**Blockchain**

**Building A Simple Blockchain Data Structure With Python**

Build a simple blockchain data structure which is the foundation of Bitcoin. This data structure only is not enough to build even a simple cryptocurrency. But we have to start somewhere.

import hashlib

hashlib.sha256(b”test data”).hexdigest()

That is how you do hashing (SHA-256) in Python. But what is actually hash? What does 256 in SHA-256 means actually?

Hashing is a process which you turn anything (as long as you can represent it as a string) into a fixed 256 bit string. In the previous example, the string “test data” has a length 9. Actually the length of “test data” is depended on how you count it. But for simplicity, we just count how many characters. That “test data” string is turned into a fixed-size string, which is ‘b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee9088f7ace2efcde9’.

I hash another string with different length.

import hashlib

hashlib.sha256(b”I am new in blockchain … trying to learn”).hexdigest()

The string “I am the best president. Ever.” has a different length than “hello world”. But the output has the same length, which is about 64 characters. So any input will be turned into 64 random characters string. Even a string which has a 23 kilometers length will be turned into a 64 random characters string.

This is a hexadecimal string. That’s why it has 64 characters. If you turn it into a bit string, it will have a 256 characters length.

So there is no apparent relation between the input and the output. Even you change the tiny bit of the input, the output will be totally different. This is the second property.

hashlib.sha256(b”1").hexdigest()

hashlib.sha256(b”2").hexdigest()

hashlib.sha256(b”3").hexdigest()

hashlib.sha256(b”11").hexdigest()

So the only way to to find different inputs which have the same output, you need to test all combination of characters with different length. “abc”, “maybe a loooooong string”, “17”, etc. It’s totally impractical.

But is there any possibility that when you hash something, the output is already same as hashing of “hello world”? Or just any two different strings but have the same hash? Of course, there is. The question is how minuscule the probability is. There are around 2²⁵⁶ possibilities of the output of SHA-256. How big is 2²⁵⁶? 115792089237316195423570985008687907853269984665640564039457584007913129639936. Or 1.15 e+77. Or roughly 10⁷⁷. If you think that number is big but not very big, I have a bad news for you. The total atom in observable universe (that is the universe that you can see up to 46 billions light years in any direction from your chair) is 10⁷⁸ to 10⁸².

To describe the hashing algorithm, it is quite a work. But someday I’ll explain the code behind SHA-256. Now that you can comprehend the vastness of hashing, let’s move on. Blockchain is like a linked list, a data structure known by many computer science students. Let’s create a block. The first block in Bitcoin is called genesis block.

import hashlib, json

block\_genesis = {

‘prev\_hash’: None,

‘transactions’: [1, 3, 4, 2]

}

The transactions represents the… well, transactions. In Bitcoin, it will be like “Jason pays 2 btc to Mary Sue”, “Kylo Rein pays 10 btc to Yoda”. For simplicity, we just put normal integers.

We serialized the block so it can be hashed.

block\_genesis\_serialized = json.dumps(block\_genesis, sort\_keys=True).encode(‘utf-8’)

block\_genesis\_hash = hashlib.sha256(block\_genesis\_serialized).hexdigest()

Now we have another block.

block\_2 = {

‘prev\_hash’: block\_genesis\_hash,

‘transactions’: [3, 3, 3, 8, 7, 12]

}

We hash the block 2.

block\_2\_serialized = json.dumps(block\_2, sort\_keys=True).encode(‘utf-8’)

block\_2\_hash = hashlib.sha256(block\_2\_serialized).hexdigest()

We build another block.

block\_3 = {

‘prev\_hash’: block\_2\_hash,

‘transactions’: [3, 4, 4, 8, 34]

}

We hash the block 3. This will be the last block, I promise.

block\_3\_serialized = json.dumps(block\_3, sort\_keys=True).encode(‘utf-8’)

block\_3\_hash = hashlib.sha256(block\_3\_serialized).hexdigest()

To make sure that data has not been tampered, I only need to check the last block’s hash, instead of checking all the data from genesis block to the last block. If it is different, than someone tried to tamper the data.

import hashlib, json

block\_genesis = {

‘prev\_hash’: None,

‘transactions’: [1, 3, 4, 2]

}

block\_2 = {

‘prev\_hash’: None,

‘transactions’: [3, 3, 3, 8, 7, 12]

}

block\_3 = {

‘prev\_hash’: None,

‘transactions’: [3, 4, 4, 8, 34]

}

def hash\_blocks(blocks):

prev\_hash = None

for block in blocks:

block[‘prev\_hash’] = prev\_hash

block\_serialized = json.dumps(block, sort\_keys=True).encode(‘utf-8’)

block\_hash = hashlib.sha256(block\_serialized).hexdigest()

prev\_hash = block\_hash

return prev\_hash

print(“Original hash”)

print(hash\_blocks([block\_genesis, block\_2, block\_3]))

print(“Tampering the data”)

block\_genesis[‘transactions’][0] = 3

print(“After being tampered”)

print(hash\_blocks([block\_genesis, block\_2, block\_3]))

The result:

Original hash

45eda4f7a76bf0f92a0acda2ce4752dfbe167473376f766f22d7ec68501cac40

Tampering the data

After being tampered

27d68dae05428be6aa244869196a481f431fca6645dd33c3df7a740afa03b7d9