Advanced Encryption Standard(AES)

AES é um algoritmo de criptografia que converte os dados em um formato ilegível para quem não tem a chave. Foi desenvolvido pelo *National Institute of Standarts and Technoloogy (NIST)* em 2001. Ele funciona com diferentes tamanhos de chave: 128, 192 ou 256 bits.

É uma cifra de blocos, o que significa que os dados são criptografados em pedaços de 128 bits por vez. Ele pega 128 bits (ou 16 bytes), processa esse bloco e gera 128 bits cifrados com base em uma lógica de substituição e permutação *Substitution-Permutation Network (SPN)*.

Blocos e bytes

Apesar de falarmos em "bits", o AES trabalha internamente com bytes. Como 1 byte = 8 bits, 128 bits = 16 bytes.

Cada bloco de 16 bytes é organizado como uma matriz 4x4:

```
[ b0  b4  b8  b12 ]
[ b1  b5  b9  b13 ]
[ b2  b6  b10 b14 ]
[ b3  b7  b11 b15 ]
```

Essa estrutura facilita as operações internas do algoritmo.

Número de Rodadas

O AES executa várias "rodadas" de transformação nos dados, dependendo do tamanho da chave:

Tamanho da chave	Número de rodadas
128 bits	10
192 bits	12
256 bits	14

Criação das chaves de rodada

Cada rodada precisa de uma chave diferente. Mas não precisamos digitar todas. O próprio AES gera essas chaves automaticamente usando um algoritmo chamado Key

Expansion (ou Agendamento de Chaves). Ele pega a chave original (ex: 256 bits) e deriva várias chaves a partir dela.

No código, usei uma chave fixa: `"minha-chave-secreta"`, que é transformada em 256 bits usando SHA-256. Depois, essa chave derivada é usada como base para o processo.

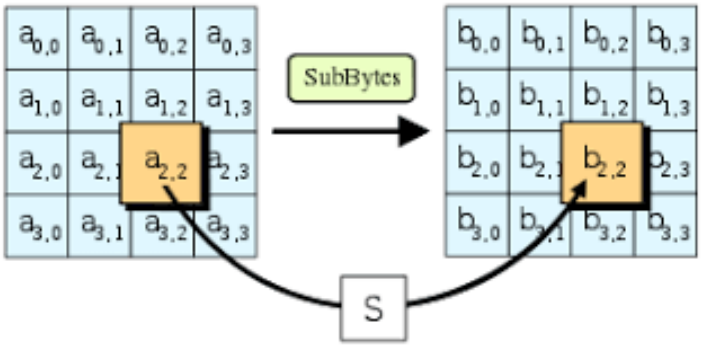
Etapas da criptografia

Cada rodada da criptografia no AES é composta por 4 etapas principais:

1. SubBytes
2. ShiftRows
3. MixColumns
4. AddRoundKey

Etapa 1: SubBytes

Faz substituição de cada byte da matriz por outro byte. Isso é feito usando uma tabela de consulta chamada S-box.



Essa substituição é feita de forma que:

- Nenhum byte seja substituído por ele mesmo
- Nenhum byte seja trocado por seu inverso

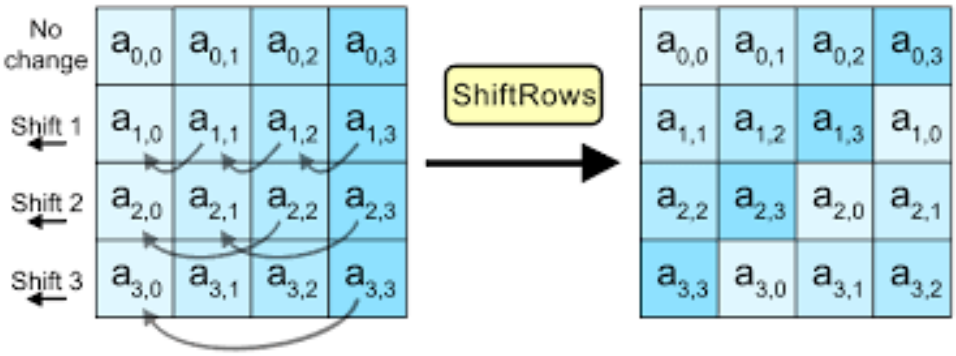
O objetivo é confundir os dados, tornando padrões difíceis de serem detectados.

Etapa 2: ShiftRows

Aqui entra a permutação. As linhas da matriz são deslocadas para a esquerda:

- Linha 1 → permanece igual
- Linha 2 → 1 deslocamento
- Linha 3 → 2 deslocamentos
- Linha 4 → 3 deslocamentos

Exemplo:

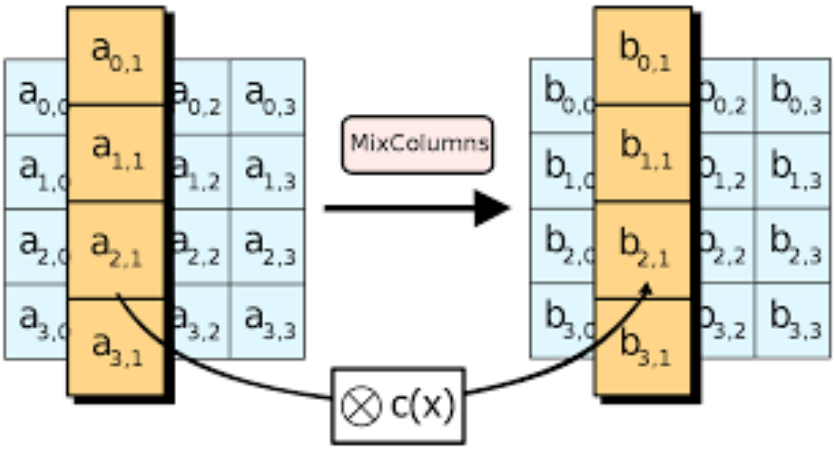


Etapa 3: MixColumns

Aqui é feita uma multiplicação de matriz em cada coluna da matriz 4x4. Cada coluna de 4 bytes é misturada usando uma matriz fixa.

Exemplo:

$$\begin{bmatrix} c0 \\ c1 \\ c2 \\ c3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} * \begin{bmatrix} b0 \\ b1 \\ b2 \\ b3 \end{bmatrix}$$

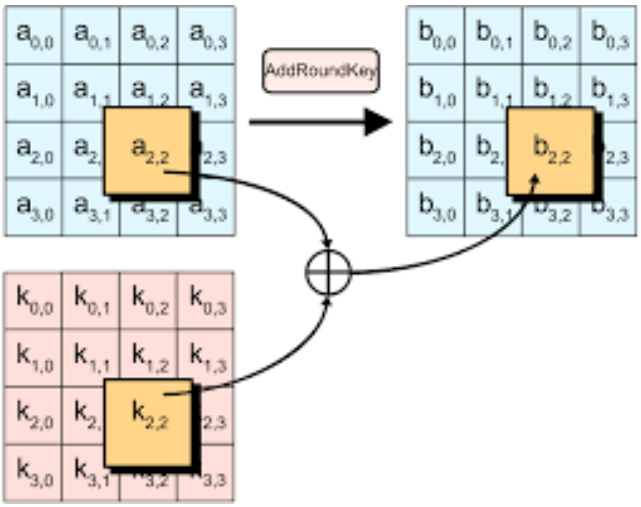


Isso embaralha os bytes entre si, aumentando a **difusão** (propagação das alterações nos bits).

Essa etapa **não acontece na última rodada**.

Etapa 4: AddRoundKey

O resultado da etapa anterior é misturado com a **chave da rodada** usando a operação XOR.



A operação XOR compara dois bits e retorna:

Bit A	Bit B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Isso reforça a segurança porque insere a "senha" no processo de forma definitiva. Sem a chave correta, os dados ficam irreconhecíveis.

Descriptografia

A descriptografia faz o **processo reverso** da criptografia:

1. **AddRoundKey**
2. **MixColumns Inverso**
3. **ShiftRows (inverso)**
4. **SubBytes (inverso)**

Cada etapa tem sua versão "inversa", o que garante que os dados originais sejam recuperados com precisão.

MixColumns Inverso

Parecido com o MixColumns normal, mas usa uma matriz diferente:

$$\begin{bmatrix} b0 \\ b1 \\ b2 \\ b3 \end{bmatrix} = \begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix} * \begin{bmatrix} c0 \\ c1 \\ c2 \\ c3 \end{bmatrix}$$

SubBytes Inverso

Usa a **S-box inversa** para desfazer a substituição feita antes. Assim, cada byte volta ao valor original.

Aplicações do AES

O AES é muito usado na prática, por ser rápido, seguro e suportado por hardware.

- **Wi-Fi (WPA2/WPA3)**: protege sua internet
- **Bancos de dados**: dados criptografados contra roubo
- **VPNs**: protege sua conexão com a internet
- **Armazenamento seguro**: HDs, SSDs, USBs e nuvem
- **Senhas**: versões criptografadas em vez de texto simples
- **Mensagens seguras**: chats, e-mails, chamadas

Modo de operação CBC

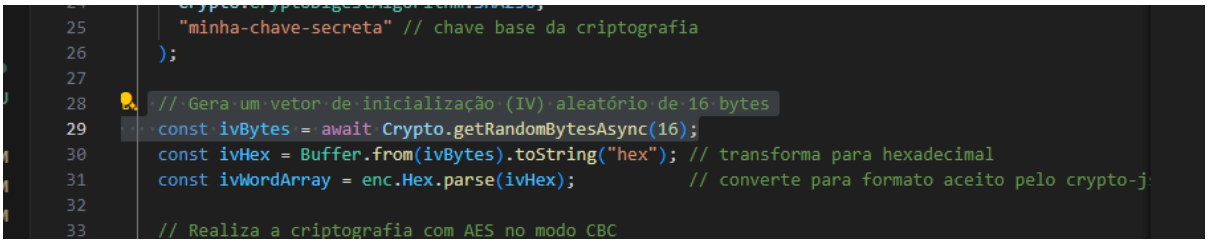
O AES puro só cifra **1 bloco de 128 bits**. Mas na prática, os dados são maiores. Por isso utilizei o *CBC (Cipher Block Chaining)*.

No modo CBC:

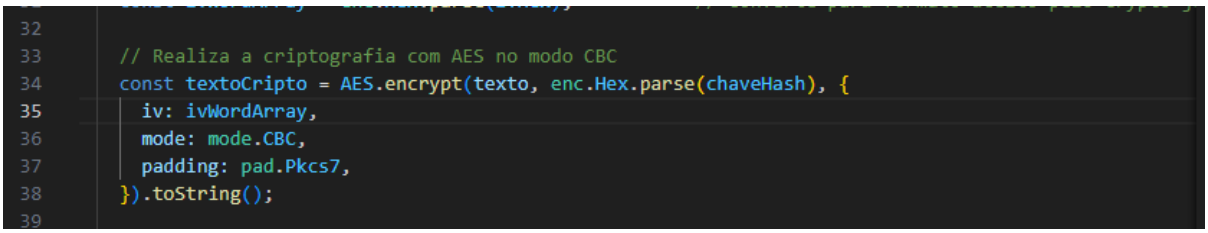
- O **primeiro bloco** é criptografado normalmente
- Os **blocos seguintes** são **XOR** com o bloco anterior **já criptografado**
- Um vetor inicial (chamado **IV**) é usado no primeiro XOR, para garantir aleatoriedade

No código:

- Geramos um IV aleatório com `Crypto.getRandomBytesAsync(16)`
- O IV é convertido em Hex e armazenado



- A função `AES.encrypt()` usa o modo CBC com PKCS7 para preencher os blocos



- Na descriptografia, usamos a mesma `chave` e o mesmo `IV` para reverter o processo

```
// Função que descriptografa o texto
function descriptografar() {
  // Verifica se os dados necessários existem
  if (!resultado || !chaveSalva || !ivSalvo) return;

  const ivWordArray = enc.Hex.parse(ivSalvo); // reconverte o IV salvo

  // Descriptografa o texto
  const bytes = AES.decrypt(resultado, enc.Hex.parse(chaveSalva), {
    iv: ivWordArray,
    mode: mode.CBC,
    padding: pad.Pkcs7,
  });

  // Converte os dados descriptografados para texto legível
  const texto = bytes.toString(enc.Utf8);
  setTextoOriginal(texto); // salva o texto original
}
```

Prática

A prática foi desenvolvida em Node.js, em um app mobile com React Native, utilizando a biblioteca Expo-Crypto, que permite gerar hash de dados de maneira equivalente à CryptoAPI principal do Node.js.

```
// Função que criptografa o texto
async function criptografar(texto) {
  // Gera o hash da chave usando SHA-256
  const chaveHash = await Crypto.digestStringAsync(
    Crypto.CryptoDigestAlgorithm.SHA256,
    "minha-chave-secreta" // chave base da criptografia
  );

  // Gera um vetor de inicialização (IV) aleatório de 16 bytes
  const ivBytes = await Crypto.getRandomBytesAsync(16);
  const ivHex = Buffer.from(ivBytes).toString("hex"); // transforma para hexadecimal
  const ivWordArray = enc.Hex.parse(ivHex);           // converte para formato aceito pelo crypto-

  // Realiza a criptografia com AES no modo CBC
  const textoCripto = AES.encrypt(texto, enc.Hex.parse(chaveHash), {
    iv: ivWordArray,
    mode: mode.CBC,
    padding: pad.Pkcs7,
  }).toString();

  // Salva o texto criptografado, a chave e o IV para posterior descriptografia
  setResultado(textoCripto);
  setChaveSalva(chaveHash);
  setIvSalvo(ivHex);
}
```

E, para otimizar a exibição dos resultados, concentrei na mesma página o texto criptografado, transformado em hexadecimal, e um outro botão ‘Descriptografar’, que chama a função ‘descriptografar’ já mencionada.