Design and Implementation of a Rudimentary Blockchain-Based Cryptocurrency

Author: Anna, Bryan, Cecil, Ran

Part 1 General introduction to code project:

1. In general, it demonstrates the basic functionalities of block-chain. Specifically, it shows how block-chain works as a rudimentary ledger-based cryptocurrency. i.e. How it stores the transaction history and allows new transactions.
2. The code comprises of 3 parts. 1. A sha-256 implementation that takes strings as parameter and spits hash values in string format. It's downloaded from the internet. 2. RSA public key digital signature implementation that uses OpenSSL's RSA algorithms. 3. Block and transaction and wallet classes that realize the functionality mentioned in 1. The data manipulation is done on two types of files. .block files store single blocks and .tx files store unverified transactions.
3. The code works on Mac OS and does not involve interaction with network or another computer. It emphasizes on block-chain technology only. Public consensus and network protocol, etc. other crucial components of cryptocurrency are not included because of the difficulty of the setup and lack of resources.

Tasks to solve:

1. It simulates many of the functionalities of bitcoin block-chain using naïve approaches.
2. A genesis block is hard-coded in as a start point.

3. It allows verification of the history of multiple unverified but valid transactions. In this manner the code prevents double spending attacks.
4. A wallet implementation allows generation of new valid transactions and appending them to the current block-chain.
5. It can print out transaction/block information to stdout.
6. It adopts bitcoin's idea of incentives and calculates the nonce of each newly generated blocks to emulate the distribution of new coins.
7. Other unspecified but necessary functionalities.

Part 2 Design of code project:

In an economy based on decentralized currency, we must ensure privacy of user transactions. If there is no sense of unique identity in utilizing a decentralized currency, then we run the risk of adversaries producing false information to attack our system. Furthermore, we need to develop a data structure to maintain the history of our system and a record of previous transactions.

Design of transaction:

In a digital currency system, money (or coins) is basically the entire history of its transactions. There are several vital elements in the design of transactions as in the specific context of blockchain-based cryptocurrency: 1. Digital signatures. Digital signatures meet the fundamental demand of a means to verify transactions. The owner of a coin transfers money by digitally signing the transaction and appending the payees' public key to the transaction. 2. Continuity with the coin's previous transaction. Each transaction has to be linked to its previous one (except the first in the chain is generated in a different format). This ensures the singularity of the coin's historical ownership and fulfills the functionality of coin serial number like identifying coins and maintaining the coin's value. In our project (also in the practical design of famous Bitcoin), each transaction includes a hash pointer to

its previous one in chronological order. 3. Capability of splitting and combining coins. Intuitively, when owners of either real or digital coins spend money, they don’t transfer exactly one bill at a time. For our first example, if A owns a 1 dollar bill and wants to transfer 50 cents to B and B’s got no exchange, A would want to split his bill. In our project, this procedure is realized by adding multiple recipients A, B to this one transaction where the amount of money transferred is specified as 50 cents for each recipient. For our second example, if A owns two 50 cents bill (so A actually owns two sequences of transactions) and he wants to transfer 1 dollar to B, A would add two hash pointers, each linking to a 50 cents bill to the transaction.

Design the blockchain:

Given the design of transactions on its own, one problem is inevitable: As the owner of a coin an agent can deliver two transactions that spend the same coin and send the transactions to the payees. As long as the payees don’t share information with each other, they can successfully verify that the transactions are valid. This is what we would call double spending. The solution to double spending is publication: With a trustworthy centralized institution that keeps all the transactions in its pocket, we can have a reliable source of information and block the attempts to double spend a coin; simply put it’s just like an online bank. But this authority can fail or people just don’t like it because anonymity is the virtue of internet and thus we don’t want centralized operations. Here’s where blockchain comes into practice: It’s an answer to both decentralization that eliminates the need for a third party and double spending.

A block in our code has the following structure, modified from Bitcoin for simplicity:

1. Header: (a) A hash pointer to the previous block. (b) Hash of the merkle root of all transactions in this block. (c) Calculated nonce as proof of work. (d) Block hash as finger print of the block (hash of nonce | merkle root hash) (e) Number of transactions N (f) Simplified time stamp that records the time of generation
2. Transactions: N-1 verified transactions and 1 new coin generating transaction.

A genesis block serving as the start point is hard-coded into the project. New blocks ending in .block type of file can be appended from that.

To verify transactions, each time the whole blockchain is traversed from the most current block back to the genesis block if needed. With correct formats and contents, a valid transaction assures that no other transactions chronologically preceding this one spend the same money (take the same transaction-index combo as one of its input). Otherwise it would be a double spending.

Decentralization is demonstrated by public consensus protocol and proof of work design. The code enables parts of the functionalities but the topic goes beyond our scope.

Part 3 Further Examples of Block Chain Applications: Implementing the Block Chain in Cloud Storage

Beyond cryptocurrencies, the block chain can be used in a wide variety of applications. For example, the block chain can play a role in cloud storage. The most popular form of cloud storage right now would include applications like Google Drive or Dropbox. Users upload files to the “cloud”, a large data center, and leave their files in the hands of a large corporation. By implementing a block chain, the large corporation can be removed from the equation, and you could still have access to your files away from your own hard drive.

High Level Overview

In order for this to be achieved, there must be a way for a user to upload their files to some decentralized authority, while keeping their files unreadable to anyone without proper permissions. For these files to be protected, they are first encrypted on the users' computers before being uploaded to a network. The encrypted file is then split up into chunks, and then distributed across a network. This means that the keys to a users' files are owned by the user, and not some corporation. This network is composed of other users who rent out their unused hard drive space.

The problem is, not all users will rent out their unused hard drive space for free; some sort of incentive is needed. Some startup companies, including Storj and Factom, are already exploring this idea. Stroj, for example, rewards users for giving up hard drive space with Storjcoin X, or SJCX. (SJCX is similar to Bitcoin in the sense that it carries a real-dollar value. Some reasons why Storj decided to create their own cryptocurrency was because Bitcoin was too expensive and it fluctuates too much).

Cost and Benefits

Relying on other users for access to your data can pose some problems. For instance, Storj has their own client app, called DriveShare that users can download. Their network is comprised of DriveShare nodes run by users around the world who rent out their unused hard drive space in exchange for SJXC. The problem with this is that a node is not guaranteed to always be online. If a node is offline, any user with files on that node cannot access it. To compensate for this problem, the solution would be spread out your files to more nodes. If one of those nodes goes offline, the Stroj network will automatically find another online node to take over, so you can still

access your files. Similar to mining cryptocurrencies, keeping a node online is not free. There is some electricity cost associated with keeping a node online. The net cost of this will depend on the cost of electricity in your area, and how much storage you are renting out. Once your node goes offline, it will begin to fail a series of network challenges and the owner of that node will no longer be paid.

Besides the extremely low cost of using the block chain as cloud storage ($0.015 GB/month from Storj versus $0.03 GB/month from Amazon), files can be accessed at an extremely fast rate. Because the network is shared, you don’t have to worry about slow download speeds coming from one place. You are, in fact, downloading multiple small files from several sources.