Blockchain Hash Function

A hash function takes an input string (numbers, alphabets, media files) of any length and transforms it into a fixed length. The fixed bit length can vary (like 32-bit or 64-bit or 128-bit or 256-bit) depending on the hash function which is being used. The fixed-length output is called a hash. This hash is also the cryptographic byproduct of a hash algorithm.

The hash algorithm has certain unique properties: -

1. It produces a unique output (or hash).
2. It is a one-way function.

In the context of cryptocurrencies like Bitcoin, the blockchain uses this cryptographic hash function's properties in its consensus mechanism. A cryptographic hash is a digest or digital fingerprints of a certain amount of data. In cryptographic hash functions, the transactions are taken as

an input and run through a hashing algorithm which gives an output of a fixed size.

SHA-256: -

Secure Hashing Algorithm, or SHA. Data and certificates are hashed with SHA, a modified version of MD5. By using bitwise operations, modular additions, and compression functions, a hashing algorithm reduces the input data into a smaller form that is impossible to comprehend. Can hashing be cracked or decrypted, you may wonder? The main distinction between hashing and encryption is that hashing is one-way; once data has been hashed, the resultant hash digest cannot be decrypted unless a brute force assault is applied.

SHA is designed to provide a different hash even if only one character in the message changes. As an illustration, consider combining the themes Heaven and Heaven Is Different. The only difference between a capital and tiny letter, though, is size.

Program: -

```
import hashlib
import time

class Block:
    def __init__(self, index, previous_hash, data):
        self.index = index
        self.timestamp = time.time()
        self.data = data
        self.previous_hash = previous_hash
        self.hash = self.calculate_hash()

    def calculate_hash(self):
        data_str = str(self.index) + str(self.timestamp) + self.data +
self.previous_hash
```

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

class Blockchain:
    def __init__(self):
        self.chain = [self.create_genesis_block()]

    def create_genesis_block(self):
        return Block(0, "0", "Genesis Block")

    def get_latest_block(self):
        return self.chain[-1]

    def add_block(self, new_block):
        new_block.previous_hash = self.get_latest_block().hash
        new_block.hash = new_block.calculate_hash()
        self.chain.append(new_block)

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

    while True:
        user_data = input("Enter data for the new block (or 'q' to quit): ")

        if user_data.lower() == 'q':
            break

        blockchain.add_block(Block(len(blockchain.chain),
blockchain.get_latest_block().hash, user_data))

    # Print the blockchain
    for block in blockchain.chain:
        print(f"Block #{block.index}")
        print(f"Timestamp: {block.timestamp}")
        print(f"Data: {block.data}")
        print(f"Previous Hash: {block.previous_hash}")
        print(f"Hash: {block.hash}")
        print()
```

Output: -

```
Enter data for the new block (or 'q' to quit): a<b=10
Enter data for the new block (or 'q' to quit): b<c=12
Enter data for the new block (or 'q' to quit): a<d=15
Enter data for the new block (or 'q' to quit): c<b=11
Enter data for the new block (or 'q' to quit): q
Block #0
Timestamp: 1698898770.3222444
Data: Genesis Block
Previous Hash: 0
Hash: 79a616567d5963e2f05e9f64a457f7de897d9fe4557cc477b3a8877c5254d4d2

Block #1
Timestamp: 1698898794.772642
Data: a<b=10
Previous Hash: 79a616567d5963e2f05e9f64a457f7de897d9fe4557cc477b3a8877c5254d4d2
Hash: 8ee9f743a5f0e5c2a3b875185bd745fd9fb658ee0258f71fea3253e082f67b3c

Block #2
Timestamp: 1698898807.598189
Data: b<c=12
Previous Hash: 8ee9f743a5f0e5c2a3b875185bd745fd9fb658ee0258f71fea3253e082f67b3c
Hash: 3790072260e62c44b3fb3698af07f45e65d9e76c7fff0f5c78d947e8a14c85e0

Block #3
Timestamp: 1698898831.7833824
Data: a<d=15
Previous Hash: 3790072260e62c44b3fb3698af07f45e65d9e76c7fff0f5c78d947e8a14c85e0
Hash: d7cabf68f8974594aa8b60549be6a98246ce756cd0be975f4886d267b5bef824

Block #4
Timestamp: 1698898843.0989652
Data: c<b=11
Previous Hash: d7cabf68f8974594aa8b60549be6a98246ce756cd0be975f4886d267b5bef824
Hash: 3739973dd5f1c992e8fd2613f4dd16010f9cc98e6fa65409687be6019820af31
```