

AES - Advanced Encryption Standard

Lucas Gabriel de Oliveira Lima - 231003406

1 de maio de 2025

Resumo

Este trabalho tem como base o algoritmo de criptografia simétrica o Advanced Encryption Standard (AES). Neste trabalho, será apresentada a implementação detalhada do Simplified AES, além do seu uso em um modo de operação Electronic Code Book. Além disso, o AES real, seguindo as definições do NIST, será usado em cinco diferentes modo de operação (ECB, CBC, CFB, OFB, CTR), com um comparativo de alguns fatores entre eles. As implementações desse trabalho foram feitas utilizando a linguagem de programação Python.

1 Introdução

O **Advanced Encryption Standard (AES)** é um dos algoritmos de criptografia mais amplamente utilizados e confiáveis no mundo da segurança da informação. Desenvolvido para substituir o antigo padrão DES (Data Encryption Standard), o AES foi adotado pelo Instituto Nacional de Padrões e Tecnologia dos Estados Unidos (NIST) em 2001 como o novo padrão para criptografia de dados. Sua eficiência, segurança e flexibilidade o tornaram uma opção melhor para proteger informações sensíveis em uma ampla gama de aplicações, desde comunicações pessoais até sistemas corporativos e governamentais.

O AES é um algoritmo de criptografia simétrica baseado em blocos, o que significa que ele utiliza a mesma chave para criptografar e descriptografar os dados. Ele opera em blocos de 128 bits e suporta chaves de 128, 192 ou 256 bits, oferecendo diferentes níveis de segurança. Além disso, o AES é projetado para ser eficiente tanto em hardware quanto em software, garantindo alto desempenho em dispositivos modernos.

Neste artigo, exploraremos os fundamentos do AES, suas principais características e modos de operação. Também abordaremos uma versão simplificada do algoritmo, conhecida como **S-AES** [Hol04], que é amplamente utilizada para fins educacionais e demonstrações práticas. Utilizaremos, também, essa implementação do modo de operação *Electronic Code Book* com essa implementação do *S-AES*.

Traremos também o código fonte das principais partes que foram solicitadas para esse projeto. As implementações dos requisitos pedidos foram feitas utilizando a linguagem de programação Python. O código-fonte do projeto pode ser encontrado no repositório a seguir: https://github.com/lucasdbr05/AES-Advanced_Encryption_Standard

2 Simplified AES

O **Simplified AES** é uma versão simplificada do AES, criada para fins educativos. No S-AES, o *plaintext* tem tamanho de 16 bits, e a chave tem 16 bits de tamanho, e retorna um *ciphertext* de também 16 bits.

O S-AES possui alguns processos, que ocorrem de maneira similar ao AES original.

O *plaintext* dividido em quatro nibbles é disposto em forma de matriz, da seguinte forma:

$$\begin{bmatrix} n_0 & n_2 \\ n_1 & n_3 \end{bmatrix}$$

2.1 Key Expansion

A chave de 16 bits é expandida, e então será gerada 3 chaves (K0, K1, K2) que serão utilizadas ao longo das rodadas do AES.

Para isso, a chave original será dividida (W0 e W1) e serão aplicadas operações de substituições e transposições, além de um *XOR* com um valor constante para cada rodada, que levarão a gerar W2 e

W3, que serão utilizadas para gerar a chave K1. De forma análoga, W2 e W3 passarão por esse processo, gerando W4 e W5, que serão usados para gerar a chave K2.

O *plaintext* é disposto em uma matriz de 2 linhas e 2 colunas, e cada nibble da matriz é disposto por coluna,

As operações ocorrem conforme a imagem a seguir:

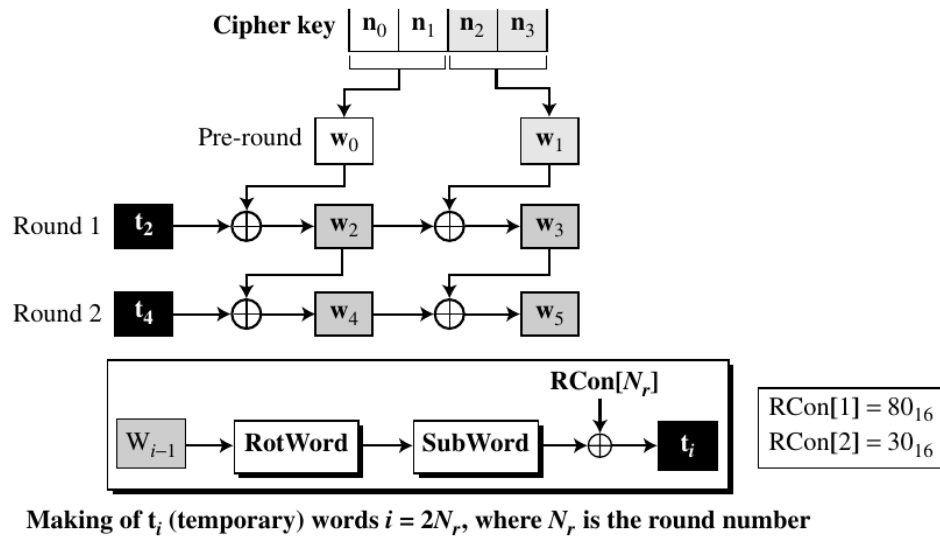


Figura 1: Expansão da Chave

2.2 Substitute Nibbles

Cada nibble da matriz é substituído por um outro nibble, seguindo o padrão de S-box, como mostrado a seguir:

Tabela 1: S-box para nibbles (4 bits)

| nibble | S-box(nibble) | nibble | S-box(nibble) |
|--------|---------------|--------|---------------|
| 0000 | 1001 | 1000 | 0110 |
| 0001 | 0100 | 1001 | 0010 |
| 0010 | 1010 | 1010 | 0000 |
| 0011 | 1011 | 1011 | 0001 |
| 0100 | 1101 | 1100 | 1100 |
| 0101 | 0001 | 1101 | 1110 |
| 0110 | 1000 | 1110 | 1111 |
| 0111 | 0101 | 1111 | 0111 |

2.3 Shift Rows

Nesse passo, ocorre uma troca dos nibbles da segunda linha da matriz, se transforma na seguinte matriz:

$$\begin{bmatrix} n_0 & n_2 \\ n_3 & n_1 \end{bmatrix}$$

2.4 Mix Columns

Essa operação realiza uma multiplicação de cada coluna da matriz com a seguinte matriz:

$$\begin{bmatrix} s_{0,0} & s_{0,1} \\ s_{1,0} & s_{1,1} \end{bmatrix} \cdot \begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} \\ s'_{1,0} & s'_{1,1} \end{bmatrix}$$

As operações são feitas em Galois-Field(2^4), módulo $2^4 + 2 + 1$. No S-AES, ela é realizada apenas na primeira rodada.

2.5 Add Round Key

É realizada uma operação de *XOR* entre um dado e uma das chaves geradas. Essa operação ocorre inicialmente antes de cada round, entre a primeira chave e o *plaintext*, e também no final de cada rodada, com a chave correspondente à respectiva rodada e o texto cifrado até o momento.

2.6 Arquitetura de encriptação S-AES

A encriptação do Simplified AES projetada da seguinte forma:

A partir de uma chave de 16 bits, ocorre uma **key expansion**, gerando três chaves (K_0, K_1, K_2).

Antes de começar as rodadas, um **add round key** entre K_0 e o *plaintext*.

Então, na primeira rodada, ocorre um **substitute nibbles**, então ocorre um **shift row**, e depois um **mix columns**. Finalizando a rodada, ocorre **add round key** com K_1 .

Na segunda rodada, ocorre um **substitute nibbles**, então ocorre um **shift row**. Finalizando a rodada, ocorre **add round key** com K_2 . E então é gerado o cipher text.

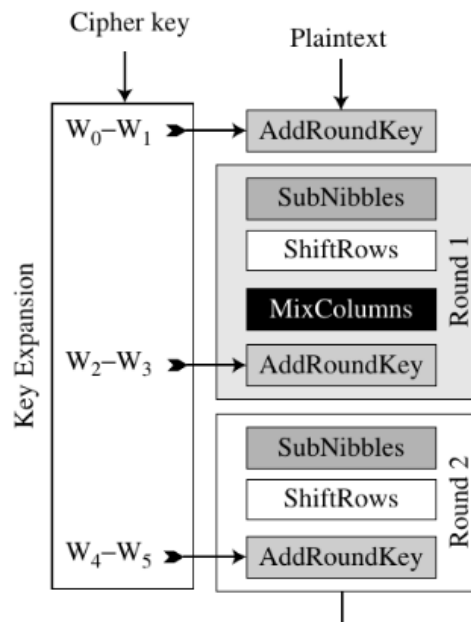


Figura 2: Encriptação com S-AES

2.7 Arquitetura de deciptação S-AES

A deciptação do Simplified AES é projetada da seguinte forma:

Antes de começar as rodadas, um **add round key** entre K_2 e o *plaintext*.

Então, na primeira rodada, ocorre um **shift rows** inverso, então ocorre um **substitute nibbles** inverso, e depois ocorre **add round key** com K_1 . Finalizando a rodada, ocorre um **mix columns** inverso.

Na segunda rodada, ocorre um **shift rows** inverso, então ocorre um **substitute nibbles** inverso. Finalizando a rodada, ocorre **add round key** com K_0 . E então é gerado o *plaintext* original.

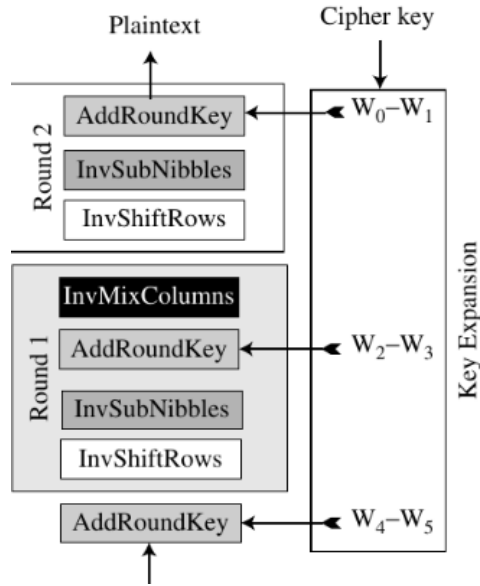


Figura 3: Deciptação com S-AES

2.8 Implementação

A seguir está partes da minha implementação do S-AES em Python, e pode ser vista por completo em ["S_AES.py"](#):

Listing 1: S-AES

```

1 class S_AES():
2     def __init__(self, key: np.uint16) -> None:
3         self.__key = key
4         self.__S_box = [ 0b1001, 0b0100, 0b1010, 0b1011, 0b1101, 0b0001, 0
                           b1000, 0b0101, 0b0110, 0b0010, 0b0000, 0b0011, 0b1100, 0b1110, 0
                           b1111, 0b0111, ]
5         self.__S_box_inv = [ 0b1010, 0b0101, 0b1001, 0b1011, 0b0001, 0b0111, 0
                              b1000, 0b1111, 0b0110, 0b0000, 0b0010, 0b0011, 0b1100, 0b0100, 0
                              b1101, 0b1110, ]
6
7         self.__K0, self.__K1, self.__K2 = self.__expand_key(key)
8
9
10    def encrypt(self, data: str) -> np.uint16:
11        # Parse string to bits
12        data = parse_str_to_int(data)
13
14        # Pre-rounds
15        data = self.__add_round_key(data, self.__K0)
16        Logger.print_saes_block_with_nibbles_matrix(data, "Pre round")
17
18        # First Round
19        data = self.__substitute_nibbles(data)
20        data = self.__shift_rows(data)
21        data = self.__mix_columns(data)
22        data = self.__add_round_key(data, self.__K1)
23        Logger.print_saes_block_with_nibbles_matrix(data, "First round")
24
25        # Second Round
26        data = self.__substitute_nibbles(data)
27        data = self.__shift_rows(data)
28        data = self.__add_round_key(data, self.__K2)
29

```

```

30     Logger.print_saes_block_with_nibbles_matrix(data, "Second round")
31
32     return data
33
34 def decrypt(self, data: str) -> np.uint16:
35     # Parse string to bits
36     data = parse_str_to_int(data)
37
38
39     # Pre-rounds
40     data = self.__add_round_key(data, self.__K2)
41     Logger.print_saes_block_with_nibbles_matrix(data, "Pre round")
42
43     # First Round
44     data = self.__shift_rows(data)
45     data = self.__inverse_substitute_nibbles(data)
46     data = self.__add_round_key(data, self.__K1)
47     data = self.__inverse_mix_columns(data)
48     Logger.print_saes_block_with_nibbles_matrix(data, "First round")
49
50     # Second Round
51     data = self.__shift_rows(data)
52     data = self.__inverse_substitute_nibbles(data)
53     data = self.__add_round_key(data, self.__K0)
54     Logger.print_saes_block_with_nibbles_matrix(data, "Second round")
55
56     return data

```

2.9 Comparativo com o AES

Tabela 2: Comparação entre S-AES e AES

| Aspecto | S-AES | AES |
|-----------------------------|--|---|
| Propósito | Usado para fins educacionais e demonstrações práticas. | Usado para segurança real em sistemas de informação. |
| Tamanho do Bloco | 16 bits (2 bytes). | 128 bits (16 bytes). |
| Tamanho da Chave | 16 bits (2 bytes). | 128, 192 ou 256 bits. |
| Número de Rodadas | 2 rodadas. | 10, 12 ou 14 rodadas, dependendo do tamanho da chave. |
| Substituição (S-Box) | S-Box simplificada de 4 bits. | S-Box de 8 bits baseada em operações no campo finito $GF(2^8)$. |
| Operações | Inclui substituição de nibbles, rotação de linhas, mistura de colunas e adição de chave. | Inclui substituição de bytes, deslocamento de linhas, mistura de colunas e adição de chave. |
| Complexidade | Simples, com operações em $GF(2^4)$. | Mais complexo, com operações em $GF(2^8)$. |
| Eficiência | Muito rápido devido ao tamanho reduzido. | Otimizado para hardware e software modernos, mas mais pesado. |
| Segurança | Não é seguro para uso prático devido ao tamanho pequeno da chave e do bloco. | Altamente seguro e amplamente utilizado em aplicações reais. |

3 Electronic Code Book (ECB) Mode

O **Electronic Code Book (ECB) Mode** é o modo de operação mais simples para ser usado com o algoritmo S-AES. Nele, a mensagem é particionada em blocos de tamanho b bits, com tamanho fixo, e cada bloco é criptografado de forma independente e com a mesma chave.

Entre suas vantagens, além da **simplicidade** de implementação, ele permite **paralelização**, pois os blocos são processados de maneira independente.

No entanto, entre suas desvantagens, se incluem a falta de difusão, pois blocos idênticos no *plaintext* resultam em blocos parecidos no *ciphertext*, revelando padrões que comprometem a segurança, e pode servir como um vetor de ataque a essa cifra.

3.1 Implementação

A seguir está partes da minha implementação do ECB, utilizando minha implementação de S-AES, em Python, e pode ser vista por completo em "ECB.py":

Listing 2: ECB Operation Mode

```
1 def encrypt_saes_ecb(text: str, key: np.int16) -> str:
2     """
3         Encrypt a given text using the Simplified AES (S-AES) algorithm in ECB
4         mode.
5         Args:
6             text (str): The input text to be decrypted. It should be a string
7             of characters.
8             key (np.int16): The encryption key used for decryption, represented
9             as a 16-bit integer.
10        Returns:
11            str: the ciphertext resulting in ECB mode encryption
12    """
13    # Initialize the S-AES instance with the provided key
14    s_aes = S_AES(key)
15    # Pad the input text to ensure it fits the block size
16    text = padding(text)
17
18    blocks = []
19    # Process the text in blocks of 2 characters
20    for i in range(0, len(text), 2):
21        block = text[i: i+2]
22        # Encrypt each block using S-AES
23        data = s_aes.encrypt(block)
24        blocks.append(data)
25
26    # Log the encrypted data for each block
27    for i in range(len(blocks)):
28        Logger.print_saes_block(blocks[i], f"Data for block {i+1}")
29
30    # Combine all encrypted blocks into a single binary string
31    return "".join(format_bin(block) for block in blocks)
32
33 def decrypt_saes_ecb(text: str, key: np.int16) -> str:
34     """
35        Decrypts a given text using the Simplified AES (S-AES) algorithm in ECB
36        mode.
37        Args:
38            text (str): The input text to be decrypted. It should be a string of
39            characters.
40            key (np.int16): The encryption key used for decryption, represented as
41            a 16-bit integer.
42        Returns:
43            str: the original plaintext resulting in ECB mode decryption
44    """
```

```

40 # Initialize the S-AES instance with the provided key
41 s_aes = S_AES(key)
42 # Pad the input text to ensure it fits the block size
43 text = padding(text)
44
45 blocks = []
46 # Process the text in blocks of 2 characters
47 for i in range(0, len(text), 2):
48     block = text[i: i+2]
49     # Decrypt each block using S-AES
50     data = s_aes.decrypt(block)
51     blocks.append(data)
52
53 # Log the decrypted data for each block
54 for i in range(len(blocks)):
55     Logger.print_saes_block(blocks[i], f>Data for block {i+1}<
56
57 # Combine all decrypted blocks into a single binary string
58 return "".join(format_bin(block) for block in blocks)

```

4 Modos de Operação com AES

O AES fornece uma criptografia de bloco segura e eficiente. No entanto, existem diferentes formas de utilizar o mesmo AES para criptografar uma mensagem grande, transformá-la em blocos e cifrá-la. Essas diferentes formas são definidas pelos modos de operação.

4.1 Electronic Code Book Mode (ECB):

Cada bloco do texto é encriptografado de forma independente e usando a mesma chave. Isso permite paralelismo, no entanto, blocos idênticos possuem resultados similares quando cifrados.

4.2 Cipher Block Chaining Mode (CBC):

Cada bloco de texto é combinado, com um *XOR*, com o bloco cifrado anterior, antes de ser criptografado. O primeiro bloco usa um vetor de inicialização (IV). Isso aumenta a difusão do texto cifrado, mas já não permite paralelizar o processo. Além disso, ele depende que o vetor de inicialização seja relativamente seguro.

4.3 Cipher FeedBack Chaining Mode (CFC):

Transforma o AES em um cifrador de fluxo. O vetor de inicialização é criptografado e combinado, por meio de um *XOR*, com o *plaintext* para produzir o *ciphertext*. Isso permite que dados de tamanho menor que o bloco sejam criptografados, no entanto, também não permite paralelismo. Além disso, caso ocorra erros, haverá uma cascata de propagação maior.

4.4 Output FeedBack Mode (OFB):

Também transforma o AES em um cifrador de fluxo. Nesse caso, o vetor de inicialização (IV) é continuamente criptografado para gerar um fluxo de chave, que é combinado, por meio de um *XOR*, com o *plaintext*. Nesse modo, o ele evita a propagação de erros, mas ainda não permite paralelismo na cifração.

4.5 Counter Mode (CTR):

Um contador criptografado para gerar um fluxo de chave, que é combinado, por meio de um *XOR*, com o *plaintext*. Cada bloco usa um valor único do contador. Esse modo permite o paralelismo no processo de criptografia, no entanto, requerer um contador único para cada bloco não é muito eficiente.

4.6 Tabela Comparativo

Tabela 3: Resumo Comparativo dos Modos de Operação

| Modo | Uso Comum | Segurança | Aleatoriedade | Execução | Eficiência | Vulnerabilidades |
|------------|---------------------|-----------|---------------|----------|------------|--------------------------|
| ECB | Não recomendado | Baixa | Nenhuma | Rápido | Alta | Padrões repetitivos |
| CBC | Segurança geral | Média | Sim (IV) | Média | Média | Propagação de erros |
| CFB | Cifradores de fluxo | Média | Sim (IV) | Média | Média | Propagação de erros |
| OFB | Cifradores de fluxo | Alta | Sim (IV) | Média | Alta | Reutilização do IV |
| CTR | Sistemas modernos | Alta | Sim (nonce) | Rápido | Alta | Reutilização do contador |

Ao executar o projeto na função de comparação de modos de operação, é possível ver, para uma dada entrada, o tempo de execução em milissegundos, e a entropia aproximada da mensagem encriptada, com base em referências do [NIST \[oSN10\]](#). A implementação da parte de modos de operação pode ser encontrada em "[AES.py](#)".

5 Conclusão

O presente trabalho apresentou a implementação do algoritmo AES (Advanced Encryption Standard) e sua versão simplificada, o S-AES, explorando diferentes modos de operação, como ECB, CBC, CFB, OFB e CTR. Essas implementações permitiram compreender os fundamentos da criptografia de blocos, bem como as vantagens e limitações de cada modo de operação.

Através do S-AES, foi possível simplificar os conceitos do AES, tornando-os mais acessíveis para fins educacionais. Já a implementação completa do AES demonstrou sua robustez e flexibilidade, sendo amplamente utilizado em aplicações reais devido à sua segurança e eficiência.

Além disso, o trabalho incluiu funcionalidades para medir a entropia e o tempo de execução de cada modo de operação, permitindo uma análise prática do desempenho e da segurança de cada abordagem. A integração de um menu interativo facilitou a experimentação e o teste das diferentes funcionalidades, tornando o projeto uma ferramenta útil tanto para aprendizado quanto para demonstrações.

Por fim, este trabalho reforça a importância da criptografia na proteção de dados e destaca o papel do AES como um dos algoritmos mais confiáveis e amplamente utilizados na segurança da informação. A implementação prática dos conceitos abordados contribuiu para um entendimento mais profundo das técnicas de criptografia e sua aplicação em cenários reais.

Referências

- [Hol04] Holden. Simplified AES. Lecture notes, Rose-Hulman Institute of Technology, 2003–2004.
- [oSN10] National Institute of Standards and Technology (NIST). A statistical test suite for random and pseudorandom number generators for cryptographic applications. Special Publication SP 800-22 Rev. 1a, National Institute of Standards and Technology, April 2010.