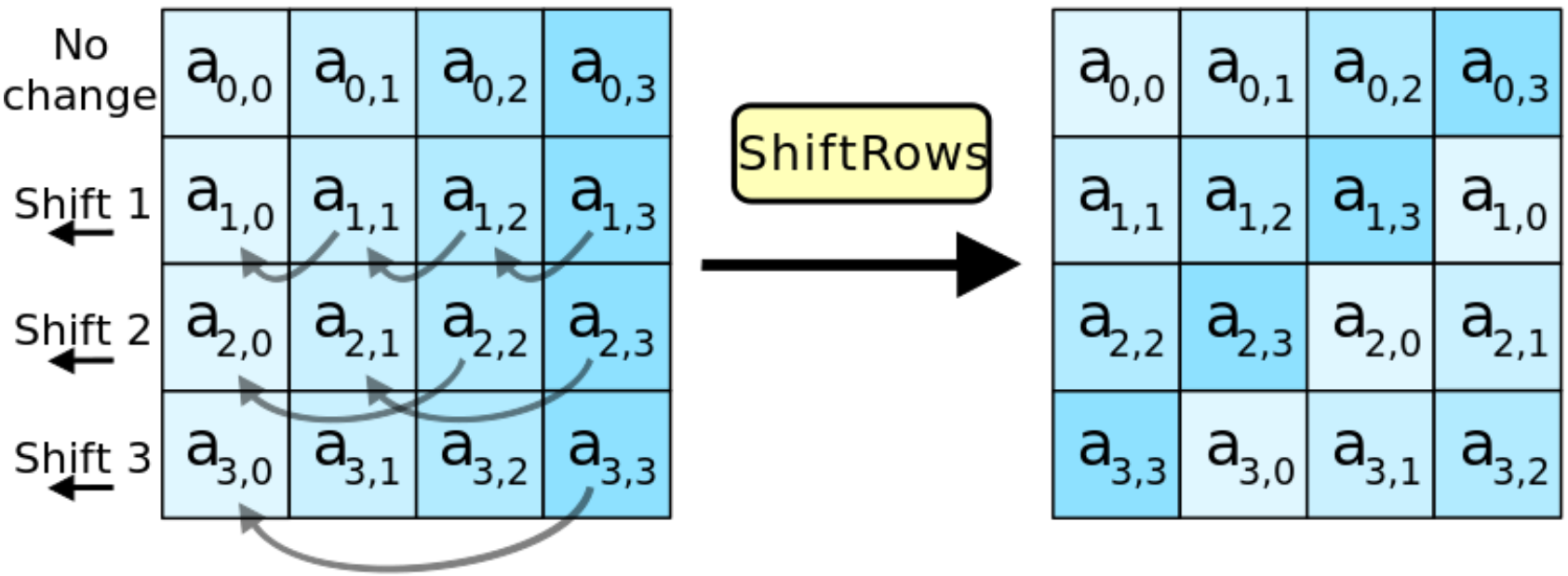


Detalhes: ByteSub

- Seja x_{in} um byte na matrix de estado que queremos trocar por x_{out}
- $x_{out} = f(x_{in})$, tal que, f usa 2 ops. não lineares:
  - Inversa da multiplicação: $x' = x_{in}^{-1}$ em $GF(2^8)$
  - Confundir bits com XOR de x' com 4 rotações cíclicas de seus próprios bit e XOR com $c = 0x63$
    - Rotações de 4,5, 6 e 7 bits à direita: $x_{out} = A \cdot x' + c$

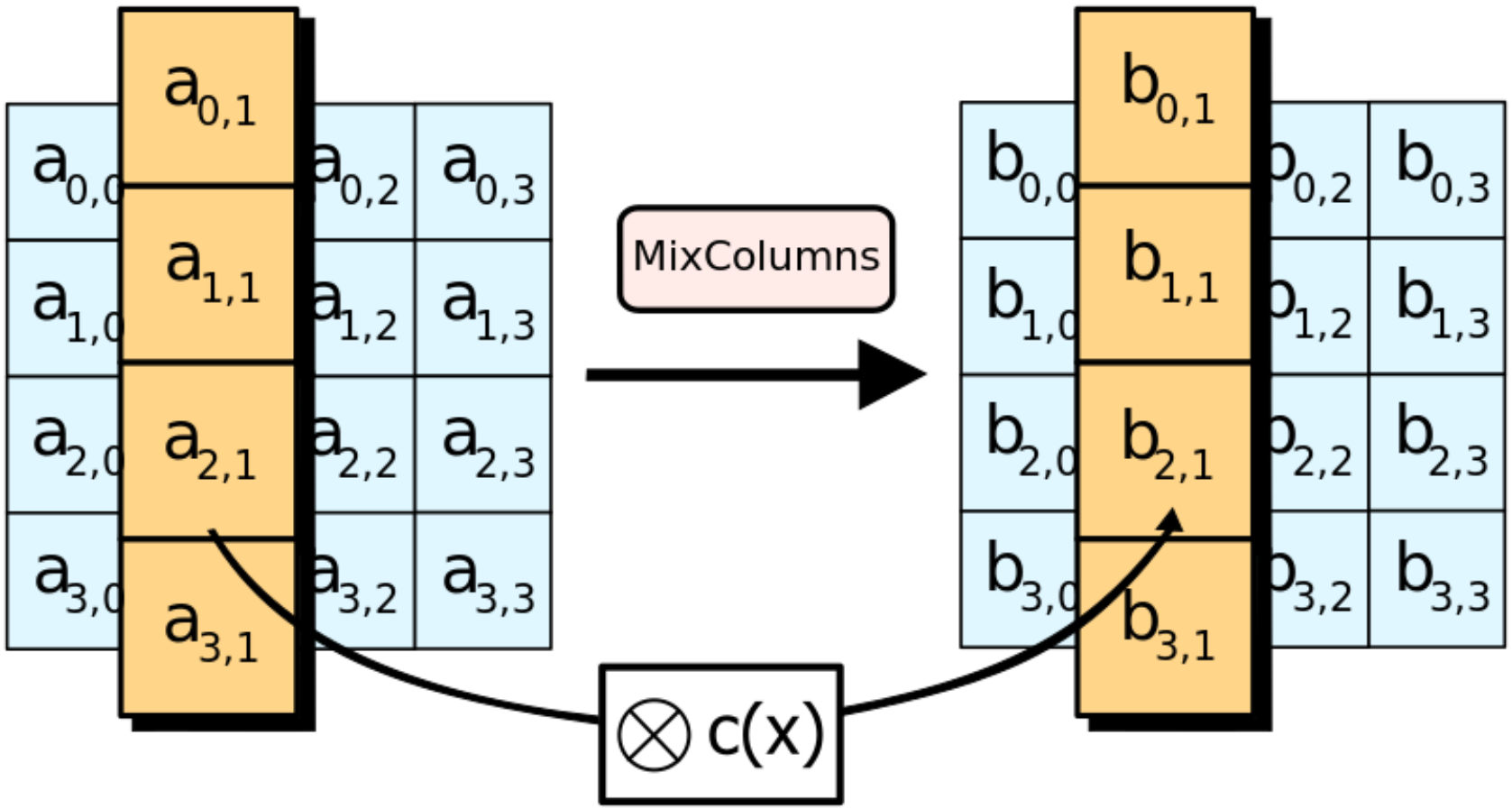
Passo 2: ShiftRows

- Desloca linhas do estados de maneira cíclica
- Objetivo: espalhar e bagunçar ordem de bytes no bloco



Passo 3 e 4: MixColumns e XOR com subchave

- Mistura bytes em cada coluna individualmente
- Objetivo: espalhar tal que cada bit de entrada influencia a posição de cada bit do texto cifrado



Funcionamento ByteSub: Estruturas algébricas

- Grupos
- Anéis
- Campos
- Campos finitos (ou Campos de Galois ou Corpos de Galois)

Grupos

- Um grupo é formado por um conjunto de elementos G e uma operacao $\circ$
- Um grupo tem as propriedades que valem para todo $a, b, c \in G$:
  - Fechamento: $a \circ b = c$
  - Associatividade: $a \circ (b \circ c) = (a \circ b) \circ c$
  - Elemento neutro: $\exists 1 \in G$ tal que $a \circ 1 = 1 \circ a = a$
  - Elemento inverso: $\forall a, \exists a^{-1}$ tal que $a \circ a^{-1} = 1$
  - Comutatividade: $a \circ b = b \circ a$
    - Se grupo aceita comutatividade, ele é abeliano.

Exemplo Grupo abeliano

- Números inteiros sob operação da adição
- Fechamento: $x + y = z$
- Associatividade: $x + (y + z) = (x + y) + z$
- Elemento neutro é o zero: $x + 0 = 0$
- Elemento inverso: x e $-x$, $x + (-x) = 0$
- Comutatividade:$x + y = y + x$

Anéis

- Nem todos elementos em G precisam ter elemento inverso
- Todos os grupos são anéis
- Nem todos os anéis são grupos
- Exemplo: operações $+$ e $\times$
  - Nem sempre é possível dividir (zero)

Campos

- Em um campo F
  - Elementos formam um grupo aditivo (operação $+$ e elemento neutro 0)
  - Elementos, exceto 0, foram um grupo multiplicativo (elemento neutro 1)
  - Distributividade vale entre as duas operações
    - $a \times (b + c) = (a \times b) + (a \times c)$
- "Conjunto de números que podemos somar, subtrair, multiplicar e dividir"
- Exemplo: $\mathbb{R}$ com $+$, $\times$, $()^{-1}$

Em conjuntos finitos

- Em criptografia, campos têm conjuntos finitos de elementos
- Quantos elementos pode ter um campo finito?
  - Todo campo finito necessariamente tem p^n elementos, onde p é um número primo e n um inteiro
  - $GF(2)$, $GF(17)$, $GF(81) = GF(3^3)$, $GF(256) = GF(2^8)$
  - AES usa $GF(2^8)$

Tipos de campos finitos

- Campos primos: campo finito com um número primo de elementos
- Campos extensíveis: campo finito com p^n elementos e $n > 1$

Campos primos $GF(p)$

- Elementos são inteiros $G = \{0, 1 \ldots p-1\}$
- Operações
  - Soma: $a + b = c \mod p$
  - Multiplicação: $a \times b = c \mod p$
  - Inversa da multiplicação: $a \times a^{-1} = 1 \mod p$
    - O zero não precisa ter inversa
    - Possível encontrar a^{-1} com o algoritmo de Euclides extendido
  - Operação mod garante que operações são fechadas em G

Campos extensíveis $GF(2^m)$

- Elementos são polinômios

$$a_{n-1}x^{n-1} + \ldots + a_1 x + a_0 = A(X) \in GF(2^n)$$

- Exemplo em $GF(2^n) = GF(8)$
  - $GF(8) = \{0, 1, x, x+1, x^2, x^2+x, x^2+1, x^2+x+1\}$

Como operar em $GF(2^n)$

- Soma

$$C(x) = A(x) + B(x) = \sum_{i=0}^{m-1} c_i x^i$$

onde $c_t = a_t + b_t \mod 2$

  - Observação: $a_t - b_t \mod 2 = a_t + b_t \mod 2$

Como operar em $GF(8)$

- Soma

$$
\begin{aligned}
& A(x) = x^2 + 1 \\
+\ & B(x) = x^2 + x + 1 \\
=\ & C(x) = 2x^2 + x + 2 \mod 2 \\
=\ & C(x) = x
\end{aligned}
$$

Como operar em $GF(2^n)$

- Multiplicação

$$C(x) = A(x) \times B(x) \mod P(x)$$

  - Sendo $P(x) = \sum_{i=0}^{n} p_i x^i$ e $p_i \in GF(2)$ um polinômio estratégico irredutível para resultado da multiplicação continuar em $GF(2^n)$

Como operar em $GF(8)$

- Multiplicação

  $A(x) = x^2 + 1$

$\times$ $B(x) = x^2 + x + 1$

$=$ $C(x) = x^4 + x^3 + x^2 + x^2 + x + 1$

$=$ $C(x) = x^4 + x^3 + 2x^2 + x + 1$

$=$ $C(x) = x^4 + x^3 + x + 1$

- Reduzir para $GF(8)$ com polinômio irredutível ("primo")

  $P(x) = x^3 + x + 1$

$=$ $C(x) = x^4 + x^3 + x + 1 \mod P(x)$

Polinômios irredutíveis

- Polinômio irredutível não é fatorável (como um número primo)
- Pode existir vários $P(x)$ para operação em um $GF(2^n)$
  - $P(x) = x^3 + x + 1$
  - $P(x) = x^3 + 1$
  - $P(x) = x^3 + x^2 + 1$
- Polinômio irredutível do AES

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

Redução na multiplicação

- Resultado depende do $P(x)$ escolhido
- $C(x) = x^4 + x^3 + x + 1 \mod P(x) = x^3 + x + 1$
- Obter resto da divisão $C(x)$ por $P(x)$
- $C(x) = x^2 + x$

Como operar em $GF(2^n)$

- Inversa de multiplicação

$$A(x) \times A^{-1}(x) = 1 \mod P(x)$$

- Encontrar pelo algoritmo de Euclides extendido

"S-Box" do AES

- A tabela de substituição S do AES troca um byte $a(x)$ por outro $b(x)$
- Troca usa 2 operações:
  - $b(x) = M \cdot a^{-1}(x) + 0x63$
  - Garante permutação tal que $b(x) \neq a(x)$
- Computar $a^{-1}(x)$ no campo finito $GF(2^8)$ com polinômio irredutível

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

Algoritmo de Euclides

- Maior divisor comum: $mdc(x, y) = ?$
- Opção 1: fatorar x e y e encontrar o maior em comum
- Opção 2: usar $mdc(x, y) = mdc(x, y \mod x)$ com $y > x$

Algoritmo de Euclides

Algorithm 1 Algoritmo de Euclides

Require: Inteiros positivos r_0 e r_1, $r_0 > r_1$

Ensure: $mcd(r_0, r_1)$

Inicialização: $i \leftarrow 1$

repeat

$i \leftarrow= i + 1$

$r_i \leftarrow r_{i-2} \mod r_{i-1}$

until $r_i \neq 0$

return $mcd(r_0, r_1) = r_{i-1}$

Exemplo: algoritmo de Euclides

- $mdc(x, y) = mdc(x, y \mod x)$
- $x = 27,\ y = 21$
- $mdc(27, 21) = mdc(6, 21) = mdc(6, 3) = 3$

$$27 = 1 \cdot 21 + 6$$
$$21 = 3 \cdot 6 + 3$$
$$6 = 2 \cdot 3 + 0$$

Exemplo 2

- $mdc(973, 301) = ?$

  $973 = 3 \cdot 301 + 70$

  $301 = 4 \cdot 70 + 21$

  $70 = 3 \cdot 21 + 7$

  $21 = 3 \cdot 7 + 0$
- $mdc(973, 301) = 7$

Algoritmo de Euclides Extendido (AEE)

- AEE acha inteiros s e t tal que $mdc(r_0, r_1) = s \times r_0 + t \times r_1$
- Ideia: computar algoritmo de Euclides comum e incluir contas para obter s e t
  - $mdc(r_0, r_1)$, $r_0 = q_1 r_1 + r_2$, $r_2 = s_2 r_0 + t_2 r_1$
  - $mdc(r_1, r_2)$, $r_1 = q_2 r_2 + r_3$, $r_3 = s_3 r_0 + t_3 r_1$

    $\vdots$
  - $mdc(r_{l-2}, r_{l-1})$, $r_{l-2} = q_{l-1} r_{l-1} + r_l$, $r_l = s_l r_0 + t_l r_1$
  - até $r_{l+1} = 0$ tal que $mdc(r_0, r_1) = r_l$

Exemplo: AEE

- Obter $mdc(r_0 = 973, r_1 = 301) = s \cdot r_0 + t \cdot r_1$.
- Considerando que $r_l = s_l r_{l-1} + t_l r_{l-1}$
- $973 = 3 \cdot 301 + 70 = 3 \cdot r_1 + r_2$

  $r_2 = 70 = (1) \cdot 973 + (-3) \cdot 301$
- $301 = 4 \cdot 70 + 21$,

  $r_3 = 21 = 301 - 4 \cdot 70 = 301 - 4 \cdot (973 - 3 \cdot 301)$

  $r_3 = (-4) \cdot 973 + (13) \cdot 301$
- $70 = 3 \cdot 21 + 7$,

  $r_4 = 7 = 70 - 3 \cdot 21 = (973 - 3 \cdot 301) - 3(-4 \cdot 973 + 13 \cdot 301)$

  $r_4 = (13) \cdot 973 + (-42) \cdot 301$
- Como $r_5 = r_3 \mod r_4 = 0$, algoritmo pára e temos:

  $mdc(r_0, r_1) = r_4 = 7 = (13) \cdot 973 + (-42) \cdot 301$

Algoritmo de Euclides Extendido (AEE)

- AEE acha inteiros s e t tal que $mdc(r_0, r_1) = s \times r_0 + t \times r_1$
- De modo geral, começa com

$$r_{i-2} = s_{i-2} r_0 + t_{i-2} r_1$$

$$r_{i-1} = s_{i-1} r_0 + t_{i-1} r_1$$

- e na próxima iteração

$$r_{i-2} = q_{i-1} r_{i-1} + r_i$$

$$r_i = r_{i-2} - q_{i-1} r_{i-1}$$

$$r_i = (s_{i-2} r_0 + t_{i-2} r_1) - q_{i-1}(s_{i-1} r_0 + t_{i-1} r_1)$$

$$r_i = (s_{i-2} - q_{i-1} s_{i-1}) r_0 + (t_{i-2} - q_{i-1} t_{i-1}) r_1$$

$$r_i = s_i r_0 + t_i r_1$$

Algoritmo de Euclides Extendido (AEE)

- AEE acha inteiros s e t tal que $mdc(r_0, r_1) = s \times r_0 + t \times r_1$
- Fórmula recursiva:
  - $s_i = s_{i-2} - q_{i-1} s_{i-1}$
  - $t_i = t_{i-2} - q_{i-1} t_{i-1}$, com $i \geq 2$
  - onde $s_0 = 1$, $s_1 = 0$, $t_0 = 0$, $t_1 = 1$

Algoritmo de Euclides Extendido

Algorithm 2 Algoritmo de Euclides Extendido

Require: Inteiros positivos r_0 e r_1, $r_0 > r_1$

Ensure: $mcd(r_0, r_1)$ e inteiros s, t tal que $mcd(r_0, r_1) = s \cdot r_0 + t \cdot r_1$

Inicialização: $i \leftarrow 1$, $s_0 = 1, s_1 = 0, t_0 = 0, t_1 = 1, i = 1$

repeat

$i \leftarrow i + 1$

$r_i \leftarrow r_{i-2} \mod r_{i-1}$

$q_{i-1} \leftarrow (r_{i-2} - r_i)/r_{i-1}$

$s_t \leftarrow s_{i-2} - q_{i-1} \cdot s_{i-1}$

$t_i \leftarrow t_{i-2} - q_{i-1} \cdot t_{i-1}$

until $r_i \neq 0$

return $mcd(r_0, r_1) = r_{i-1}, s = s_{i-1}, t = t_{i-1}$

Usar AEE para divisão $\mod n$

- Queremos: $a^{-1} = ? \mod n$

  tal que $a^{-1} \cdot a = 1 \mod n$
- Temos que: $mdc(n, a) = 1$
- Usando o AEE sabendo que n é primo e $n > a$:

  $mdc(n, a) = s \cdot n + t \cdot a = 1$
- Aplicar $\mod n$:

  $mdc(n, a) \mod n = (s \cdot n + t \cdot a) \mod n = 1 \mod n$

  $s \cdot n \mod n + t \cdot a \mod n = 1 \mod n$

  $t \cdot a = 1 \mod n$
- Assim obtemos $t = a^{-1}$, a inversa da multiplicação

"S-Box" do AES

- "S-box" do AES troca um byte $a(x)$ por outro $b(x)$
  - $b(x) = M \cdot a^{-1}(x) + 0x63$
- Computar $a^{-1}(x)$ no campo finito $GF(2^8)$ com polinômio irredutível

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

- Usar Algoritmo de Euclides extendido adaptado para polinômios
  - Para achar $a^{-1}(x)$, obter $mdc(a(x), P(x)) = s \cdot P(x) + t \cdot a(x)$
  - Fazer $a^{-1}(x) = t \cdot a(x) \mod P(x)$