Cryptocurrency

Cryptocurrency

What is it?

What problems does it solve?

What problems does it introduce?


What is
Bitcoin?

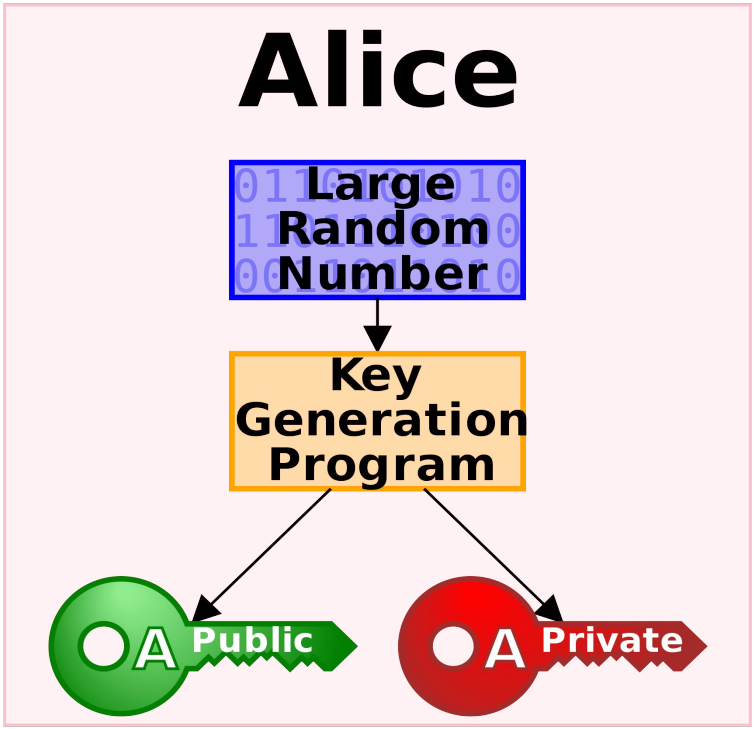
Concepts

Digital Signature (Public Key Cryptography)

Mining (Proof of work)

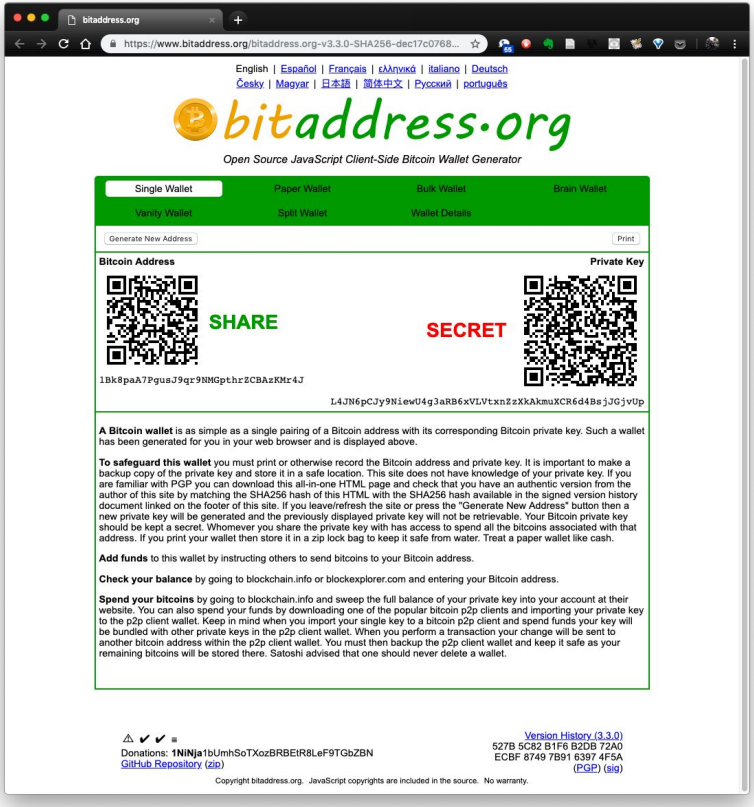
Blockchain (Double-spend problem)

Public/Private Key Cryptography

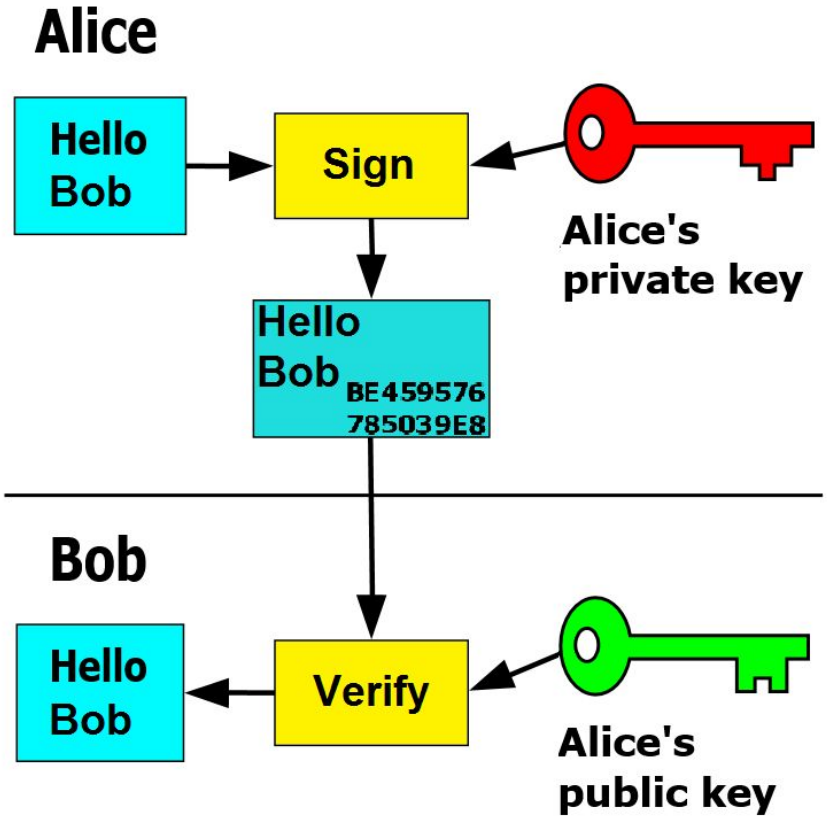


KohanX [Public domain]

Bitcoin Wallet

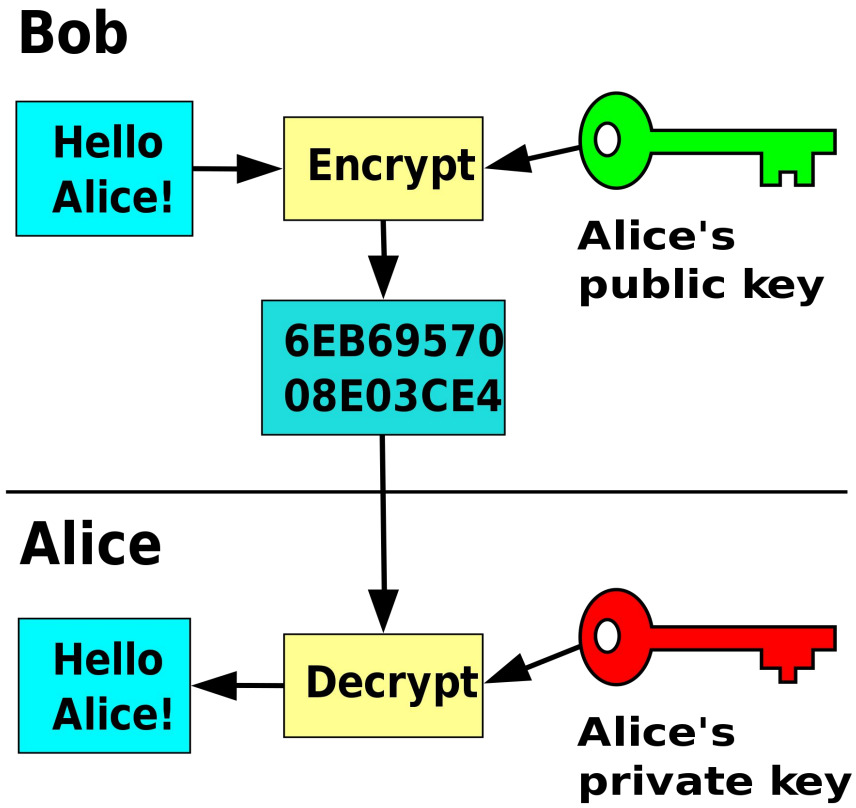


Hello Bob, I want to buy your book for 0.01XBT.

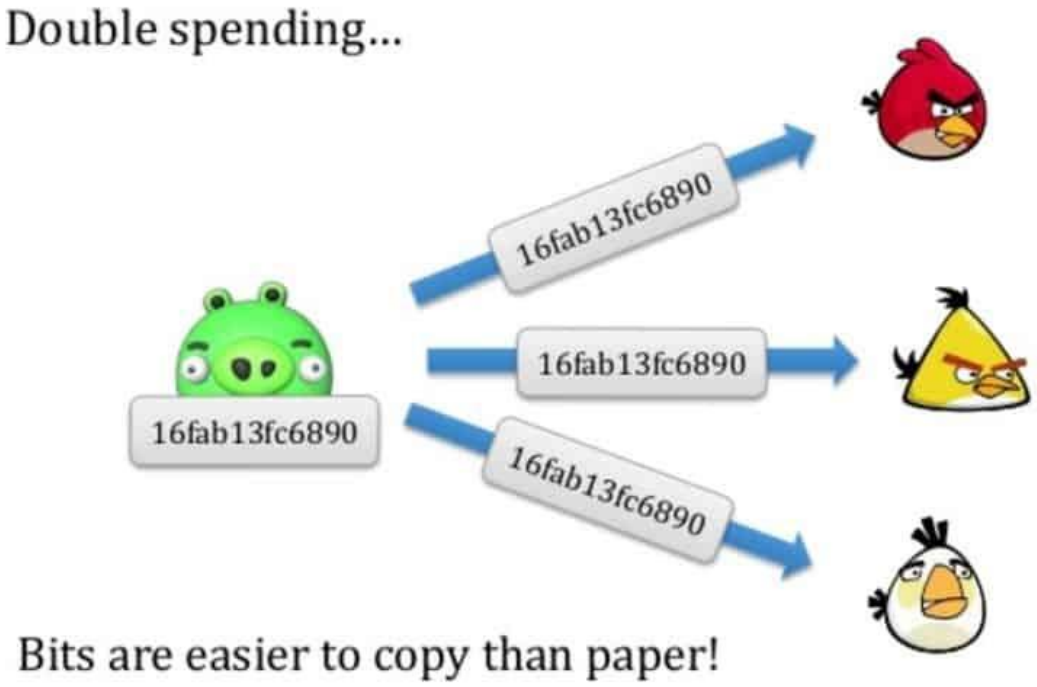


FlippyFlink [Public domain]

Hello Alice, here's the book.

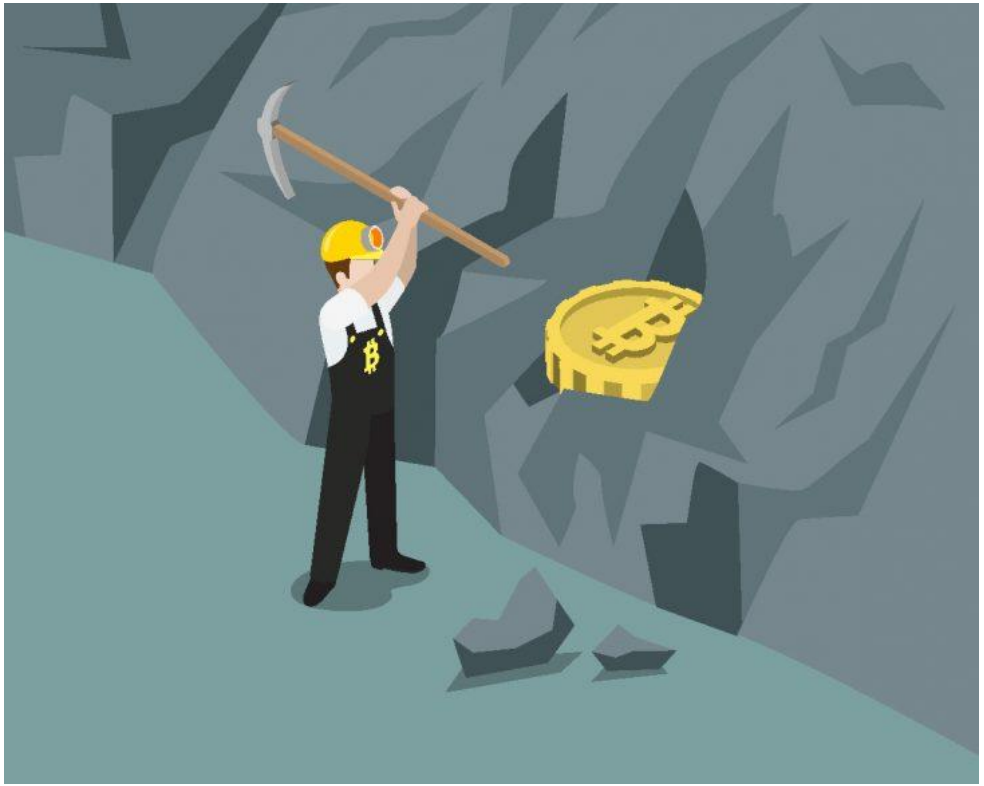


Double-Spend



https://coinsutra.com/bitcoin-double-spending/

Mining



Hash

SHA256:

“The quick brown fox jumps over the lazy dog.”

ef537f25c895bfa782526529a9b63d97aa631564d5d789c2b765448c8635fb6c

“The quick brown fox, jumps over the lazy dog.”

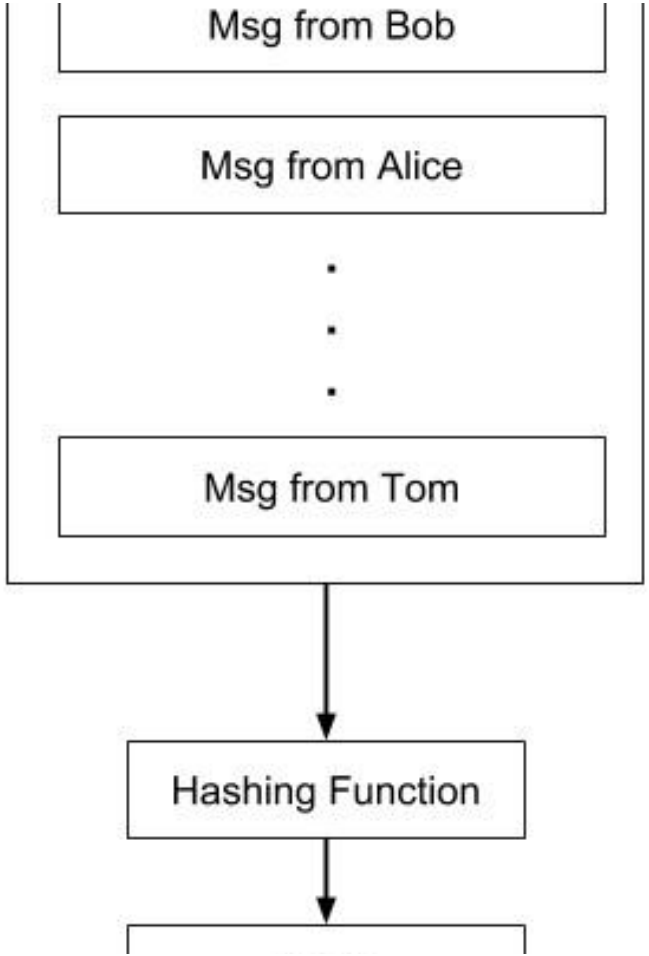
392bfd94c083e025e46d0be3ff9258c8bfc33bcd3296156f76c722f339a98dfb

The text of “War and Peace” http://www.gutenberg.org/files/2600/2600-0.txt

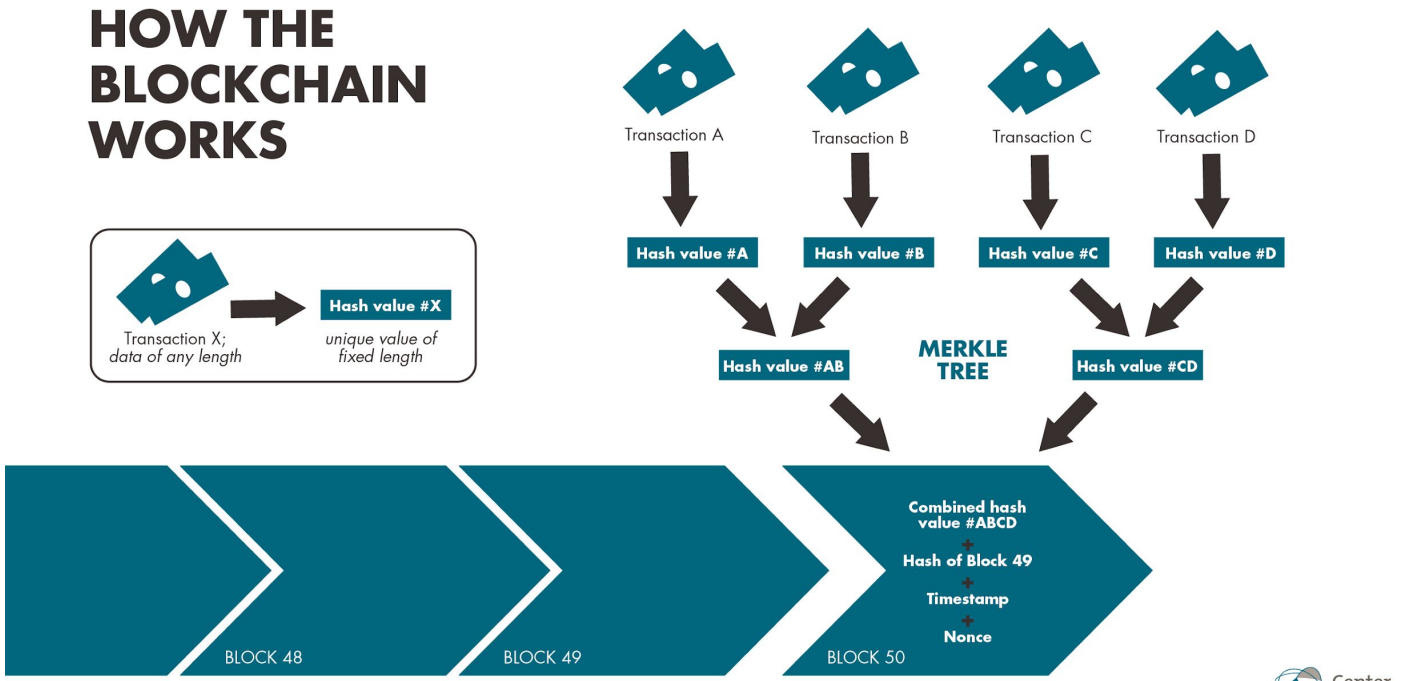
1feba561bf9106f3cbf6d78dd0c6056eef6ab59f15a30e64530ea6aea91d4e07

Mining Process - 12.5 XBT per block

https://www.tutorialspoint.com/blockchain/bitcoin_mining.htm



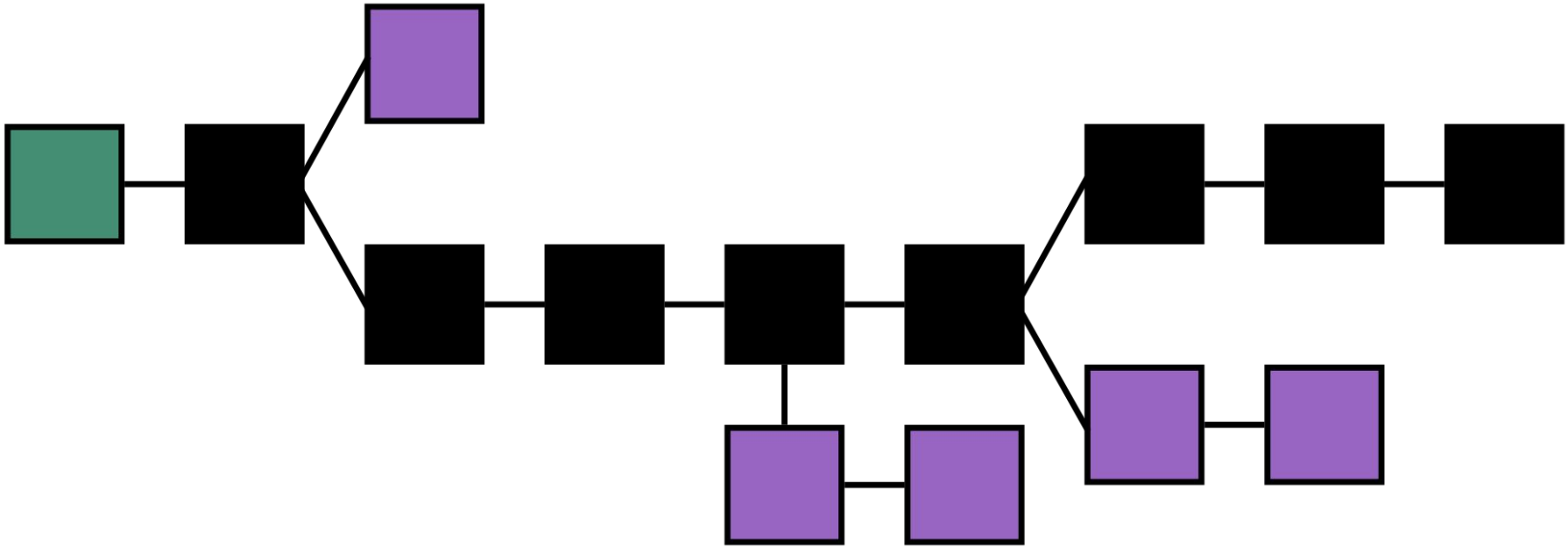
Blockchain



Reproduction of an original figure in "The Great Chain of Being Sure About Things" by the Economist

Center for Global Development

Longest Chain wins



What problems does it introduce?

Irreversible

Hackable

Not really anonymous

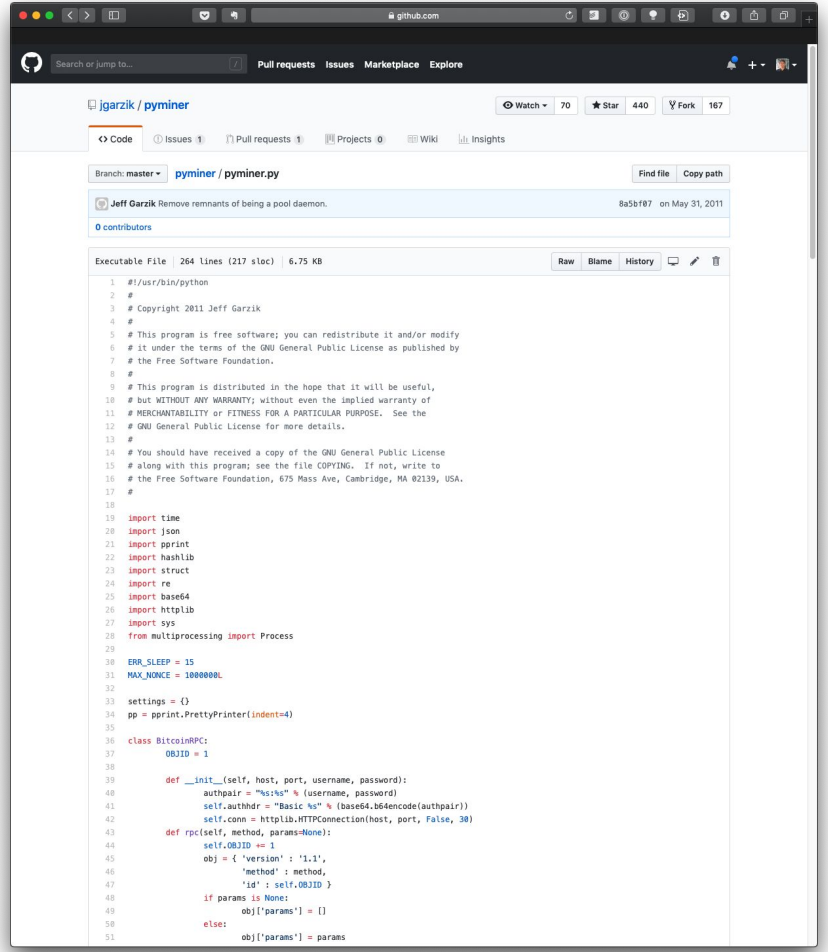
Not really decentralized

Table 4. Normalized global results for Energy, Time, and Memory

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

"*Energy Efficiency across Programming Languages: How does Energy, Time and Memory Relate?*", Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. In *Proceedings of the 10th International Conference on Software Language Engineering (SLE '17)*

https://github.com/greensoftwarelab/Energy-Languages#energy-efficiency-in-programming-languages


github.com
Search or jump to...
Pull requests
Issues
Marketplace
Explore
jgarzik / pyminer
Watch 70
Star 440
Fork 167
Code
Issues 1
Pull requests 1
Projects 0
Wiki
Insights
Branch: master
pyminer / pyminer.py
Find file
Copy path
Jeff Garzik Remove remnants of being a pool daemon.
8a5bf07 on May 31, 2011
0 contributors
Executable File | 264 lines (217 sloc) | 6.75 KB
Raw
Blame
History
#!/usr/bin/python
#
Copyright 2011 Jeff Garzik
#
This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation.
#
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
#
You should have received a copy of the GNU General Public License
along with this program; see the file COPYING. If not, write to
the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
#
import time
import json
import pprint
import hashlib
import struct
import re
import base64
import httplib
import sys
from multiprocessing import Process
ERR_SLEEP = 15
MAX_NONCE = 1000000L
settings = {}
pp = pprint.PrettyPrinter(indent=4)
class BitcoinRPC:
OBJID = 1
def __init__(self, host, port, username, password):
authpair = "%s:%s" % (username, password)
self.authhdr = "Basic %s" % (base64.b64encode(authpair))
self.conn = httplib.HTTPConnection(host, port, False, 30)
def rpc(self, method, params=None):
self.OBJID += 1
obj = { 'version' : '1.1',
'method' : method,
'id' : self.OBJID }
if params is None:
obj['params'] = []
else:
obj['params'] = params

Solidity by Example

Voting

The following contract is quite complex, but showcases a lot of Solidity's features. It implements a voting contract. Of course, the main problems of electronic voting is how to assign voting rights to the correct persons and how to prevent manipulation. We will not solve all problems here, but at least we will show how delegated voting can be done so that vote counting is **automatic and completely transparent** at the same time.

The idea is to create one contract per ballot, providing a short name for each option. Then the creator of the contract who serves as chairperson will give the right to vote to each address individually.

The persons behind the addresses can then choose to either vote themselves or to delegate their vote to a person they trust.

At the end of the voting time, `winningProposal()` will return the proposal with the largest number of votes.

```
pragma solidity >=0.4.22 <0.6.0;

/// @title Voting with delegation.
contract Ballot {
    // This declares a new complex type which will
    // be used for variables later.
    // It will represent a single voter.
    struct Voter {
        uint weight; // weight is accumulated by delegation
        bool voted;  // if true, that person already voted
        address delegate; // person delegated to
        uint vote;   // index of the voted proposal
    }

    // This is a type for a single proposal.
    struct Proposal {
        bytes32 name;   // short name (up to 32 bytes)
        uint voteCount; // number of accumulated votes
    }

    address public chairperson;

    // This declares a state variable that
    // stores a `Voter` struct for each possible address.
    mapping(address => Voter) public voters;

    // A dynamically-sized array of `Proposal` structs.
    Proposal[] public proposals;

    /// Create a new ballot to choose one of `proposalNames`.
    constructor(bytes32[] memory proposalNames) public {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;

        // For each of the provided proposal names,
        // create a new proposal object and add it
        // to the end of the array.
        for (uint i = 0; i < proposalNames.length; i++) {
            // `Proposal({...})` creates a temporary
            // Proposal object and `proposals.push(...)`
            // appends it to the end of `proposals`.
            proposals.push(Proposal({
                name: proposalNames[i],
                voteCount: 0
            }));
        }
    }
```