Nome: Joao Otavio Rodrigues de Castro Manieri
Matricula: 12021BSI263

Relatório: Avaliação do Efeito Avalanche no AES-128 (Modo ECB)

Objetivo: O objetivo deste experimento foi avaliar o efeito avalanche no algoritmo AES-128 operando no modo ECB, ou seja, observar como uma pequena alteração no texto claro afeta significativamente o texto cifrado.

Procedimentos:

1. **Geração dos textos claros (M1 a M10):** Foram criados 10 arquivos texto contendo exatamente 16 bytes cada. Exemplo: "abcdefghijklmno1", "abcdefghijklmno2", etc.
2. **Geração da chave AES de 128 bits:** Chave utilizada: `a3f536b587c72ad2bb2f18619d768ec9`
3. **Geração dos textos modificados (M1_p a M10_p):** Foi alterado 1 byte no final de cada um dos textos claros para gerar versões ligeiramente diferentes.
4. **Cifragem:** Todos os arquivos foram cifrados utilizando o OpenSSL com os parâmetros `-aes-128-ecb -nosalt -nopad`. Geraram-se os arquivos `C1` a `C10` e suas versões modificadas `C1_p` a `C10_p`.
5. **Comparacão e análise:** Um script em Python foi utilizado para comparar byte a byte os textos cifrados e contar quantos bits eram diferentes entre cada par.

Resultados:

C1 x C1_p → 66 bits diferentes (51.56%)
C2 x C2_p → 60 bits diferentes (46.88%)
C3 x C3_p → 59 bits diferentes (46.09%)
C4 x C4_p → 58 bits diferentes (45.31%)
C5 x C5_p → 63 bits diferentes (49.22%)
C6 x C6_p → 68 bits diferentes (53.12%)
C7 x C7_p → 69 bits diferentes (53.91%)
C8 x C8_p → 67 bits diferentes (52.34%)
C9 x C9_p → 72 bits diferentes (56.25%)
C10 x C10_p → 70 bits diferentes (54.60%)

Conclusão: O experimento demonstrou claramente o efeito avalanche no AES. Mesmo com uma única alteração de byte no texto claro, mais de 45% dos bits no texto cifrado foram alterados em todos os testes. Esse comportamento confirma que o AES é altamente sensível a mudanças de entrada, uma característica desejável em algoritmos criptográficos modernos, pois garante imprevisibilidade e segurança contra ataques
Fotos dos comandos

```
PS C:\Users\jotam\Downloads\openssl_test\ex13> python .\criar_textos_corrigido.py
✅ Arquivos criados com exatamente 16 bytes.
PS C:\Users\jotam\Downloads\openssl_test\ex13> python .\test.py
M1.txt: 16 bytes | M1_p.txt: 16 bytes
M2.txt: 16 bytes | M2_p.txt: 16 bytes
M3.txt: 16 bytes | M3_p.txt: 16 bytes
M4.txt: 16 bytes | M4_p.txt: 16 bytes
M5.txt: 16 bytes | M5_p.txt: 16 bytes
M7.txt: 16 bytes | M7_p.txt: 16 bytes
M8.txt: 16 bytes | M8_p.txt: 16 bytes
M9.txt: 16 bytes | M9_p.txt: 16 bytes
M10.txt: 16 bytes | M10_p.txt: 16 bytes
```

```
PS C:\Users\jotam\Downloads\openssl_test\ex13> python .\cifrar_arquivos_aes.py
[OK] M1.txt e M1_p.txt cifrados.
[OK] M2.txt e M2_p.txt cifrados.
[OK] M3.txt e M3_p.txt cifrados.
[OK] M4.txt e M4_p.txt cifrados.
[OK] M5.txt e M5_p.txt cifrados.
[OK] M6.txt e M6_p.txt cifrados.
[OK] M7.txt e M7_p.txt cifrados.
[OK] M8.txt e M8_p.txt cifrados.
[OK] M9.txt e M9_p.txt cifrados.
[OK] M10.txt e M10_p.txt cifrados.
```

```
PS C:\Users\jotam\Downloads\openssl_test\ex13> python .\comparar_efeito_avalanche.py
Comparando os arquivos cifrados (efeito avalanche):

C1 x C1_p → 66 bits diferentes (51.56%)
C2 x C2_p → 60 bits diferentes (46.88%)
C3 x C3_p → 59 bits diferentes (46.09%)
C4 x C4_p → 58 bits diferentes (45.31%)
C5 x C5_p → 63 bits diferentes (49.22%)
C6 x C6_p → 68 bits diferentes (53.12%)
C7 x C7_p → 69 bits diferentes (53.91%)
C8 x C8_p → 67 bits diferentes (52.34%)
C9 x C9_p → 72 bits diferentes (56.25%)
C10 x C10_p → 70 bits diferentes (54.69%)
PS C:\Users\jotam\Downloads\openssl_test\ex13>
```

Arquivos utilizados

criar_textos_corrigido.py:

```
from pathlib import Path

for i in range(1, 11):
    Path(f"M{i}.txt").unlink(missing_ok=True)
    Path(f"M{i}_p.txt").unlink(missing_ok=True)

original = [
    "abcdefghijklmno1",
    "abcdefghijklmno2",
    "abcdefghijklmno3",
```

```
    "abcdefghijklmno4",
    "abcdefghijklmno5",
    "abcdefghijklmno6",
    "abcdefghijklmno7",
    "abcdefghijklmno8",
    "abcdefghijklmno9",
    "abcdefghijklmnoA"
]

modificado = [
    "abcdefghijklmnoX",
    "abcdefghijklmnoY",
    "abcdefghijklmnoZ",
    "abcdefghijklmnoW",
    "abcdefghijklmnoQ",
    "abcdefghijklmnoE",
    "abcdefghijklmnoR",
    "abcdefghijklmnoT",
    "abcdefghijklmnoU",
    "abcdefghijklmnoI"
]

for i in range(10):
    with open(f"M{i+1}.txt", "wb") as f:
        f.write(original[i].encode("utf-8"))
    with open(f"M{i+1}_p.txt", "wb") as f:
        f.write(modificado[i].encode("utf-8"))

print("✅ Arquivos criados com exatamente 16 bytes.")
```

cifrar_arquivos_aes.py:

```
import subprocess
from pathlib import Path



chave = "a3f536b587c72ad2bb2f18619d768ec9"

for i in range(1, 11):
    arquivo_normal = f"M{i}.txt"
    arquivo_modificado = f"M{i}_p.txt"
```

```
    if not Path(arquivo_normal).exists():
        print(f"[ERRO] {arquivo_normal} não encontrado.")
        continue
    if not Path(arquivo_modificado).exists():
        print(f"[ERRO] {arquivo_modificado} não encontrado.")
        continue

    saida_normal = f"C{i}.bin"
    saida_modificado = f"C{i}_p.bin"

    subprocess.run([
        "openssl", "enc", "-aes-128-ecb", "-nosalt", "-nopad",
        "-in", arquivo_normal,
        "-out", saida_normal,
        "-K", chave
    ])

    subprocess.run([
        "openssl", "enc", "-aes-128-ecb", "-nosalt", "-nopad",
        "-in", arquivo_modificado,
        "-out", saida_modificado,
        "-K", chave
    ])

    print(f"[OK] M{i}.txt e M{i}_p.txt cifrados.")
```

comparar_efeito_avalanche.py:

```
import os

def contar_bits_diferentes(bin1, bin2):
    dif = 0
    for b1, b2 in zip(bin1, bin2):
        xor = b1 ^ b2
        dif += bin(xor).count('1')
    return dif

print("Comparando os arquivos cifrados (efeito avalanche):\n")

for i in range(1, 11):
    nome1 = f"C{i}.bin"
```

```
    nome2 = f"C{i}_p.bin"
    if not os.path.exists(nome1) or not os.path.exists(nome2):
        print(f"[!] Arquivos {nome1} ou {nome2} não encontrados.")
        continue

    with open(nome1, "rb") as f1, open(nome2, "rb") as f2:
        c1 = f1.read()
        c2 = f2.read()
        bits_dif = contar_bits_diferentes(c1, c2)
        taxa = bits_dif / 128 * 100
        print(f"C{i} x C{i}_p → {bits_dif} bits diferentes
({taxa:.2f}%)")
```