Lab 4: Minimum Viable Blockchain

Revised: February 23, 2021

Introduction

The high-level goal of this project is to create a simulation of a Minimum Viable Blockchain (MVB) called ZachCoin (ZC)[1]. An MVB is defined by the minimum functionality required to support a cryptocurrency that behaves something like Bitcoin. This project is based on the following blog post https://www.igvita.com/2014/05/05/minimum-viable-block-chain/ so it's not a bad idea to read through it before beginning.

For this project, you must implement an MVB that provides:

- Authentic transactions that are resistant to stealing and deniability, and which are then "broadcast" to all ZC Nodes (i.e. clients) in the network.
- An open competition amongst Nodes to validate transactions that: detects double spending and other attacks, provides a link to the previous verified transaction (creating an ordered chain), and uses a proof-of-work to raise the cost of running attacks against the network. The proof-of-work should be computationally difficult to solve, but easy to verify.
- "Broadcasting" verified transactions to the "network," and achieving chain consensus by adding verified transactions to a Node's chain as they arrive, while detecting forks in the chain and automatically switching to the longest chain.

ZachCoin may be a simulation, i.e. it need not actually be implemented as a P2P network. In other words, you may ignore the networking aspects required by a real blockchain implementation, and instead simulate all Nodes on a single computer. Your Nodes should be implemented as separate threads or processes, whose speed and "intentions" (malicious vs. cooperative) may be independently controlled. Transactions, their correctness, and the order they "arrive" in the system will also be controlled by the simulation.

It is strongly suggested that you implement this project in Python, or some other high level language, which has support for cryptographic hash functions, digital signatures, and threading.

Task I: Define a Transaction

Start your project by creating a definition for a ZC transaction block (each block contains exactly one transaction). Use JSON (which easily becomes a Python dictionary) and include fields for: a transaction identifier (a SHA256 hash), an input (a pointer to one or more prior transactions outputs), an output (a set of public key-coin values), signature(s), a hash pointer to a previous

[1] You don't actually have to call it this.

transaction, a nonce, and a proof of work (another SHA256 hash). Your system should support at least three transaction types: **transfer** (one input, one output), **merge** (multiple inputs from a single entity, one output), and **join** (multiple inputs from multiple entities, one output).

Transactions that arrive in your system will be defined by an input file that defines the transaction number, the type of transaction, its inputs and outputs, and a verifying signature. These unverified transactions should be added to a global Unverified Transaction Pool. The order transactions appear in the file will determine the order in which they arrive in the "network," but not necessarily the order in which they will be verified. See Task III below for additional details.

Task II: Create Verifying Nodes

Implement a Node that will:

1. Select at random a transaction from a global Unverified Transaction Pool (UTP), and check that the transaction is valid. If the transaction is invalid due to double-spending, report an error and permanently discard it from the network. If the transaction is invalid because the input doesn't yet exist, return it to the UTP.
2. If the transaction is valid, add a hash pointer to the last transaction on the Node's chain, and verify the transaction by running a proof of work.
3. Once mined, add the nonce, and final proof of work value to the transaction object. Then "broadcast" the transaction by placing it (if it does not already exist) in a global Verified Transaction Pool and removing it from the Unverified Transaction Pool. (You may consider "notifying" other nodes that they should stop mining.)
4. Nodes should add any additional transactions in the Verified Transaction Pool to their chain in the correct order, verifying that each transaction is valid, that the proof of work is correct, and that its hash pointer correctly points to the previous transaction. Nodes must support and detect forks in their chain, and remove all verified transactions from the shorter branch in their chain, returning them to the Unverified Transaction Pool.
5. If there are more unverified transactions, return to Step 1, otherwise the Node should sleep for 5 seconds before returning to Step 1.

A valid transaction is one in which:

- The signature verifies the transaction
- Each input is only used once on the chain
- The amount of coins in the output is satisfied by the number of coins in the input

Other details:

- Your proof of work algorithm should randomly choose a nonce, append the nonce to a serialized representation of the unverified transaction, and compute a SHA256 hash.

  You should repeat this operation until the hexadecimal value of your hash is less than or equal to the value (copy and paste it into your code):
  `0x00000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF`

- The hash to a previous transaction should be calculated as the SHA256 hash of the complete, serialized contents of a verified transaction.
- All Nodes should begin their chain with the Genesis Block (see Task III below), which defines the total number of coins in the system. No new coins will ever be generated.
- Your Node will need to support forks in their chain, meaning that a Node will not be able resolve a fork until at least one more transaction is added to one of the forked transactions. Only then will a "longer" chain be identifiable, and the verified transaction for the shorter chain returned to the Unverified Transaction Pool.

Task III: Transaction File

Either as a second program or as an option prior to starting your simulation, create the Transaction File. Start by creating at least five identities (as defined by five distinct public key pairs). Then generate and write to the Transaction File, either programmatically or randomly, a set of at least ten transactions, including at least one transfer, join, and merge operation. The set of transactions should also include at least one malicious transaction (e.g. a double spend), and at least one invalid transaction (*e.g.* one that is improperly signed, transfers more coins then are possessed, disobeys the conservation of coins, etc.).

The structure of a transaction in the Transaction File should be a serialized JSON object with the following keys:

- NUMBER: the hash of the input, output, and signature fields.
- TYPE: either TRANS, JOIN, or MERGE
- INPUT: a set of transaction numbers
- OUTPUT: a set of (value:public key) pairs
- SIGNATURE: a set of cryptographic signatures on the TYPE, INPUT, and OUTPUT fields

The first transaction in the Transaction File will always be assumed to be the Genesis Block, and accepted to be trusted and verified by all Nodes as the first verified transaction in their chain (and therefore need not have a valid signature). The Genesis Block should always be a transfer of 25 coins, with a null input and an output to one or more identities.

Task IV: Simulation Driver

Write a driver for your simulation that creates at least ten Nodes running as parallel threads.

After launching the Nodes, your driver program should open the Transaction File, adding them to the Unverified Transaction Pool, and sleeping between zero and two seconds (chosen at random) between transactions.

The simulation should run so long as there are transactions in the Unverified Transaction Pool. At the end of the simulation, when all transactions have been handled, each node should print out its transaction chain and exit. (Hopefully, all the chains agree!)

Task V: Additional Features

OPTIONAL FOR WINTER 2021

In addition to the ZachCoin simulation described above, you must implement at least one additional feature from the list below:

- Use a P2P library to make Nodes communicate over a network.
- Give Nodes identities, and create and award coins as part of the verification process.
- Create a web interface for creating transactions,
- Create a web-based visualizing the transaction chain, similar to blockchain.info
- Create a transaction chain forensics tool that is capable of performing queries against the chain, such as determining who has the most coins, who has more than X coins, list all the identities that owned a particular coin.
- Implement an alternative proof-of-work algorithm.
- Implement a coin wallet for your system.
- Use the cyberlab infrastructure to build a prototype implementation that could be used by future classes.

Task VI: Demonstration

You will demonstrate ZachCoin and additional features to the class on the last day of the course. Be prepared to answer questions about how your system and additional features work.

Please use the following guidelines for your final presentation:

- Aim for ~5-7 minutes in length (plus some additional time for questions)
- Discuss how you designed and implemented ZachCoin; be sure to address your design choices, the structures and algorithms you used, any challenges you experienced, why you stopped calling it ZachCoin, and any design decision that you think might be worth sharing. Be sure to address the features and limitations of your implementation.
- Be sure to present on the design and implementation of your "additional" feature. A demonstration of your feature working would also be useful. Screenshots are OK; a live demo is better.

Resources

ECDSA in Python

PyNACL: python binding to the libsodium library (an alternative to ECDSA)
Threading in Python, and some Examples
JSON in Python
Cryptographic Hash Functions in Python