Class 168

Sida Zhong, Peilu Liu, Yan Chen

Implementing Blockchain using Python

10th December 2020

Introduction

Blockchain is invented for Bitcoin, which is a digital currency that solves the double-spending issue without a central server. This technique has been used in other applications, such as financial services and video games. Blockchains are designed to be readable to the public and widely used in other cryptocurrencies like Etherum and smart contract. Blockchain is made of a growing link of transactions (blocks) with cryptography. It is managed by a peer-to-peer network and distributed timestamping server. Javascript and Solidity are two of the most commonly used programming languages to implement Blockchain. In this project, we implement a blockchain using Python and simulate basic functionalities: making transactions, mining, and transaction validating using the Merkle tree algorithm.

Mining Pool

Our blockchain allows multiple miners to mine at each round. The mining functionality is simulated using the multi-threading technique, and each miner is a single thread. First, each miner reads past transactions from a JSON file that stores transaction records. Every thread has several non-shareable variables such as the

nonce, Merkle tree, and pre-hash, etc. These variables are initialized after reading historical transactions. Next, each miner validates signatures using the public key and private key from transaction records. Then, the miner builds its own Merkle tree and uses the proof-of-work mechanism to update the hash and build the block. This round is completed when the first miner finishes running through the whole process. We implemented the communication between threads using global variables instead of broadcasting. There are also some challenges using multi-threading to implement the mining functionality. Collisions might occur when multiple miners are reading from the transaction file, modifying global variables, or calling the same function at the same time. Overall, the probability of collision is very low if the blockchain has less than five miners. For future works, we will continue to look into this issue.

Merkle Tree

In our project, the transactions are hashed by ECC and the result hashes are stored in a Merkle tree. Each node of the tree is the hash of its two children, and the leaves are the hashes of the transactions. The top node is called the root hash, which is stored in each block. This tree structure prevents data from being modified since any change in the leaf will also affect the root hash, as illustrated in Fig. 1. Another advantage of the Merkle tree is that it is fast to insert, delete, or search for a node. The complexity is O(log(n)).

To implement the Merkle tree, we used the idea of a segment tree, which stores a segment of data in each node. Apply to the Merkle tree, each node stores the hash of a segment of transactions, and its left child stores the hash of the left half of the segment, while its right child stores the hash of the right half of the segment. For

example, if we have data of length 4, then the root, which is numbered as node 0, takes care of segment 0 to 3. The left node will be numbered as 2 * 0 + 1, which is node 1 and takes care of segment 0 to 1. And the right node will be numbered as 2 * 0 + 2, which is node 2 and takes care of segment 2 to 3.

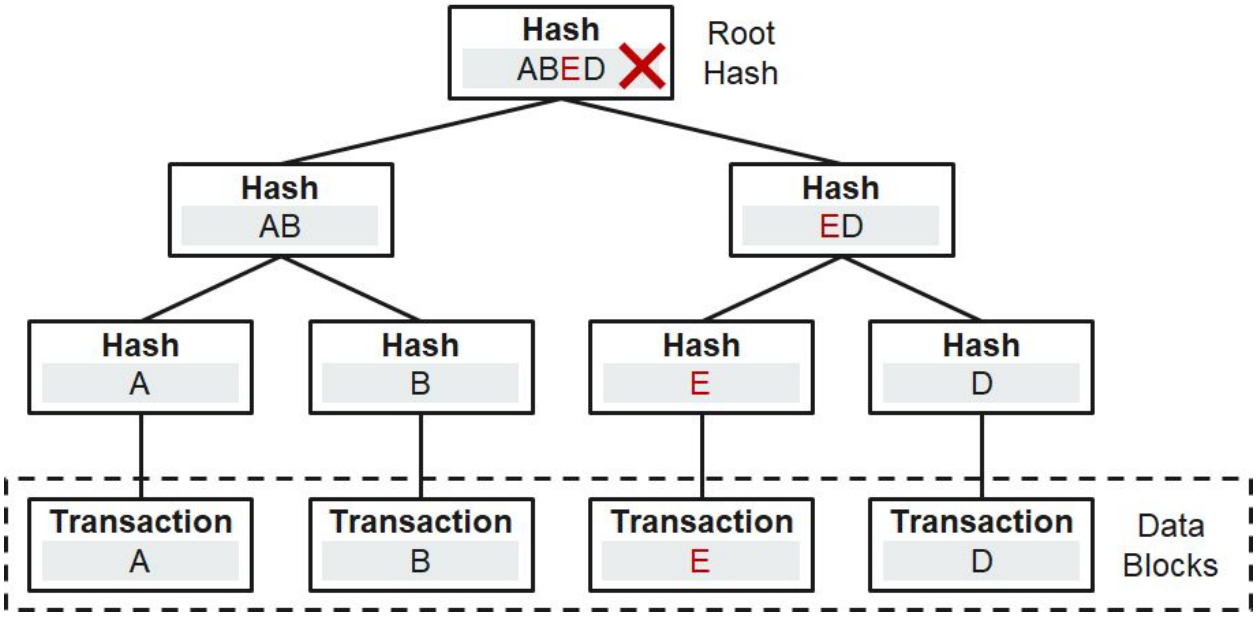

Fig. 1: Structure of the Merkle tree with 4 transactions.
The third transition was C, if it is changed to E, the root hash will also change.

Therefore, to build a tree, we need the data, the tree, which is empty initially, and the current node, starting from node 0, with the starting and ending point of the segment. The left part of the tree is built (hashed) recursively first, which means the left node becomes the current node and the midpoint becomes the ending point. Recursion stops when the starting point is equal to the ending point, which means we reach the leaves, so we only need to hash the data itself. Similarly, the right part is also built recursively. The order is always left first, then right. Note that, we still hashed from leaf to root, since we build the tree recursively, and the hashing process is similar to depth-first search, as shown in Fig. 2.

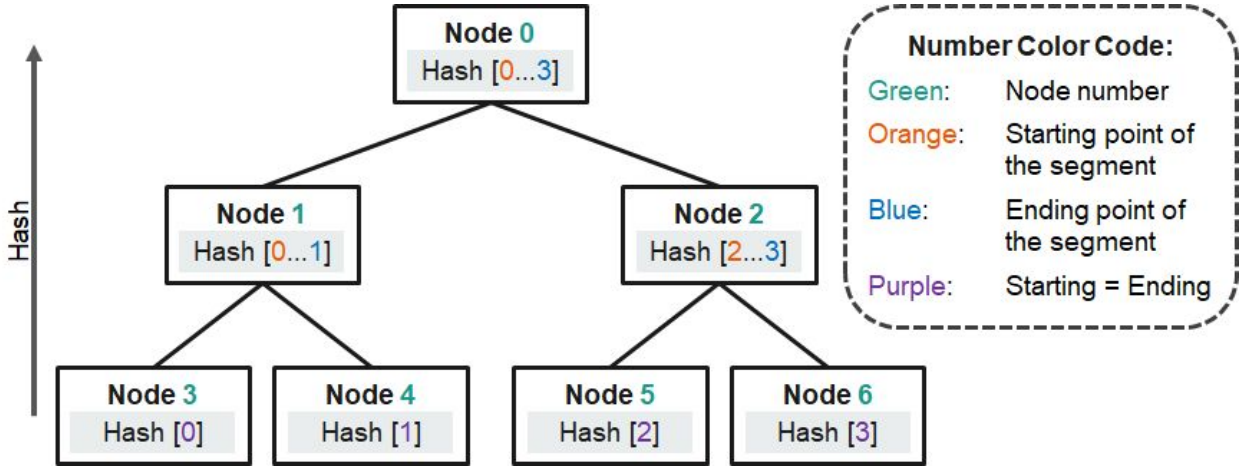

Fig. 2: Implementation of the Merkle tree of 4 transactions using a segment tree. The order of node being built is: 0 → 1→ 3→ 4→ 2→ 5→ 6 (DFS)

Our implementation differs from the original Bitcoin Merkle tree slightly. In the Bitcoin Merkle tree, if there exists a layer that contains an odd number of nodes, the unpaired node will be duplicated and hashed with itself. That is, suppose the node itself has a hash value of n, then its parent node will be hash(n + n). However, in our implementation, the unpaired node will not be duplicated, so its parent node will be hash(n) if the node has a hash value of n. So far, we didn't see any major impact of this difference.

Client side

In the client, there are two test users, Alice and Bob. They have their own wallets. Inside the wallet is their private key that can unlock the remaining currency in UTXO[1]. "bank.py" file can view their total currency value. "client.py" is a program for user transactions. When Alice and Bob conduct a transaction, based on the UTXO

[1] UTXO stands for Unspent Transaction (TX) Output. Every on-chain bitcoin transaction sends bitcoin to one or more addresses, from at least zero (in case of a coinbase transaction) addresses. A bitcoin wallet balance is actually the sum of the UTXOs controlled by the wallet's private keys.

concept of Bitcoin[2], the program will extract and burn the currency based on their private key in the wallet. After that, the program will generate a new secret key, in which the public key is added to UTXO as a change to the user, and the private key is put into their wallet, as shown in Fig. 3. The transaction record will be stored in the "transaction.json" file. This project needs to store a batch of transaction records in a block with a Merkle tree structure. In order to ensure that multiple users generate transaction records at the same time, the project simulated 600 transaction records in a "transaction.json" file as a batch of transactions to be processed by miners. Among them, only the transaction records of Alice and Bob are real, and the others are just for testing and have no real UTXO value, as shown in Fig. 4.

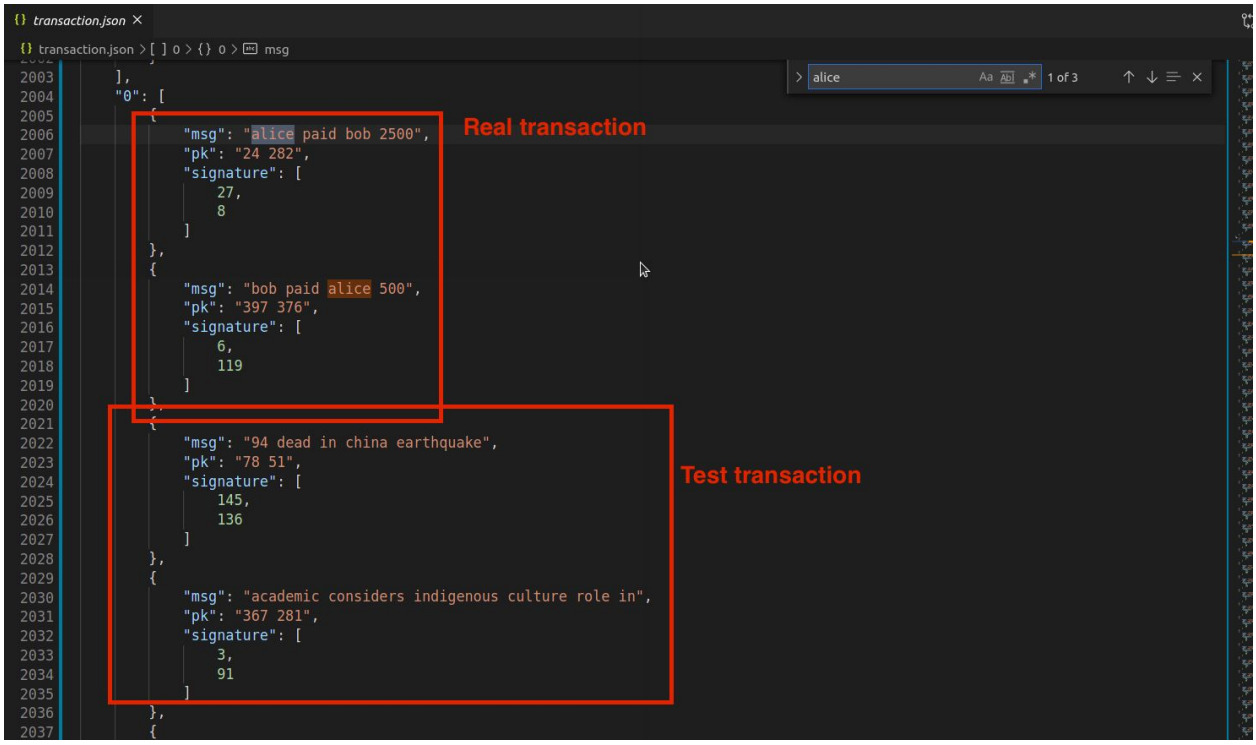

Fig. 3: Simulate record transaction files. The transaction records of Alice and Bob are real, while the others are for testing and have no real UTXO value

[2] Bitcoin is a cryptocurrency invented in 2008 by an unknown person or group of people using the name Satoshi Nakamoto and started in 2009 when its implementation was released as open-source software.

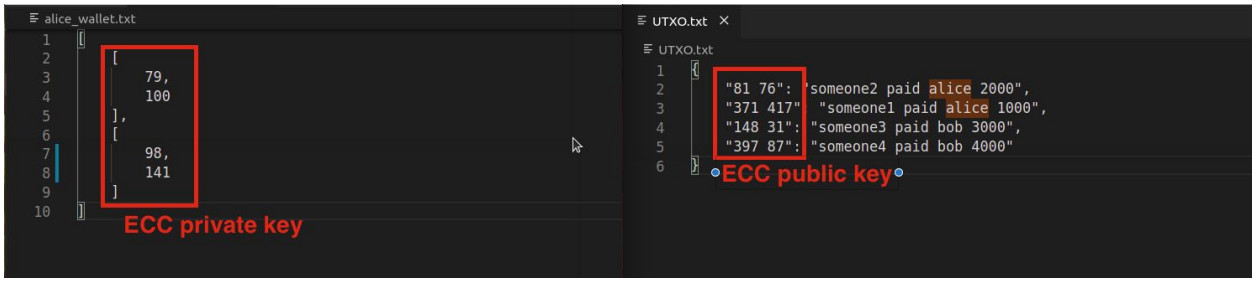

Fig. 4: Client wallet (left, contains private key) and UTXO (right, contains public key).

Elliptic Curve Cryptography

In this project, the client's UTXO public and private key encryption uses ECC[3] instead of RSA[4]. Fig. 5 demonstrated ECC encryption. ECC is currently the most popular encryption algorithm. The advantages are more security, shorter key length, and faster hashing speed. Among them, Compressing the size of the key is the most significant advantage. The biggest bottleneck of blockchain is that the throughput is too slow. For a world-class bank like VISA INC[5], the transaction volume per second is about 6000 transactions. In Bitcoin, on average, miners dig a block in 10 minutes, and the size of a block is only 1MB. 1MB of space can only store about 4000 transaction records, an average of fewer than 10 transactions per second, this speed is very slow. Part of the reason is the transaction hash keys take too much space. Although there is no specific data, according to the usual 256-bit RSA key generated, it may occupy about 30%. So the key size is very important. The "ECC.py" file contains the generation and verification of the ECC key. In this project, for the convenience of testing, some basic lib

[3] Elliptic-curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. ECC allows smaller keys compared to non-EC cryptography (based on plain Galois fields) to provide equivalent security.

[4] RSA (Rivest–Shamir–Adleman) is a public-key cryptosystem that is widely used for secure data transmission.

[5] Visa Inc. (also known as Visa, stylized as VISA) is an American multinational financial services corporation headquartered in Foster City, California, United States.

parameters are modified, such as the starting point P of the elliptic curve, so that the tested transaction key generation is simpler and faster.

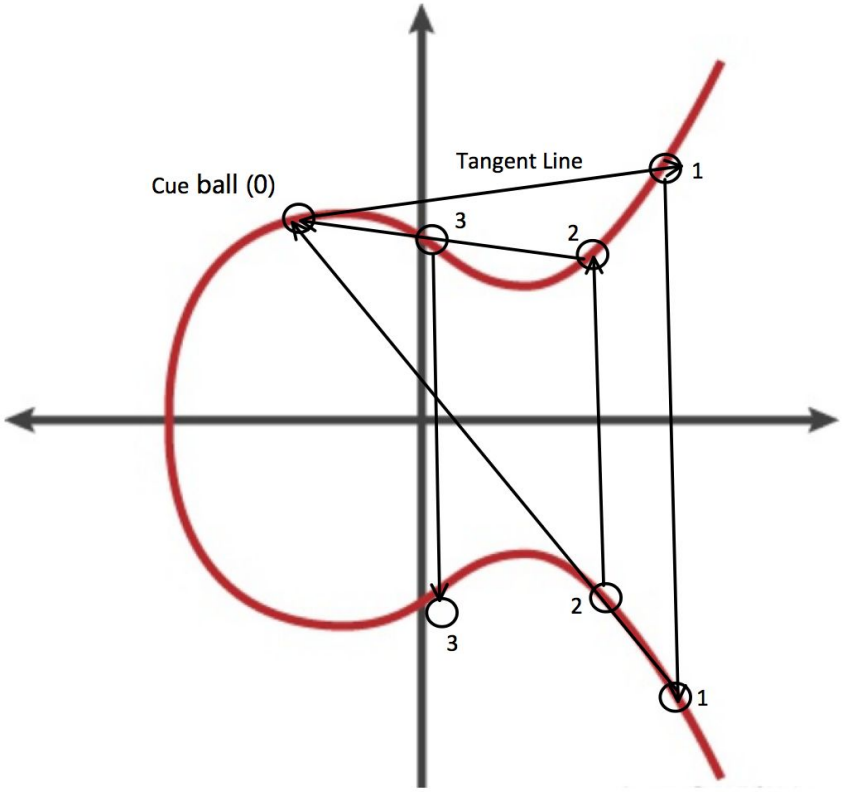

Fig. 5: ECC encryption explanation diagram. Given an elliptic curve and a starting point (Cue ball 0). The public key pairs are the starting point and the ending point after “bouncing” on the curve n times; the number of bounces (n) is the private key.

Zero Knowledge Proof

There are some follow-ups to this project. If there is more time, ZKP[6] can be used to hide the UTXO transactions. Bitcoin's current anonymity mechanism is already very insecure. The so-called anonymity just cannot map the user's real name and transaction address. But if a user uses the same transaction address multiple times, there are many ways to find clues and expose privacy. ZKP has done some research,

[6] In cryptography, a zero-knowledge proof or zero-knowledge protocol is a method by which one party (the prover) can prove to another party (the verifier) that they know a value x, without conveying any information apart from the fact that they know the value x.

and the best way is to use Sudoku[7] games to hide transactions between customers. Because the rules of the Sudoku game limit the number of rows and columns to be unique, it is more suitable for random proof.

Conclusion

As discussed above, we were able to implement a blockchain with basic functionalities using Python. We used ECC for public and private key encryption, and the hashes are stored in a Merkle tree that is implemented using a segment tree. However, we faced some challenges when implementing mining functionality in multi-threading since collisions might occur when multiple miners accessing the same resource at the same time. In the future, we will try to solve this collision problem, as well as implementing ZKP, and simulating a 51% attack.

[7] Sudoku originally called Number Place is a logic-based, combinatorial number-placement puzzle.