CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA
CELSO SUCKOW DA FONSECA – CEFET/RJ
UNIDADE PETRÓPOLIS

ENGENHARIA DE COMPUTAÇÃO

Henrique Brasil
Kalebe dos Santos
Vitor Vasconcellos

DESENVOLVIMENTO DE UMA BLOCKCHAIN EM C

PETRÓPOLIS
2025

RESUMO

Este trabalho apresenta o desenvolvimento de uma blockchain funcional em C, implementando conceitos como Proof of Work (PoW), Merkle Tree, segurança de dados e um sistema interativo para manipulação da cadeia de blocos. A proposta do projeto é demonstrar, na prática, como funciona uma cadeia de blocos descentralizada e segura, garantindo a integridade dos dados através da criptografia e validação distribuída. A implementação inclui desde a criação da estrutura de blocos até a mineração e verificação da blockchain.

Palavras-chave: blockchain, Proof of Work, Merkle Tree, mineração, segurança.

Sumário

Capítulo 1

INTRODUÇÃO

Blockchain é uma tecnologia que permite o armazenamento seguro e imutável de dados em uma estrutura descentralizada. Este projeto explora os principais conceitos técnicos por trás do funcionamento de uma blockchain, como:

- Estruturas de dados para representação de blocos;
- Implementação de algoritmos de mineração baseados em Proof of Work (PoW);
- Criação de uma Merkle Tree para validação de transações;
- Menu interativo para manipulação da blockchain;
- Simulação de ataques para análise da segurança.

Capítulo 2

DESENVOLVIMENTO

Este capítulo apresenta o desenvolvimento da blockchain, abordando sua estrutura de dados, o processo de mineração utilizando o algoritmo Proof of Work (PoW), a implementação da Merkle Tree para validação das transações e as funções para validação e persistência dos blocos. Além disso, será apresentado um menu interativo para permitir a interação do usuário com o sistema.

2.1 Estrutura dos Dados

A blockchain é uma estrutura encadeada de blocos, onde cada bloco contém informações essenciais para garantir a integridade e segurança dos dados armazenados. Cada bloco possui um identificador único (hash), que é gerado com base nas informações do próprio bloco e do bloco anterior, formando uma cadeia de blocos imutável.

Os blocos possuem os seguintes atributos:

- **Índice**: Posição do bloco na cadeia.
- **Hash do bloco**: Código gerado a partir dos dados do bloco, garantindo sua autenticidade.
- **Hash do bloco anterior**: Referência ao hash do bloco anterior, garantindo o encadeamento da cadeia.
- **Nonce**: Valor numérico ajustado durante a mineração para atender ao critério de dificuldade do PoW.
- **Raiz Merkle**: Hash que representa todas as transações do bloco.
- **Lista de transações**: Conjunto de operações registradas no bloco.
- **Timestamp**: Data e hora da criação do bloco.

A implementação da estrutura de um bloco em C é dada por:

Listing 2.1: Estrutura de um bloco na blockchain

```
typedef struct {
    int index;
    char hash[65];
    char hash_anterior[65];
    unsigned int nonce;
    char raiz_merkle[65];
    char **transacoes;
    int cont_transacoes;
    time_t timestamp;
} Block;
```

A blockchain, por sua vez, é composta por um conjunto de blocos armazenados de forma dinâmica:

Listing 2.2: Estrutura da blockchain

```
typedef struct {
    Block **blocks;
    int cont_block;
} Blockchain;
```

2.2 Mineração e Proof of Work

A mineração consiste no processo de encontrar um **nonce** que, ao ser combinado com os dados do bloco e submetido a uma função hash (SHA-256), gere um hash que atenda a um critério específico de dificuldade. Esse critério geralmente exige que o hash gerado comece com um determinado número de zeros à esquerda.

O processo de mineração pode ser descrito da seguinte forma: 1. Inicializa-se o nonce com um valor arbitrário (geralmente 0). 2. Calcula-se o hash do bloco utilizando a função `SHA-256`. 3. Se o hash gerado satisfaz a condição de dificuldade, a mineração é concluída. 4. Caso contrário, o nonce é incrementado e o processo é repetido até que um hash válido seja encontrado.

A seguir, a implementação da função de mineração:

Listing 2.3: Função de mineração

```
void Minerador(Block *block, int dificuldade) {
    char target[65] = {0};
    memset(target, '0', dificuldade);
    target[dificuldade] = '\0';
    do {
        block->nonce++;
        calcula_blockHash(block);
    } while (strncmp(block->hash, target, dificuldade) != 0);
}
```

Este método assegura que a criação de novos blocos requer um esforço computacional significativo, tornando a blockchain resistente a ataques de falsificação e modificações indevidas.

2.3 Merkle Tree

A Merkle Tree é uma estrutura de dados utilizada para consolidar e verificar a integridade das transações de um bloco. Seu funcionamento baseia-se na criação de uma árvore binária onde cada nó folha contém o hash de uma transação e os nós superiores são formados a partir da combinação dos hashes inferiores. O nó raiz da árvore, chamado de **raiz Merkle**, representa um resumo criptográfico de todas as transações do bloco.

A implementação do cálculo da raiz da Merkle Tree é feita da seguinte maneira:

Listing 2.4: Função para calcular a raiz de Merkle

```
char *calculaRaizMerkle(char **transacoes, int
   cont_transacoes) {
    if (cont_transacoes == 0) return strdup("");
    char *hashes = (char *)malloc(cont_transacoes * sizeof(
       char *));
    for (int i = 0; i < cont_transacoes; i++) {
        hashes[i] = (char *)malloc(65);
        calculaSHA256(transacoes[i], hashes[i]);
    }
    while (cont_transacoes > 1) {
        int novo_cont = (cont_transacoes + 1) / 2;
        for (int i = 0; i < novo_cont; i++) {
            char combined[130] = {0};
            strncpy(combined, hashes[i * 2], 65);
            if (i * 2 + 1 < cont_transacoes) {
                strncat(combined, hashes[i * 2 + 1], 65);
            }
            calculaSHA256(combined, hashes[i]);
        }
        cont_transacoes = novo_cont;
    }
    char *raizMerkle = strdup(hashes[0]);
    free(hashes);
    return raizMerkle;
}
```

Essa estrutura permite que qualquer alteração em uma transação seja detectada, pois modificaria a raiz Merkle do bloco.

2.4 Validação da Blockchain

A validação da blockchain assegura que os blocos armazenados mantêm sua integridade e que não houve adulteração de informações. O processo de validação envolve a verificação dos seguintes critérios:

- O hash armazenado em cada bloco deve ser recalculado e comparado com o valor esperado.
- O **hash do bloco anterior** deve corresponder ao hash efetivo do bloco anterior na cadeia.
- A dificuldade do Proof of Work deve ser respeitada, ou seja, o hash do bloco deve atender ao critério de dificuldade estipulado.

A função de validação é implementada conforme mostrado abaixo:

Listing 2.5: Função de validação da blockchain

```
int validaBlockchain(Blockchain *blockchain) {
    for (int i = 1; i < blockchain->cont_block; i++) {
        if (strncmp(blockchain->blocks[i]->hash_anterior,
           blockchain->blocks[i - 1]->hash, 65) != 0) {
            printf("Erro: Bloco %d invÃ¡lido! Hash anterior
               nÃ£o bate com o hash do bloco %d.\n", i, i -
               1);
            return 0;
        }
    }
    printf("Blockchain vÃ¡lida!\n");
    return 1;
}
```

2.5 Persistência da Blockchain

Para garantir que a blockchain possa ser armazenada e carregada posteriormente, implementamos funções para gravar e recuperar os dados em um arquivo.

2.5.1 Salvar Blockchain em Arquivo

A blockchain pode ser salva em um arquivo binário para posterior recuperação.

Listing 2.6: Função para salvar a blockchain em um arquivo

```
void salvaBlockchain(Blockchain *blockchain, const char *
   arquivo) {
    FILE *fp = fopen(arquivo, "w");
    if (!fp) {
        printf("Erro ao abrir arquivo para escrita!\n");
        return;
    }
    fwrite(blockchain, sizeof(Blockchain), 1, fp);
    fclose(fp);
}
```

2.5.2 Carregar Blockchain de um Arquivo

Para restaurar a blockchain de um arquivo salvo:

Listing 2.7: Função para carregar a blockchain de um arquivo

```
Blockchain *carregaBlockchain(const char *arquivo) {
    FILE *fp = fopen(arquivo, "r");
    if (!fp) {
        printf("Arquivo de blockchain nÃ£o encontrado.
           Criando nova blockchain...\n");
        return inicializaBlockchain();
    }
```

```
    Blockchain *blockchain = (Blockchain *)malloc(sizeof(
        Blockchain));
    fread(blockchain, sizeof(Blockchain), 1, fp);
    fclose(fp);
    return blockchain;
}
```

2.6 Menu Interativo

O menu interativo permite realizar operações na blockchain:

Listing 2.8: Menu interativo da blockchain

```
void menu(Blockchain *blockchain) {
    int opcao;
    do {
        printf("\n--- MENU ---\n");
        printf("1. Inserir transacao\n");
        printf("2. Minerar bloco\n");
        printf("3. Exibir blockchain\n");
        printf("4. Verificar transacao\n");
        printf("5. Simular ataque\n");
        printf("6. Validar blockchain\n");
        printf("7. Sair\n");
        printf("Escolha uma opcao: ");
        scanf("%d", &opcao);
        ...
    } while (opcao != 7);
}
```

Capítulo 3

CONCLUSÃO

Este trabalho demonstrou a implementação de uma blockchain funcional em C, desde a estruturação dos blocos até a mineração e validação das transações. Além disso, foram abordadas estratégias de segurança para impedir ataques e manter a integridade da cadeia de blocos.