(../../../../author/adil-moujahid.html)

Adil Moujahid

Buy Me a Coffee (https://ko-fi.com/W7W4K1LZ)

Follow @AdilMouja

Published

Wed 14 March 2018

←Home (/)

# A Practical Introduction to Blockchain with Python

// tags python (../../../../tag/python/) blockchain (../../../../tag/blockchain/)

Blockchain is arguably one of the most significant and disruptive technologies that came into existence since the inception of the Internet. It's the core technology behind Bitcoin and other crypto-currencies that drew a lot of attention in the last few years.

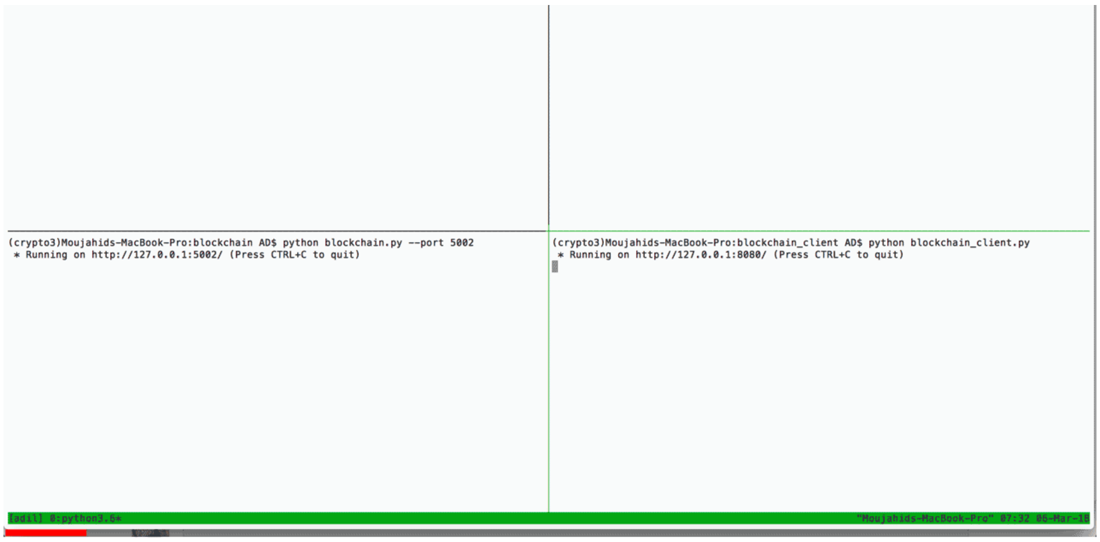
As its core, a blockchain is a distributed database that allows direct transactions between two parties without the need of a central authority. This simple yet powerful concept has great implications for various institutions such as banks, governments and marketplaces, just to name a few. Any business or organization that relies on a centralized database as a core competitive advantage can potentially be disrupted by blockchain technology.

Putting aside all the hype around the price of Bitcoin and other cryptocurrencies, the goal of this blog post is to give you a practical introduction to blockchain technology. Sections 1 and 2 cover some core concepts behind blockchain, while section 3 shows how to implement a blockchain using Python. We will also implement 2 web applications to make it easy for end users to interact with our blockchain.

Please note that I'm using Bitcoin here as a medium for explaning the more general technology of "Blockchain", and most of the concepts described in this post are applicable to other blockchain use cases and crypto-currencies.

Below is an animated gif of the two web apps that we will build in section 3.

Go Top



```
(crypto3)Moujahids-MacBook-Pro:blockchain AD$ python blockchain.py --port 5002
 * Running on http://127.0.0.1:5002/ (Press CTRL+C to quit)
```

```
(crypto3)Moujahids-MacBook-Pro:blockchain_client AD$ python blockchain_client.py
 * Running on http://127.0.0.1:8080/ (Press CTRL+C to quit)
```

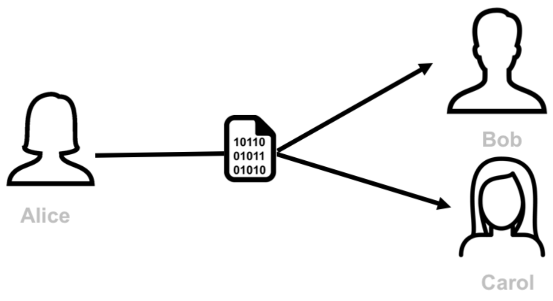
```
[adil] 0:python3.6*                                "Moujahids-MacBook-Pro" 07:32 06-Mar-18
```

# 1. A Crash Course in Blockchain

It all started with a white paper released in 2008 by an unknown person or entity using the name Satoshi Nakamoto. The white paper was titled “Bitcoin: A Peer-to-Peer Electronic Cash System” (https://bitcoin.org/bitcoin.pdf) and it laid the foundation of what later became known as Blockchain. In the original Bitcoin white paper, Satoshi described how to build a peer-to-peer electronic cash system that allows online payments to be sent directly from one party to another without going through a centralized institution. This system solves an important problem in digital money called double-spending.

## 1.1. What is Double-Spending?

Suppose that Alice wants to pay Bob 1$. If Alice and Bob use physical cash, then Alice will not longer have the 1$ after the transaction is executed. If Alice and Bob use digital money, then the problem gets more complicated. Digital money is in digital form and can be easily duplicated. If Alice sends a digital file worth 1$ to Bob by email for example, Bob cannot know for sure if Alice has deleted her copy of the file. If Alice still has the 1$ digital file, then she can choose to send the same file to Carol. This problem is called double-spending.



**Double-Spending Problem:** If Alice sends money in digital format to Bob, Bob cannot know for sure if Alice has deleted her copy of the file and she can choose to send the same file to Carol.

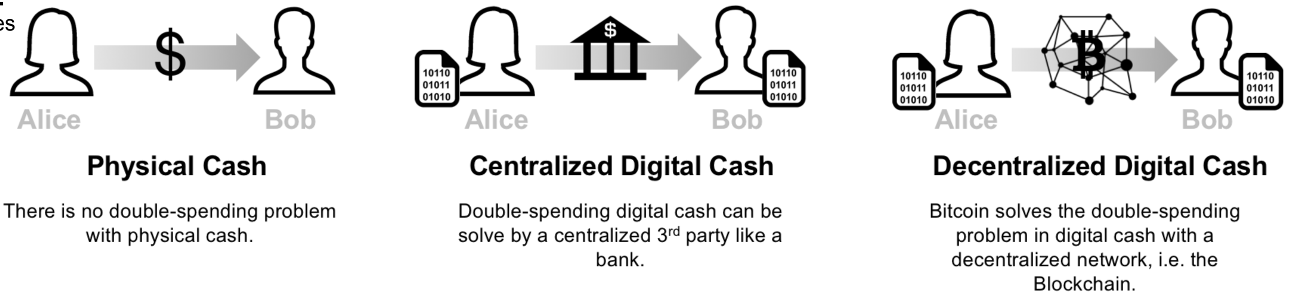
One way of solving the double-spending problem is to have a trusted third party (a bank for example) between Alice, Bob and all other participants in the network. This third party is responsible for managing a centralized ledger that keeps track of and validates all the transactions in the network. The drawback of this solution is that for the system to function, it requires trust in a centralized third party.

## 1.2. Bitcoin: A Decentralized Solution for the Double-Spending Problem

To solve the double-spending problem, Satoshi proposed a public ledger, i.e., Bitcoin’s blockchain to keep track of all transactions in the network. Bitcoin’s blockchain has the following characteristics:

Go Top

the transactions are added to the blockchain.
- Immutable: The blockchain can be changed in append only fashion. In other words, transactions can only be added to the blockchain but cannot be deleted or modified.
- Uses Proof of Work (PoW): A special type of participants in the network called miners compete on searching for the solution to a cryptographic puzzle that will allow them to add a block of transactions to Bitcoin's blockchain. This process is called Proof of Work and it allows the system to be secure (more on this later).


Alice
Bob
Physical Cash
There is no double-spending problem with physical cash.
Alice
Bob
Centralized Digital Cash
Double-spending digital cash can be solve by a centralized 3rd party like a bank.
Alice
Bob
Decentralized Digital Cash
Bitcoin solves the double-spending problem in digital cash with a decentralized network, i.e. the Blockchain.

Sending bitcoin money goes as follows:

- Step 1 (one-time effort): Create a bitcoin wallet. For a person to send or receive bitcoins, she needs to create a bitcoin wallet. A bitcoin wallet stores 2 pieces of information: A private key and a public key. The private key is a secret number that allows the owner to send bitcoin to another user, or spend bitcoins on services that accept them as payment method. The public key is a number that is needed to receive bitcoins. The public key is also referred to as bitcoin address (not entirely true, but for simplicity we will assume that the public key and the bitcoin address are the same). Note that the wallet doesn't store the bitcoins themselves. Information about bitcoins balances are stored on the Bitcoin's blockchain.
- Step 2: Create a bitcoin transaction. If Alice wants to send 1 BTC to Bob, Alice needs to connect to her bitcoin wallet using her private key, and create a transaction that contains the amount of bitcoins she wants to send and the address where she wants to send them (in this case Bob's public address).
- Step 3: Broadcast the transaction to Bitcoin's network. Once Alice creates the bitcoin transaction, she needs to broadcast this transaction to the entire Bitcoin's network.
- Step 4: Confirm the transaction. A miner listening to Bitcoin's network authenticates the transaction using Alice's public key, confirms that Alice has enough bitcoins in her wallet (in this case at least 1 BTC), and adds a new record to Bitcoin's Blockchain containing the details of the transaction.
- Step 5: Broadcast the blockchain change to all miners. Once the transaction is confirmed, the miner should broadcast the blockchain change to all miners to make sure that their copies of the blockchain are all in sync.

# 2. A Technical Deep Dive on Blockchain

The goal of this section is to go deeper into the technical building blocks that power the blockchain. We will cover public key cryptography, hashing functions, mining and security of the blockchain.

## 2.1. Public Key Cryptography

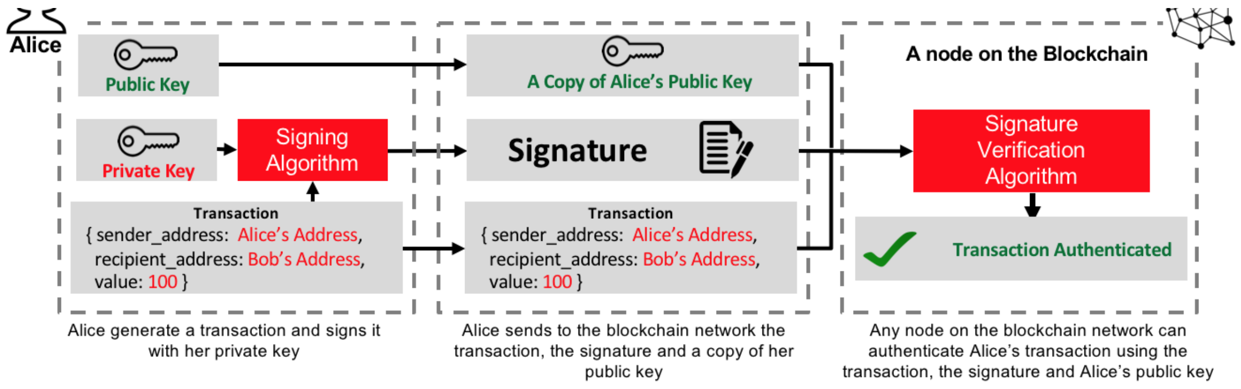
Public-key cryptography, or asymmetrical cryptography, is any cryptographic system that uses pairs of keys: public keys which may be disseminated widely, and private keys which are known only to the owner. This accomplishes two functions: authentication, where the public key verifies a holder of the paired private key sent the message, and encryption, where only the paired private key holder can decrypt the message encrypted with the public key. [1]

RSA (https://en.wikipedia.org/wiki/RSA_(cryptosystem)) and Elliptic Curve Digital Signature (ECDSA) (https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm) are the most popular public-key cryptography algorithms.

In the case of Bitcoin, ECDSA algorithm is used to generate Bitcoin wallets. Bitcoin uses a variety of keys and addresses, but for the sake of simplicity, we will assume in this blog post that each Bitcoin wallet has one pair of private/public keys and that a Bitcoin address is the wallet's public key. I recommend this article (https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses), if you're interested in the complete technical details of Bitcoin wallets.

To send or receive BTCs, a user starts by generating a wallet which contains a pair of private and public keys. If Alice wants to send Bob some BTCs, she creates a transaction in which she enters both her and Bob's public keys, and the amount of BTCs she wants to send. She then sign the transaction using her private key. A computer on the blockchain uses Alice's public key to verify that the

Go Top


Alice
Public Key
Private Key
Signing Algorithm
Transaction
{ sender_address: Alice's Address, recipient_address: Bob's Address, value: 100 }
Alice generate a transaction and signs it with her private key
A Copy of Alice's Public Key
Signature
Alice sends to the blockchain network the transaction, the signature and a copy of her public key
A node on the Blockchain
Signature Verification Algorithm
Transaction Authenticated
Any node on the blockchain network can authenticate Alice's transaction using the transaction, the signature and Alice's public key

**Authentication Process for Transactions on the Blockchain**

## 2.2. Hashing Functions and Mining

All Bitcoin transactions are grouped in files called blocks. Bitcoin adds a new block of transactions every 10 minutes. Once a new block is added to the blockchain, it becomes immutable and can't be deleted or modified. A special group of participants in the network called miners (computers connected to the blockchain) are responsible for creating new blocks of transactions. A miner has to authenticate each transaction using the sender's public key, confirm that the sender has enough balance for the requested transaction, and add the transaction to the block. Miners are completely free to choose which transactions to include in the blocks, therefore the senders need to include a transaction fee to incentivise the miners to add their transactions to the blocks.

For a block to be accepted by the blockchain, it needs to be "mined". To mine a block, miners need to find an extremely rare solution to a cryptographic puzzle. If a mined block is accepted by the blockchain, the miner receive a reward in bitcoins which is an additional incentive to transaction fees. The mining process is also referred to as Proof of Work (PoW), and it's the main mechanism that enables the blockchain to be trustless and secure (more on blockchain security later).

### Hashing and Blockchain's Cryptographic Puzzle

To understand the blockchain's cryptographic puzzle, we need to start with hash functions. A hash function is any function that can be used to map data of arbitrary size to data of fixed size. The values returned by a hash function are called hashes. Hash functions are usually used to accelerate database lookup by detecting duplicated records, and they are also widely used in cryptography. A cryptographic hash function allows one to easily verify that some input data maps to a given hash value, but if the input data is unknown, it is deliberately difficult to reconstruct it by knowing the stored hash value. [2]

Bitcoins uses a cryptographic hash function called SHA-256. SHA-256 is applied to a combination of the block's data (bitcoin transactions) and a number called nonce. By changing the block data or the nonce, we get completely different hashes. For a block to be considered valid or "mined", the hash value of the block and the nonce needs to meet a certain condition. For example, the four leading digits of the hash needs to be equal to "0000". We can increase the mining complexity by making the condition more complex, for example we can increase the number of 0s that the hash value needs to start with.

The cryptograhic puzzle that miners need to solve is to find a nonce value that makes the hash value satisfies the mining condition. You can use the app below to simulate block mining. When you type in the "Data" text box or change the nonce value, you can notice the change in the hash value. When you click the "Mine" button, the app starts with a nonce equals to zero, computes the hash value and checks if the leading four digits of the hash value is equal to "0000". If the leading four digits are not equal to "0000", it increments the nonce by one and repeats the whole process until it finds a nonce value that satisfy the condition. If the block is considered mined, the background color turns green.

**Nonce:**

72608

**Data:**

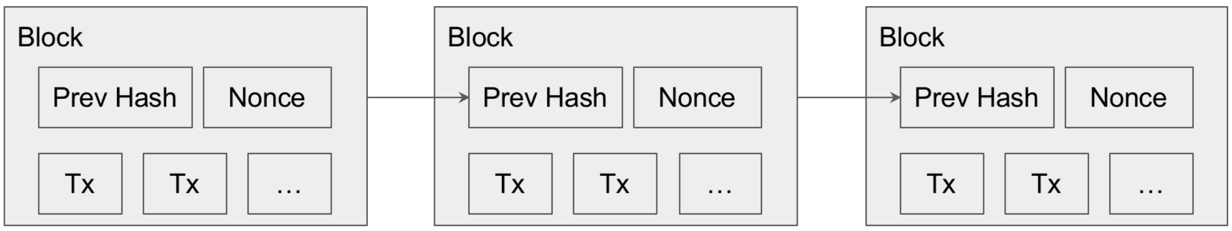
Go Top

Mine

## 2.3. From Blocks to Blockchain

As discussed in the previous section, transactions are grouped in blocks and blocks are appended to the blockchain. In order to create a chain of blocks, each new block uses the previous block's hash as part of its data. To create a new block, a miner selects a set of transactions, adds the previous block's hash and mines the block in a similar fashion described above.

Any changes to the data in any block will affect all the hash values of the blocks that come after it and they will become invalid. This give the blockchain its immutability characteristic.



Blocks are chained together using the previous block's hash to form a Blockchain.

You can use the app below to simulate a blockchain with 3 blocks. When you type in the "Data" text box or change the nonce value, you can notice the change in the hash value and the "Prev" value (previous hash) of the next block. You can simulate the mining process by clicking on the "Mine" button of each individual block. After mining the 3 blocks, try changing the data in block 1 or 2, and you will notice that all the blocks that come after become invalid.

**Block:**

| | |
|---|---|
| # | 1 |

**Nonce:**

1131

**Data:**

Mohammad

**Prev:**

0000000000000000000000000000000000000000000000000000000000000000

**Hash:**

6322bfdc3e78a708261ede40c961b0bb283c753011fe9bf2acd126947221d1ba

Mine

**Nonce:**

35230

**Data:**

**Prev:**

6322bfdc3e78a708261ede40c961b0bb283c753011fe9bf2acd126947221d1ba

**Hash:**

d4bb1bbe8ed296b42f344202c6e9176e0f5ba961239d09b5614cb62bf6ec3899

Mine

**Block:**

| # | 3 |
|---|---|

**Nonce:**

12937

**Data:**

**Prev:**

d4bb1bbe8ed296b42f344202c6e9176e0f5ba961239d09b5614cb62bf6ec3899

**Hash:**

4052a226d8a08c04b67c85e9bbadb6bf24621d76dd799f80ad421c41e04d3c6a

Mine

Both mining simulators above were adapted from Anders Brownworth's excellent Blockchain Demo (https://anders.com/blockchain/blockchain.html).

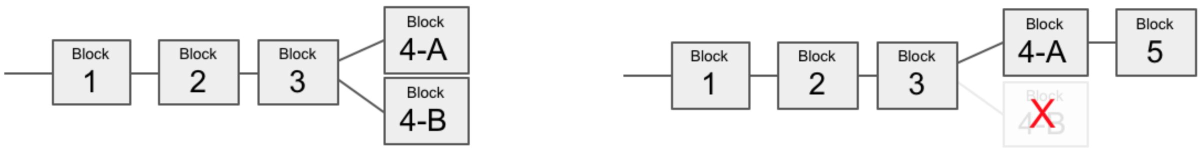
## 2.4. Adding Blocks to the Blockchain

Go Top

receives the reward in bitcoins. But what happens if two miners or more submit their blocks at the same time?

## Resolving Conflicts

If 2 miners solve a block at almost the same time, then we will have 2 different blockchains in the network, and we need to wait for the next block to resolve the conflict. Some miners will decide to mine on top of blockchain 1 and others on top of blockchain 2. The first miner to find a new block resolves the conflict. If the new block was mined on top of blockchain 1, then blockchain 2 becomes invalid, the reward of the previous block goes to the miner from blockchain 1 and the transactions that were part of blockchain 2 and weren't added to the blockchain go back to the transactions pool and get added to the next blocks. In short, if there is a conflict on the blockchain, then the the longest chain wins.



Resolving conflicts - The longest chain wins

## 2.5. Blockchain and Double-Spending

In this section, we will cover the most popular ways for performing double-spending attacks on the blockchain, and the measures that users should take to prevent damages from them.

### Race Attack

An attacker sends the same coin in rapid succession to two different addresses. To prevent from this attack, it is recommended to wait for at least one block confirmation before accepting the payment. [3]

### Finney Attack

An attacker pre-mines a block with a transaction, and spends the same coins in a second transaction before releasing the block. In this scenario, the second transaction will not be validated. To prevent from this attack, it is recommended to wait for at least 6 block confirmations before accepting the payment. [3]

### Majority Attack (also called 51% attack)

In this attack, the attacker owns 51% of the computing power of the network. The attacker starts by making a transaction that is brodcasted to the entire network, and then mines a private blockchain where he double-spends the coins of the previous transaction. Since the attacker owns the majority of the computing power, he is guaranteed that he will have at some point a longer chain than the "honest" network. He can then release his longer blockchain that will replace the "honest" blockchain and cancel the original transaction. This attack is highly unlikely, as it's very expensive in blockchain networks like Bitcoin. [4]

# 3. A Blockchain Implementation in Python

In this section, we will implement a basic blockchain and a blockchain client using Python. Our blockchain will have the following features:

- Possibility of adding multiple nodes to the blockchain
- Proof of Work (PoW)
- Simple conflict resolution between nodes
- Transactions with RSA encryption

Our blockchain client will have the following features:

Go Top

- Wallets generation using Public/Private key encryption (based on RSA algorithm)
- Generation of transactions with RSA encryption

• Blockchain Client for users to generate wallets and send coins

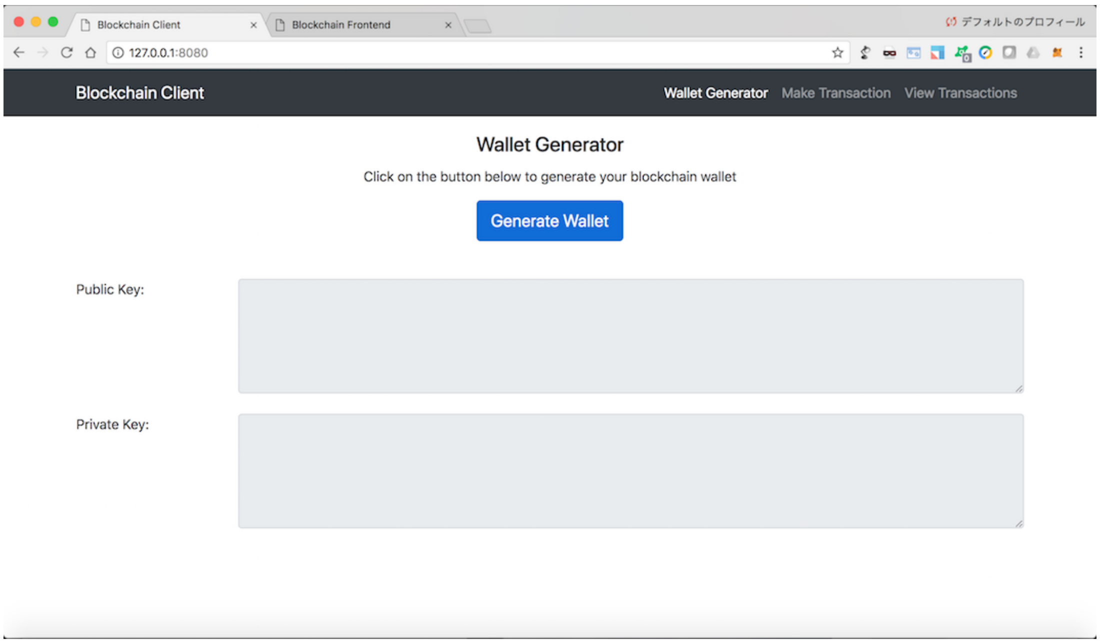
The blockchain implementation is mostly based on this github project (https://github.com/dvf/blockchain). I made a few modifications to the original code in order to add RSA encryption to the transactions. Wallet generation and transaction encryption is based on this Jupyter notebook (https://github.com/julienr/ipynb_playground/blob/master/bitcoin/dumbcoin/dumbcoin.ipynb). The 2 dashboard are implemented from scratch using HTML/CSS/JS.

You can download the complete source code from https://github.com/adilmoujahid/blockchain-python-tutorial (https://github.com/adilmoujahid/blockchain-python-tutorial).

Please note that this implementation is for educational purposes only and shouldn't be use in production as it doesn't have good security, doesn't scale well and lacks many important features.

## 3.1. Blockchain Client Implementation

You can start the blockchain client from the terminal by going to the `blockchain_client` folder, and typing `python blockchain_client.py` . In your browser, go to `http://localhost:8080` and you'll see the dashboard below.



The dashboard has 3 tabs in the navigation bar:

- Wallet Generator: To generate wallets (Public/Private keys pair) using RSA encryption algorithm
- Make Transaction: To generate transactions and send them to a blockchain node
- View Transasctions: To view the transactions that are on the blockchain

In order to make or view transactions, you will need at least one blockchain node running (to be covered in next section).

Below is some explanation of the most important parts in the `blockchain_client.py` code.

We define a python class that we name `Transaction` that has 4 attributes `sender_address` , `sender_private_key` , `recipient_address` , `value` . These are the 4 pieces of information that a sender needs to create a transaction.

The `to_dict()` method returns the transaction information in a Python dictionary format (without the sender's private key). The `sign_transaction()` method takes the transaction information (without the sender's private key) and signs it using the sender's private key.

Go Top

```
        self.sender_address = sender_address
        self.sender_private_key = sender_private_key
        self.recipient_address = recipient_address
        self.value = value

    def __getattr__(self, attr):
        return self.data[attr]

    def to_dict(self):
        return OrderedDict({'sender_address': self.sender_address,
                            'recipient_address': self.recipient_address,
                            'value': self.value})

    def sign_transaction(self):
        """
        Sign transaction with private key
        """
        private_key = RSA.importKey(binascii.unhexlify(self.sender_private_key))
        signer = PKCS1_v1_5.new(private_key)
        h = SHA.new(str(self.to_dict()).encode('utf8'))
        return binascii.hexlify(signer.sign(h)).decode('ascii')
```

Tne line below initate a `Python Flask` app that we will use to create different APIs to interact with the blockchain and its client.

```
pp = Flask(__name__)
```

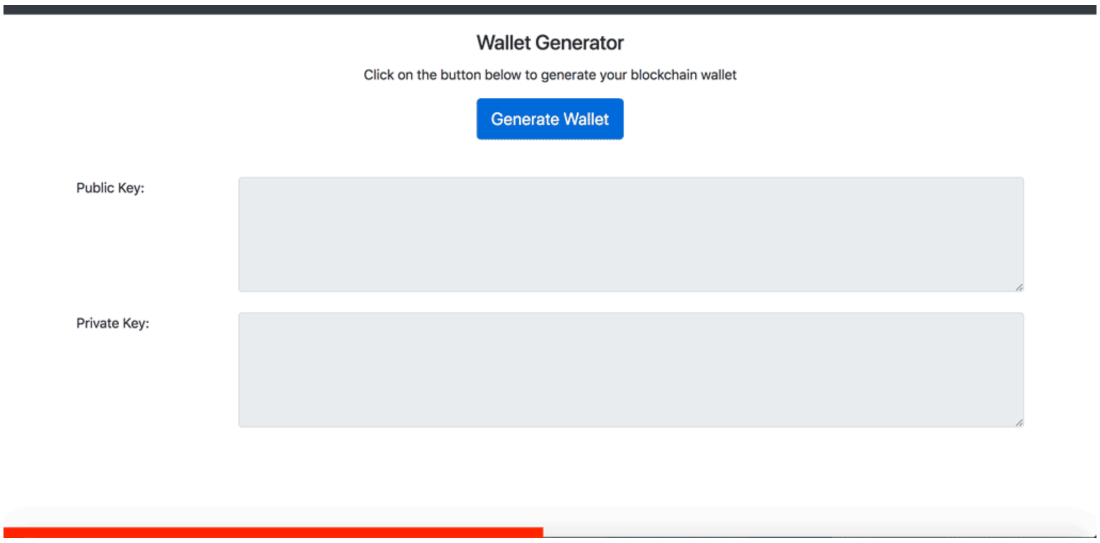
Below we define the 3 `Flask` routes that returns html pages. One html page for each tab.

```
@app.route('/')
def index():
  return render_template('./index.html')

@app.route('/make/transaction')
def make_transaction():
    return render_template('./make_transaction.html')

@app.route('/view/transactions')
def view_transaction():
    return render_template('./view_transactions.html')
```

Below we define an API that generates wallets (Private/Public keys pairs).

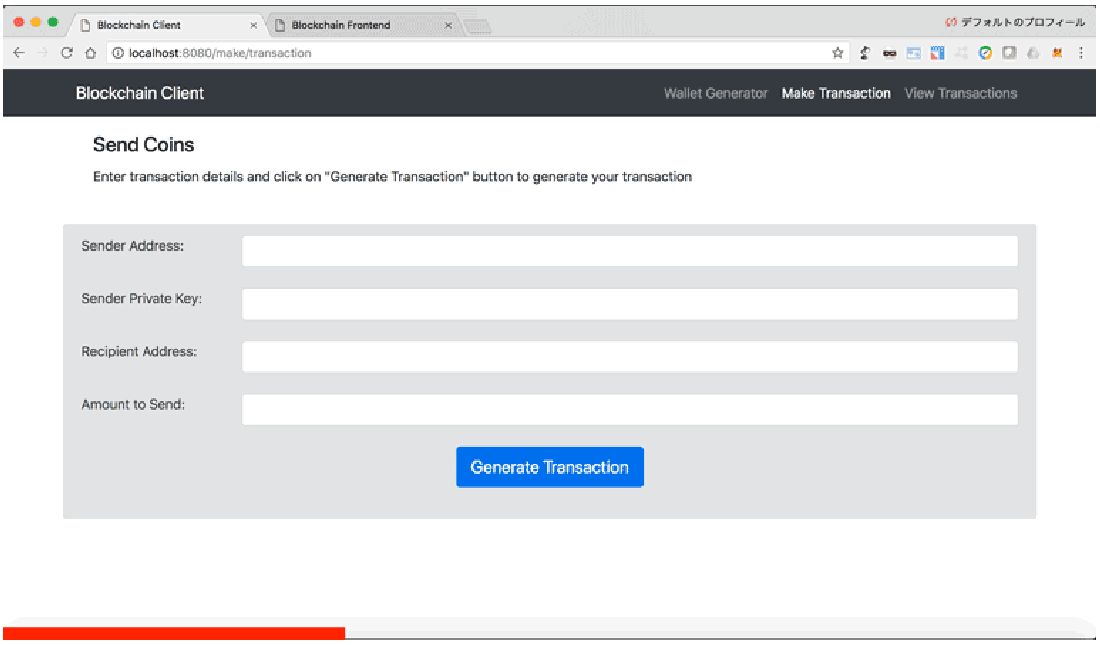
```
@app.route('/wallet/new', methods=['GET'])
def new_wallet():
  random_gen = Crypto.Random.new().read
  private_key = RSA.generate(1024, random_gen)
  public_key = private_key.publickey()
  response = {
    'private_key': binascii.hexlify(private_key.exportKey(format='DER')).decode('ascii'),
    'public_key': binascii.hexlify(public_key.exportKey(format='DER')).decode('ascii')
  }

  return jsonify(response), 200
```

Go Top


Wallet Generator
Click on the button below to generate your blockchain wallet
Generate Wallet
Public Key:
Private Key:

Below we define an API that takes as input `sender_address` , `sender_private_key` , `recipient_address` , `value` , and returns the transaction (without private key) and the signature.

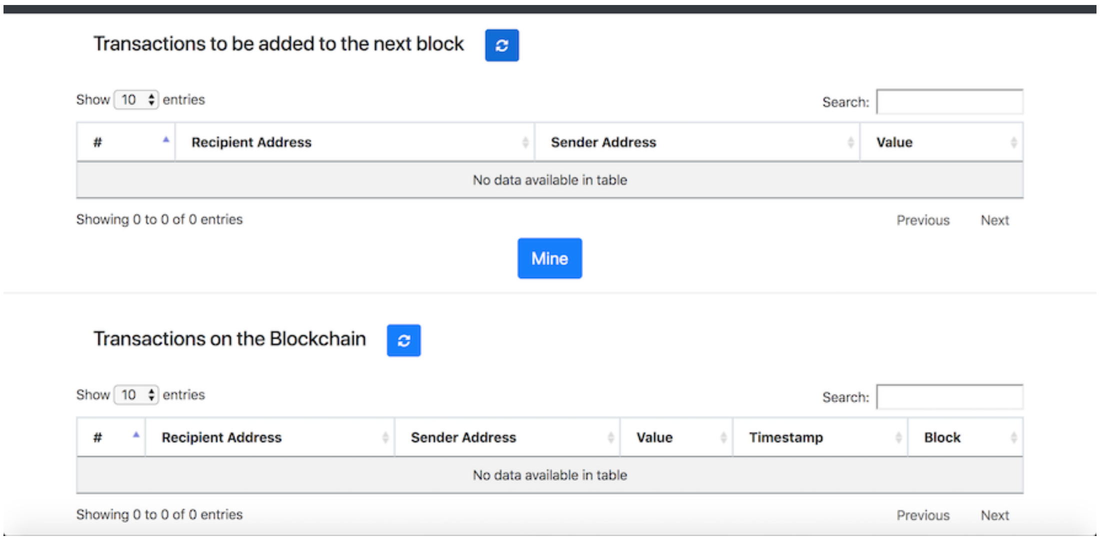
```
@app.route('/generate/transaction', methods=['POST'])
def generate_transaction():

  sender_address = request.form['sender_address']
  sender_private_key = request.form['sender_private_key']
  recipient_address = request.form['recipient_address']
  value = request.form['amount']

  transaction = Transaction(sender_address, sender_private_key, recipient_address, value)

  response = {'transaction': transaction.to_dict(), 'signature': transaction.sign_transaction()}

  return jsonify(response), 200
```



## 3.2. Blockchain Implementation

You can start a blockchain node from the terminal by going to the `blockchain` folder, and type `python blockchain_client.py` or `python blockchain_client.py -p <PORT NUMBER>` . If you don't specify a port number, it will default to port 5000. In your browser, go to `http://localhost:<PORT NUMBER>` to see the blockchain frontend dashboard.

Go Top


Transactions to be added to the next block

Show 10 entries Search:

| # | Recipient Address | Sender Address | Value |
|---|---|---|---|
| No data available in table | | | |

Showing 0 to 0 of 0 entries Previous Next

Transactions on the Blockchain

Show 10 entries Search:

| # | Recipient Address | Sender Address | Value | Timestamp | Block |
|---|---|---|---|---|---|
| No data available in table | | | | | |

Showing 0 to 0 of 0 entries Previous Next

The dashboard has 2 tabs in the navigation bar:

- Mine: For viewing transactions and blockchain data, and for mining new blocks of transactions.
- Configure: For configuring connections between the different blockchain nodes.

Below is some explanation of the most important parts in the `blockchain.py` code.

We start by defining a `Blockchain` class that has the following attributes:

- `transactions` : List of transactions that will be added to the next block.
- `chain` : The actual blockchain which is an array of blocks.
- `nodes` : A set containing node urls. The blockchain uses these nodes to retrieve blockchain data from other nodes and updates its blockchain if they're not in sync.
- `node_id` : A random string to identify the blockchain node.

The `Blockchain` class also implements the following methods:

- `register_node(node_url)` : Adds a new blockchain node to the list of nodes.
- `verify_transaction_signature(sender_address, signature, transaction)` : Checks that the provided signature corresponds to transaction signed by the public key (sender_address).
- `submit_transaction(sender_address, recipient_address, value, signature)` : Adds a transaction to list of transactions if the signature verified.
- `create_block(nonce, previous_hash)` : Adds a block of transactions to the blockchain.
- `hash(block)` : Create a SHA-256 hash of a block.
- `proof_of_work()` : Proof of work algorithm. Looks for a nonce that satisfies the mining condition.
- `valid_proof(transactions, last_hash, nonce, difficulty=MINING_DIFFICULTY)` : Checks if a hash value satisfies the mining conditions. This function is used within the proof_of_work function.
- `valid_chain(chain)` : checks if a bockchain is valid.
- `resolve_conflicts()` : Resolves conflicts between blockchain's nodes by replacing a chain with the longest one in the network.

Go Top

```
        self.transactions = []
        self.chain = []
        self.nodes = set()
        #Generate random number to be used as node_id
        self.node_id = str(uuid4()).replace('-', '')
        #Create genesis block
        self.create_block(0, '00')

    def register_node(self, node_url):
        """
        Add a new node to the list of nodes
        """
        ...

    def verify_transaction_signature(self, sender_address, signature, transaction):
        """
        Check that the provided signature corresponds to transaction
        signed by the public key (sender_address)
        """
        ...

    def submit_transaction(self, sender_address, recipient_address, value, signature):
        """
        Add a transaction to transactions array if the signature verified
        """
        ...

    def create_block(self, nonce, previous_hash):
        """
        Add a block of transactions to the blockchain
        """
        ...

    def hash(self, block):
        """
        Create a SHA-256 hash of a block
        """
        ...

    def proof_of_work(self):
        """
        Proof of work algorithm
        """
        ...

    def valid_proof(self, transactions, last_hash, nonce, difficulty=MINING_DIFFICULTY):
        """
        Check if a hash value satisfies the mining conditions. This function is used within the proof_of_work function.
        """
        ...

    def valid_chain(self, chain):
        """
        check if a bockchain is valid
        """
        ...

    def resolve_conflicts(self):
        """
        Resolve conflicts between blockchain's nodes
        by replacing our chain with the longest one in the network.
        """
        ...
```

The line below initate a `Python Flask` app that we will use to create different APIs to interact with the blockchain.

```
app = Flask(__name__)
CORS(app)
```

Next, we initiate a Blockchain instance.

```
blockchain = Blockchain()
```

Below we define the 2 `Flask` routes that return the html pages for our blockchain frontend dashboard.

Go Top

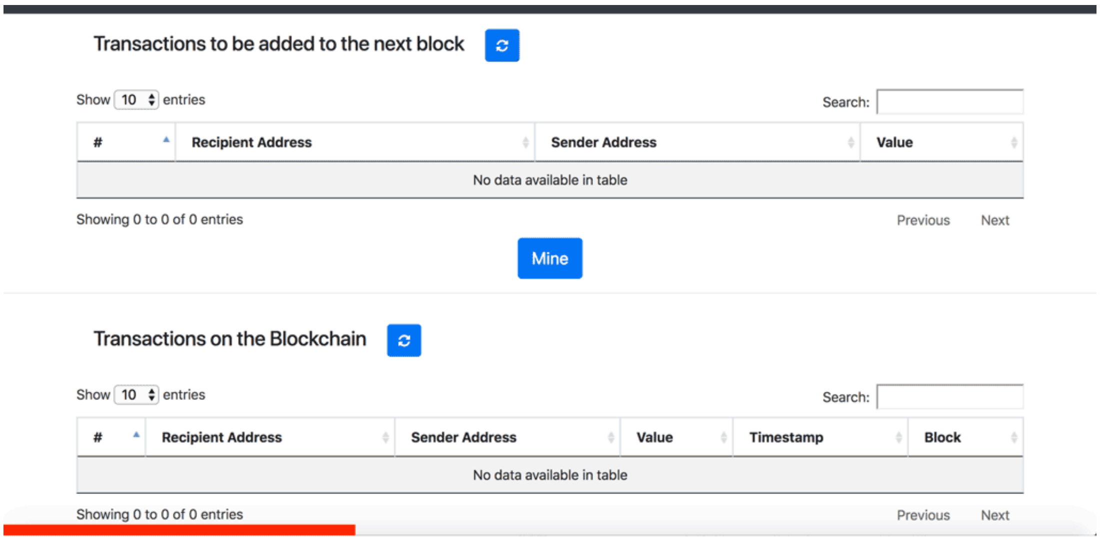
```
@app.route('/configure')
def configure():
    return render_template('./configure.html')
```

Below we define `Flask` APIs to manage transactions and mining the blockchain.

- `'/transactions/new'` : This API takes as input `'sender_address'` , `'recipient_address'` , `'amount'` and `'signature'` , and adds the transaction to the list of transactions that will be added to next block if the signature is valid.
- `'/transactions/get'` : This API returns all the transactions that will be added to the next block.
- `'/chain'` : This API returns all blockchain data.
- `'/mine'` : This API runs the proof of work algorithm, and adds the new block of transactions to the blockchain.

```
@app.route('/transactions/new', methods=['POST'])
def new_transaction():
    values = request.form

    # Check that the required fields are in the POST'ed data
    required = ['sender_address', 'recipient_address', 'amount', 'signature']
    if not all(k in values for k in required):
        return 'Missing values', 400
    # Create a new Transaction
    transaction_result = blockchain.submit_transaction(values['sender_address'], values['recipient_address'], values['amount'], values['si
gnature'])

    if transaction_result == False:
        response = {'message': 'Invalid Transaction!'}
        return jsonify(response), 406
    else:
        response = {'message': 'Transaction will be added to Block '+ str(transaction_result)}
        return jsonify(response), 201

@app.route('/transactions/get', methods=['GET'])
def get_transactions():
    #Get transactions from transactions pool
    transactions = blockchain.transactions

    response = {'transactions': transactions}
    return jsonify(response), 200

@app.route('/chain', methods=['GET'])
def full_chain():
    response = {
        'chain': blockchain.chain,
        'length': len(blockchain.chain),
    }
    return jsonify(response), 200

@app.route('/mine', methods=['GET'])
def mine():
    # We run the proof of work algorithm to get the next proof...
    last_block = blockchain.chain[-1]
    nonce = blockchain.proof_of_work()

    # We must receive a reward for finding the proof.
    blockchain.submit_transaction(sender_address=MINING_SENDER, recipient_address=blockchain.node_id, value=MINING_REWARD, signature="")

    # Forge the new Block by adding it to the chain
    previous_hash = blockchain.hash(last_block)
    block = blockchain.create_block(nonce, previous_hash)

    response = {
        'message': "New Block Forged",
        'block_number': block['block_number'],
        'transactions': block['transactions'],
        'nonce': block['nonce'],
        'previous_hash': block['previous_hash'],
    }
    return jsonify(response), 200
```

Go Top


Transactions to be added to the next block

Show 10 entries Search:

| # | Recipient Address | Sender Address | Value |
|---|---|---|---|
| No data available in table | | | |

Showing 0 to 0 of 0 entries Previous Next

Transactions on the Blockchain

Show 10 entries Search:

| # | Recipient Address | Sender Address | Value | Timestamp | Block |
|---|---|---|---|---|---|
| No data available in table | | | | | |

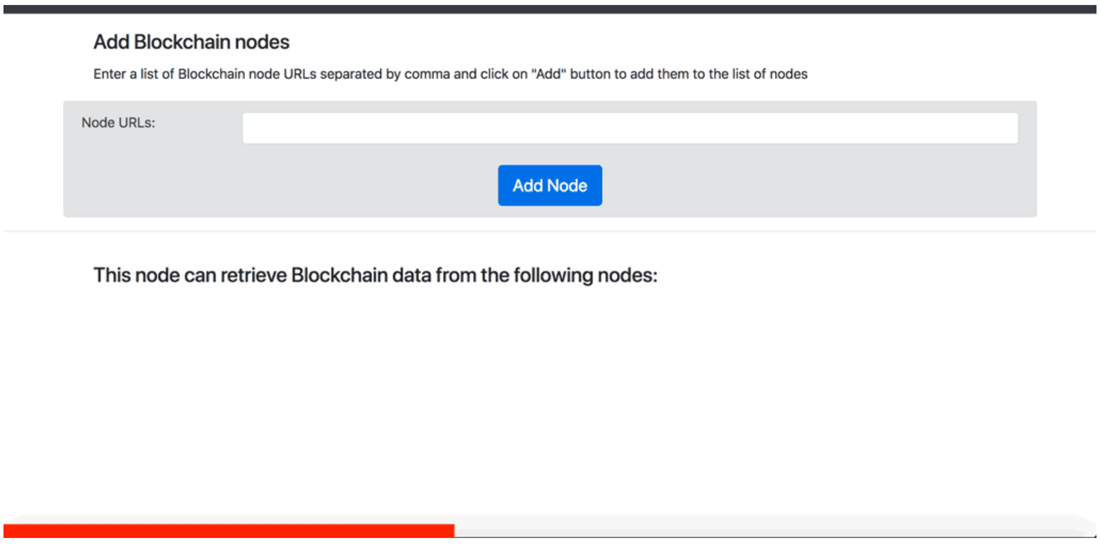
Showing 0 to 0 of 0 entries Previous Next

Below we define Flask APIs to manage blockchain nodes.

- `'/nodes/register'` : This API takes as input a list of node urls, and adds them to the list of nodes.
- `'/nodes/resolve'` : This API resolves conflicts between blockchain nodes by replacing a local chain with the longest one available in the network.
- `'/nodes/get'` : This API returns the list of nodes.

```
@app.route('/nodes/register', methods=['POST'])
def register_nodes():
    values = request.form
    nodes = values.get('nodes').replace(" ", "").split(',')

    if nodes is None:
        return "Error: Please supply a valid list of nodes", 400

    for node in nodes:
        blockchain.register_node(node)

    response = {
        'message': 'New nodes have been added',
        'total_nodes': [node for node in blockchain.nodes],
    }
    return jsonify(response), 201


@app.route('/nodes/resolve', methods=['GET'])
def consensus():
    replaced = blockchain.resolve_conflicts()

    if replaced:
        response = {
            'message': 'Our chain was replaced',
            'new_chain': blockchain.chain
        }
    else:
        response = {
            'message': 'Our chain is authoritative',
            'chain': blockchain.chain
        }
    return jsonify(response), 200


@app.route('/nodes/get', methods=['GET'])
def get_nodes():
    nodes = list(blockchain.nodes)
    response = {'nodes': nodes}
    return jsonify(response), 200
```

Go Top


Add Blockchain nodes
Enter a list of Blockchain node URLs separated by comma and click on "Add" button to add them to the list of nodes
Node URLs:
Add Node
This node can retrieve Blockchain data from the following nodes:

## Conclusion

In this blog post, we covered some core concepts behind blockchain and we learned how to implement one using Python. For the sake of simplicity, I didn't cover some technical details, for example: Wallet addresses and Merkel trees. If you want to learn more about the subject, I recommend reading the original Bitcoin white paper and follow up with bitcoin wiki (https://en.bitcoin.it/wiki/Main_Page) and Andreas Antonopoulos's excellent book: Mastering Bitcoin: Programming the Open Blockchain (https://www.amazon.com/gp/product/1491954388/ref=as_li_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=1491954388&linkCode=as2&tag=adilmoujahid-20&linkId=bd776f9224715e8a022d4984909d6a69).

## References

- 1 - Wikipedia - Public-key cryptography (https://en.wikipedia.org/wiki/Public-key_cryptography)
- 2 - Wikipedia - Hash function (https://en.wikipedia.org/wiki/Hash_function)
- 3 - Bitcoin Stackexchange - What happens to a transaction once generated? (https://bitcoin.stackexchange.com/questions/58687/what-happens-to-a-transaction-once-generated)
- 4 - Bitcoin Wiki - Majority attack (https://en.bitcoin.it/wiki/Majority_attack)

Name

722
Shares

**ajayigbolahan** • 5 months ago

Great article but you didn't point out in your transaction class how the input is form from previous output and how these input are sent to recipient as output

24 ^ | ˅ • Reply • Share ›

616

**JeffWeakley** • 5 months ago

This is fantastic. On a lot of levels. Thanks a bunch!

16 ^ | ˅ • Reply • Share ›

**Sreehari B S** • 10 months ago

Thanks for the nice writeup !

But still in confusion ::

Suppose there are three Transactions T1, T2, T3 in the network, of which T2 is invalid (can be double spent / insufficient credits). Assume that I am able to create a block and validates all the transactions including T2. Let other minors build on top of my block. Hence my invalid transaction T2 is now confirmed. (Please refer the image below)

Is there any mechanism available to identify this invalid transaction.

see more

15 ^ | ˅ • Reply • Share ›

**Charlie Miller** ➦ Sreehari B S • 10 months ago

the decentralised nature of the blockchain eliminates the chance of a double spend ever happening so your t2 would never get confirmed onto block #1

^ | ˅ • Reply • Share ›

**Gabriel** ➦ Charlie Miller • 4 months ago

The nature of the blockchain doesn't fully eliminate this possibility, it is just something very difficult to achieve. Assuming you are able to modify a transaction and successfully mine all the way to the current block, you can effectively double spend, and because of the way the chain

Go Top

why it isn't a real concern, is the part where you mine all the way to the top before anyone else. If you think about it, you'll notice how that's just unrealistic, although technically possible.

• Reply • Share ›

**QI Yuqi** • 10 months ago

Thanks Adil, here's a Chinese version of this post.
中文翻译版本：http://qiyuqi.com/article/b...

2 • Reply • Share ›

**Milan D** • 6 months ago

Both commands fail for me. Some broken dependency?

```
python blockchain.py -p 5000
Traceback (most recent call last):
File "blockchain.py", line 24, in <module>
import Crypto
ImportError: No module named Crypto
```

1 • Reply • Share ›

**Matej Tacer** → Milan D • 3 months ago

you need to install Crypto library: pip install pycryptodome

• Reply • Share ›

**Nikita Petrov** • a year ago

Very nice and informative post, thanks! Anders Brownworth's Blockchain Demo is actually excellent.

I have one question about miners reward. Is it really taken from commissions of all transactions in block? As I heard before, mining is the procedure which is needed to generate new coins in the system. So this means that miners reward is taken from nothing but not from transaction commissions. Is it right?

1 • Reply • Share ›

**Adil Moujahid** Mod → Nikita Petrov • a year ago

In the case of Bitcoin, when a new block is discovered, the system generates new coins that are rewarded to the miner. As of March 2018, the reward is 12,5 BTC for each block. The Bitcoin block mining reward halves every 210,000 blocks. It's estimated that on 31 May 2020, the reward will drop to 6,25 BTC. [1]

Because of this reward structure, the number of bitcoins in existence will not exceed 21 million BTC. The closer the system gets to this theoretical limit, the more it will rely on transaction fees to incentivize the miners.

Go Top

**SEBASTIAN RIOS RIVERA** • a year ago

Muchas gracias ADIL, excelentes tutoriales.

1 ^ | v • Reply • Share ›

**Adil Moujahid** Mod → SEBASTIAN RIOS RIVERA
• a year ago

Gracias Sebastian! Me alegra que lo hayas disfrutado.

^ | v • Reply • Share ›

**Masayori Shoji** • a year ago

Hi,
I am Head of Fintech promotion supprot office in KPMG Japan.
If you have some interest in our team, please see this web site and contact by mail.
www.kpmg.com/jp/fintech
Thanks
Masayori Shoji

1 ^ | v • Reply • Share ›

**Sai Chand Akella** • 2 months ago

how to read original data from the blockchain?

^ | v • Reply • Share ›

**jayant isswani** • 3 months ago

How does this implementation ensure integrity of data in the chain among different nodes? I mean, any node can change the data in the chain. How is this solved?

^ | v • Reply • Share ›

**Vidya Krishnan M** • 3 months ago

I tried to run this code and it executed without an error. I did all the steps as mentioned and when i make a transaction and click confirm transaction terminal is showing error 406(invalid transaction). what could be the possible reason?

^ | v • Reply • Share ›

**Bharath Athithiya** • 5 months ago

Great article manh!!
But my doubt is that if incentives must be given to miners to validate our transactions, then it is almost same as a bank which charges for our transactions right?!

^ | v • Reply • Share ›

**Mery Lily** • 5 months ago

Hi Adil.. thank you its really a helpful post i did learned a lot.. i have a question how i can create a python script to only collect ip address from the nodes net_addr ?? i am very new to this

Go Top

Join my Data in Practice Newsletter Enter your email address Subscribe Now ✕

722 Shares

616

**Mike Hellawaits** • 6 months ago
Gracias, muy buena Informaciòn!
Reply • Share ›

**Mohamed Arbi Mabrouk** • 7 months ago
can I send bitcoin to blockchain.info
Reply • Share ›

**crypto raj** • 7 months ago
Typo Correction.
"The mining process is also referred to as Proof of Work (PoW), and it's the main mechanism that enables the blockchain to be **trustless** and secure" . It should be trustful right?
Reply • Share ›

**Gabriel** ➜ crypto raj • 4 months ago
"Trustless" as in you don't need to trust anyone in particular to be sure that everything is in order in the chain.
Reply • Share ›

**myFatherIsKing** • 8 months ago
Thanks for this Adil. Quite interesting given all the rave about blockchains

I did follow the tutorial and that of dvf which you referenced. I implemented it in Django rather than flask. My implementation is online here http://parousia.pythonanywh..., and sourcecode is on github https://github.com/immensit...

I initialize the blockchain in my settings file

from chain.blockchain_client import Blockchain

BLOCKCHAIN = Blockchain()

But the problem I have is that each time I restart the server the blockchain disappears and everything is lost. Any ideas as to how I could potentially perpetuate the state of the blockchain?
Reply • Share ›

**Carlos** • 8 months ago
Many thanks for the useful information!
At me such question: and how to make so that this all was not locally on one computer, and globally? That is, run for example nodes on different computers and that they somehow synchronize the blocks among themselves?
Reply • Share ›

Go Top

but it seems it is not available (Unable to Connect). Any hints?
Thank you very much

Reply • Share ›

**Yan Grey** • 10 months ago

Help me find a more detailed reference in the encyclopedia http://bitcoinwiki.org/ about the python

Reply • Share ›

**Francisco Renteria** • a year ago

gracias por este excelente post... me ayudo a entender mucho mas :) saludos

Reply • Share ›

**kubilay** • a year ago

Hi Adil,

Great job thanks,

I have a few questions:
1) I could not understand how you verify the sender has enough coins during the mining process. Is this part missing?
2) After mining a block, should the node broadcast the new chain? I could not see this part. So, if you create multiple nodes, the chains are different in each node, also the consensus does not work due to this problem.

I am not expert in the blockchain, but the parts above are confusing for me, if you help me about them, I would appreciate that.

BR
Kubilay

Reply • Share ›

**Adil Moujahid** Mod → kubilay • a year ago

Hi Kubilay,

Both features are needed for a real blockchain, but they're not implemented here.

May you can extend the source code to cover both :)

25 • Reply • Share ›

**Mohamed Arbi Mabrouk** → Adil Moujahid • 7 months ago

can I send bitcoin to blockchain.info and other wallet help me

Reply • Share ›

Go Top

I have one question in Section 2.2, you said:
and it's the main mechanism that enables the blockchain to be trustless and secure.

How to understand the word "trustless"?

^ | ⌄ • Reply • Share ›

**Adil Moujahid** Mod ➦ QI Yuqi • a year ago

By "trustless" I mean that it doesn't require trust in a third party

^ | ⌄ • Reply • Share ›

**QI Yuqi** ➦ Adil Moujahid • a year ago

Aha, the beginning of this post mentions this issue: https://medium.com/@preethi...

^ | ⌄ • Reply • Share ›

**QI Yuqi** ➦ Adil Moujahid • a year ago

なるほど😊 but "trustless" usually means untrusted or untrustworthy, I think? a little ambiguity

^ | ⌄ • Reply • Share ›

**KMagnum** • a year ago

Great post, but I think you've confused RSA encryption with RSA signatures. They're both based on the difficulty of the RSA problem, but it looks like you're using RSA signatures and not RSA encryption. You briefly mention them as distinct things, but then you mention 5 times that you're using RSA encryption. Your diagrams only show use of RSA signatures, and I don't see the utility of RSA encryption for your use case.

An RSA signature on a transaction would allow only you to sign the transaction, but anyone to verify that you've signed (and presumably authorized) the transaction.

RSA encryption of a transaction would allow anyone to encrypt a transaction such that only you could read the contents of the transaction. There's not much utility (disregarding homomorphic encryption and zero-knowledge proofs of properties of the transaction) in having a transaction on the pubic blockchain that is only readable by one party.

Also, if you use 1024-bit RSA keys, readers could easily make the mistake of thinking this size is secure. Browsers are in the process of distrusting certificates signed with 1024-bit RSA. You should either use 2048-bit RSA, or use something much smaller and efficient and make it much more clear to readers that the key size isn't safe.

^ | ⌄ • Reply • Share ›

Go Top

Hi,

Could you explain the following line of code:

**722**
Shares

616

Go Top