UnB – Campus Darcy Ribeiro

CIC0087 - TOPICOS AVANCADOS EM COMPUTADORES - Turma 02 - 2025/1

Discente: Guilherme Araújo de Oliveira

Matrícula: 190125748

S-AES

(Simplified - Advanced Encryption Standard)

Brasília, 2025

UnB – Campus Darcy Ribeiro

CIC0087 - TOPICOS AVANCADOS EM COMPUTADORES - Turma 02 - 2025/1

Discente: Guilherme Araújo de Oliveira

Matrícula: 190125748

1. Introdução

O algoritmo S-AES é uma versão simplificada do algoritmo AES muito utilizada para fins didáticos e foi concebido de forma análoga ao algoritmo S-DES para a compreensão do algoritmo DES. Os algoritmos AES e S-AES possuem complexidade O (n*r) se forem seguir a risca suas definições mas uma vez que passam a considerar tamanhos variáveis passa a ter complexidade O (m) onde m se refere ao tamanho da mensagem.

2. Detalhes do S-AES

O algoritmo S-AES consiste em um algoritmo de cifragem em blocos usando chave e blocos de 16 bits. Além disso usa duas rodadas e uma expansão de chave para geração de outras duas chaves adicionais com base na chave inicial de 16 bits. As duas rodadas fazem a fragmentação da chave em blocos sendo que na primeira rodada faz quatro blocos e na segunda faz três blocos.

2.1. Principais diferenças entre o S-AES e o AES

Detalhes	S-AES	AES
Tamanho de bloco (r)	16 bits	128 bits
Tamanho de chave	16 bits	128, 192 ou 256 bits
Número de rounds (n)	2	10 para chave de 128 bits 12 para chave de 192 bits 14 para chave de 256 bits

2.2. Operações do S-AES

As operações do S-AES são:

- **AddRoundKey**
- **SubstituteNibles**
- **ShiftRows**
- **MixColums**
- **ExpansionKey**

Antes de continuar é importante ressaltar que essas operações são feitas sobre o estado de 16 bits, como assim? Independente da mensagem que for passada a chave que a acompanha deve ter 16 bits. Segue os detalhes de cada operação:

1. **AddRoundKey**

   Essa operação consiste em aplicar a operação lógica **XOR** a uma chave de **16 bits** (2 bytes). Ao aplicar a operação **XOR** ocorre uma combinação bit a bit de duas entradas, pois a chave é dividida pela metade, retornando um novo valor com base em suas diferenças, e para isso ocorre a divisão das chaves de 8 bits (1 byte) em nibbles, que consistem em 4 bits, uma vez que é feita essa conversão a chave

passa a formar uma matriz 4x4. Para melhor compreensão segue a tabela verdade do XOR e um exemplo abaixo de como ocorre essas operações.

A	B	X = A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

Dado em bits	Dado em bits	XOR	Resultado
3 (0011)	7 (0111)	3 ⊕ 7	4 (0100)
1 (0001)	4 (0100)	1 ⊕ 4	5 (0101)
F (1111)	1 (0001)	F ⊕ 1	E (1110)
B (1011)	9 (1001)	B ⊕ 9	2 (0010)

Note que:

- Na primeira e na segunda linha são feitas as conversões de decimal para binário
- Na terceira e na quarta linha ocorrem as conversões de hexadecimal para binário

Após essa conversão e aplicação do XOR, considera-se que o resultado agora é uma chave de 16 bits, segue exemplo de implementação orientada a objetos com Python. Interessante notar que o operador XOR pode ser representado pelo caractere “^”

```
def add_round_key(self, bloco, chave):
    return [b ^ k for b, k in zip(bloco, chave)]
```

2. SubstituteNibbles

Essa é uma operação pode ser considerada como elemento-chave no algoritmo do S-AES, pois envolve concepções matemáticas mais complexas. A ideia dessa operação é substituir bit a bit usando uma tabela randômica que chamamos de S-box, matematicamente pode ser definida como uma função booleana vetorial não linear. Com esse embaralhamento não linear o que se espera é alcançar a confusão de Shannon, que faz com que cada bit cifrado seja dependente da chave, segue um exemplo de como as S-Boxes funcionam com as matrizes 4x4 vistas anteriormente:

Considere que queremos embaralhar o seguinte dado 45E2, de cara não se pode concluir nada, por enquanto vamos apenas converter esses números de base hexadecimal para nibbles

Hexadecimal	Nibbles
0x4	0100
0x5	0101
0xE	1110
0x2	0010

Show! Agora levando em conta que o S-AES opera no campo finito GF(2^4) podemos concluir que o módulo é um polinômio irreducível de grau 4 e como são elementos representados com 4 bits (

	00 (0)	01 (1)	10 (2)	11 (3)
00 (0)	1001 (9)	0100 (4)	1010 (A)	1011 (B)
01 (1)	1101 (D)	0001 (1)	1000 (8)	0101 (5)
10 (2)	0110 (6)	0010 (2)	0000 (0)	0011 (3)
11 (3)	1100 (C)	1110 (E)	1111 (F)	0111 (7)

Note que:

- Após o embaralhamento de 45E2, é feita a checagem na tabela S-Box randômica para geração da nova chave.

Segue exemplo de implementação orientada a objetos com Python.

```
def sub_nibbles(self, bloco):
    return [self.s_box[n] for n in bloco]
```

3. ShiftRows

Essa operação consiste em uma troca de posição entre os nibbles, essa operação é auto inversa, como assim? Quer dizer que a operação pode ser revertida para descriptografia, ou seja, enquanto as outras funções do S-AES precisam de uma função a parte para descriptografia essa função *shift_rows* pode ser usada para ir e vir. Essa função opera nos dois últimos blocos de dados. Segue exemplo de implementação com orientação a objetos em Python.

```
def shift_rows(self, bloco):
    return [bloco[0], bloco[1], bloco[3], bloco[2]]
```

4. Mix Colums

Essa função é responsável por fazer o embaralhamento dos dados disposto sob a matriz 2x2 com coeficientes fixos, um exemplo de seu uso seria receber a matriz [1,4] e retornar a matriz com os seguintes valores [4,1].

```
def mix_columns(self, bloco):
    def mult(a, b):
        p = 0
        for _ in range(4):
            if b & 1: p ^= a
            hi_bit = a & 0x8
            a = (a << 1) & 0xF
            if hi_bit: a ^= 0x3
            b >>= 1
        return p

    return [
        mult(self.mix_columns_matrix[0][0], bloco[0]) ^ mult(self.mix_columns_matrix[0][1], bloco[2]),
        mult(self.mix_columns_matrix[0][0], bloco[1]) ^ mult(self.mix_columns_matrix[0][1], bloco[3]),
        mult(self.mix_columns_matrix[1][0], bloco[0]) ^ mult(self.mix_columns_matrix[1][1], bloco[2]),
        mult(self.mix_columns_matrix[1][0], bloco[1]) ^ mult(self.mix_columns_matrix[1][1], bloco[3]),
    ]
```

5. **Key Expansion**

Essa função é responsável por transformar a chave original em um sequência de subcahves que são utilizadas em diferentes fases da criptografia, no S-AES essa função gera duas chave (*w0* e *w1*), no AES real pode operar com S-Boxes, rotações e até mesmo com XORs, garantindo assim que a chave gerada seja única e imprevisível. Segue exemplo de implementação básica dessa operação com Python.

```
def key_expansion(self, chave):
    return [chave[:4], chave[4:]]
```