UNIT UNIT UNIT

BAU BAU BAU

X.Teams X.Teams X.Teams

BLOCKCHAIN
HUB Academy

UNIT UNIT UNIT

BAU BAU BAU

X.Teams X.Teams X.Teams

UNIT UNIT UNIT

BAU BAU BAU

X.Teams X.Teams X.Teams

Satoshi way

XBIP: 2
Title: Bitcoin - Under the Hood
Created: 2019

CONTENTS

GENERAL INSTRUCTIONS

Theoretical tasks:

Hereinafter, the role of theoretical tasks is to gradually form a holistic understanding of the entire Bitcoin ecosystem. Keep in mind that when checking knowledge acquired during the course week, theoretical questions will be also included.

Practical tasks:

Hereinafter, in order to perform practical tasks, you need to use the Python programming language, version 3.5 and higher. This language offers easier access to the possibility of modeling and prototyping blockchain ecosystem and understanding the basic principles of its work. Production-ready projects mainly use low-level and system languages.

Each item of the task assumes modularity, that is, if the task is to write a script, then it must be a separate script with the extension * .py, or in a separate directory in which scripts are grouped according to their intended purpose. Not at all libraries are allowed to be used, pay attention to restrictions mentioned in the tasks.

Technical organization:

Submit your finished tasks to the GIT repository.
https://xteams-gitlab.unit.ua/module-N-login

Only work submitted to the repository will be considered for peer-evaluation.
Any extraneous files will be considered against you if this is not justified by any serious cause.

Deadline:

Access to the repository for making records closes after 7 days since tasks were presented at 09:01:00 AM GMT+2

XBIP: 2 - 04:02:2019 at 09:01:00 AM GMT+2

Peer-evaluation in Unit Factory: 04:02:2019 11:00 GMT+2

BEHIND THE WAY

Looking back at the first week, we can assume that its result was a working full node, which ensures the operation of the Proof-of-Work consensus algorithm (mining, resolving chain conflicts, emissions), building and validating transactions, building blocks and a basic wallet for working with transactions and the definition of balance.

This week will focus on the details of some components of the full node.

Important note

If according to the results of the first week some parts were not implemented, then it is possible to implement them immediately in the format that is offered in the second week (nevertheless, taking into account the logic of work from the first week).

On the one hand, this opens up the possibility of a partial switch from the tasks of the first week to the second week, on the other hand - it sets up more complex and voluminous tasks of the next level in relation to both specific task and its architectural component. It is worth remembering that one of the results of the course is supposed to be a prototype of a functioning full node. Due to the great coherence throughout the development period, it is allowed to add certain abstractions to speed up the development of all interconnections. For example, a script can be written to validate the original transaction from the second week, when the transaction itself is not yet ready. At the same time, mock data is sent to the input for script validation.

PART 01:
Transaction

At this stage, it is necessary to deal with the transaction in more detail and bring it to the form in which it can live in the Bitcoin network - **raw transaction**, as well as being built in accordance with the format and execution of the P2PKH script. An important point in the transaction is the inputs / outputs system. This contributes to invalidation of the transaction and prevents attempts of Bitcoin double spending.

When implementing it is forbidden to use ready-made libraries to form raw transaction. It is also worth considering the implementation of the transaction without updating SegWit.

Practical tasks:

It is necessary to bring the transaction to the original form in the Bitcoin network. Update transaction.py script - add missing fields, implement methods for calculating these fields and preparing transaction for serialization (as a visual example, you can use the picture below, for specific detailed information you need to read the documentation). As a signature, use ready-made methods from the wallet, update of which we will discuss later.

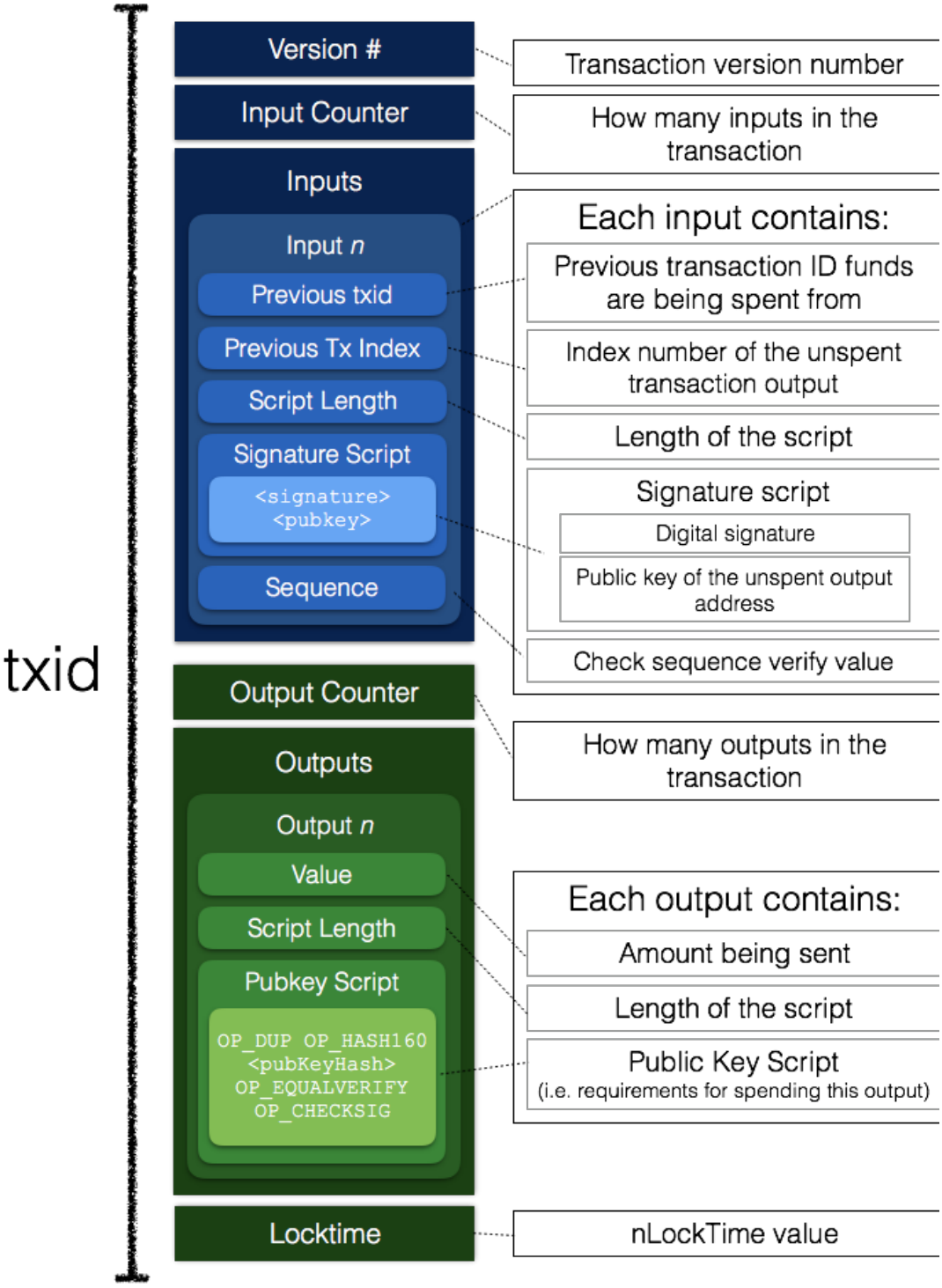
Bring the coinbase transaction to the original form in the Bitcoin network. Pay special attention to the formation of the fields scriptSig and scriptPubKey, which are different from the usual transaction and have other differences.

Implement transaction formation for the P2PKH script using the necessary opcodes. Next, we will cover the processing of these opcodes.

Update serialization of transaction in serializer.py according to original principles. A serialized transaction must be a Bitcoin raw transaction. For byte representation, it is allowed to use any suitable tools in Python ecosystem.

Bitcoin Transaction Schema

Pay-To-Public-Key-Hash (P2PKH)



PART 02:
Wallet

The wallet is almost unchanged in the interface, it retains its basic functions, updating their implementation.

Practical tasks:

Update the send method, based on the fact that the transaction is now in Bitcoin raw transaction format. This implies updating the processes of its formation, a specific signature of the required parameter and its definition in a special place in the transaction, as well as serialization of the above transaction format.

Update the broadcast method with the option to send a flag that determines where you want to broadcast – to the Pitcoin network or to the Bitcoin testnet. To connect to Bitcoin testnet, it is allowed to use ready proxy libraries and publicly accessible nodes.

PART 03:
Bitcoin SCRIPT

For implementing scenarios embedded in transactions, Turing Incomplete programming language Bitcoin Script is used. At this stage, it is necessary to implement a certain part of it - to implement the stack, formalize and execute the opcodes, based on data in such transaction fields as scriptSig and scriptPubKey.

Practical tasks:

You need to write script.py, which implements the following features:

- Provides LIFO stack operation for further adding to it and testing incoming commands
- Formalizes opcodes and presents them in encoding for consistency
- Implementing each opcode needed to ensure the execution of the P2PKH script
- Provides reading and execution of opcodes for P2PKH in the context of transaction validation (lock + unlock)

To encapsulate a specific logic in the context of transaction validation, the tx_validator.py script was previously used. Given this approach, you can replace the transaction validation itself with an implementation using the script described above without disturbing the network logic - the transaction will be validated in the place where the system requires it, using the capabilities of script.py. Make this replacement and implement transaction validation based on the scriptSig, scriptPubKey, and other important parameters.

PART 04:
UTXO set

With the addition to the transaction of its original capabilities, the opportunity to update the way to determine the balance of the address has appeared. To do this, you need to implement utxo pool, which will update the unspent outputs storage after adding a block to the chain and provide a programming interface for determining the balance.

Practical tasks:

- Write a utxo_set.py script that will process the save logic and quick access to unspent outputs
- When adding a block to a circuit, call the execution of the methods from the script above, which is necessary to save the new unspent outputs and delete used ones
- Update the method of obtaining balance in the wallet, based on data from the UTXO pool. The wallet interface still takes the address parameter, finds it in the UTXO pool and returns the result which value to spend. Consider that there might be several unspent exits.

PART 05:
Testing

According to the result of the past week, all parts of the ecosystem should be covered with tests. At this stage it is proposed to continue the test coverage.

Practical tasks:

Provide test coverage of newly created use cases - raw transaction, bitcoin script implementation, transaction validation, utxo set
If you couldn’t make some parts on time, you will be able to add the mock data instead of original flow and write tests with it

PART 06:
Check workflow

To test the results of this week, it was necessary to ensure the functioning of the Pitcoin testnet. Let’s recall the flow that was suggested to verifier for ensuring functioning of the network:

0. Download the repository from personal gitlab
1. With one command install the dependencies required for the project and enshrined in the requirements.txt
2. Launch the command line interface ./wallet_cli.py
3. Use the new command to generate a new private key and public address (or execute the import command with the private key parameter in WIF format). The result of the work of both teams must be the output of the private key in the WIF format and a valid public address (in the bitcoin mainnet format). The public address is saved to the file address, private key
4. Call the send command in wallet cli with two parameters - sender address and amount
5. After send command, serialized transaction would appears in console
6. Use the broadcast command for broadcast of signed and serialized transaction
7. Go to localhost:port/transaction/pending and find my transaction
8. In the next tab run miner cli and start mining process by calling mine with RPC server. If mining success, it should return newly block hash
9. Check pending pool route for transaction absence and get full chain by route /chain
10. Run Pitcoin infrastructure on another ports in another directory
11. Add node with miner cli and call consensus command
12. Mine two new blocks and check this chain by the previous web URL
13. Return to first running Pitcoin blockchain and call command consensus as a miner
14. Check fetch result in this chain

It has to be noted that the priority of these two weeks is to ensure the interaction between all parts of the system. This means that it is more important to take the path from creating a transaction to adding a block to the chain and resolving conflicts, than the implementation of a separate part without taking into account the integral interaction. In case of impossibility of full coverage of the functional, the intentional assumption of abstraction is permitted - mock data, templates. This will certainly affect the assessment, but to a much less extent than the absence of coherent ecosystem.

The priority task of this week is to ensure a transaction of a form which will be suitable for the Bitcoin network and the possibility of it entering the testnet block.

Verification and weight

This week's check is provided by the students themselves. To do this, you must perform the use cases described below. The weight is indicated within the framework of this week, the functioning of the network and the implementation of tasks from the first week will be assessed separately and affect the total result in the amount.

Use cases:

35% - the wallet collects raw transaction and it is transmitted over the network to Bitcoin testnet, where it successfully passes validation
25% - ensuring full network operation according to the flow described above (15 points) + test coverage (if the network is operational according to the result of the first week, then only adding tests is left)
20% - transaction validation on the Pitcoin network with a designated implementation of the Bitcoin Script and P2PKH script
20% - UTXO pool and ensuring balance at the user's address