COMP4137
MiniBlockChain Project Report

Group Members:
UCHE Destiny Nnanna
TAM Kar Nam
FUNG Hok Chun
CHAN Hok Ting

April 21, 2025

1 Introduction and Contributions

The MiniBlockchain project is a simplified implementation of a blockchain system designed to demonstrate the core principles and functionalities of blockchain technology. This project includes essential components such as blocks, transactions, a mempool, a miner, and a blockchain ledger. It also incorporates features like proof-of-work, Merkle trees for transaction verification, and digital signatures for secure transactions.

The project is structured to simulate a basic blockchain network where users can create transactions, miners can validate and add blocks to the blockchain, and the integrity of the blockchain can be verified. The implementation is written in Java and provides a clear and modular design, making it an excellent educational tool for understanding blockchain concepts.

Key features of the MiniBlockchain project include:

- **Block and Blockchain Management**: Blocks store transaction data, and the blockchain maintains a sequential ledger of all blocks.
- **Proof-of-Work**: A mining mechanism ensures that blocks meet a specific difficulty target before being added to the blockchain.
- **Merkle Trees**: Used to verify the integrity of transactions within a block.

- **Digital Signatures**: Transactions are signed using RSA encryption to ensure authenticity and prevent tampering.
- **User and Transaction Management**: Users can create transactions, and a mempool temporarily stores pending transactions before they are mined into blocks.

This project serves as a foundation for exploring blockchain technology and can be extended to include more advanced features such as smart contracts, consensus algorithms, and distributed networking.

Table 1: Contributions to the MiniBlockChain Project

Group Member	**Contribution**
UCHE Destiny Nnanna	User and Transaction Management: Implementation of user creation, wallet management, and transaction generation.
TAM Kar Nam	Blockchain Core: Development of the block structure, blockchain management, and mining logic.
FUNG Hok Chun	Verification and Security: Implementation of transaction and block verification mechanisms.
CHAN Hok Ting	Merkle Tree: Integration of Merkle Tree for transaction integrity verification.

2 Related Tools/Libraries

The project relies on the following tools and libraries:

- **Java Standard Library**: For core functionalities like data structures (e.g., `HashMap`, `ArrayList`, `LinkedList`), cryptography, and I/O operations.
- **javax.crypto**: For RSA encryption and decryption using the `Cipher` class.
- **java.security**: For generating cryptographic keys (e.g., `KeyPairGenerator`, `PublicKey`, `PrivateKey`) and hashing algorithms (e.g., `MessageDigest` for SHA-256).
- **java.time**: For timestamp generation using `Instant`.
- **java.math.BigInteger**: For handling large numbers in proof-of-work calculations.

3 Pipeline Flow of the Blockchain System

The pipeline flow of the blockchain system is designed to simulate the core functionalities of a blockchain. Below is a detailed explanation of each step:

3.1 User Creation

Users are created with unique public-private key pairs and wallet addresses. This ensures that each user has a secure identity within the blockchain system.

3.2 Transaction Generation

Users create transactions by specifying the amount to transfer and the recipient's address. Each transaction is digitally signed by the sender to ensure authenticity and integrity.

3.3 Transaction Validation

Transactions are validated using cryptographic techniques. This includes verifying the sender's digital signature and checking wallet balances to ensure the transaction is legitimate and untampered.

3.4 MemPool Management

Valid transactions are added to the mempool, a temporary storage for pending transactions waiting to be included in a block.

3.5 Mining

The miner collects transactions from the mempool, validates them, and creates a new block. The miner performs proof-of-work by solving a cryptographic puzzle to find a valid nonce.

3.6 Blockchain Update

Once a block is successfully mined, it is added to the blockchain after verification. The blockchain is updated to include the new block, and the mempool is cleared of the included transactions.

3.7 Verification

The entire blockchain is periodically verified to ensure its integrity. This includes checking the hashes, Merkle roots, and proof-of-work for each block.

4 Overall Architecture of the Blockchain System

The architecture of the blockchain system is modular and consists of the following components:

4.1 UserManager

Manages user data, including public-private key pairs and wallet balances. Provides methods for user creation, retrieval, and management.

4.2 Transaction

Represents a transaction with details such as sender, receiver, amount, and digital signature. Ensures data integrity through unique transaction IDs.

4.3 MemPool

A temporary storage for pending transactions. Provides methods to add and collect transactions for mining.

4.4 Block

Represents a block in the blockchain. Contains metadata such as the previous block hash, timestamp, nonce, difficulty, Merkle root, and transactions.

4.5 BlockChain

Manages the chain of blocks. Ensures that blocks are added sequentially and that proof-of-work is satisfied.

4.6 Miner

Handles the mining process, including proof-of-work and block creation. Collects valid transactions from the mempool and attempts to solve the cryptographic puzzle.

4.7 Verifier

Validates transactions, blocks, and the entire blockchain. Ensures that the blockchain remains tamper-proof and consistent.

4.8 MerkleTree

Calculates the Merkle root for a set of transactions. Ensures transaction integrity within a block.

This modular design ensures that each component is responsible for a specific functionality, making the system easier to maintain and extend.

5 Design and Implementation Details

5.1 Overview

The MiniBlockChain project is a simplified blockchain implementation. It includes the following components:

- **Block**: Represents a single block in the blockchain.
- **BlockChain**: Manages the chain of blocks.
- **Transaction**: Represents a transaction between users.
- **User**: Represents a user in the system with a wallet and cryptographic keys.
- **UserManager**: Manages users in the system.
- **MemPool**: A memory pool for pending transactions.
- **Miner**: Mines new blocks by solving a proof-of-work problem.
- **Verifier**: Verifies blocks, transactions, and the blockchain.
- **MerkleTree**: Generates a Merkle root for transactions in a block.

5.2 Data Structures

- **Block**: Contains properties such as block number, hash, previous block hash, timestamp, nonce, difficulty, Merkle root, and transactions.
- **Transaction**: Includes transaction ID, data (amount), signature, sender address, and receiver address.

- **BlockChain**: Maintains a list of blocks and the mining target value.
- **User**: Stores user details such as name, private key, public key, address, and wallet balance.
- **MemPool**: A queue for managing pending transactions.

5.3 Algorithms and Pseudocode

5.3.1 Block Creation

```
function createBlock(previousBlockHash, transactions, miningTargetValue):
    block.previousBlockHash = previousBlockHash
    block.timestamp = current time
    block.nonce = 0
    block.difficulty = miningTargetValue
    block.merkleRoot = calculateMerkleRoot(transactions)
    block.transactions = transactions
    return block
```

5.3.2 Proof of Work

```
function solveProofOfWork(block, miningTargetValue):
    startTime = current time
    while true:
        blockHash = generateHash(block)
        if blockHash < miningTargetValue:
            block.setHash(blockHash)
            return true
        block.incrementNonce()
        if current time - startTime > TIMEOUT:
            return false
```

5.3.3 Transaction Verification

```
function verifyTransaction(transaction):
    sender = getUserByAddress(transaction.sender_address)
    dataHash = hash(transaction.data)
    decryptedHash = decrypt(transaction.signature, sender.publicKey)
    return dataHash == decryptedHash
```

5.3.4 Block Verification

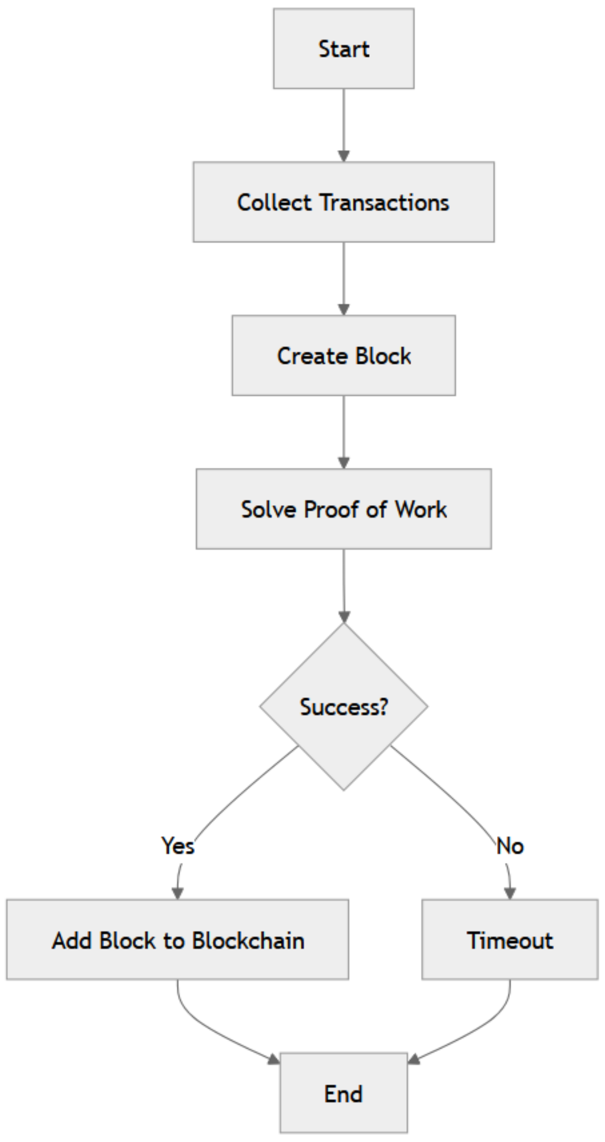
```
function verifyBlock(block, blockchain):
    if generateHash(block) != block.getStoredHash():
        return false
    if block.blockNumber > 0:
        previousBlock = blockchain.getBlock(block.blockNumber - 1)
        if block.previousBlockHash != previousBlock.getStoredHash():
            return false
    else:
        if block.previousBlockHash != all zeros:
            return false
    return true
```

5.3.5 Merkle Tree Construction

```
function calculateMerkleRoot(transactions):
    currentLevel = transactions
    while currentLevel.size > 1:
        nextLevel = []
        for i in range(0, currentLevel.size, 2):
            left = currentLevel[i]
            right = currentLevel[i+1] or left
            nextLevel.append(hash(left + right))
        currentLevel = nextLevel
    return currentLevel[0]
```

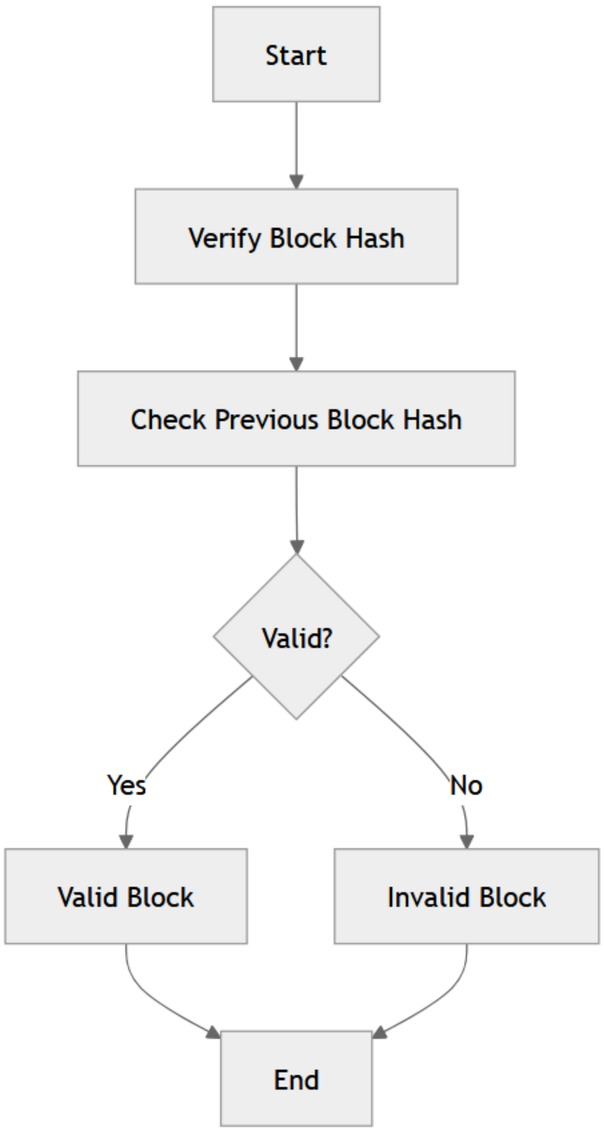
5.4 Flow Charts

5.4.1 Mining Process



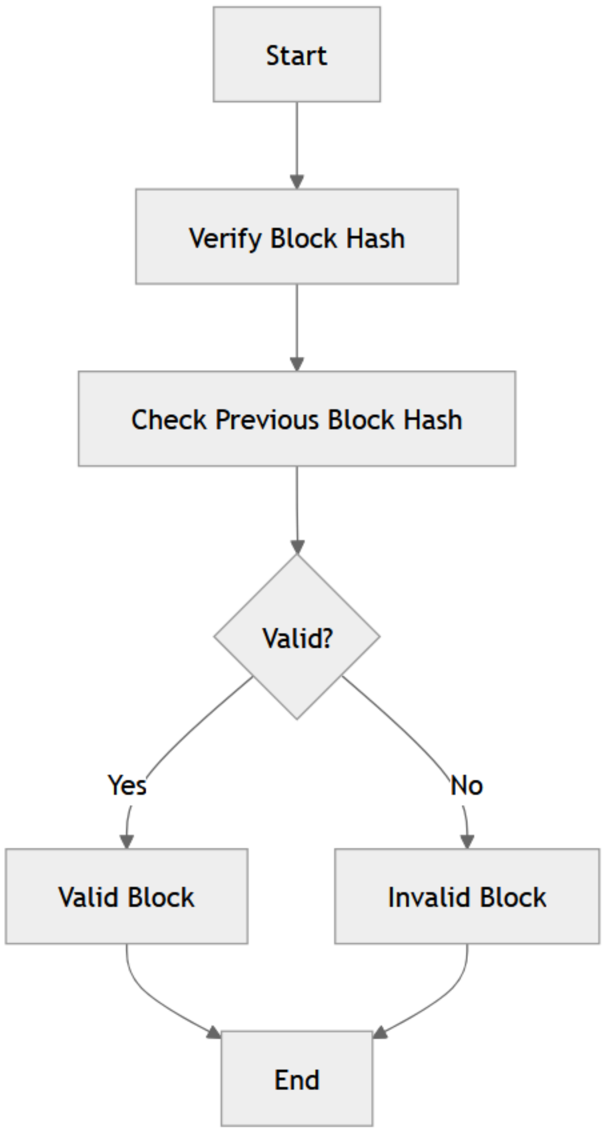
```
Start -> Collect Transactions -> Create Block -> Solve Proof of Work
    -> [Success?] -> Yes -> Add Block to Blockchain -> End
                     No -> Timeout -> End
```

5.4.2 Transaction Verification



```
Start -> Retrieve Sender -> Hash Transaction Data -> Decrypt Signature
    -> [Match?] -> Yes -> Valid Transaction -> End
                   No -> Invalid Transaction -> End
```

5.4.3 Block Verification



```
Start -> Verify Block Hash -> Check Previous Block Hash
    -> [Valid?] -> Yes -> Valid Block -> End
                   No -> Invalid Block -> End
```

5.5 Implementation Details

- **Cryptography**: RSA is used for digital signatures, and SHA-256 is used for hashing.
- **Singletons**: `UserManager`, `MemPool`, and `BlockChain` are implemented as singletons for centralized management.
- **Timeouts**: Mining has a timeout to prevent infinite loops.
- **Error Handling**: Exceptions are used for invalid operations, such as insufficient funds.

This design ensures modularity, security, and scalability for the blockchain system.

6 Experimental Results

6.1 Simulation

- **Transaction Validation**: Verified transactions before and after tampering.
- **Mining**: Successfully mined blocks with valid transactions.
- **Blockchain Verification**: Ensured the integrity of the blockchain after adding blocks.

6.2 Observations

- Mining time increases with the difficulty of proof-of-work.
- Tampered transactions are successfully detected and rejected.

7 Conclusion

The MiniBlockChain project demonstrates the core functionalities of a blockchain system, including transaction validation, mining, and block verification. The system ensures data integrity and security through cryptographic techniques and proof-of-work.

8 References

1. Java Cryptography Architecture (JCA) Documentation.
2. Blockchain Basics by Daniel Drescher.
3. Merkle Tree Concepts in Cryptography.

9 Appendices

9.1 Codebase

Full source code of the project is available in the `src/` directory.

9.2 Simulation Logs

Output logs from the experimental results are included in the project repository.