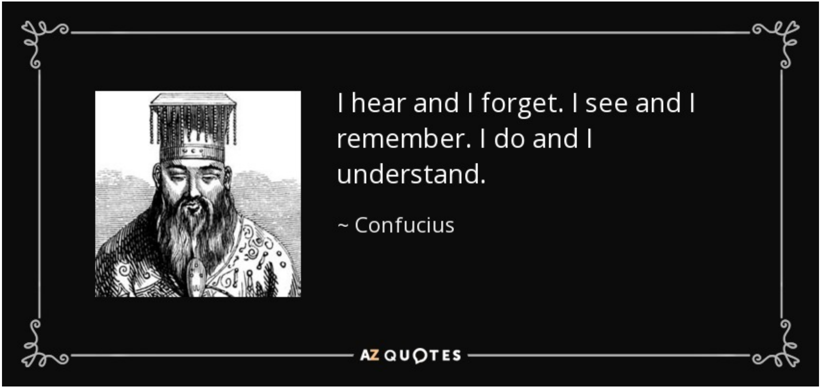
FINE 434: FinTech

Lecture 16

Professor Fahad Saleh

McGill University - Desautels

Learning By Doing



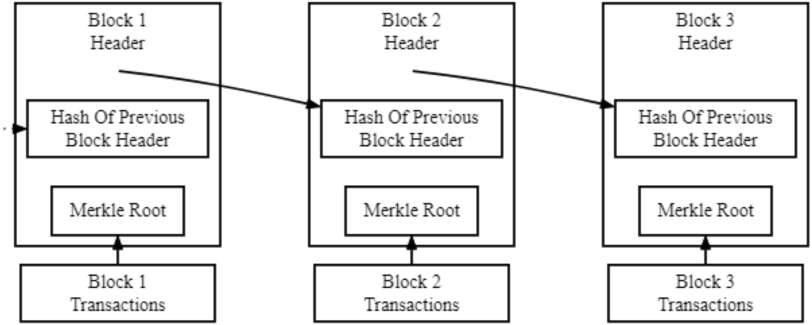
Overview

We will build a Proof-of-Work (PoW) Payments Blockchain

The Blockchain consists of Blocks

What do blocks consist of?

Blocks


Block 1
Header
Hash Of Previous
Block Header
Merkle Root
Block 1
Transactions
Block 2
Header
Hash Of Previous
Block Header
Merkle Root
Block 2
Transactions
Block 3
Header
Hash Of Previous
Block Header
Merkle Root
Block 3
Transactions

Block (Header) Hash

A Block Hash serves as a sufficient statistic for the entire blockchain up to that point

Therefore, it must include something that summarizes the new transactions - the Merkle Root

It must also include something that summarizes the history of transactions - the previous block hash

Block (Header) Hash

A Block Hash serves as a sufficient statistic for the entire blockchain up to that point

Therefore, it must include something that summarizes the new transactions - the Merkle Root

It must also include something that summarizes the history of transactions - the previous block hash

How do we determine if a block is valid?

Block (Header) Hash

A Block Hash serves as a sufficient statistic for the entire blockchain up to that point

Therefore, it must include something that summarizes the new transactions - the Merkle Root

It must also include something that summarizes the history of transactions - the previous block hash

How do we determine if a block is valid? PoW

Block Header

Table 2. The structure of the block header

Size	Field	Description
4 bytes	Version	A version number to track software/protocol upgrades
32 bytes	Previous Block Hash	A reference to the hash of the previous (parent) block in the chain
32 bytes	Merkle Root	A hash of the root of the merkle tree of this block's transactions
4 bytes	Timestamp	The approximate creation time of this block (seconds from Unix Epoch)
4 bytes	Difficulty Target	The Proof-of-Work algorithm difficulty target for this block
4 bytes	Nonce	A counter used for the Proof-of-Work algorithm

```python
import hashlib

class Blockchain(object):
    def __init__(self):
        # Initialize the blockchain

    def new_transaction(self, sender, recipient, amount):
        # Adds a new transaction to the list of transactions

    def mine(self, difficulty = 2**248):
        # Add new blocks to the blockchain

    def __repr__(self):
        # Special function for display purposes
```

Initializing the Blockchain

What do we need in _init_(self)?

Initializing the Blockchain

What do we need in _init_(self)?

How many blocks are there initially?

How many transactions are there?

init

```
import hashlib

class Blockchain(object):
    def __init__(self):
        # Initialize the blockchain
        self.chain = [] # initial chain
        self.current_transactions = [] # initial set of transactions
```

Adding Transactions

Transactions must be valid.

Validation involves a Digital Signature Algorithm (see Lectures 8 - 11)

We will overlook validation and assume transaction validity

What's left to do?

Adding Transactions

Transactions must be valid.

Validation involves a Digital Signature Algorithm (see Lectures 8 - 11)

We will overlook validation and assume transaction validity

What's left to do?

Record Transactions

new_transaction

```
def new_transaction(self, sender, recipient, amount):
    # Adds a new transaction to the list of transactions
    tx = {"sender": sender,
          "recipient": recipient,
          "amount": amount}  #store tx as a dictionary
    self.current_transactions.append(tx) # add tx to tx list
    return(None)
```

Note: This is a simplification.

Adding Blocks

Blocks must be valid.

Validation requires that underlying transactions must be valid and not double spend

We overlook both requirements and focus on block validity separate from that (i.e., does the block solve the PoW puzzle?)

Therefore, our task is to “mine” for a new block...

mine

```python
def mine(self, difficulty = 2**248):
    # Add new blocks to the blockchain
    self.new_transaction("None","Me",1) # Block Reward
    previous_hash = "None" # default value for previous hash if first block
    if len(self.chain) > 0: # use previous hash if there is a previous block
        previous_hash = hashlib.sha256(str(self.chain[-1]).encode()).hexdigest()
    proposed_block = {'index': len(self.chain),
              "transactions": self.current_transactions,
              "nonce": 0,
              "previous_hash": previous_hash} # store block as a dictionary
    while int(hashlib.sha256(str(proposed_block).encode()).hexdigest(),16) > difficulty:
        proposed_block["nonce"] = proposed_block["nonce"] + 1
    self.chain.append(proposed_block) # add valid block to blockchain
    self.current_transactions = [] # reset new transactions
    return(proposed_block)
```

One More Thing...

It would be nice to see that this actually works...

```
b = Blockchain()
print(b)
```

We want the last line to give us something readable.

One More Thing...

It would be nice to see that this actually works...

b = Blockchain()
print(b)

We want the last line to give us something readable.

repr is a special function that returns a str such that print(b) actually produces print(b._repr_())

Therefore, we want to write _repr_ to return a human-readable str that reveals the blockchain's content

repr

```
def __repr__(self):
    # Special function for display purposes
    representation = ""
    for block in self.chain: # Loop over all blocks
        representation = representation + hashlib.sha256(str(block).encode()).hexdigest()+"\n"+ str(block) +"\n\n"
        ## Add the block's hash and the block header's content to the display
    return(representation)
```

This code loops through the chain and concatenates the block hash and block header for each block sequentially

Creating a Blockchain Instance

```
b = Blockchain()
b.new_transaction("Alice","Bob",1)
b.mine()
```

Creating a Blockchain Instance

```
b = Blockchain()
b.new_transaction("Alice","Bob",1)
b.mine()
```

... but does it work?

Moment of Truth

```
b = Blockchain()
b.new_transaction("Alice","Bob",1)
b.mine()
print(b)
```

```
00071cabffa1b0e59ceb5243c5d0e1cd89660fb4ad55b0b4496c0009599f69d3
{'index': 0, 'transactions': [{'sender': 'Alice', 'recipient': 'Bob', 'amount': 1}, {'sender': 'None', 'recipient': 'Me', 'amou
nt': 1}], 'nonce': 110, 'previous_hash': 'None'}
```

Does this look correct?

More

```
b = Blockchain()
b.new_transaction("Alice","Bob",1)
b.mine()
b.mine()
b.new_transaction("Bob","Alice",1)
b.new_transaction("Chris","Alice",12)
b.mine()
b.mine()
print(b)
```

```
00071cabffa1b0e59ceb5243c5d0e1cd89660fb4ad55b0b4496c0009599f69d3
{'index': 0, 'transactions': [{'sender': 'Alice', 'recipient': 'Bob', 'amount': 1}, {'sender': 'None', 'recipient': 'Me', 'amou
nt': 1}], 'nonce': 110, 'previous_hash': 'None'}

00d8c18e8dcd81c7f46f6d5a368edee2ca49b497cc22a1bdf0ddc834a1f4e801
{'index': 1, 'transactions': [{'sender': 'None', 'recipient': 'Me', 'amount': 1}], 'nonce': 92, 'previous_hash': '00071cabffa1b
0e59ceb5243c5d0e1cd89660fb4ad55b0b4496c0009599f69d3'}

0086147b22b61cff1c8b54fa6aa0d95371c7ae1a7109dcd3b90b60003bb1715f
{'index': 2, 'transactions': [{'sender': 'Bob', 'recipient': 'Alice', 'amount': 1}, {'sender': 'Chris', 'recipient': 'Alice',
'amount': 12}, {'sender': 'None', 'recipient': 'Me', 'amount': 1}], 'nonce': 143, 'previous_hash': '00d8c18e8dcd81c7f46f6d5a368
edee2ca49b497cc22a1bdf0ddc834a1f4e801'}

00d6afd6ec1d19b5258810e84e659746cfc5874dd2ce3fcb391c2b6a0b38c804
{'index': 3, 'transactions': [{'sender': 'None', 'recipient': 'Me', 'amount': 1}], 'nonce': 258, 'previous_hash': '0086147b22b6
1cff1c8b54fa6aa0d95371c7ae1a7109dcd3b90b60003bb1715f'}
```

Does this look correct?

This is an important lecture!

Make sure you understand everything in this lecture with specific attention to how to modify the code for different environments.

This is a rich testing ground... and the conceptual center of the course.

Questions for <u>You</u>

How would you know if I tampered with a blockchain instance?

Can you identify the first point of tampering if such an instance exists?

Can you verify if I send you a block without the appropriate "Proof-of-Work"?

How would the code differ if we required a different "Proof-of-Work" or a different consensus protocol entirely?

How would the code differ if we needed to verify transactions?