ALU Medical Blockchain System

Documentation

Title: Blockchain-Based Medical Records System for ALU Clinic
Student Name: King Chukwumere
ALU ID: 817957251
Course: Software Engineering Low-Level Specialization
Date: June 8, 2024

---

1. Introduction and Problem Background

1.1 Problem Statement

The ALU Clinic requires a secure and immutable system for managing patient medical records. Traditional medical record systems face several challenges: - Data tampering and unauthorized modifications - Lack of transparency in record access - Difficulty in maintaining audit trails - Centralized storage vulnerabilities

1.2 Solution Overview

The ALU Medical Blockchain System addresses these challenges by implementing: - A blockchain-based architecture for immutable record storage - Proof of Work consensus mechanism for security - Command-line interface for easy interaction - SHA-256 hashing for data integrity - **Persistence and backup/restore for data reliability**

1.3 Objectives

1. Create a secure medical records system
2. Implement blockchain technology for data immutability
3. Provide a user-friendly interface
4. Ensure data integrity through cryptographic hashing
5. Enable easy verification of record authenticity
6. **Ensure blockchain data is not lost between runs (persistence)**

---

2. System Design

2.1 Data Structures

2.1.1 Block Structure

```
typedef struct Block {
    uint32_t id;                    // Block identifier
    time_t timestamp;               // Block creation time
```

```c
    Transaction transactions[MAX_TRANSACTIONS];  // Array of transactions
    int transaction_count;           // Number of transactions
    char previous_hash[HASH_SIZE + 1];  // Hash of previous block
    char hash[HASH_SIZE + 1];        // Current block hash
    uint32_t nonce;                  // Proof of work nonce
    struct Block* next;              // Pointer to next block
} Block;
```

2.1.2 Transaction Structure

```c
typedef struct {
    char patient_id[32];
    char record_type[32];  // e.g., "diagnosis", "prescription"
    char data[256];        // Medical record data
    time_t timestamp;
} Transaction;
```

2.1.3 Blockchain Structure

```c
typedef struct {
    Block* genesis;           // First block
    Block* latest;           // Most recent block
    uint32_t block_count;    // Total blocks
    int difficulty;          // Mining difficulty
} Blockchain;
```

2.2 Algorithms

2.2.1 Proof of Work The system implements a simple Proof of Work algorithm: 1. Calculate block hash using SHA-256 2. Check if hash meets difficulty requirement (leading zeros) 3. Increment nonce and repeat until valid hash found

```c
int mine_block(Blockchain* chain, Block* block) {
    while (1) {
        calculate_block_hash(block);
        if (is_valid_hash(block->hash, chain->difficulty)) {
            break;
        }
        increment_nonce(&block->nonce);
    }
    return 1;
}
```

2.2.2 Hash Calculation

```c
void calculate_block_hash(Block* block) {
    char data[1024];
```

```c
    // Combine block data
    snprintf(data, sizeof(data), "%u%ld%u%s",
        block->id,
        block->timestamp,
        block->nonce,
        block->previous_hash);

    // Add transaction data
    for (int i = 0; i < block->transaction_count; i++) {
        // Append transaction data
    }

    // Calculate SHA-256 hash
    sha256(data, block->hash);
}
```

2.3 CLI Features

2.3.1 Command Structure

```c
typedef struct {
    const char* name;
    const char* description;
    int (*handler)(Blockchain* chain, int argc, char** argv);
} Command;
```

2.3.2 Available Commands

1. `add` - Add medical record
   - Usage: `add <patient_id> <record_type> <data>`
   - Example: `add P001 diagnosis "Common cold symptoms"`
2. `mine` - Mine new block
   - Usage: `mine`
   - Creates new block with pending transactions
3. `view` - View blockchain
   - Usage: `view`
   - Displays entire blockchain
4. `verify` - Verify chain integrity
   - Usage: `verify`
   - Checks block hashes and links
5. `backup` - Create a backup of the blockchain (**new**)
   - Usage: `backup`
   - Creates a timestamped backup of the blockchain files
6. `restore` - Restore blockchain from latest backup (**new**)
   - Usage: `restore`
   - Restores the blockchain from the latest backup
7. `help` - Show help

    - Usage: `help`
    - Displays available commands
8. `exit` - Exit program
    - Usage: `exit`
    - Safely terminates program

2.3.3 Persistence and Backup/Restore (new)

- The blockchain is automatically saved to disk after every session.
- On startup, the system loads the blockchain from disk if available.
- The `backup` command creates a timestamped backup of the blockchain files.
- The `restore` command restores the blockchain from the latest backup.

---

3. Sample Outputs

3.1 Adding a Medical Record

```
medblockchain> add P001 diagnosis "Patient shows symptoms of common cold"
Success: Transaction added successfully
```

3.2 Mining a Block

```
medblockchain> mine
Success: New block mined successfully
```

3.3 Viewing Blockchain

```
Blockchain Status:
Total Blocks: 2
Current Difficulty: 4
Chain Valid: Yes

Block #0
Timestamp: 2024-06-08 10:58:23
Previous Hash: 0000000000000000000000000000000000000000000000000000000000000000
Hash: 3d70886532a7cf5f362c6aba297cd909dc80841081cca3681b047680e34847e5
Nonce: 259826
Transactions: 1

Transaction #1:
  Patient ID: P001
  Type: diagnosis
  Data: Patient shows symptoms of common cold
  Timestamp: 2024-06-08 10:58:24
```

3.4 Backup and Restore (new)

```
medblockchain> backup
Success: Blockchain backup created successfully

medblockchain> restore
Success: Blockchain restored from backup successfully
```

---

4. Challenges and Solutions

4.1 Technical Challenges

4.1.1 OpenSSL Integration **Challenge:** Implementing SHA-256 hashing with OpenSSL **Solution:** Used OpenSSL's SHA-256 functions and proper library linking

4.1.2 Memory Management **Challenge:** Managing dynamic memory for blockchain structure **Solution:** Implemented proper allocation and deallocation functions

4.1.3 Input Validation **Challenge:** Ensuring data integrity and security **Solution:** Added comprehensive input validation functions

4.2 Design Challenges

4.2.1 Block Size **Challenge:** Determining optimal block size **Solution:** Implemented configurable transaction limit

4.2.2 Mining Difficulty **Challenge:** Balancing security and performance **Solution:** Implemented adjustable difficulty level

---

5. Conclusion and Future Improvements

5.1 Conclusion

The ALU Medical Blockchain System successfully implements: - Secure medical record storage - Immutable data structure - User-friendly interface - Data integrity verification - **Persistence and backup/restore for reliability**

5.2 Future Improvements

5.2.1 Technical Improvements

1. Add network support for multiple nodes
2. Enhance encryption for sensitive data

3. Implement user authentication
4. **Encrypt backups and persistent storage**

5.2.2 Feature Additions

1. Smart contracts for automated record management
2. Web interface for easier access
3. Mobile application support
4. Integration with existing medical systems

5.2.3 Security Enhancements

1. Advanced encryption algorithms
2. Role-based access control
3. Audit logging system
4. Backup and recovery mechanisms

References

1. Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System
2. OpenSSL Project. (2024). OpenSSL Documentation
3. C Programming Language Standard (C11)
4. Blockchain Technology in Healthcare: A Systematic Review