Course Title: Introduction to Blockchain Technologies

Course Code (CEN-429)

Assignment No 1

Assignment Title: Blockchain Fundamentals and Security Analysis

Name: **Muhammad Sohail**

Roll no: **01-132212-029**

Department/Program: BCE-08

Assignment Title: Blockchain Fundamentals and Security Analysis

Objective

- Understand UTXOs and balance retrieval in Bitcoin and Ethereum.
- Explore SHA-256 hashing and its impact on data integrity.
- Develop a simple blockchain and analyze tampering effects.
- Implement a peer-to-peer blockchain network and test data consistency.
- Evaluate proof-of-work's role in blockchain security.

Introduction:

Blockchain is a decentralized ledger ensuring secure and tamper-proof transactions. It relies on cryptography, consensus mechanisms, and distributed validation. Security analysis helps identify threats like 51% attacks and smart contract vulnerabilities.

Link to the Github repository:

All the codes including this project report can be found on the public Github repository at:

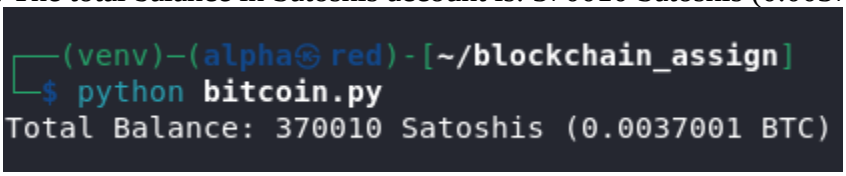
https://github.com/sohailkhan47/blockchain_ass1/

Tasks

1. Fetching UTXOs and Balances

1. Use `accounting-models/get-utxo.py` to fetch UTXOs for **"1Dorian4RoXcnBv9hnQ4Y2C1an6NJ4UrjX"**. What is the total balance in Satoshis?

   Answer: The total balance in Satoshis account is: 370010 Satoshis (0.0037001 BTC)



2. Use `accounting-models/get-eth-balance.py` to fetch the balance of **"0xEeC84548aAd50A465963bB501e39160c58366692"**. What is the total balance in Weis?

```
Balance of Weis account is: 196549219251000
Balance in ETH: 0.000196549219251 ETH
```

2. Hashing with SHA-256

3. Use `dissecting-blockchain/sha256.py` to compute the SHA-256 hash of **"CEN-429–Blockchain for Fintech"**. Write the hash value.

SHA-256 Hash: 3cda54b8aba369653621940d99a4e243c86ef4bcc831e81f9e5fb04e9ca5c858

```
└─$ python hashing.py
SHA-256 Hash: 3cda54b8aba369653621940d99a4e243c86ef4bcc831e81f9e5fb04e9ca5c858
--------------------------------------------------------------------------------
```

4. Modify the text to **"COMP1830-Blockchain for Fintech!"** and compute its SHA-256 hash. Write the new hash value.

Modified SHA-256 Hash:
52c7ab834474b736205fb3b3c49421dffb2b5d03ffc203c3923315a3b72f2ae1

```
--------------------------------------------------------------------------------
Modified SHA-256 Hash:  52c7ab834474b736205fb3b3c49421dffb2b5d03ffc203c3923315a3b72f2ae1
```

3. Creating a Blockchain

5. Use `dissecting-blockchain/block.py` to create a blockchain of 4 blocks (including the genesis block) with these data values:
   - "Blockchain"
   - "Is"
   - "Awesome"

   What is the hash of the last block?

Original last block hash:
ba25c2e0ac375c0852519c54305f36377858c695fdab3fa30d01f0a821d90bdd

6. Modify the 3rd block's data to **"Is very"** and check blockchain validity. What should the correct final hash be?

Corrected last block hash:
48b762ca8d36e148cebf2b0eb1f66b5f1b2ddb2ec81ad8f882262145fa9ebc89

Output of Screenshot:

```
┌──(venv)─(alpha㉿red)-[~/blockchain_assign]
└─$ python dissecting-blockchain/block.py
Original last block hash: ba25c2e0ac375c0852519c54305f36377858c695fdab3fa30d01f0a821d90bdd
Block 2 modified. New hash: 7111da2ce976cf9607aa28155a2ea8a20f1878c6107d3a877943f3377debdb80
Blockchain is INVALID due to modification!
Corrected last block hash: 48b762ca8d36e148cebf2b0eb1f66b5f1b2ddb2ec81ad8f882262145fa9ebc89
```

4. Peer-to-Peer Blockchain Network

7. Use `dissecting-blockchain/peer.py` to launch 3 peers. Explain their functionality.

Answer:

Each peer in the network functions as an independent node that maintains a copy of the blockchain. Peers communicate via API endpoints, allowing them to share and update their blockchain data. The `/chain` endpoint returns the current blockchain, while `/receive_chain` validates and updates the local chain if a longer, valid one is received. The `/add_peer` endpoint registers new peers, helping nodes discover and connect with each other. This setup enables decentralized consensus without a central authority, ensuring that the longest valid chain is accepted across the network.

Output showing three peers:

First peer terminal output of flask server running on port 5000

```
┌──(venv)─(alpha㉿red)-[~/blockchain_assign]
└─$ python dissecting-blockchain/peer.py 5000
 * Serving Flask app 'peer'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 567-605-744
```

First peer server output:


```
[
  {
    "data": "Genesis Block",
    "hash": "b40c10dfccf195ba5d551aaed3f718ef3ad5f85e62bfcf4b8d94bfe15a01ab82",
    "index": 0,
    "previous_hash": "0",
    "timestamp": 1741644996.0810606
  }
]
```

Second peer terminal output of flask server running on port 5001

```
┌──(venv)─(alpha㉿red)-[~/blockchain_assign]
└─$ python dissecting-blockchain/peer.py 5001
 * Serving Flask app 'peer'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5001
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 111-548-998
127.0.0.1 - - [11/Mar/2025 03:16:56] "GET /chain HTTP/1.1" 200 -
```

Second peer server output:


```
[
  {
    "data": "Genesis Block",
    "hash": "26fdf7eb8ff0322bbca6332e0b245bef13a3cc99f83a54bfef863e61d6a7658b",
    "index": 0,
    "previous_hash": "0",
    "timestamp": 1741644992.6846368
  }
]
```


Third peer terminal output of flask server running on port 5003

```
┌──(venv)─(alpha㉿red)-[~/blockchain_assign]
└─$ python dissecting-blockchain/peer.py 5002
 * Serving Flask app 'peer'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5002
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 561-511-120
127.0.0.1 - - [11/Mar/2025 03:16:46] "GET /chain HTTP/1.1" 200 -
127.0.0.1 - - [11/Mar/2025 03:17:01] "GET /chain HTTP/1.1" 200 -
```

Third peer server output:


```
[
  {
    "data": "Genesis Block",
    "hash": "b40c10dfccf195ba5d551aaed3f718ef3ad5f85e62bfcf4b8d94bfe15a01ab82",
    "index": 0,
    "previous_hash": "0",
    "timestamp": 1741644996.0810606
  }
]
```

8. Create a blockchain of 4 blocks (as in Q5) and broadcast it to the network.

```
┌──(venv)─(alpha㉿red)-[~/blockchain_assign]
└─$ python dissecting-blockchain/broadcast.py
Sent to http://127.0.0.1:5000, Response: {
  "message": "Blockchain updated"
}

Sent to http://127.0.0.1:5001, Response: {
  "message": "Blockchain updated"
}

Sent to http://127.0.0.1:5002, Response: {
  "message": "Blockchain updated"
}
```

Server output after 4 blocks are broadcasted to peers


```
[
  {
    "data": "Genesis Block",
    "hash": "26af5183b1bb01aa379e4fadede710e324e0c3e9d2fa482b658b0a53bd865b97",
    "index": 0,
    "previous_hash": "0",
    "timestamp": 1741645551.76506
  },
  {
    "data": "Blockchain",
    "hash": "e05b5fac38d5174205ad7f5e035db84502100d8f7ba82615f8bac4e526b5e587",
    "index": 1,
    "previous_hash": "26af5183b1bb01aa379e4fadede710e324e0c3e9d2fa482b658b0a53bd865b97",
    "timestamp": 1741645551.76509
  },
  {
    "data": "Is",
    "hash": "f65b38904861f740f98d277e8cd6b22287f1d83bd7fa6f75dfd3642a87fe0c09",
    "index": 2,
    "previous_hash": "e05b5fac38d5174205ad7f5e035db84502100d8f7ba82615f8bac4e526b5e587",
    "timestamp": 1741645551.7651
  },
  {
    "data": "Awesome",
    "hash": "182794a4799c0616b0f089f279f769698a6573a0c097e0e5215075007dd06d23",
    "index": 3,
    "previous_hash": "f65b38904861f740f98d277e8cd6b22287f1d83bd7fa6f75dfd3642a87fe0c09",
    "timestamp": 1741645551.76511
  }
]
```

9. Modify the second block's data to **"Is Very"** and broadcast it. Do the peers accept the tampered data?

Answer: Yes the peer will accept the tampered data since the chain **"Blockchain is very awesome"** is longer than the original chain **"Blockchain is awesome"**

Screenshot output of original chain:


```
[
  {
    "data": "Genesis Block",
    "hash": "8816ebacdb56ecc028707c4275b9acc6e2acd7e26297165b1e0801a5f05fe3b9",
    "index": 0,
    "previous_hash": "0",
    "timestamp": 1741650156.3808858
  },
  {
    "data": "Blockchain",
    "hash": "a0b84a7c5590ab52d92b1d7f58266663633a932c655c95f27022c6a7895d3618",
    "index": 1,
    "previous_hash": "8816ebacdb56ecc028707c4275b9acc6e2acd7e26297165b1e0801a5f05fe3b9",
    "timestamp": 1741650156.3809183
  },
  {
    "data": "Is",
    "hash": "348fdd1981ca79898c15fd1bcd7da990e8fe6f2c6cf5694fb54f00da368b1993",
    "index": 2,
    "previous_hash": "a0b84a7c5590ab52d92b1d7f58266663633a932c655c95f27022c6a7895d3618",
    "timestamp": 1741650156.380927
  },
  {
    "data": "Awesome",
    "hash": "213a2612db127e9a19478f63e88285147771aa072442475a601c3934e60f7df6",
    "index": 3,
    "previous_hash": "348fdd1981ca79898c15fd1bcd7da990e8fe6f2c6cf5694fb54f00da368b1993",
    "timestamp": 1741650156.3809319
  }
]
```


screenshot output of the peers not accepting the tampered data:

```
┌──(venv)─(alpha㉿red)-[~/blockchain_assign]
└─$ python dissecting-blockchain/modified_chain.py
Sent to http://127.0.0.1:5000, Response: {
  "message": "Received chain is not longer"
}

Sent to http://127.0.0.1:5001, Response: {
  "message": "Received chain is not longer"
}

Sent to http://127.0.0.1:5002, Response: {
  "message": "Received chain is not longer"
}
```

10. Can you force the peers to accept tampered data without restarting them? How long does it take without a consensus mechanism?

Answer: Yes, you can force peers to accept tampered data without restarting them if they lack validation. Broadcasting a modified full chain via `/receive_chain` can overwrite their

blockchain. Without a consensus mechanism, this happens almost instantly, making the system vulnerable.

Each peer stores its own blockchain, syncs with others via `/receive_chain`, and registers new peers using `/add_peer`. Peers accept a longer valid chain but reject shorter or equal-length chains. Without a consensus mechanism, the longest chain always wins, making it vulnerable to attacks if a malicious node controls the majority.

Here is a screenshot output of sending the data with one extra block

```
# Create the blockchain
blockchain = [Block(0, "0", "Genesis Block")]
blockchain.append(Block(1, blockchain[-1].hash, "Blockchain"))
blockchain.append(Block(2, blockchain[-1].hash, "Is very"))
blockchain.append(Block(3, blockchain[-1].hash, "Awesome"))
blockchain.append(Block(4, blockchain[-1].hash, "Awesome"))
```

```
┌──(venv)─(alpha㉿red)-[~/blockchain_assign]
└─$ python dissecting-blockchain/modified_chain.py
Sent to http://127.0.0.1:5000, Response: {
  "message": "Blockchain updated"
}

Sent to http://127.0.0.1:5001, Response: {
  "message": "Blockchain updated"
}

Sent to http://127.0.0.1:5002, Response: {
  "message": "Blockchain updated"
}
```

```
[
  {
    "data": "Genesis Block",
    "hash": "b254209fe749a6f2ef0061527ccf78266b6727f67bd73bd8e67db65600b9f6c1",
    "index": 0,
    "previous_hash": "0",
    "timestamp": 1741650354.1812093
  },
  {
    "data": "Blockchain",
    "hash": "b20a2972496c0e399a1beca67e9a0d95f598814043d05f1b05ab338b4b407828",
    "index": 1,
    "previous_hash": "b254209fe749a6f2ef0061527ccf78266b6727f67bd73bd8e67db65600b9f6c1",
    "timestamp": 1741650354.1812418
  },
  {
    "data": "Is very",
    "hash": "78cbc75333287ac1cb55d36c72ebb87236f1c05db6605cbab8ca84fe20279d81",
    "index": 2,
    "previous_hash": "b20a2972496c0e399a1beca67e9a0d95f598814043d05f1b05ab338b4b407828",
    "timestamp": 1741650354.1812503
  },
  {
    "data": "Awesome",
    "hash": "58e7a6b5cf2a80f5b0ebcd3dc6c33251da5186842f503322aed272b95def6dac",
    "index": 3,
    "previous_hash": "78cbc75333287ac1cb55d36c72ebb87236f1c05db6605cbab8ca84fe20279d81",
    "timestamp": 1741650354.1812546
  },
  {
    "data": "Awesome",
    "hash": "411f021b215977e327b11816bea9ca451f4626b7b8736e8c0e637a7e5d867ea4",
    "index": 4,
    "previous_hash": "58e7a6b5cf2a80f5b0ebcd3dc6c33251da5186842f503322aed272b95def6dac",
    "timestamp": 1741650354.1812577
  }
]
```


5. Proof-of-Work and Blockchain Security

11. Use `dissecting-blockchain/pow_peer.py` to launch 3 peers and repeat the attack from Q9.

12. Compare the attack duration in Q9 (without proof-of-work) vs. Q11 (with proof-of-work). What do you observe?
Answer: Without proof of work the chain will accept the tampered data if its chain is longer and will reject the data if the chain is not longer.

13. Write a Python script that:

- Creates a 10-block blockchain.
- Performs the attack from Q9 and Q11 on the second block while maintaining validity.

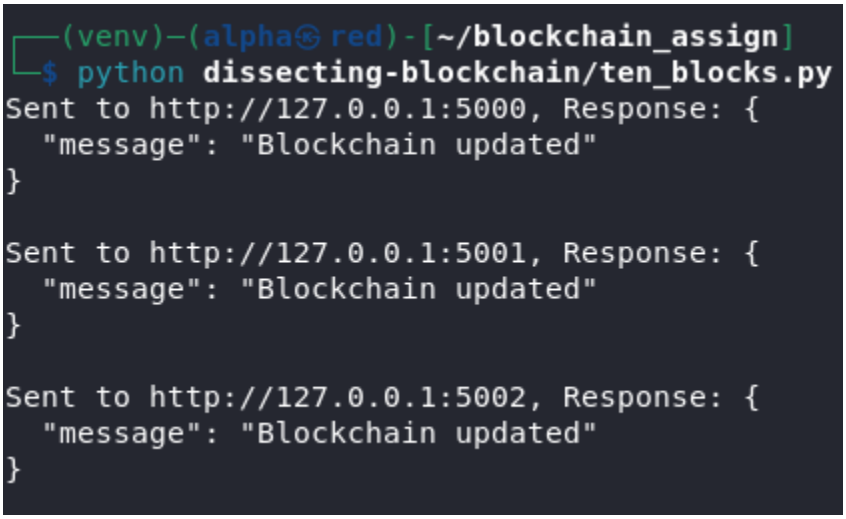
```
import requests

import json

import time

import hashlib



# List of peer nodes

PEERS = ["http://127.0.0.1:5000", "http://127.0.0.1:5001", "http://127.0.0.1:5002"]



# Block structure

class Block:

    def __init__(self, index, previous_hash, data, timestamp=None):

        self.index = index

        self.previous_hash = previous_hash

        self.data = data

        self.timestamp = timestamp or time.time()

        self.hash = self.compute_hash()



    def compute_hash(self):

        block_string = f"{self.index}{self.previous_hash}{self.data}{self.timestamp}"

        return hashlib.sha256(block_string.encode()).hexdigest()
```

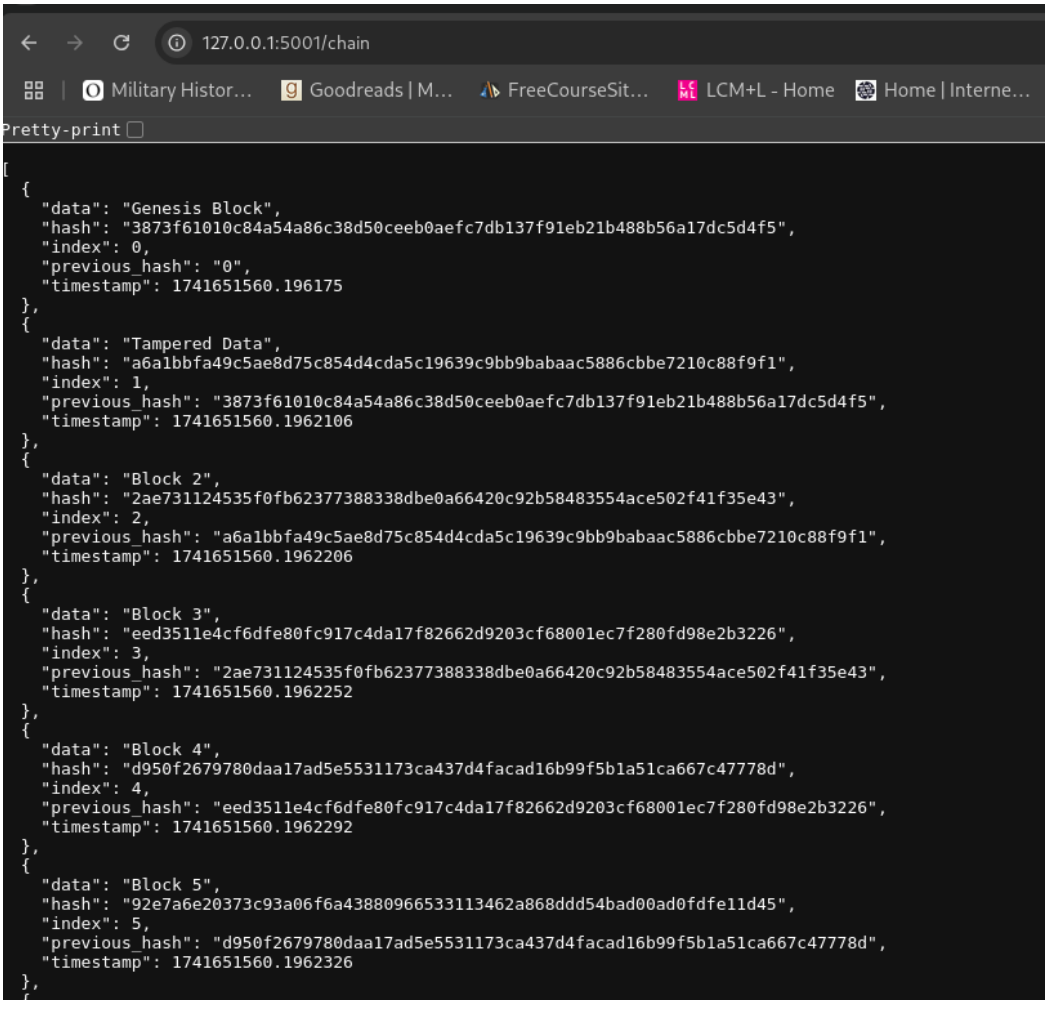
```
# Create blockchain with 10 blocks
blockchain = [Block(0, "0", "Genesis Block")]

for i in range(1, 10):
    blockchain.append(Block(i, blockchain[-1].hash, f"Block {i}"))

# Perform attack: Modify the second block's data
blockchain[1].data = "Tampered Data"
blockchain[1].hash = blockchain[1].compute_hash()

# Recalculate hashes for all subsequent blocks
for i in range(2, len(blockchain)):
    blockchain[i].previous_hash = blockchain[i - 1].hash
    blockchain[i].hash = blockchain[i].compute_hash()

# Convert blockchain to JSON
blockchain_json = json.dumps([block.__dict__ for block in blockchain])

# Broadcast the modified blockchain
for peer in PEERS:
    try:
        response = requests.post(f"{peer}/receive_chain", json={"chain": blockchain_json})
        print(f"Sent to {peer}, Response: {response.text}")
    except requests.exceptions.RequestException as e:
```

print(f"Failed to send to {peer}: {e}")

The script ensures that the tampered data is accepted because the entire blockchain remains valid by updating the hashes.

Outputs:



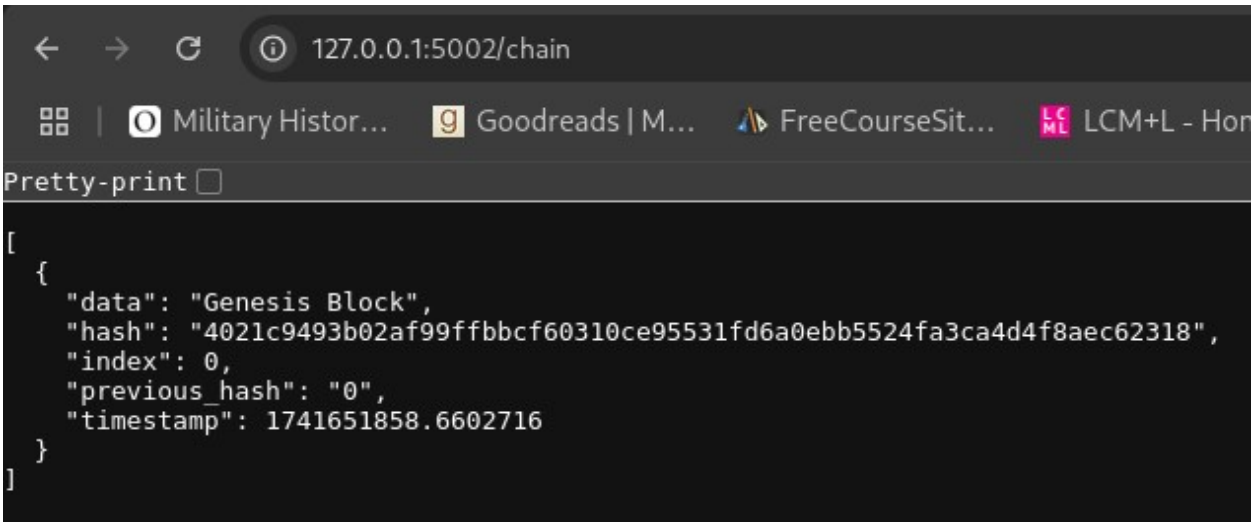


14. Modify the proof-of-work algorithm to change the number of required leading zeros in the hash.

- How does reducing the prefix affect performance?
- How does increasing the prefix affect performance?

Output:

```
┌──(venv)─(alpha㉿red)-[~/blockchain_assign]
└─$ python dissecting-blockchain/modified_pof.py
Difficulty: 2, Hash: 00b606adc19d37dda5780cf1116b75be9e3b9deacd71070f9c7e76f402294549, Time Taken: 0.0006 seconds
Difficulty: 4, Hash: 00003c6ccdc626c318d9151bf56a8f226f4f85a3f051cf484a90f76ca379e131, Time Taken: 0.1497 seconds
Difficulty: 6, Hash: 00000098037954f7f6bc71937a35f06522d89af46bddbc17b9edad0309c98451, Time Taken: 56.8148 seconds
```



Effects of Changing the Prefix Length

Reducing the Prefix (e.g., from 4 to 2 zeros)

- Performance: Faster block mining since fewer computations are needed.
- Security: Weaker security, as an attacker can manipulate blocks more easily.
- Increasing the Prefix (e.g., from 4 to 6 zeros)

Increasing the Prefix (e.g., from 4 to 6 zeros)

- Performance: Slower block mining due to increased computational effort.
- Security: Stronger security, as altering a block requires significantly more work.
- This demonstrates the trade-off between security and efficiency in proof-of-work blockchains.