

# Projeto 2 - AES 128

Cláudio Barros, 190097591

<sup>1</sup>Dep. Ciéncia da Computaçáo – Universidade de Brasília (UnB)  
CIC0201 - Segurança Computacional

190097591@aluno.unb.br

**Abstract.** This project presents the implementation of AES-128, the Advanced Encryption Standard. The project focuses on understanding the intricacies of the AES-128 algorithm and offers an implementation that works correctly and allows a custom number of main rounds for the cipher.

**Resumo.** Este projeto apresenta a implementaçáo do AES-128, o Advanced Encryption Standard. O projeto teve como objetivo principal compreender as complexidades do algoritmo AES-128 e oferece uma implementaçáo que funciona corretamente e permite um númeroo personalizado de rodadas principais para a cifraçáo.

## 1. Introduçáo

O objetivo do projeto é implementar o algoritmo de cifraçáo simétrica AES-128 (Advanced Encryption Standard), que opera em blocos fixos de tamanho 128 bits e utiliza uma chave de 128 bits.

A especificaçáo oficial do algoritmo especifica 9 rodadas principais para um bloco de 128 bits e uma chave de 128 bits. Porém, para este trabalho, o algoritmo implementado permite a especificaçáo de quantas rodadas principais o algoritmo deve rodar.

### 1.1. Visão Geral do Algoritmo

#### Expansão da Chave:

- **KeyExpansion.** Expande a chave de 128-bits de acordo com um algoritmo de expansão para que seja grande o suficiente para todos os rounds e não se repita em cada round.

#### Rodada Inicial:

- **AddRoundKey.** XOR byte a byte do estado com a chave expandida.

#### Rodadas Principais:

- **SubBytes.** Substitui cada byte do estado com um byte de uma look-up table pré definida chamada de S-Box. Na decifraçáo, a look-up table é a inversa da S-Box.
- **ShiftRows.** Considerando o estado como uma matriz-coluna, desloca de maneira circular as linhas da matriz. A primeira linha se mantém igual, a segunda linha é deslocada uma posição para a esquerda, a terceira linhas duas posições e quarta três posições.
- **MixColumns.** Cada coluna é multiplicada por um polinômio pertencente a um Campo de Galois e depois reduzido módulo  $x^4 + 1$ .

- **AddRoundKey**. XOR byte a byte do estado com a chave expandida.

### Rodada Final:

- **SubBytes**
- **ShiftRows**
- **AddRoundKey**

## 1.2. Modo de Cifração CTR

O modo de cifração CTR permite transformar uma cifra de bloco, como o AES, em uma cifra de fluxo. Nesse caso, uma keystream é gerada a partir de um número gerado aleatoriamente (nonce) que é incrementado em blocos do mesmo tamanho do bloco de cifração, essa keystream é cifrada com a chave utilizando o AES no modo de operação ECB. Em seguida, um XOR da mensagem original com a keystream é feita, gerando a mensagem cifrada.

## 2. Implementação

### 2.1. Cifrador

```
void aes128(U128* in, U128* out, U128* key, uint rounds)
{
    uint numKeyBlocks = 11;
    if(rounds > 10)
        numKeyBlocks += rounds - 10 + 1;

    std::vector<U128> expKey(numKeyBlocks, {0});
    expandKey(key->vec8u, expKey[0].vec8u, rounds);

    U128 state = *in;

    //initial round:
    addRoundKey(&state, &expKey[0]);

    //intermediate rounds:
    for(uint i = 1; i <= rounds; i++)
    {
        subBytes(&state);
        shiftRows(&state);
        mixColumns(&state);
        addRoundKey(&state, &expKey[i]);
    }

    //final round
    subBytes(&state);
    shiftRows(&state);
    addRoundKey(&state, &expKey[10]);
```

```
    *out = state;  
}
```

## 2.2. KeyExpansion

```
void expandKey(u8* keyIn, u8* keyOut, uint rounds)  
{  
    // keyIn -> 16 bytes  
    // keyOut -> 176 bytes  
  
    uint n = 44;  
    if(rounds > 10)  
        n += 4*(rounds - 10) + 1;  
  
    for(uint i = 0; i < n; i++)  
    {  
        if(i < 4)  
        {  
            u8 tmp[4]{};  
            GetKeyOffset(keyIn, tmp, i*4);  
            memcpy(&keyOut[i*4], tmp, sizeof(u8) * 4);  
            continue;  
        }  
  
        if(i%4 == 0)  
        {  
            u8 a[4]{};  
            u8 b[4]{};  
            u8 c[4]{};  
  
            u8 tmp[4]{};  
            GetKeyOffset(keyOut, tmp, (i-1)*4);  
            rotWord(tmp, a);  
            memcpy(tmp, a, sizeof(u8) * 4);  
            subWord(tmp, a);  
  
            rcon(b, (i/4)-1);  
  
            GetKeyOffset(keyOut, c, (i-4)*4);  
  
            keyOut[(i*4) + 0] = a[0] ^ b[0] ^ c[0];  
            keyOut[(i*4) + 1] = a[1] ^ b[1] ^ c[1];  
            keyOut[(i*4) + 2] = a[2] ^ b[2] ^ c[2];  
            keyOut[(i*4) + 3] = a[3] ^ b[3] ^ c[3];  
        }  
    }  
}
```

```

        continue;
    }

    else
    {
        u8 a[4] {};
        u8 b[4] {};
        GetKeyOffset(keyOut, a, (i-1)*4);
        GetKeyOffset(keyOut, b, (i-4)*4);

        keyOut[(i*4) + 0] = a[0] ^ b[0];
        keyOut[(i*4) + 1] = a[1] ^ b[1];
        keyOut[(i*4) + 2] = a[2] ^ b[2];
        keyOut[(i*4) + 3] = a[3] ^ b[3];
    }
}
}

```

### 2.3. AddRoundKey

```

void addRoundKey(U128* state, U128* key)
{
    for(uint i = 0; i < 16; i++)
    {
        state->vec8u[i] = state->vec8u[i] ^ key->vec8u[i];
    }
}

```

### 2.4. SubBytes

```

void subBytes(U128* state)
{
    for(int i = 0; i < 16; i++)
        state->vec8u[i] = SBox[state->vec8u[i]];
}

```

### 2.5. ShiftRows

```

void shiftRows(U128* state)
{
    u8 r0[4] = {state->vec8u[0], state->vec8u[4], state->vec8u[8], state->vec8u[12]};
    u8 r1[4] = {state->vec8u[1], state->vec8u[5], state->vec8u[9], state->vec8u[13]};
    u8 r2[4] = {state->vec8u[2], state->vec8u[6], state->vec8u[10], state->vec8u[14]};
}

```

```

u8 r3[4] = {state->vec8u[3], state->vec8u[7], state->vec8u[11], stat

    //shift r1 1 position to the left
u8 tmp[4]{};
tmp[0] = r1[1];
tmp[1] = r1[2];
tmp[2] = r1[3];
tmp[3] = r1[0];
memcpy(r1, tmp, sizeof(u8) * 4);

    //shift r2 2 positions to the left
tmp[0] = r2[2];
tmp[1] = r2[3];
tmp[2] = r2[0];
tmp[3] = r2[1];
memcpy(r2, tmp, sizeof(u8) * 4);

    //shift r3 3 positions to the left
tmp[0] = r3[3];
tmp[1] = r3[0];
tmp[2] = r3[1];
tmp[3] = r3[2];
memcpy(r3, tmp, sizeof(u8) * 4);

for(int i = 0; i < 16; i+=4)
{
    state->vec8u[i] = r0[i/4];
    state->vec8u[i+1] = r1[i/4];
    state->vec8u[i+2] = r2[i/4];
    state->vec8u[i+3] = r3[i/4];
}
}

```

## 2.6. MixColumns

```

void GMixColumn(u8* in)
{
    // multiplication matrix
    // {2, 3, 1, 1,
    //  1, 2, 3, 1,
    //  1, 1, 2, 3,
    //  3, 1, 1, 2}

    u8 tmp[4]{};

```

```

        tmp[0] = GMul2[in[0]] ^ GMul3[in[1]] ^ in[2]           ^ in[3];
        tmp[1] = in[0]           ^ GMul2[in[1]] ^ GMul3[in[2]] ^ in[3];
        tmp[2] = in[0]           ^ in[1]           ^ GMul2[in[2]] ^ GMul3[in[3]];
        tmp[3] = GMul3[in[0]] ^ in[1]           ^ in[2]           ^ GMul2[in[3]];

        memcpy(in, tmp, sizeof(u8) * 4);
    }

void mixColumns(U128* state)
{
    for(uint i = 0; i < 16; i+=4)
        GMixColumn(&state->vec8u[i]);
}

```

## 2.7. Decifrador

O decifrador possui funções similares ao cifrador, mas que invertem as operações realizadas pelo cifrador.

```

void aes128Inv(U128* in, U128* out, U128* key, uint rounds)
{
    uint numKeyBlocks = 11;
    if(rounds > 10)
        numKeyBlocks = rounds - 10 + 1;

    std::vector<U128> expKey(numKeyBlocks, {0});
    expandKey(key->vec8u, expKey[0].vec8u, rounds);

    U128 state = *in;

    //initial round:
    addRoundKey(&state, &expKey[10]);

    //intermediate rounds:
    for(uint i = rounds; i >= 1; i--)
    {
        invShiftRows(&state);
        invSubBytes(&state);
        addRoundKey(&state, &expKey[i]);
        invMixColumns(&state);
    }

    //final round
    invShiftRows(&state);
    invSubBytes(&state);
    addRoundKey(&state, &expKey[0]);
}

```

```
    *out = state;  
}
```

## 2.8. Modo de Operação CTR

```
void aes128_ctr_enc(U128* in,  
                     U128* out,  
                     u64 nonce,  
                     uint sizeBytes,  
                     U128* key,  
                     uint rounds)  
{  
    uint n;  
    (sizeBytes % 16 == 0) ? n = sizeBytes/16 : n = sizeBytes/16 + 1;  
  
    std::vector<U128> keystream(n, {0});  
    for(uint i = 0; i < n; i++)  
    {  
        U128 block{};  
        block.vec64u[0] = nonce;  
        block.vec64u[1] = i;  
  
        aes128(&block, &keystream[i], key, rounds);  
    }  
  
    for(uint i = 0; i < n; i++)  
    {  
        for(uint byte = 0; byte < 16; byte++)  
        {  
            out[i].vec8u[byte] =  
                in[i].vec8u[byte] ^ keystream[i].vec8u[byte];  
        }  
    }  
}
```

## 3. Testes

Todos os testes realizados abaixo estão disponíveis no arquivo `tests.cpp` presente no repositório do GitHub. Os arquivos de entrada para os testes estão localizados na pasta `testfiles` e devem ser colocados na mesma pasta do binário do programa.

### 3.1. Cifrar e Decifrar um Arquivo

O objetivo principal deste teste é checar a corretude da cifração e decifração. Para isso, um arquivo com o livro *Franskenstein* em UTF-8 foi cifrado e decifrado para realizar o teste, resultando em uma operação bem sucedida.

### **3.2. Selfie**

O objetivo principal deste teste é observar o efeito do número de rodadas principais do algoritmo na qualidade da randomização dos dados cifrados. Para isso, uma selfie foi cifrada no modo de operação CTR utilizando 1, 5, 9 e 13 rodadas do AES, com os resultados obtidos apresentados abaixo. É possível observar que a foto cifrada com 1 rodada ainda é distinguível como uma selfie, mas a partir de 5 rounds não conseguimos mais distinguir o conteúdo da mesma.



**Figura 1. Selfie usada como imagem de teste**



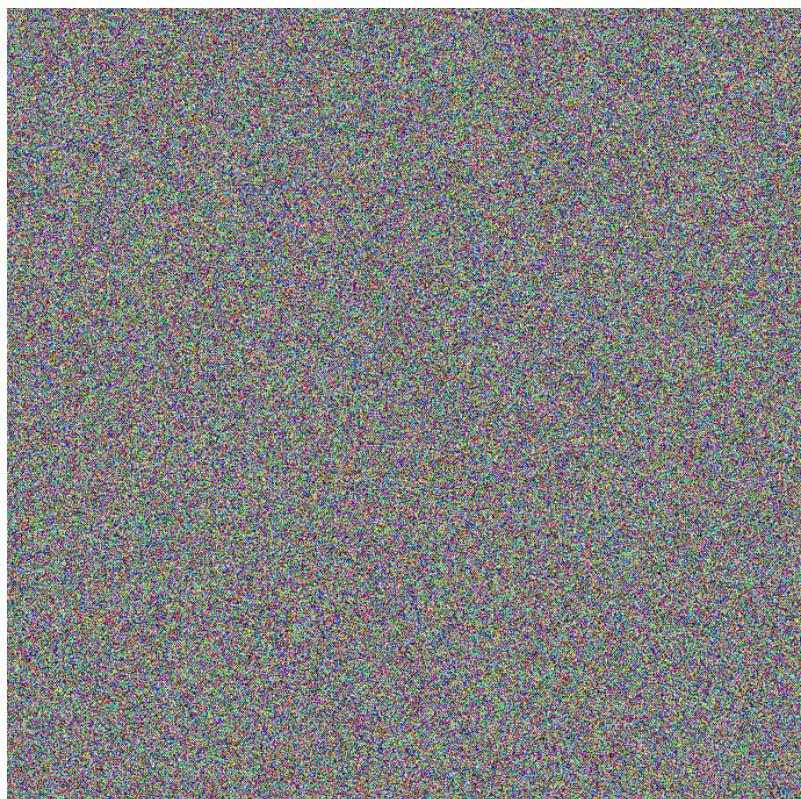
**Figura 2. Selfie cifrada em modo CTR com 1 rodada do AES**



**Figura 3. Selfie cifrada em modo CTR com 5 rodadas do AES**



**Figura 4.** Selfie cifrada em modo CTR com 9 rodadas do AES



**Figura 5.** Selfie cifrada em modo CTR com 13 rodadas do AES