University of Melbourne

COMP90043 Cryptography and Security

Decentralizing Security: Implementing SHA-256 in Blockchain for Enhanced Data Integrity

Student :
Langze Lu 1185039
Wenda Zhang 1126164
Yilin Chen 1239841

Prof :
Udaya Parampalli

October 19, 2024

Contents

1 Reflective Statement **3**
1.1 Langze Lu 1185039 . . . 3
1.2 Wenda Zhang 1126164 . . . 3
1.3 Yilin Chen 1239841 . . . 3

2 Abstract **5**

3 Introduction **6**
3.1 Background . . . 6
3.2 Research Objective and Methods . . . 6
3.3 Research Impact . . . 6

4 Theoretical Background and Foundations **7**
4.1 Cryptographic Hash Functions Overview . . . 7
4.1.1 Fundamentals and Key Properties of Cryptographic Hash Functions 7
4.1.2 Evolution of Hash Function Families in Blockchain . . . 7
4.2 In-depth Analysis of SHA-256 . . . 8
4.3 Blockchain and Decentralization . . . 9
4.3.1 Relationship Between Decentralization, Blockchain, and SHA-256 . 9

5 Implementation **10**
5.1 Overview . . . 10
5.2 Hash Function Performance Comparison . . . 10
5.3 Justification for Choosing SHA-256 . . . 10
5.4 Implementation Details . . . 11
5.4.1 Implementation Approach . . . 11
5.4.2 Data Structures . . . 11
5.4.3 Proof of Work Mechanism . . . 13
5.4.4 Blockchain Class . . . 13
5.4.5 Transaction Handling . . . 14
5.4.6 Blockchain Visualization . . . 14
5.4.7 Blockchain Execution . . . 15
5.4.8 Chain Structure . . . 16
5.4.9 Significance of Visualization . . . 17
5.4.10 Decentralization and Security . . . 17
5.4.11 Tampering Demonstration: Impact of Modifying a Block . . . 17

6 Evaluation **19**
6.1 Security Analysis . . . 19
6.1.1 Tamper Resistance Evaluation . . . 19
6.1.2 Potential Attack Analysis . . . 19
6.2 Performance Evaluation . . . 19
6.2.1 Hash Computation Speed . . . 19
6.2.2 Overall System Efficiency . . . 21
6.3 Limitations of the Current Implementation . . . 21

7 Discussion **22**
7.1 Advantages and Innovations 22
7.2 Limitations and Challenges 22
7.3 Future Directions 22

8 Conclusion **23**

9 References **24**

10 Appendix **26**
10.1 Prerequisite 26
10.2 Running the Program 26
10.3 Troubleshooting 26
10.4 Notes 27

1 Reflective Statement

1.1 Langze Lu 1185039

In this study, I focused on the detailed implementation section, explaining how our system integrates SHA-256 to secure transactions and maintain the integrity of our decentralized network. My responsibilities included outlining the data structures used, such as the blocks and the blockchain itself, and describing how these are integrated with the SHA 256 to create a secure, immutable chain. I also implemented the Proof-of-Work mechanism, which underpins the consensus model of our blockchain, illustrating how it deters fraud and ensures equitable participation across the network. This involved writing code snippets to demonstrate the practical application of these concepts and creating visualizations to help illustrate the chain's response to data tampering, which emphasized the robustness of our blockchain against attacks and unauthorized changes. During the completion process of my part, I aimed to provide a clear, technical detail of our blockchain's workings, grounding the theoretical concepts of our team in practice, and create an executable code file to showcase the real-world applicability and efficiency of our design.

1.2 Wenda Zhang 1126164

For this research project, my primary contribution focused on the in-depth analysis of cryptographic hash functions, particularly their application in blockchain technology. I was responsible for writing the "Cryptographic Hash Functions Overview" section, which detailed the fundamental principles, key properties, and evolution of hash functions in blockchain. I conducted a comprehensive examination of SHA-256, exploring its security strengths and potential weaknesses, and investigated its relationship with blockchain decentralization.

Additionally, I contributed to the overall project's literature research and support. This ensures that our study was grounded in solid academic foundations. In sum, this work deepened my understanding of cryptography and blockchain technology and gave me valuable knowledge and practical insights in these rapidly evolving fields.

Beyond theoretical research, I handled the technical implementation aspects of the project. I set up the GitHub repository to host our code and created a Docker image, ensuring the project could run consistently across different environments. This enhanced the project's reproducibility and accessibility.

1.3 Yilin Chen 1239841

In this project, I mainly wrote the abstract, introduction, evaluation, discussion, and conclusion sections. Abstract and introduction are the overall introduction to this research, including background introduction, research objectives, and research impact. The Evaluation and Discussion sections mainly evaluate the various results of the implementation, including safety, running speed, and limitations, and discuss the evaluation results to summarize the advantages and innovations as well as limitations and challenges of this technology

My responsibility is to provide an overall summary of this study and further discuss it, in order to ensure that the article has critical analysis for the selection of techniques

2 Abstract

Cryptographic hash functions and the concept of decentralization are two very fundamental concepts in blockchain technology. This study investigates the use of the encrypted hash function such as SHA-256 to secure the integrity and confidentiality of decentralized blockchain systems. We analyzed how hashing algorithms secure the centralized consensus in Proof of Work (PoW) mechanisms and show how encryption of hash functions can be incorporated with decentralization through small-scale blockchain implementation.

This work also assessed the security and performance of SHA-256 and PoW in blockchain environments, investigated the pros and cons of this architecture, and recommended the future improvement directions such as anti-quantum cryptography and more efficient consensus mechanisms to strengthen system security and scalability.

3 Introduction

3.1 Background

Blockchain technology has affected many sectors such as finance, data security, and agriculture, and the reason for this is its strong features. Defined as "a series of blocks linked and secured using cryptography in a decentralized and distributed network"[1], blockchain excels due to its decentralization, immutability, transparency, and auditability. These features make it possible to process transactions in a secure way by ensuring that tampering cannot take place.

One of the blockchain's innovations is the principle of decentralization, which eliminates any single authority's control over the network. Subsequently, a collective of nodes are involved in both the maintenance and authentication of the ledger. Consensus algorithms like Proof of Work (PoW) are the main components of this architecture, where hash functions are the ones responsible for the security of each block.

3.2 Research Objective and Methods

This analysis is concerned with the primary role of hash functions in protecting the security and integrity of data in a decentralized blockchain infrastructure. We seek to analyze the security that is guaranteed by hash functions, their effect on the efficiency of the system, and we will also illustrate these dynamics through a real blockchain deployment.

3.3 Research Impact

This study aims to provide the security aspect of hash functions and how they integrate with decentralized systems to ensure data integrity, which enhances blockchain's immutability. The insights gained from this research provide a deeper understanding of the mechanisms that make blockchain tamper-resistant, thereby promoting its reliability across various industries.

4 Theoretical Background and Foundations

4.1 Cryptographic Hash Functions Overview

4.1.1 Fundamentals and Key Properties of Cryptographic Hash Functions

The hash functions aim to generate a digital representation that can be considered a "fingerprint" of data structures such as messages and data blocks[2]. The process of hash functions can be shown as below:

$$H : \{0,1\}^* \to \{0,1\}^n$$

A hash function is designed to accept an arbitrary-length string as the input and further output a fixed length (n) of string[3]. Furthermore, the cryptographic hash function must satisfy three fundamental security properties: preimage resistance, second preimage resistance, and collision resistance.

Firstly, the preimage resistance required that for the hash function (H) and any hash value (h), it is computationally infeasible to find any input message (M) such that $h = H(M)$.

Secondly, the second preimage resistance indicates that given an input message (M), finding another message (M') such that $H(M) = H(M')$ is computationally infeasible.

Lastly, the collision resistance property defines that finding two different inputs $(M \neq M')$ that produce the same output $(H(M) = H(M'))$ is computationally infeasible[4].

These properties make every output considered "unique," making hash functions widely used in several domains, such as data integrity protection, password protection, digital signatures, blockchain, etc.

4.1.2 Evolution of Hash Function Families in Blockchain

Over the years, hash functions have constantly evolved to meet cybersecurity needs with technological advancements. The following section examines several significant hash functions from the past to the present that have shaped the field of cryptography and blockchain technology.

Cryptographic hash functions have significantly evolved, bolstering digital system security. Initially favored, MD5 (1992)[5] and SHA-1 (1995)[6] became obsolete due to vulnerabilities to collision attacks[7],[8]. In response, the SHA-2 family, developed by the NSA and released by NIST in 2002[9], includes more secure variants like SHA-224, SHA-256, SHA-384, and SHA-512[10]. NIST's continued endorsement highlights SHA-256's robustness[11], pivotal in blockchain applications like Bitcoin, where it secures the proof-of-work mechanism detailed in the Bitcoin whitepaper[12]. Its reliability and efficiency have led to widespread adoption in blockchain technologies[13],[14],[15]. The following section delves into SHA-256's critical role in blockchain.

4.2 In-depth Analysis of SHA-256

After examining the evolution of cryptographic hash functions in the blockchain and their fundamental properties, we now focus on a detailed analysis of SHA-256. SHA-256's prominence in modern cryptographic applications, particularly in blockchain technology, justifies this focus.

Security Strength: SHA-256, as part of the SHA-2 family, has been widely recognized for its robust security properties. Its key strengths lie in:

- **Collision Resistance**: SHA-256 is crafted to ensure that the probability of two different inputs yielding the same hash is extremely low, with an expected work factor of $\mathcal{O}(2^{128})$. Current computational capabilities cannot feasibly break this, and extensive cryptanalysis has yet to find practical collision attacks, unlike its predecessors MD5 and SHA-1. Mendal et al.'s research further corroborates SHA-256's resilience against differential cryptanalysis[16].
- **Pre-image and Second Pre-image Resistance:** Pre-image resistance makes it computationally prohibitive to deduce the original input from its hash[17]. Second pre-image resistance similarly impedes discovering a new input that produces the same hash as a given input. The effort required to find a second pre-image is roughly $\mathcal{O}(2^{256})$, rendering brute-force attacks ineffective.

Potential Weaknesses: Although SHA-256 is highly secure, certain theoretical vulnerabilities and potential future threats should be noted:

- **Quantum Computing Threats**: As quantum computing evolves, SHA-256's vulnerability to **Grover's Algorithm**—a quantum search algorithm—increases. This algorithm can theoretically reduce the complexity of finding pre-images from $\mathcal{O}(2^{256})$ to $\mathcal{O}(2^{128})$ by requiring only $\mathcal{O}(\sqrt{N})$ evaluations of the function, where N is the size of the function's domain[13]. Although current quantum capabilities do not yet pose a practical threat, this highlights the need for future blockchain technologies to integrate post-quantum cryptographic methods to maintain long-term security.
- **Collision Resistance Concerns**: Theoretical advancements could potentially reduce the effectiveness of SHA-256's collision resistance by altering the number of processing rounds. While SHA-256 remains unbroken in practical terms, a future theoretical breakthrough could compromise blockchain integrity by enabling the manipulation of block hashes.
- **Security in Resource-Constrained Environments**: SHA-256 might be computationally demanding for devices with limited processing power, such as IoT nodes[18], [19]. The risk is that such devices could become the weak links in a decentralized network, potentially leading to security breaches. Ensuring that the hashing process is optimized for low-power environments is critical for maintaining the security and integrity of decentralized systems.

4.3 Blockchain and Decentralization

Decentralization is one of the core principle of Blockchain, this enables secure, transparent, and immutable record-keeping without a central authority. The structure of blockchain consists of a sequence of blocks secured by the hash function[13], inherently distributes control and bolsters data integrity.

The distributed nature ensures that no entity has complete control over the system, providing integrity. By ensuring that each participant in the network plays a role in validating transactions and maintaining the ledger's integrity, a trust-less environment is created (where trust is placed in the cryptography protocol rather than in individual participants). In this environment, consensus among participants replaces the need for a trusted central authority.

4.3.1 Relationship Between Decentralization, Blockchain, and SHA-256

SHA-256 is critical in seamlessly integrating decentralization with the functional integrity of blockchain systems. It enhances network security by uniquely identifying and linking each block, making unauthorized alterations computationally impractical[20]. In consensus frameworks like PoW, exemplified by Bitcoin, SHA-256 is crucial for fostering decentralized consensus and thwarting any single party's dominance[12].

Furthermore, SHA-256 supports fairness in mining, a vital element in decentralized networks. Mining requires solving cryptographic puzzles to find a nonce that results in the desired hash, ensuring the fair distribution of new digital currency and securing the blockchain[21].

5 Implementation

5.1 Overview

Our implementation aims to develop a small-scale blockchain system that integrates cryptographic hash functions with decentralization, focusing on demonstrating security breach conditions and evaluating performance.

5.2 Hash Function Performance Comparison

To determine the optimal hash function for our blockchain, we tested the performance of SHA-256, SHA-224, SHA-1, and MD5 using varied input sizes—1 KB, 1 MB, and 1 GB—to represent different transaction and block sizes in the blockchain.

Evaluation Method: Hash computation times were measured for each input size across 1000 iterations to maintain consistency, reflecting the range of data managed in typical blockchain operations.

Input Size	**SHA-256 (s)**	**SHA-224 (s)**	**SHA-1 (s)**	**MD5 (s)**
1 KB	0.000001	0.000001	0.000001	0.000002
1 MB	0.000685	0.000688	0.000614	0.001272
1 GB	0.666032	0.664819	0.619302	1.276272

Table 1: Performance comparison of different hash functions at varying input sizes (averaged over 1000 iterations)

Analysis: Performance tests reveal minor differences among SHA-256, SHA-224, and SHA-1 across various input sizes. However, MD5 lags in both speed and security, performing significantly slower and lacking robustness as input size increases. In contrast, the SHA family maintains consistent performance, with SHA-256 slightly trailing SHA-1 and SHA-224.

5.3 Justification for Choosing SHA-256

SHA-256 was selected as our blockchain's hash function based on its superior security and performance:

- **Security**: SHA-256 offers robust defense against collision and preimage attacks, essential for ensuring the integrity of the blockchain.
- **Performance**: Though slightly slower than SHA-1 and SHA-224 at larger data sizes, SHA-256 provides a negligible performance difference while offering enhanced security with its 256-bit output—superior to the lower bit outputs of other SHA variants. This makes SHA-256 an optimal balance of security and performance for our blockchain needs.

5.4 Implementation Details

5.4.1 Implementation Approach

We developed a small-scale blockchain system in Python to illustrate the fundamental principles of blockchain technology, emphasizing the role of SHA-256 in maintaining data integrity and immutability in a decentralized environment. Each block in the chain includes critical data like index, timestamp, transactions, and hash values, and features a basic PoW mechanism to demonstrate consensus building.

Rather than simulating attacks, our focus is on demonstrating the impact of data modifications on the chain's integrity. We include a simple visualization to show how SHA-256 upholds data consistency across the blockchain. The following sections detail the key data structures and the integration of the hash function, which are central to our blockchain architecture.

5.4.2 Data Structures

The blockchain is built around a fundamental data structure, the `Block`, which contains the following fields:

- **index**

  The position of the block in the blockchain. This ensures the correct ordering of blocks.
- **timestamp**

  The time when the block is created, helping to track the chronology of the chain.
- **data**

  The transaction or payload data stored in the block. This contains the core information being processed on the blockchain.
- **previousHash**

  The hash of the previous block, linking blocks together to form the chain and ensuring immutability.
- **nonce**

  A number used once for proof-of-work, which ensures that blocks are mined according to the difficulty requirement.
- **hash**

  The hash of the current block, calculated from all other fields except the `hash` itself.

Block creation

```
# The version of Python used is 3.11.4
import hashlib
import json
import time
# For visualisation purposes
import matplotlib.pyplot as plt
from matplotlib.patches import FancyBboxPatch

class Block:
    def __init__(self, index, timestamp, transactions, previousHash,
    nonce=0):
        self.index = index
        self.timestamp = timestamp
        self.transactions = transactions  # List of transactions
        self.previousHash = previousHash
        self.nonce = nonce
        self.hash = self.computeHash()
```

Hash Function Integration To securely integrate the SHA-256 hashing algorithm into the blockchain, the following function is implemented:

- **Function**

  `computeHash()`

- **Input**

  All block attributes, except for the `hash` field.

- **Process**

  Concatenate the block's contents (such as `index`, `timestamp`, `data`, `previous_hash`, and `nonce`) into a string and apply the SHA-256 hash function.

- **Output**

  A fixed-size 256-bit hash value that uniquely represents the contents of the block.

Compute SHA 256 Hash

```
    def computeHash(self):
        # Serialize block's data and compute the SHA-256 hash
        block_string = json.dumps({
            'index': self.index,
            'timestamp': self.timestamp,
            # Serialize transactions
            'transactions': [t.__dict__ for t in self.transactions],
            'previous_hash': self.previousHash,
            'nonce': self.nonce
        }, sort_keys=True).encode()
        return hashlib.sha256(block_string).hexdigest()
```

5.4.3 Proof of Work Mechanism

To secure the blockchain and validate blocks, we implemented a basic PoW mechanism. Each block's hash must begin with a predetermined number of leading 0s (the higher the difficulty level, the more 0s in front), adjusted via the nonce. This process, aided by SHA-256, distributes computational tasks evenly across the network, bolstering fairness and security. The requirement to re-mine all subsequent blocks for any data alteration makes tampering computationally impractical, thus safeguarding the blockchain's integrity and deterring unauthorized changes. In addition, PoW requires substantial computational feasibility to mine a block, preventing easy and malicious alterations to the chain.

Proof-of-Work Function

```
def proofOfWork(self, block):
    block.nonce = 0
    computedHash = block.computedHash()
    while not computedHash.startswith('0' * self.difficulty):
        block.nonce += 1
        computedHash = block.computedHash()
    return computedHash
```

5.4.4 Blockchain Class

The `Blockchain` class is responsible for managing the chain of blocks and adding new blocks through a PoW consensus mechanism.

Each time a block is added, the hash of the previous block is included to ensure that the chain is tamper-proof. If any block in the chain is modified, the hashes of all subsequent blocks will no longer match, making the blockchain invalid.

Blockchain Class

```
class Blockchain:
    def __init__(self):
        self.chain = []  # List to store the chain of blocks
        self.difficulty = 2  # Difficulty level for mining
        self.createFirstBlock()

    def createFirstBlock(self):
        # Create the first block in the blockchain with fixed values
        firstBlock = Block(0, time.time(), [], "0")
        self.chain.append(firstBlock)

    def addBlock(self, transactions):
        # Add a new block to the blockchain
        previousBlock = self.chain[-1]
        newBlock = Block(index=previousBlock.index + 1,
                         timestamp=time.time(),
                         transactions=transactions,
                         previousHash=previousBlock.hash)
        newBlock.hash = self.proofOfWork(newBlock)
        self.chain.append(newBlock)
```

5.4.5 Transaction Handling

Each block in the blockchain stores a list of transactions. The following `Transaction` class defines the structure for a transaction, where each transaction has a sender, receiver, and amount.

Transaction Class

```
class Transaction:
    def __init__(self, sender, receiver, amount):
        self.sender = sender
        self.receiver = receiver
        self.amount = amount

    def __repr__(self):
        return f"Transaction({self.sender} ->
        {self.receiver}: {self.amount})"
```

5.4.6 Blockchain Visualization

In order to provide a better understanding of how the blocks are linked in the blockchain, we implemented a simple visualization using `Matplotlib`. Each block is represented as a box, and arrows show the connections between the blocks, indicating the chaining based on the `previousHash`.

Visualization Function

```
def visualizeBlockchain(blockchain):
    fig, ax = plt.subplots(figsize=(20, 7))

    # Set the color map and block spacing
    colors = plt.cm.coolwarm
    block_spacing = 4
    arrow_offset = 2

    for i, block in enumerate(blockchain.chain):
        color = colors(i / len(blockchain.chain))
        box = FancyBboxPatch((i * block_spacing, 1), 2.8, 1,
        boxstyle="round,pad=0.3", edgecolor="black", facecolor=color,
        linewidth=2, linestyle='--')
        ax.add_patch(box)

        # block details
        block_details = f"Index: {block.index}\nHash:
        {block.hash[:8]}...\n" + "\n".join([f"{t.sender} ->
        {t.receiver}: {t.amount}" for t in block.transactions])
        ax.text(i * block_spacing + 0.2, 1.5, block_details,
        fontsize=12, verticalalignment='center', family='monospace')

        if i > 0:
            conn = ConnectionPatch(xyA=((i-1) * block_spacing +
            2.8, 1.5), xyB=(i * block_spacing - arrow_offset, 1.5),
                                   coordsA='data', coordsB='data',
                                   axesA=ax, axesB=ax,
                                   arrowstyle="->", shrinkB=5,
                                   linewidth=2, color='gray',
                                   linestyle=':')
            ax.add_patch(conn)

    ax.set_xlim(0, len(blockchain.chain) * block_spacing)
    ax.set_ylim(0, 3)
    ax.axis('off')
    plt.title('Blockchain Visualization', fontsize=18,
    fontweight='bold')
    plt.show()
```

5.4.7 Blockchain Execution

In the following example, we demonstrate how to create transactions, mine blocks, and visualize the blockchain.

Main Execution

```
if __name__ == "__main__":
    blockchain = Blockchain()

    # Create some sample transactions
    transactions1 = [Transaction("Alice", "Bob", 50),
                     Transaction("Bob", "Charlie", 20)]
    transactions2 = [Transaction("Charlie", "Dave", 30),
                     Transaction("Dave", "Eve", 40)]
    transactions3 = [Transaction("Eve", "Frank", 25),
                     Transaction("Frank", "Alice", 10)]

    # Mine blocks and add transactions to the blockchain
    print("Mining block 1...")
    blockchain.addBlock(transactions1)

    print("Mining block 2...")
    blockchain.addBlock(transactions2)

    print("Mining block 3...")
    blockchain.addBlock(transactions3)

    # Visualize the blockchain
    visualizeBlockchain(blockchain)
```

5.4.8 Chain Structure

In figure 1 the arrows between the blocks represent the linkage in the blockchain. Each block contains the hash of its preceding block (shown above each arrow), which secures the chain:

If any block's data were altered, its hash would change, thereby invalidating the subsequent blocks due to mismatched previous hash values. This chaining mechanism is what makes the blockchain tamper-evident.

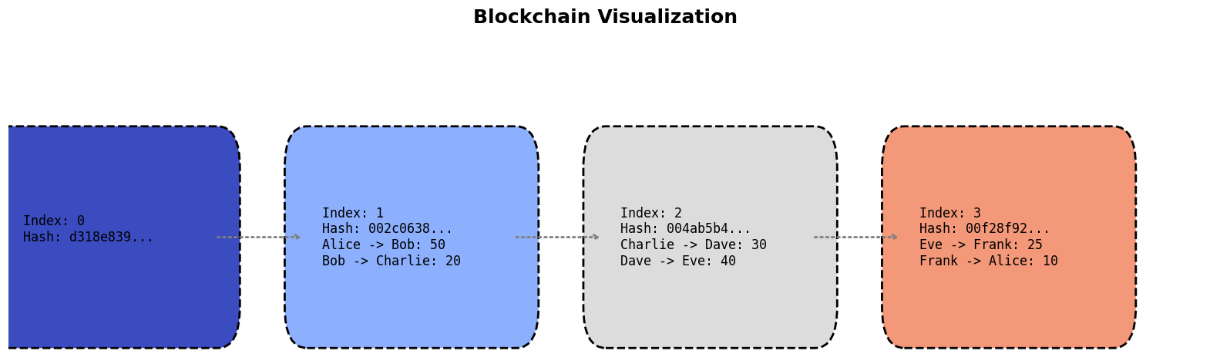

Figure 1: Blockchain Visualization: Each block displays its index, hash, and contained transactions, linked by its predecessor's hash to demonstrate the security and continuity of the blockchain.

5.4.9 Significance of Visualization

The visualization demonstrates key blockchain features, emphasizing the security, transparency, and traceability of transactions. It showcases how blockchain technology ensures a secure, decentralized framework for data management.

5.4.10 Decentralization and Security

Our blockchain architecture integrates the hash chain and PoW mechanism to enhance security and uphold the system's decentralized nature.

- **Proof-of-Work (PoW) Mechanism**: The PoW mechanism requires that each block's hash starts with a predetermined number of zeros, which necessitates considerable computational effort to achieve, thus protecting against unauthorized modifications.

- **The Role of the Hash Chain**: Each block in the blockchain references the hash of its predecessor, ensuring the integrity of the chain. Any modification in the block data disrupts this chain, rendering subsequent blocks invalid and demonstrating the system's resistance to unauthorized changes.

5.4.11 Tampering Demonstration: Impact of Modifying a Block

This demonstration simulates tampering with a block to show how such actions compromise the integrity of the blockchain. Modifying block data disrupts the hash chain, invalidating all subsequent blocks and highlighting the blockchain's robust security measures.

Tampering Demonstration

```
# Modify the data of a block (e.g., block 2)
blockchain.chain[1].transactions = [Transaction("Hacker", "Bob", 1000)]

# Recompute the hash for the modified block
blockchain.chain[1].hash = blockchain.chain[1].computeHash()

# Update the subsequent blocks
for i in range(2, len(blockchain.chain)):
    blockchain.chain[i].previousHash = blockchain.chain[i-1].hash
    blockchain.chain[i].hash = blockchain.chain[i].computeHash()

# Visualize the blockchain after tampering
visualizeBlockchain(blockchain)
```

This demonstration shows that altering the transaction data in block 2 causes its hash and all subsequent block hashes to be recalculated, effectively breaking the hash chain. This breakage illustrates the blockchain's tamper-evident feature—any unauthorized change is quickly detectable and renders the chain invalid unless all affected blocks are recalculated.

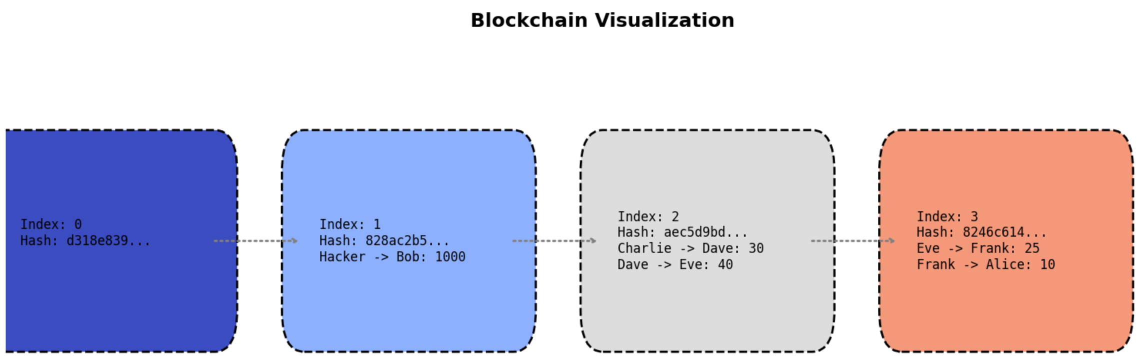

Figure 2: Visualization of Blockchain After Tampering: The hash chain is broken after modifying a block, as indicated by the mismatched hashes. This demonstrates the blockchain's tamper-evident property.

This practical demonstration underlines the critical role of hashes in maintaining the security and integrity of decentralized data within blockchain applications. It also reinforces the blockchain's inherent capability to resist and reveal any tampering, ensuring data immutability.

6 Evaluation

6.1 Security Analysis

6.1.1 Tamper Resistance Evaluation

The implementation effectively demonstrated the tamper-evident nature of blockchain through the integration of the SHA-256 hash function. Each block's hash relies on the previous block's hash, creating a hash chain that ensures data immutability. Any modification to a block's data invalidates all subsequent blocks, illustrating the importance of hashes in maintaining the integrity of the blockchain. This tamper-proof property is essential, particularly in decentralized systems where data integrity must be preserved across distributed nodes.

6.1.2 Potential Attack Analysis

Although SHA-256 provides strong collision and preimage resistance, real-world blockchain deployments face risks from attacks such as Sybil attacks and 51% attacks. One of the most famous attacking methods in the blockchain is the 51% attack[22]. In a 51% attack, if a single entity controls more than 50% of the network's computational power, they could potentially alter the blockchain's history. However, the concept of PoW disperses computing power among many nodes or participants, making it difficult for a single entity to control most resources, which can effectively resist 51% attacks. Sybil's attacks involve creating numerous fake identities to overwhelm the network. Here, an attacker tries to fool the blockchain network by generating and controlling multiple identities that are considered genuine in the blockchain network[23]. These attacks were not simulated in the current implementation, but they remain significant security concerns in practical blockchain systems.

6.2 Performance Evaluation

6.2.1 Hash Computation Speed

In order to evaluate the computational complexity and required time of different hash algorithms, the team tested them under different input sizes. The testing includes SHA-256, SHA-224, SHA-1, and MD5 algorithms, which are evaluated for input data sizes of 1KB, 1MB, and 1GB, respectively, to simulate different data scenarios that blockchain may handle.

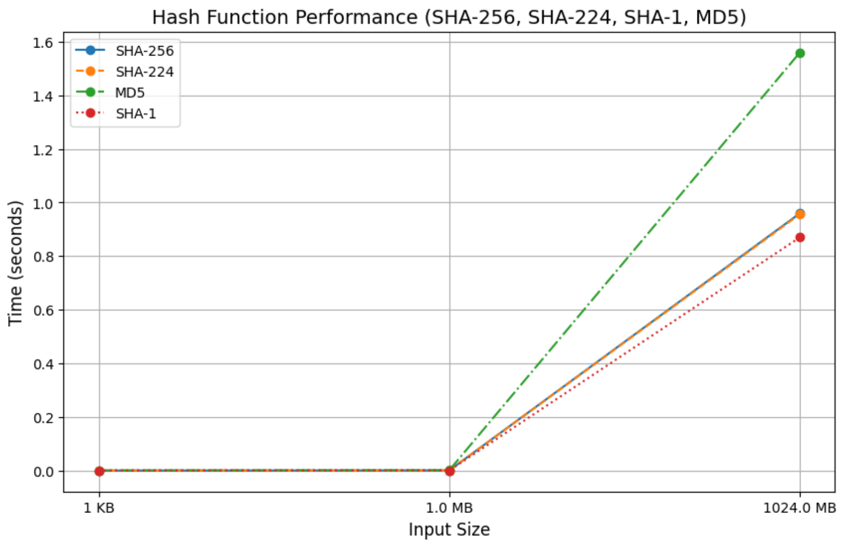

Figure 3: Different Hash Functions Performance in different Input Sizes

In Figure 3, it can be seen that although the difference in computation time between different hash algorithms is not significant with smaller input data (1KB and 1MB), as the input data increases to 1GB, the computation time of MD5 is significantly higher than that of SHA family algorithms. SHA-256, SHA-224, and SHA-1 have relatively small performance differences under large data volumes, but SHA-1 is slightly better than SHA-256 and SHA-224.

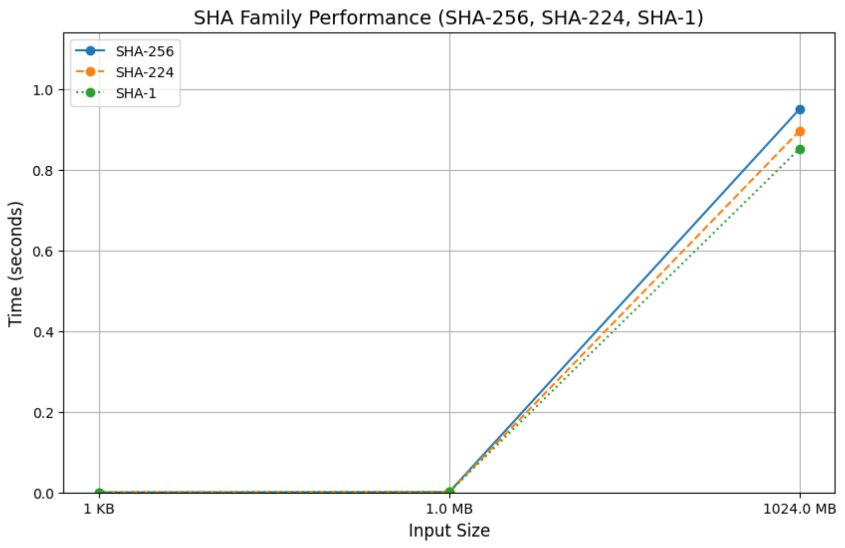

Figure 4: Different SHA Functions Performance in different Input Sizes

In Figure 4, only three algorithms of the SHA family (SHA-256, SHA-224, and SHA-1) are shown. It can be observed that their performance is almost consistent under large input data, indicating that they have similar efficiency in processing larger data blocks.

These performance evaluation results indicate that MD5 lacks speed and security compared to the SHA family, which is not suitable as a hash function for blockchain applications due to its lack of security (such as vulnerabilities in collision attacks). In contrast, SHA-256 not only provides stronger security but also has stable performance when processing large-scale data, making it suitable for scenarios that require high security, such as data integrity protection in blockchain.

6.2.2 Overall System Efficiency

PoW requires nodes to carry out iterative searches in a certain range of integers expected to be time-consuming[24]. Therefore, although PoW provides strong security, it also introduces significant overhead, especially as the blockchain scale grows. This method of ensuring security by solving password problems reduces system efficiency and consumes much energy. This inefficiency becomes a limiting factor for system expansion, especially in environments with limited computing resources, such as IoT devices or mobile platforms

In addition, the SHA-256 algorithm provides strong security, but the speed difference between it and other algorithms (SHA-224, SHA-1) is not significant. Considering the difference in the number of generated hash bits, this is acceptable.

6.3 Limitations of the Current Implementation

The current implementation does not simulate advanced attacks like Sybil or 51% attacks, and it struggles with scalability due to the computational cost of the PoW mechanism. These limitations suggest the need for further optimization, particularly in enhancing performance for resource-constrained environments and addressing real-world attack scenarios.

7 Discussion

7.1 Advantages and Innovations

The study integrate SHA-256 and PoW to showcase the tamper-resistant and secure nature of blockchain technology decentralized architecture. Demonstrating enhanced security compared to hash functions like MD5 and SHA1 to SHA256, SHA 256 offers a notable efficiency improvement. The visual representation of block structures serves as a depiction of the fundamental security principles underlying blockchain technology. Ultimately acting as a proof of concept for grasping the core mechanisms of technology.

7.2 Limitations and Challenges

Even though the implementation showcased the basics of blockchain enough the excessive energy usage associated with the PoW method still poses a major drawback. This disparity becomes glaring when compared to alternative eco-friendly consensus methods like Proof of Stake (PoS). Besides, while PoW offers protection against specific security risks, it isn't entirely invulnerable to potential weaknesses. If one entity manages to control most of the power on the network, what is known as a 51 per cent attack could disrupt transaction verification and allow for double spending to occur. Moreover, our systems defences are not fully equipped to handle sophisticated attack strategies like Sybil attacks. Another issue states the vulnerability of the SHA-256 against advancements in quantum computing that could potentially weaken its ability to resist preimage and collision challenges over time.

7.3 Future Directions

The future aims for this study could focus on improving the system's scalability, particularly by exploring quantum-resistant algorithms and more efficient consensus mechanisms like PoS. Security measures should also be implemented to defend against advanced attack scenarios.

8 Conclusion

In conclusion, the study explored why SHA-256 helps in strengthening security and providing reliability in blockchain. Our findings highlighted that SHA-256 effectively ensures data integrity and provides a fundamental for the PoW mechanism. Although robust, poses scalability and efficiency challenges, particularly in energy consumption. Looking forward, enhancing blockchain's resilience against emerging quantum threats and exploring energy-efficient consensus mechanisms like Proof of Stake (PoS) is crucial. This research not only reaffirms the strength of hash functions in blockchain but also points to significant pathways for its evolution and wider adoption.

9 References

[1] A. A. Monrat, O. Schelén, and K. Andersson, "A survey of blockchain from the perspectives of applications, challenges, and opportunities," *IEEE Access*, vol. 7, pp. 117 134–117 151, 2019. DOI: 10.1109/ACCESS.2019.2936094.

[2] S. Yunling and M. Xianghua, "An overview of incremental hash function based on pair block chaining," in *2010 International Forum on Information Technology and Applications*, vol. 3, 2010, pp. 332–335. DOI: 10.1109/IFITA.2010.332.

[3] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 6th. USA: Prentice Hall Press, 2013, ISBN: 0133354695.

[4] M. Naor and M. Yung, "Universal one-way hash functions and their cryptographic applications," in *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, ser. STOC '89, Seattle, Washington, USA: Association for Computing Machinery, 1989, pp. 33–43, ISBN: 0897913078. DOI: 10.1145/73007.73011. [Online]. Available: https://doi.org/10.1145/73007.73011.

[5] R. Rivest, "The md5 message-digest algorithm," Tech. Rep., 1992.

[6] D. R. Ignatius Moses Setiadi, A. Faishal Najib, E. H. Rachmawanto, C. Atika Sari, K. Sarker, and N. Rijati, "A comparative study md5 and sha1 algorithms to encrypt rest api authentication on mobile-based application," in *2019 International Conference on Information and Communications Technology (ICOIACT)*, 2019, pp. 206–211. DOI: 10.1109/ICOIACT46704.2019.8938570.

[7] T. Xie, F. Liu, and D. Feng, *Fast collision attack on MD5*, Cryptology ePrint Archive, Paper 2013/170, 2013. [Online]. Available: https://eprint.iacr.org/2013/170.

[8] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, *The first collision for full SHA-1*, Cryptology ePrint Archive, Paper 2017/190, 2017. [Online]. Available: https://eprint.iacr.org/2017/190.

[9] N. Sklavos and O. Koufopavlou, "On the hardware implementations of the sha-2 (256, 384, 512) hash functions," in *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.*, vol. 5, 2003, pp. V–V. DOI: 10.1109/ISCAS.2003.1206214.

[10] M. Kammoun, M. Elleuchi, M. Abid, and M. S. BenSaleh, "Fpga-based implementation of the sha-256 hash algorithm," in *2020 IEEE International Conference on Design Test of Integrated Micro Nano-Systems (DTS)*, 2020, pp. 1–6. DOI: 10.1109/DTS48731.2020.9196134.

[11] N. NIST, "Fips 180-4 secure hash standard (shs)," *Gaithersburg, Montgomery County, Maryland: National Institute of Standards and Technology.*, pp. 180–4, 2015. DOI: http://dx.doi.org/10.6028/NIST.FIPS.

[12] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system bitcoin: A peer-to-peer electronic cash system," *Bitcoin. org. Disponible en https://bitcoin. org/en/bitcoin-paper*, 2009.

[13] D. K.N. and R. Bhakthavatchalu, "Parameterizable fpga implementation of sha-256 using blockchain concept," in *2019 International Conference on Communication and Signal Processing (ICCSP)*, 2019, pp. 0370–0374. DOI: 10.1109/ICCSP.2019.8698069.

[14] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *2017 IEEE International Congress on Big Data (BigData Congress)*, 2017, pp. 557–564. DOI: 10.1109/BigDataCongress.2017.85.

[15] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, "A survey on the security of blockchain systems," *Future Generation Computer Systems*, vol. 107, pp. 841–853, 2020, ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2017.08.020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17318332.

[16] F. Mendel, T. Nad, and M. Schläffer, "Improving local collisions: New attacks on reduced sha-256," May 2013, ISBN: 978-3-642-38347-2. DOI: 10.1007/978-3-642-38348-9_16.

[17] J. Mo, X. Xiao, M. Tao, and N. Zhou, "Hash function mapping design utilizing probability distribution for pre-image resistance," in *2012 IEEE Global Communications Conference (GLOBECOM)*, 2012, pp. 862–867. DOI: 10.1109/GLOCOM.2012.6503221.

[18] C. E. B. Santos, L. M. D. d. Silva, M. F. Torquato, S. N. Silva, and M. A. C. Fernandes, "Sha-256 hardware proposal for iot devices in the blockchain context," *Sensors*, vol. 24, no. 12, 2024, ISSN: 1424-8220. DOI: 10.3390/s24123908. [Online]. Available: https://www.mdpi.com/1424-8220/24/12/3908.

[19] S. Abed, R. Jaffal, B. J. Mohd, and M. Al-Shayeji, "An analysis and evaluation of lightweight hash functions for blockchain-based iot devices," *Cluster Computing*, Jun. 2021. DOI: 10.1007/s10586-021-03324-1. [Online]. Available: https://doi.org/10.1007/s10586-021-03324-1.

[20] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *International journal of web and grid services*, vol. 14, no. 4, pp. 352–375, 2018.

[21] A. M. Antonopoulos, *Mastering Bitcoin: unlocking digital cryptocurrencies.* " O'Reilly Media, Inc.", 2014.

[22] K. Kaşkaloğlu, "Near zero bitcoin transaction fees cannot last forever," in *SDIWC, The International Conference on Digital Security and Forensics (DigitalSec2014)*, 2014, pp. 91–99.

[23] P. Swathi, C. Modi, and D. Patel, "Preventing sybil attack in blockchain using distributed behavior monitoring of miners," in *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2019, pp. 1–6. DOI: 10.1109/ICCCNT45670.2019.8944507.

[24] Z. Ouyang, J. Shao, and Y. Zeng, "Pow and pos and related applications," in *2021 International Conference on Electronic Information Engineering and Computer Science (EIECS)*, 2021, pp. 59–62. DOI: 10.1109/EIECS53707.2021.9588080.

10 Appendix

This appendix provides instructions for running the blockchain demonstration program using Docker. The code implementation can be accessed via `https://github.com/WendaZhang08/24SM2-COMP90043-Research-Project-Blockchain-Hash-Fusion`.

10.1 Prerequisite

Ensure that:

1. Docker installed on your system (Windows, macOS, or Linux).

2. Internet connection for downloading the Docker image.

10.2 Running the Program

To run the blockchain demonstration, follow these steps:

1. Install Docker on your system if not already installed

Visit `https://www.docker.com/products/docker-desktop` for installation instructions specific to your operating system.

2. Open a terminal or command prompt

3. Pull the Docker image by running:

```
docker pull damonzzz/blockchain-demo:latest
```

4. Run the program using:

```
docker run -it --rm damonzzz/blockchain-demo:latest
```

This command will download the image (if not already present) and start a container that runs the blockchain demonstration program. The "-it" flag allows for interactive output, and "–rm" removes the container after it exits.

10.3 Troubleshooting

If you encounter any issues:

1. Ensure Docker is properly installed and running on your system.

2. Check your internet connection if the image fails to download.

3. If the program doesn't start, try removing the existing image and pulling it again:

```
docker rmi damonzzz/blockchain-demo:latest
docker pull damonzzz/blockchain-demo:latest
```

10.4 Notes

The program will generate visualizations of the blockchain. These will be displayed in the terminal output.

The Docker image contains all necessary dependencies and code to run the demonstration, ensuring consistency across different systems.

By following these instructions, you should be able to run the blockchain demonstration program in a Docker container, allowing for easy project evaluation.