AES - Advanced Encryption Standard

Lucas Gabriel de Oliveira Lima - 231003406
Pedro Lucas Pereira Neris - 231018964

15 de junho de 2025

Universidade de Brasília (UnB)
Departamento de Ciência da Computação (CIC)
Disciplina: Segurança Computacional
Semestre: 2025.1
Professora: Lorena de Souza Bezerra Borges

Resumo

Este trabalho tem como base o algoritmo de criptografia simétrica Advanced Encryption Standard (AES). Será apresentada a implementação detalhada do Simplified AES (S-AES), além do seu uso com o modo de operação Electronic Code Book (ECB). 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 Pyhton.

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 trabalho, 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 requisistos pedidos foram feitas utilizando a linguagem de programção 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 são geradas 3 chaves (K0, K1, K2), que serão utilizadas ao longo das rodadas do AES.

Para isso, a chave original é dividida (W0 e W1) e são aplicados 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 utizadas para gerar a chave K1. De forma análoga, W2 e W3 passam 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:

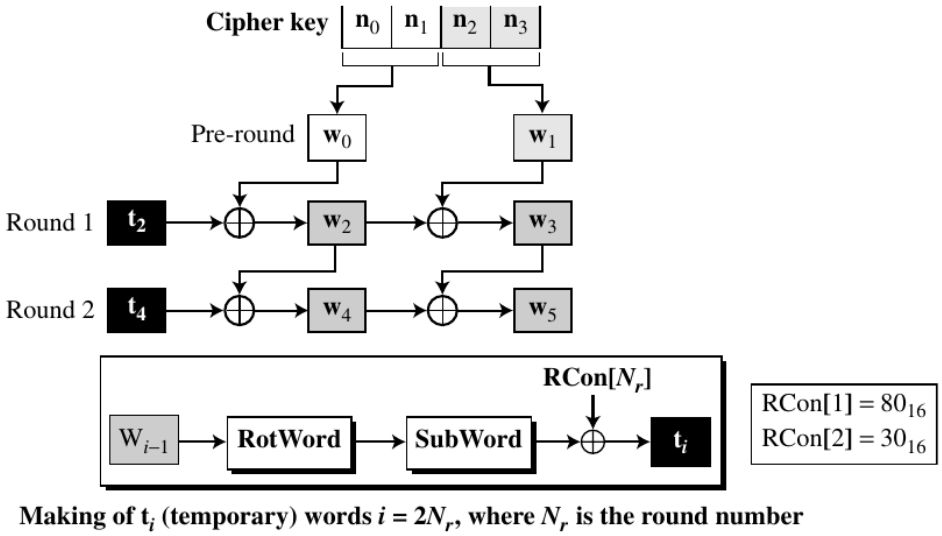

Figura 1: Expansão da Chave

2.2 Substitute Nibbles

Cada *nibble* da matriz é substituido 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 transformando 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 é 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, ocorre um **add round key** entre $K0$ 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 $K1$.

Na segunda rodada, ocorre um **substitute nibbles**, então ocorre um **shift row**. Finalizando a rodada, ocorre **add round key** com $K2$. E então é gerado o *cipher text*.

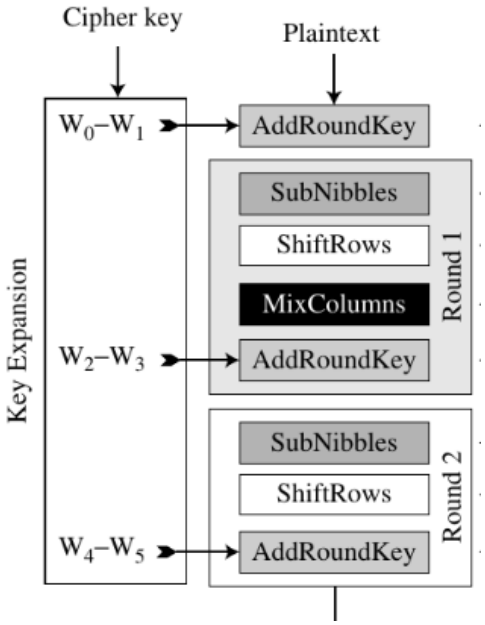

Figura 2: Encriptação com S-AES

2.7 Arquitetura de decriptação S-AES

A decriptação do Simplified AES é projetada da seguinte forma:

Antes de começar as rodadas, um **add round key** entre $K2$ e o *plaintext*.

Então, na primeira rodada, ocorre um **shift rows** inverso, então ocorre um **substitute nibbles** inverso, e deopis ocorre **add round key** com $K1$. 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 $K0$. E então é gerado o *plaintext* original.

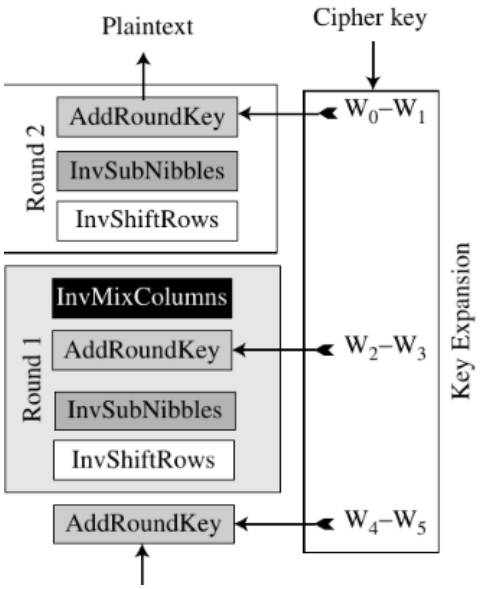

Figura 3: Decriptação com S-AES

2.8 Implementação

A seguir estão partes da implementação do S-AES em Python, que pode ser vista por completo em "S_AES.py":

Listing 1: S-AES

```
class S_AES():
    def __init__(self, key: np.uint16) -> None:
        self.__key = key
        self.__S_box =  [ 0b1001, 0b0100, 0b1010, 0b1011, 0b1101, 0b0001, 0
            b1000, 0b0101, 0b0110, 0b0010, 0b0000, 0b0011, 0b1100, 0b1110, 0
            b1111, 0b0111, ]
        self.__S_box_inv = [ 0b1010, 0b0101, 0b1001, 0b1011, 0b0001, 0b0111, 0
            b1000, 0b1111, 0b0110, 0b0000, 0b0010, 0b0011, 0b1100, 0b0100, 0
            b1101, 0b1110, ]

        self.__K0, self.__K1, self.__K2 =  self.__expand_key(key)


    def encrypt(self, data: str) -> np.uint16:
        # Parse string to bits
        data = parse_str_to_int(data)

        # Pre-rounds
        data = self.__add_round_key(data, self.__K0)
        Logger.print_saes_block_with_nibbles_matrix(data, "Pre round")

        # First Round
        data = self.__substitute_nibbles(data)
        data = self.__shift_rows(data)
        data = self.__mix_columns(data)
        data = self.__add_round_key(data, self.__K1)
        Logger.print_saes_block_with_nibbles_matrix(data, "First round")

        # Second Round
        data = self.__substitute_nibbles(data)
        data = self.__shift_rows(data)
        data = self.__add_round_key(data, self.__K2)

```

```
        Logger.print_saes_block_with_nibbles_matrix(data, "Second round")

        return data

    def decrypt(self, data: str) -> np.uint16:
        # Parse string to bits
        data = parse_str_to_int(data)


        # Pre-rounds
        data = self.__add_round_key(data, self.__K2)
        Logger.print_saes_block_with_nibbles_matrix(data, "Pre round")

        # First Round
        data = self.__shift_rows(data)
        data = self.__inverse_substitute_nibbles(data)
        data = self.__add_round_key(data, self.__K1)
        data = self.__inverse_mix_columns(data)
        Logger.print_saes_block_with_nibbles_matrix(data, "First round")

        # Second Round
        data = self.__shift_rows(data)
        data = self.__inverse_substitute_nibbles(data)
        data = self.__add_round_key(data, self.__K0)
        Logger.print_saes_block_with_nibbles_matrix(data, "Second round")

        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 **Eletronic 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 estão partes da implementação do ECB, utilizando a implementação de S-AES, em Python, e pode ser vista por completo em "ECB.py":

Listing 2: ECB Operation Mode

```
def encrypt_saes_ecb(text: str, key: np.int16) -> str:
    """
        Encrypt a given text using the Simplified AES (S-AES) algorithm in ECB
            mode.
        Args:
            text (str): The input text to be decrypted. It should be a string
                of characters.
            key (np.int16): The encryption key used for decryption, represented
                 as a 16-bit integer.
        Returns:
            str: the ciphertext resulting in ECB mode encryptation
    """
    # Initialize the S-AES instance with the provided key
    s_aes = S_AES(key)
    # Pad the input text to ensure it fits the block size
    text = padding(text)

    blocks = []
    # Process the text in blocks of 2 characters
    for i in range(0, len(text), 2):
        block = text[i: i+2]
        # Encrypt each block using S-AES
        data = s_aes.encrypt(block)
        blocks.append(data)

    # Log the encrypted data for each block
    for i in range(len(blocks)):
        Logger.print_saes_block(blocks[i], f"Data for block {i+1}")

    # Combine all encrypted blocks into a single binary string
    return "".join(format_bin(block) for block in blocks)

def decrypt_saes_ecb(text: str, key: np.int16) -> str:
    """
    Decrypts a given text using the Simplified AES (S-AES) algorithm in ECB
        mode.
    Args:
        text (str): The input text to be decrypted. It should be a string of
            characters.
        key (np.int16): The encryption key used for decryption, represented as
            a 16-bit integer.
    Returns:
        str: the original plaintext resulting in ECB mode decryptation

    """
```

```
    # Initialize the S-AES instance with the provided key
    s_aes = S_AES(key)
    # Pad the input text to ensure it fits the block size
    text = padding(text)

    blocks = []
    # Process the text in blocks of 2 characters
    for i in range(0, len(text), 2):
        block = text[i: i+2]
        # Decrypt each block using S-AES
        data = s_aes.decrypt(block)
        blocks.append(data)

    # Log the decrypted data for each block
    for i in range(len(blocks)):
        Logger.print_saes_block(blocks[i], f"Data for block {i+1}")

    # Combine all decrypted blocks into a single binary string
    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):

No modo de operação ECB, a mensagem em claro é dividida em blocos de tamanhos iguais, e cada bloco é encriptado utilizando a mesma chave. Ao final, todos os blocos são concatenados na mensagem cifrada final.

Cabe ressaltar que, no ECB, as mensagens são criptografadas de forma independente, ou seja, a encriptação de um bloco não interfere na criptografia de nenhum outro. Isto faz com que dois textos em claro iguais retornem o mesmo texto cifrado, tornando o ECB vulnerável à análise de padrões. Porém, por cada bloco ser encriptado independentemente de outro, é possível utilizar paralelismo na encriptação e decriptação, melhorando o desempenho deste modo de operação.

4.2 Cipher Block Chaining Mode (CBC):

Para o 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 inicalização (IV). Isso aumenta a difusão do texto cifrado, visto que um mesmo bloco de texto pode ter textos cifrados distintos a depender da fase onde ocorre sua encriptação e o(s) bloco(s) anterior(es) a ele, 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 criptpografados, 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 cifragem.

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 Comparativa

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 milisegundos, 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 Análise de segurança do AES

Como citado anteriormente, o AES é amplamente utilizado em aplicações reais e é considerado extremamente seguro. Possíveis fraquezas ou formas de quebrar a criptografia do AES incluem ataques de força bruta, avanço da computação quântica e implementação incorreta do algoritmo, como o uso de modos de operação inadequados [Por].

Os ataques de força bruta são menos custosos computacionalmente quanto menor for o tamanho de chave utilizada (teoricamente, visto que, mesmo que se utilize chaves de 128 bits ao invés de 192 ou 256 bits, quebrar o AES por força bruta ainda é inviável). Porém, com o advento de computadores quânticos, o poder computacional exigido para ataques de força bruta pode diminuir consideravelmente, sendo necessário adaptar o algoritmo do AES para lidar com este cenário.

Já para a implementação incorreta do algoritmo, é necessário que se utilize modos de operação suficientemente seguros com o AES. Como visto anteriormente, o ECB expõe padrões da mensagem, visto que um mesmo texto em claro sempre irá produzir o mesmo texto cifrado utilizando a mesma chave, logo, este modo não é adequado para o uso com o AES. Além disto, para os outros modos de operação analisados, é necessário garantir uma aleatoriedade minimamente segura para elementos utilizados na criptografia, como o vetor de inicialização IV.

6 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.

[Por] Portnox. What is Advanced Encryption Standard (AES)? Disponível em: https://www.portnox.com/cybersecurity-101/what-is-advanced-encryption-standard-aes/. Acesso em: 07 de julho de 2025.