1. Descrição do Trabalho

Este documento descreve a implementação de um sistema criptográfico baseado em funções de geração de chave (GEN), criptografia (ENC) e descriptografia (DEC) utilizando **Python 3.10**. O objetivo foi criar um algoritmo que encriptasse/descriptasse um texto, mas que também apresentasse alta qualidade de **difusão** e **confusão**.

2. Implementação das Funções

2.1 Geração de Chave (GEN)

A função `GEN(seed)` utiliza a *seed* fornecida para inicializar o `random` do python (`random.seed`). Em seguida ele gera uma lista binária de tamanho `4 x len(seed)` para ser utilizada como chave do nosso processo de cifragem. * **Segurança**: O uso do estado interno do gerador garante que a mesma semente sempre produza a mesma chave, mas que uma semente minimamente diferente gere uma chave completamente distinta (Confusão).

2.2 Criptografia (ENC) com Modo CBC

Para obter melhores resultados de **difusão** e **confusão**, foi implementado o modo **Cipher Block Chaining (CBC)**. * **Processo**: Cada bloco da mensagem passa por uma operação XOR com o bloco cifrado anterior (ou um Vetor de Inicialização - IV) antes de ser cifrado com a chave. * **Vetor de Inicialização (IV)**: Utilizou-se a biblioteca `secrets` para gerar um IV aleatório, garantindo que a mesma mensagem cifrada duas vezes com a mesma chave resulte em cifras diferentes.

2.3 Descriptografia (DEC)

A função `DEC` reverte o processo de encriptação, extraindo o IV do início da cifra e realizando as operações XOR inversas para recuperar a mensagem original.

---

3. Código Implementado

```
import random, secrets

def _xor_bytes(b1, b2):
    return [a ^ b for a, b in zip(b1, b2)]

def GEN(seed: list[int]) -> list[int]:
    random.seed(str(seed))
    key_len = 4 * len(seed)
    return [random.randint(0, 1) for _ in range(key_len)]
```

```python
def ENC(K: list[int], M: list[int]) -> list[int]:
    BLOCK_SIZE = len(K)
    IV = [secrets.randbelow(2) for _ in range(BLOCK_SIZE)] # Versão para bits
    C = []
    prev_block = IV
    for i in range(0, len(M), BLOCK_SIZE):
        block = M[i:i + BLOCK_SIZE]
        mixed = _xor_bytes(prev_block, block)
        cypher = _xor_bytes(K, mixed)
        C.extend(cypher)
        prev_block = cypher
    return IV + C

def DEC(K: list[int], C: list[int]) -> list[int]:
    BLOCK_SIZE = len(K)
    IV = C[:BLOCK_SIZE]
    cifra_corpo = C[BLOCK_SIZE:]
    M = []
    prev_block = IV
    for i in range(0, len(cifra_corpo), BLOCK_SIZE):
        block = cifra_corpo[i:i + BLOCK_SIZE]
        mixed = _xor_bytes(K, block)
        msg_block = _xor_bytes(mixed, prev_block)
        M.extend(msg_block)
        prev_block = block
    return M
```

---

4. Resultados dos Testes

Os testes foram realizados seguindo os critérios de avaliação:

Teste	Resultado
Tempo de Execução	0.29116s (3000 ciclos)
Chaves Equivalentes	0 Colisões
Teste de Difusão	~63.81 bits alterados
Teste de Confusão	~63.25 bits alterados

Conclusão

A implementação obteve resultados interessantes. Com uma média de ~64 bits alterados utilizando de um estrutura em blocos, o algoritmo mostra que pequenas mudanças na entrada (tanto na mensagem quanto na semente) tornam a saída imprevisível, cumprindo os requisitos de qualidade.