Czech University of Life Sciences

Faculty of Economics and Management

Department of Information Engineering

TRANSPARENT AND IRREVERSIBLE DATA USING BLOCK CHAIN

BACHELOR THESIS

Marek ŠÍP

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

BACHELOR THESIS ASSIGNMENT

Marek Šíp

Informatics

Thesis title

Transparent and irreversible data using Blockchain

Objectives of thesis

This work will briefly introduce the principles of current and past distributed databases and provide their comparison. The main part will be dedicated to analysis of Blockchain technology for transparent and irreversible data storage. The usage of Blockchain technology will be demonstrated by a smart-contract application.

Methodology

Methodology of this thesis is based on analysis and study of various information sources. Based on the synthesis of the gained knowledge an overview of available distributed databases will be provided, with special focus on the Blockchain technology. The usage of this technology will be demonstrated on an example application. The application will be tested and deployed and the results gained from deployment will be evaluated.

The proposed extent of the thesis

35-40 pages

Keywords

bitcoin, blockchain, consensus, open-source, distributed database

---

Recommended information sources

ANTONOPOULOS, Andreas M. Mastering bitcoin. First edition. O'Reilly Media, 2014, xxi, 272 pages. ISBN 978-149-1902-608.

BITCOIN PROJECT.Bitcoin Documentation [online]. 2009 [cit. 2015-06-04]. Dostupné z: https://bitcoin.org/en/developer-documentation

ETHEREUM SWITZERLAND GMBH. Ethereum Documentation [online]. 2013 [cit. 2015-06-04]. Dostupné z: https://github.com/ethereum/wiki/wiki

---

Expected date of thesis defence

2015/16 SS – FEM

The Bachelor Thesis Supervisor

Ing. Jiří Brožek, Ph.D.

Supervising department

Department of Information Engineering

Electronic approval: 20. 2. 2016

Ing. Martin Pelikán, Ph.D.

Head of department

Electronic approval: 20. 2. 2016

Ing. Martin Pelikán, Ph.D.

Dean

Prague on 08. 03. 2016

Transparent and Irreversible data using Block chain

This thesis will be dedicated to analysis of Block chain technology as a trust-free system for transparent and irreversible data exchange and storage. The principles of current and past distributed transaction ledgers will be provided to accent the huge improvement within the last years, mainly due to invention of Bitcoin, the decentralized currency system. Since that time, block chain has been adapted and modified into decentralized voting, token exchange or domain name registration systems.

Currently, the Block chain technology is considered to be as important as the invention of internet itself. Block chain is considered to have potential to succeed due to its transparent, privacy-enhancing, decentralize, trust-free and open-source development nature.

The work will be introducing advantages of decentralized applications used to overcome imperfections of centralized systems such as, single point of failure and lack of transparency. More specifically, this thesis will describe the engine and structure behind block chain. Moreover, how society in general could benefit from its usage. The usage of Block chain technology will be demonstrated by a smart-contract application built on top of Ethereum platform.

Key words

Block chain, Ethereum, Bitcoin, dapp, consensus, decentralization, distributed timestamp database, peer to peer and decentralized computing

Veřejná, neměnná data pomocí block chain technologie

Tato práce je věnována analýze technologii blockchain, sloužící k trvalému uchování veřejně přístupných dat uložených v uzlech peer-to-peer sítě. V současné době je důležitost této technologie, prvně použité pro Bitcoin protokol, přirovnávána k samotnému vynálezu World Wide Web, protože zcela mění pojem využívání internetu a současnou architekturu klient-server na bezpečnou peer-to-peer komunikaci za pomocí kryptografie. Stavebními kameny této technologie jsou vývojářská a uživatelská transparentnost a decentralizovaná síť bez nutnosti důvěry v centrálního správce.

Od vynálezu Blockchain v roce 2009 byl tento protokol převzat, aby plnil funkci registrací DNS domén, nebo implementaci transparentních a snadno auditovaných bank. Velmi zajímavé využití navrhla společnost Ethereum Foundation., která vyvinula platformu pro úložiště decentralizovaných aplikací. Tato platforma umožňuje využít blockchain pro jakoukoliv aplikaci, např. finanční či demokratický systém.

Tato práce představí výhody decentralizovaných aplikací, které napravují hlavní nedostatky současných centralizovaných systému, jako jsou riziko technického nebo protiprávního selhání poskytovatele služby, MITM útoky a nedostatek transparentnosti.

Klíčová slova

Block chain, Ethereum, Bitcoin, dapp, konsenzus, decentralizace, distribuovaná časově označená databáze, peer to peer a decentralizovaný výpočetní systémy

Declaration

I declare that I **Marek Šíp** have worked on my bachelor thesis titled “Transparent and Irreversible data using Block chain” by myself and I have used only the sources mentioned at the end of the thesis. As the author of the bachelor thesis, I declare that the thesis does not break copyrights of any their person.

In Prague on 14.3.2016 _____

Acknowledgement

I would like to thank Ing. Jiří Brožek, Ph.D for supervising my thesis, to ArrowSys s.r.o. for providing exceptional testing environment and to all participants in block chain related discussions in Paralelní Polis.

Table of Contents

1. Introduction 5

2. Objectives and Methodology 6

2.1 Objectives 6

2.2 Methodology 6

3. Account for technology 7

3.1 Bitcoin Block chain 7

3.1.1 Introduction 7

3.1.2 Block 8

3.1.3 Proof of work 8

3.1.4 Difficulty 10

3.1.5 Block structure 11

3.1.6 Block rewards 13

3.1.7 Merkle Trees 15

3.1.8 Transactions 16

3.1.9 Addresses 18

3.1.10 Hierarchical addresses 19

3.1.11 Account to Economic effect 20

3.1.12 Smart Contracts 20

3.2 Ethereum 21

3.2.1 Intention and goals 21

3.2.2 Smart contracts 22

3.2.3 Accounts 25

3.2.4 Transactions 25

3.2.5 Ethereum Virtual Machine 26
3.2.6 Gas policy 27
3.2.7 Scripting languages 28
3.2.8 Merkle Patricia trees 29
3.2.9 Differences from other block chains 29
3.2.10 Frontier release 30
3.3 Other similar platforms 30
3.3.1 Blockstream 30
3.3.2 Eris project 30
3.3.3 Maidsafe 30
3.4 Potential 31
4. Practical usage of Ethereum Block chain 32
4.1 PureVote - Decentralized Voting Application 32
4.1.1 Prerequisites 33
4.1.2 Classes 34
4.1.3 Events 34
4.1.4 Methods 35
4.1.5 Publishing contract 35
4.1.6 Executing contract functions 35
4.1.7 Gas usage summary and cost 38
4.1.8 Confirmations 38
4.1.9 Evaluating results 39
5. Used environment 41
6. Discussion and Results 42
7. Conclusion 44

8. Bibliography 45

9. Appendix 47

9.1 Acronyms and used abbreviations 47

10. Supplements 48

List of Equations

Equation 1: Bitcoin block difficulty calculation (ANTONOPOULOS, 2014) 11

Equation 2: Bitcoin block reward halving (Harding, 2015) 14

List of Tables

Table 1: SHA256 computation example (Bitcoin Wikipedia community, 2016) 9

Table 2: Structure of block data (Harding, 2015) 12

Table 3: Structure of coinbase transaction (Bitcoin Wikipedia community, 2016) 14

Table 4: Gas costs per operation 28

Table 5: Contract testing fee overview 38

List of Figures

Figure 1: Block chain visualization (Harding, 2015) 8

Figure 2: Block chain difficulty historical chart (Blockchain Ltd., 2016) 10

Figure 3: Block #100 data 12

Figure 4: Merkle Tree scheme. Scheme drawn by author inspired from (ANTONOPOULOS, 2014) 16

Figure 5: Mist interface after publishing contract 36

Figure 6: Contract state after termination of Poll 40

1. Introduction

In 2008 a revolutionary brand new currency system was proposed in a paper called *Bitcoin: Peer-to-Peer Electronic cash system*. The system allows participants to transfer value within a decentralized network, instead of trusting central authority. Thus, it became obvious that more than a money, the concept that was Bitcoin built on top of, is a decentralized trust system, rather than ordinary currency, which was in its case *just* an application.

Until the invention of Bitcoin a person willing to use their cash electronically had to trust *central authorities* such as banks. These entities were trusted to guard and transfer values according to owner's decisions and keep the entire history of transactions as a proof of client's account balance.

In the Bitcoin system, the transaction records and ownership authentications are proved and managed by chain of digital signatures within a distributed database called Block chain (Nakamoto, 2008).

Unlikely from physical token money, Bitcoin developers invented units called bitcoins that only exists in binary form attached to a particular digital key pair. Lacking of any physical value, such units can be easily copied and reused if no supervisory system is present.

In a decentralized network, a certain mechanism was needed to be introduced to prevent the peers that already transferred the ownership, to use the same digital coins for any future transaction, simply to prevent double-spending problem.

The paper proposed, among many other important algorithms, solution to achieve consensus about all past transactions, a transparent proof-of-work transaction ledger called *Block chain* that will be synchronized across every peer involved in the network.

2. Objectives and Methodology

2.1 Objectives

This bachelor thesis will explore the potential of block chain and decentralized applications that are already, or could be built using this system. Moreover, the huge development since introduction of Bitcoin in 2009. The thesis will mainly focus on the Ethereum platform, its implementation of block chain, capabilities and opportunities of applications that are planned to be built on top of it. Furthermore, main aspects differentiating decentralized transactions ledgers from each other will be described. Also how society in general could benefit from using them.

2.2 Methodology

Methodology of this thesis is based on analysis and study of various information sources. Based on the synthesis of the gained knowledge and overview of available distributed transaction platforms will be provided, with special focus on the Block chain technology. The usage of this technology will be demonstrated on an example application. The application will be tested and deployed and the results gained from deployment will be evaluated.

3. Account for technology

3.1 Bitcoin Block chain

3.1.1 Introduction

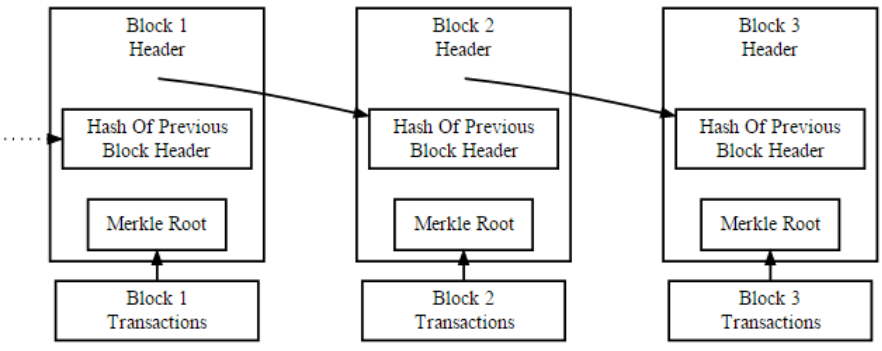
The network timestamps the transactions by hashing them into an on-going chain of hash-based proof-of-work, creating a record that cannot be rewritten without redoing the proof-of-work (Nakamoto, 2008)

Block chain is a distributed peer to peer timestamp database that stores formatted history of digital signatures. After the invention of Bitcoin, the block chain was adopted to serve various purposes other than electronical cash system.

The database could contain any type of additional metadata together with the digital signatures. In the Bitcoin network, the metadata is the value of coins transferred, Namecoin's network metadata contains domain registration info. One of the most interesting usages of block chain is Ethereum network which stores application bytecodes generated from Ethereum source code compilers.

In the block chain the transactions are formed into separate *blocks*. Blocks can be stored in a plain file or any database engine. Bitcoin Wallet, the original software built according to proposal from 2009 uses Google's LevelDB (Bitcoin Wikipedia community, 2016), nonetheless, the metadata can be stored in any form as long as integrity is kept.

Each block contains a Merkle root hash of its transactions and the hash of previous block as a reference, known as the parent block. Thus, the blocks can be visualized stacked on top of each other. The only block that does not have the reference to its parent is the first block, called *Genesis block* that is statically defined within the software (Harding, 2015).



Simplified Bitcoin Block Chain

Figure 1: Block chain visualization [1] (Harding, 2015)

3.1.2 Block

Block is an element of continuous linear sequence of block chain ledger. Block contains list of recent transaction data collected from other participating nodes. Each block is added to the end of the chain as pictured on Figure 1. Blocks are considered irreversible once they are accepted by all nodes within the network, because their propagation to network is backed up by CPU proof-of-work.

Blocks are propagated by a single network node that collects the transactions from other nodes that are not willing to provide CPU proof-of-work to propagate blocks themselves. The nodes propagating blocks are referenced as "miners".

The node propagating the block has to provide as much CPU proof-of-work as is computed according to the time it took to propagate previous blocks. The bitcoin network adjusts to consistently keep propagation time at 10 minutes, meaning one block is publishes every ten minutes. Maximum block size is 1 MB (Harding, 2015).

3.1.3 Proof of work

Proof of work is computational hashing power required by the node in order to publish a block to the network. The hashing is in form of completing enough SHA256[1] hashing computations to satisfy previous block's conditions. The condition is known as *nonce*. Nonce is 32-bit field value included in previous block header. Miners compute many values to find the value that meets the criteria of number of required leading zero bits – the nonce. (ANTONOPOULOS, 2014)

SHA256 example

[1] Secure hashing algorithm (SHA) is a hash function comparing the computed „hash" to the expected value.

The goal is to find value that has three leading zeroes. The base string for computing will be “Hello, world!” First column states base string and second the hash itself.

Table 1: SHA256 computation example (Bitcoin Wikipedia community, 2016)

Hello, world!0	1312af178c253f84028d480a6adc1e25e81caa44c749ec81976192e2ec934c64
Hello, world!1	e9afc424b79e4f6ab42d99c81156d3a17228d6e1eef4139be78e948a9332a7d8
Hello, world!4250	**0000**c3af42fc31103f1fdc0151fa747ff87349a4714df7cc52ea464e12dcd4e9

This took 4251 SHA256 computations[2] to find the desired output hash that meets the condition.

For a record, an exceptional personal computer with superefficient graphical card ASUS HD7990-6GD5 can do 1518 kHash operations per second, which is 1 518 000 hashes per second[3].

Currently required computational power on Bitcoin network is 144,116,447,847.[4] This means that it would take such a computer in average 26 hours to find desired hash that would match the criteria.

[2] Application used for computing SHA256 http://www.movable-type.co.uk/scripts/sha256.html

[3] Page with hash computation data https://litecoin.info/Mining_hardware_comparison#NVIDIA

[4] Block chain difficulty chart: https://blockchain.info/charts/difficulty

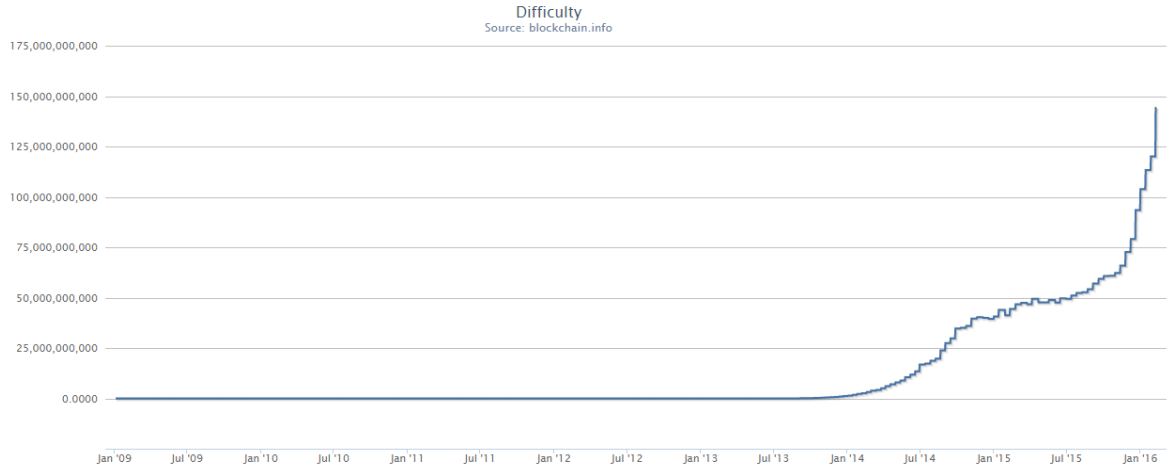


Figure 2: Block chain difficulty historical chart (Blockchain Ltd., 2016)

3.1.4 Difficulty

Difficulty measures how difficult is to find a hash complying with certain criteria. Originally, with the Genesis block difficulty started at 1. New blocks are added to the block chain if their difficulty is high enough to what is expected by the consensus protocol. In the Bitcoin network, ideally, 2,016 blocks should take about 1,209,600 seconds (two weeks). Every 2,016 blocks, network uses timestamps stored in the block headers of the first (1.) and last (2,016.) block to calculate seconds that elapsed between propagation of these blocks to calculate new difficulty rate. (Bitcoin Wikipedia community, 2016)

<u>There exists two scenarios</u>

1. It took less than two weeks to generate 2,016 blocks – difficulty value is increased by as much as 300% so that next 2,016 blocks would take about 14 days to be generated.
2. It took more than two weeks to generate 2,016 blocks – difficulty value is decreased by as much as 75%.

As stated in official Bitcoin developer documentation[5] in official implementation is small scale bug that currently calculates previous difficulty on data collected from last 2015 blocks. (Harding, 2015)

Bitcoin block difficulty calculation

$$New\ Difficulty = Old\ Difficulty * \frac{Actual\ time\ of\ Last\ 2016\ Blocks}{20160\ minutes}$$

Equation 1: Bitcoin block difficulty calculation (ANTONOPOULOS, 2014)

3.1.5 Block structure

Block among other metadata coins information about recently collected transactions and its block header.

For an example let's take block with height 100[6]. For a record, at the time of writing (9.2.2016) the block height is at 397514.

Bitcoin block data can be obtained using Bitcoin daemon API[7] distributed with original Bitcoin Core[8] software. The Bitcoin console can be accessed in Debug window.

Fetching block number #100 will require knowing the block's hash. Command *getblockhash _blocknumber_* will return it. Further command *getblock _hash_* will return parsed block data.

[5] https://bitcoin.org/en/developer-guide#proof-of-work

[6] http://blockr.io/block/info/100 Parsed information about block #100

[7] Original Bitcoin API https://en.bitcoin.it/wiki/Original_Bitcoin_client/API_calls_list

[8] https://bitcoin.org/en/choose-your-wallet Bitcoin Core download

Figure 3: Block #100 data

Block data[9]

Table 2: Structure of block data (Harding, 2015)

Attribute	**Byte size**	**Description**	**Header (H) / body (B)**
size	4	Bytes size of block data – header and transactions	B
Magic No.	4	Identifies network ID, value is always 0xD9B4BEF9. Each network has its ow e.g. bitcoin testnet ID is 0xDAB5BFFA	B
transaction count	1-9	Number of transactions included in the block. Displayed as array of transactions.	B

[9] https://en.bitcoin.it/wiki/Block block structure description

transactions		Non-empty list of transactions – first transaction is defined by miner - further described in 3.1.6	B
version	4	Version of Bitcoin Core system	H
previousBlockHash	32	Previous block header hash	H
merkleroot	32	Hash calculated on all data about transactions	H
time	4	UNIX timesamp, seconds elapsed since 1st February 1970.	H
nonce	4	Count of hashes required to produce valid SHA256 hash.	H
bits	4	Difficulty	H

Some attribute descriptions are not provided as they are part of additional block chain querying such as *nextblockhash* or number of *confirmations*.

3.1.6 Block rewards

Miners who collect transactions and provide their hardware computational power to propagate new blocks are using their own resources such as hardware and electricity. The network obviously cannot be kept alive as a lossmaking activity without any reward to the miners.

An incentive was proposed in the original paper to reward each valid block propagation with the release of brand new coins. This concept was necessary as there is no other authority to issue them. In the Bitcoin network there is predictable release of new coins and is defined in the protocol to be steadily decreasing.

Controlled supply started at 50 coins reward per block and being halved every 210,000 blocks – about 4 years if average block takes 10 minutes.

Equation 2: Bitcoin block reward halving (Harding, 2015)

$$10 * \frac{210{,}000}{60 * 24 * 365} = 3{,}99\ years$$

Bitcoin protocol was published on 1st of January 2009 with block reward 50 units, first halving was on 28th of November 2012 when reaching 210,000th block. Next is planned to be at in the middle of 2016.

Additionally, a transaction fee system was introduced to increase miners' rewards. Once, all the predetermined coins are distributed, note that Bitcoin protocol is defined to have solely 21,000,000 coins distributed, the miners will only be reward from the transaction fees.

The first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. (Nakamoto, 2008).

As stated in the block structure, the first transaction specifies additional block data and output address (3.1.9) of block issuer. Such transaction is called *coinbase transaction*.

Sometimes when the nonce – number of SHA256 tries overflows the 32bit integer, parameter *extraNonce* is added to the Coinbase data attribute.

Structure of coinbase transaction

Table 3: Structure of coinbase transaction (Bitcoin Wikipedia community, 2016)

Attribute	**Byte size**	**Description**

Transaction hash	32	Hash containing zero bits as all transaction has no input UTXO[10] 3.1.7
Output index	4	All bits are ones: 0xFFFFFFFF
Coinbase date size	1-2	Length of coinbase data itself
Doinbase data	2-100	2-100 bytes
Sequence number	4	Set to 0xFFFFFFFF

3.1.7 Merkle Trees

Merkle tree is a data structure for minimization of data storage and verification of integrity. Merkle trees are used to summarize all transaction in block chain through a single hash called *merkle root hash 3.1.2*. Every branch except leaves contains hash of previous two branches.

As every transaction has transaction identification - hash, merkle root is calculated by hashing as many times is needed, two previous hashes, starting from transaction hashes (in Merkle tree scheme called merkle leaves). These initial hashes produce values known as branches. Two branches are hashed over and over to produce single hash, the merkle root. These calculations will produce each time same result. SHA256 produces always 64 characters string with only hex characters ([0-9], [A-F])

Because merkle tree is binary, even number of leaf nodes is required. In case of odd number of transactions, the remaining transaction hash will be duplicated, this is known as *balanced tree*.

This scheme allows light-weight nodes Simplified Payment Verification (SPV) because they only need to download block's merkle root in order to check if particular

[10] Unspent Transaction output

transaction was included in a block. “Downloading entire block requires over 500,000 bytes whereas block headers are only 140 bytes” (Harding, 2015).

Merkle trees allow for efficiently verifiable proof that a particular transaction is included in a block. If a node wants to verify a transaction, it does not have to download the whole block, but only a branch of a tree. Each branch in a tree is hash of two other branches below. The goal is to minimize the data that client needs to process in order to find relevant transaction information.

This scheme was not introduced with bitcoin, Ralph C. Merkle proposed it in 1979 (Merkle, 1979).

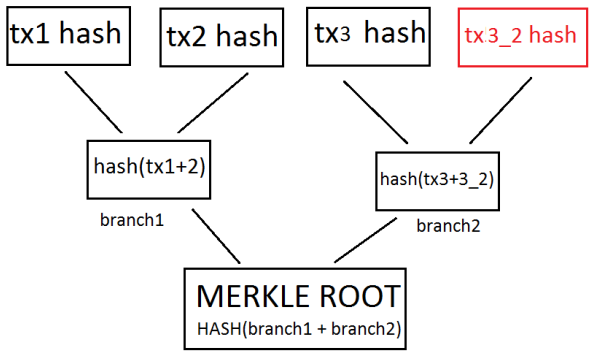


Figure 4: Merkle Tree scheme. Scheme drawn by author inspired from (ANTONOPOULOS, 2014)

3.1.8 Transactions

Transactions are the most important part of the protocol. Each transaction is publicly recorded into the block chain. Each transaction composes of output(s), input(s) and digital signature(s).

Transaction inputs are unspent transactions outputs (UTXO) from previous transactions that couldn’t be used more than once in order to prevent double-spending attack.

Outputs are bitcoins receivable by recipient’s address that will become UTXO after validating the transaction. One transaction can include arbitrary amount of UTXO and

outputs as far as it provides digital signature and pays fee which is calculated per byte of data.

UTXO is a fundamental of efficient block composing and transaction verification. “It represents chunk of bitcoins locked to a specific owner and recorded on block chain” (ANTONOPOULOS, 2014). This citation clears out common misconception people may have when encountering bitcoin protocol for the first time. Bitcoin does not store current balance of an account, whereas the network keeps track of unspent transaction outputs. Each miner keeps list of currently unspent transaction outputs as it is more efficient than searching whole block chain each time transaction is received.

In essence, transaction is about 300-400 bytes of data, depending on the amount of UTXO and destination addresses.

Transaction contains following fields:

- Version of rules (bitcoin core version)
- The amount of transaction inputs (UTXO before propagating) with UTXO data
- The amount of transaction outputs (new UTXO) with UTXO data
- Locktime which allows originator to specify that the transaction shouldn’t be include into block before specific block height or UNIX[11] timestamp passed.

Once transaction is generated, it is broadcasted to connected peers. These nodes check if all rules are met e.g. UTXO haven’t been yet spent, transaction includes a fee for miners, data have been digitally signed by private keys corresponding to particular UTXO. If transaction is validated by one node, it is then propagated to other nodes and finally reaches all nodes in the network. “A new validated transaction injected into any node on the network will be sent to three to four of the neighbouring nodes” (ANTONOPOULOS, 2014).

There exists also extended types of transactions that require more than one signature for spending UTXO, those are known as M-of-N. M represents threshold that is minimum of required signees from total N possible signees.

[11] UNIX timestamp is a number of seconds that passed since 1.1.1970

Balances are formed from inputs and outputs rather than subtracting and adding balance corresponding to specific address. This allows to keep separate storage of unspent transaction outputs.

3.1.9 Addresses

There are different types of address used in block chains, but all of them are in a form of asymmetric[12] key-pair. Standard address is a key pair that is formed from *Public* and *Private* parts, which is the same form as used for the digital certificates used in Czech Republic to electronically communicate with government. Those have to be generated from Central Authorities. Unlikely, key pair for bitcoin address can be generated by available algorithms.

More complex addresses can combine more than one private key these are called *multi signature addresses* and are prefixed with number 3, standard addresses with 1 in the final format.

Public key is a derived from the private key, which can be freely shared across the network, to untrusted entities. Public key serves for common purpose as e-mail address or bank account, it does not give the recipient of this information the opportunity to unlock the account funds. Moreover, it becomes mathematically impossible to "rollback" or compute the private key from public key, even though the value is derived. ECDSA[13] formula is used for generating the public key which ensures high-standard security across "time and space". Using private-key that has bit length 100, requires 2^{100} operations to find out the private key from public-key (Harding, 2015).

Private key is the part that is used to create digital signatures in order to prove third parties that specific public key corresponds to the person who supplied the digital signature.

Digital signature is a cryptographic scheme for authenticating sender of a message. Sender usually sings message with his private key giving the recipient opportunity to verify the signature and thus believe that the sender is the owner of the public key that is usually attached in unencrypted state.

---

[12] Asymmetric cryptography ensures that from the Public key part cannot be resolved the Private part.

[13] Elliptic Curve Digital Signature Algorithm

Several other functions and technical improvements in formatting are used in order to simply the format as much as possible, those will be described in following datagram.

Generation of Bitcoin address (ANTONOPOULOS, 2014)

1. Generate ECDSA key pair with optional input string to be taken as see
2. Then the result is put through SHA-256 and its result
3. Hash through RIPEMD-160[14]
4. To this result is attached network specification represented as leading byte.
5. Next SHA-256 is computed
6. On the previous result another SHA-256. The first 4 bytes are stored as later address checksum.
7. Checksum is attached to the end of result from stage 4. And Bitcoin address is produced.

Lastly, the result is transformed into the Base58 format which is alphanumerical set excluding zero, O and I and l (lowercase L), in order to prevent misinterpretation of characters while reading.

In the Bitcoin console address key pair is generated using command *getnewaddress* or *addmultisigaddress*.

Standard address example: **1**Meum86kuhVe4nNeSJo3fgjutqEUnc3bpB, HD wallet example: **3**GyK2Km65tmfG5gXYUAh2DyupYCtSv47Jp

[15]

3.1.10 Hierarchical addresses

Special type of address schema called Hierarchical Deterministic[16] address supports derivation of sub addresses from either master private key or master public key. Feature of

[14]

[15] https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses

[16] Bitcoin Improvement Proposal (BIP) 32: Hierarchical Deterministic addresses https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

deriving sub addresses is very useful in scenarios where public key needs to be given to untrusted third parties without any privileges to manage available funds. (Marek Palatinus, 2014). On the other hand sub addresses generated from master private key gives full control of funds, limited to particular key, this does not apply to the master key.

3.1.11 Account to Economic effect

With stable and predictable unit supply, bitcoin is considered to have deflationary attitude, as fixed amount of units would value the units over time, thus decrease the price level of goods and services purchased by bitcoin.

3.1.12 Smart Contracts

Contracts are transactions recorded in the block chain with a different purpose than exchanging electronical cash, e.g. to achieve financial agreement.

A simple contract can be written on the bitcoin block chain protocol using nLock (Lock time) parameter of the transaction, which will propagate transaction only if the expected amount of time (measured by elapsed blocks) is reached. An example is to propagate transaction with nLockTime = 400,100, means that the miners will include the transaction to the block after block 400,099 has been constructed, and no miner would include invalid transaction because it is not profitable to provide computation power on blocks that will be declined by the network.

Other extended contracts can be written using Scripts[17] that has Fortran-like commands, but lack of Turing-complete support, which means no loops can be made.

"There is also another, equally important part of the Satoshi's grand experiment, other than decentralized electronical-cash: the concept of a proof of work-based block chain to allow for public agreement on the order of transactions (…) And now, attention is rapidly starting to shift toward this second part of Bitcoin's technology, and how the block chain concept can be used for more than just money". (Buterin, 2014) According to the White

[17] https://en.bitcoin.it/wiki/Script using scripts

Paper written by Vitalik Buterin in late 2013, the block chain can be reused to decentralize digital assets, registries, exchanges, financial derivatives, voting and much more.

3.2 Ethereum

Ethereum is smart contract platform inspired by block chain technology introduced in 2008. Its elemental unit is called *ether*. Ether (symbol Ξ from Greek alphabet Xi), similarly to bitcoin is divisible up to 10^{-18}, its smallest subunit is called *wei*. Due to the *fee-by-computation*[18] policy, Ether (abbr. ETH) is sometimes referred as the fuel of Ethereum (ETHEREUM SWITZERLAND GMBH., 2015).

3.2.1 Intention and goals

Ethereum Development team takes the block chain usage onto significantly general level. The goal as written the White Paper, is to provide Turing-complete programming language[19] that enables developers to create applications with any rules defined by the smart contract, from finance systems, register systems to democracy voting systems.

The reason behind this was that Bitcoin protocol does not extend its state transition system which is only limited to "decrease amount on this address and increase it on this address".

Vitalik Buterin summarized the reason why he decided not to build such a complex platform on top of Bitcoin protocol into 4 main insufficiencies the protocol (Buterin, 2014)

1) Lack of Turing-completeness – Mainly missing loop functions
2) Value-blindness – Very difficult to bind real-world values such as BTC/USD exchange rate into the block chain

[18] In ethereum network, the contracts have to pay per processing power required to execute its bytecode.

[19] A programming language with complete instruction set including conditional and cycling abilities

3) Lack of state - UTXO[20] can either be spent or unspent, there is no space for another stage, and thus it is hard to implement more complex system like DAO[21] that manages the contracts and funds only by rules encoded within the block chain.
4) Block chain-blindness – lack of randomness based on in the nonce and previous block hash.

The intention of Ethereum is to merge together enhanced scripting possibilities, meta protocol and time stamped database to allow development of an arbitrary application. As stated they call this platform an "Ultimate abstract foundation layer: a block chain with a built-in Turing-complete programming language" (Buterin, 2014). The developers could write smart contracts by defining all the rules themselves.

This might lead to having the opportunity to decentralize any possible service on Ethereum layer as boundaries would be unlimited, unlikely from bitcoin protocol.

To summarize, the key difference from other block chain protocols is built-in programming language, various types of accounts and unlimited variation of application that can be built on top of it.

3.2.2 Smart contracts

Similarly to real-life contracts in business and finance, Ethereum enables to digitalize rules on the block chain that are transparently and globally enforced by participating nodes.

Ethereum allows much broader scale of contracting system known as Distributed Autonomous Organization. This is a scheme that encodes whole organizational structure (e.g. reward system tied to specific actions) into block chain.

As an example we can take open source development, where developers are paid from funds collected by supporters. The funds are publicly and without central authority

[20] Unspent transaction output

[21] Decentralized Autnomous Organization – type of application that is capable to manage contracts and funds without human interaction.

distributed to the developers by the amount of effort measured in lines of code produced and validated as meaningful by rest of the nodes.

Vitalik Buterin divides the applications that can be built on top of Ethereum in three categories (Buterin, 2014):

1) Financial applications – a powerful way for transparently manage their funds using smart contracts. Can include sub-currencies, financial derivatives, funding platforms such as Kickstarter, P2P insurance or gambling models.
2) Semi-financial applications – systems where money are involved in form of tokens. Such sub-currencies can bind current stock value to USD or gold, or any other property. Fundamentally, these applications are on model of subtract or add unit from particular account.
3) Democracy applications – decentralized governance such are voting and reputation systems where no monetary value is present.

From the scope of applications currently built[22] we can see one huge category of applications that use block chain simply to store data. For instance to replicate centrally controlled systems like Dropbox.

The key differences from current central-based solutions is that this platform based applications are out-of-the-box public and has open-code. For contract to stay permanently enforceable, it is necessary that it is not removed from block chain, although its data storage can be modified by explicitly defined nodes.

Once the application is created, anyone can interact with it by sending transactions which include the contract address and functions with parameters to be called. This way it is possible to change states (write to the contract) and invoke functions. The contracts' code can be execute by propagating a transaction with destination address of a specific contract or standard account.

[22] http://dapps.ethercasts.com/

Contracts are allowed to send ETH to other contracts, read/write storage and call other contracts. If destination address is standard account, the ETH is simply transferred to other account managed by private key. Unlikely from accounts, contracts are controlled by invoking new transaction with their contract hash and if such a transaction is invoked, the bytecode gets executed.

Every node in the network stores processes of the transactions and stores complete states of contracts. Nodes executing the contracts code are rewarded by using their resources for executing the code, from transaction fees.

For each computational step is charged fee from transaction balance. This is also implemented to prevent infinite cycle loops that would end up in blocking the node from further transaction processing.

In advance, it is impossible to detect how many computational steps it would take to fully execute contract's bytecode. Therefore, a gas limit is specified to state the maximum amount of gas the sender is willing to consume for code execution. If the gas limit is exceeded and function cycle is not finished, the process of execution is aborted and contract state is reverted to previous state. However, the consumed gas is not "refunded". This function prevents the sender from running out of funds if incomprehensible or buggy bytecode is published.

Therefore, each transaction specifies *gas price* that indicates the number of ETH to be paid per *gas unit* for all computation costs incurred as a result of the execution of this transaction. (Wood, 2013) Such a prevention is included in both contract-initiated messages and those propagated from standard transaction.

The gas price is also used by miners to rank transactions for inclusion in the block chain. Furthermore, the miner who propagated the block votes either to up vote or down vote the current gas limit.

The transaction fields gas is used for maximum computational steps, gasPrice and value used for ETH balance.

If contract data are not filled, the fee is simply calculated as number of bytes multiplied by the gasprice.

3.2.3 Accounts

In Ethereum, the state is defined by the objects named “accounts”, those are 20-bytes addresses and the state transitions are the transfers between such accounts.

Example of Ethereum address: 0xc2b1918bc7a2c398ec6f20b754992d7c10d3e2cb

Account contains four elemental fields:

1) The nonce – counter ensuring each transaction is processed only once
2) Ethereum balance
3) Contract code – optional depending if account is used as contract or as a standard transactions. Contracts specify hash of the bytecode, for standard transaction is used empty string.
4) Storage – space for contract bytecode

Furthermore, Ethereum has different types of accounts, essentially divided into two categories. Both of these accounts possesses of ether balance. (ETHEREUM SWITZERLAND GMBH., 2016)

1) Externally owned accounts (EOE), controlled by private keys
    - Has ability to send messages by creating and signing transactions
2) Contract accounts (CA), controlled by their contract code
    - Every time the contract receives message from EOE it activates its code and is allowed to read and write to internal storage.
    - It is also capable of sending messages to other contracts or to create new contracts.

3.2.4 Transactions

In a scope of Ethereum, transactions refer to signed data package to be sent throughout the network. (Buterin, 2014) There are several important difference from transactions in bitcoin protocol. First, the message can be broadcasted from either EOE or contract. Second, it is optional to include the message data. Finally, recipient (CA) has option to response to the message.

Transactions includes message to be sent and digital signature identifying the sender, amount of ETH and contract data. Particularly, data are divided into *nonce*, *gasprice*, *startgas*, *to*, *value*, *data*, and *ECDSA signature values*.

Unique incrementing nonce specifies the index of transaction sent from specific account. The values start from 0. Gasprice specifies the cost of computational unit. Startgas is maximum gas consumable 3.2.2. To sets the address of receiving address. Value is the amount of ETH sent. Data field of bytes without limit, naturally, the higher the size is higher will be the transaction cost.

There are special logs of contract execution called *Receipts* that are hashed and included into the block chain. Each transaction has its own receipt. Receipt has Intermediate state root which is hash of the state after the contract execution. Cumulative gas used with value of total gas that has been used for the execution. Logs is a special feature designed for light-weight Ethereum clients, which only parse the block headers in which logs are stored and obtain information about recent transactions. Logs contain 32 bytes space for data called "topics" those are intended for easy protocol filtering[23] (Buterin, 2014).

Logs is a special storage not accessible by the contract. Unlikely from variables that are set in contract storage, the logs are append-only for nodes to quickly scan the block chain.

3.2.5 Ethereum Virtual Machine

The EVM can be thought of as a large decentralized computer containing millions of objects, called "accounts", which have the ability to maintain an internal database, execute code and talk to each other. (Ethereum Development - Github, 2015)

EVM runs at every node and serves as a global virtual execution environment for contracts.

Before publish a contract, the code is compiled into stack-based programming language[24]. The contract can be written in one of the higher-level languages like Serpent,

[23] Bloom filters

[24] Language relying on machine (for ethereum its emulated machine) that pushdown stack for registering operations. E.g. addition of two numbers will be programmed like 3 4 +. (Lin, 2003)

which has similar syntax to Python, Solidity (JavaScript based) or LLL (much more low level and ideal for writing storage-efficient code) (ETHEREUM SWITZERLAND GMBH., 2015).

As stated in white paper, the operations executed in EVM have access to three types of spaces in which to store data

1) Stack
2) Memory – unlimitedly expandable byte array
3) Contract's irreversible storage. Unlikely from stack and memory, storage persists after execution of code is finished.

Code can access data included in the incoming message and block header (block number, time, mining difficulty, etc.), these data are known as Environment variables.

In order to write sustainable application it is necessary to use a style of programming that consumes as little of memory as is necessary, since for every computational step is charged a fee.

3.2.6 Gas policy

Average gas price can be found on Etherscan.io[25] Currently the gas price is set to 10 szabo (10^{-6} ETH).

Gas costs table[26]

Operation name	**Cost (in gas)**	**Description**
step	1	Default amount of gas to pay for an execution cycle.
stop	0	Nothing paid for the STOP operation.
suicide	0	Nothing paid for the SUICIDE operation.

[25] https://etherscan.io/charts/gaspri/ce

[26] http://ether.fund/tool/gas-fees

sha3	20	Paid for a SHA3 operation.
sload	20	Paid for a SLOAD operation.
sstore	100	Paid for a normal SSTORE operation (doubled or waived sometimes).
balance	20	Paid for a BALANCE operation.
create	100	Paid for a CREATE operation.
call	20	Paid for a CALL operation.
memory	1	Paid for every additional word when expanding memory.
txdata	5	Paid for every byte of data or code for a transaction.
transaction	500	Paid for every transaction.

Table 4: Gas costs per operation

3.2.7 Scripting languages

Ethereum platform supports various programming languages to extend developer possibilities and also not to support only one language and thus enhance advantage of choosing the right programming language for different development purpose.

Solidity is a high-level programming language with syntax similar to JavaScript.

Ethereum provides many programming languages

- Umbrella is C++ client [27]

[27] https://github.com/ethereum/webthree-umbrella

- EthereumJ is Java implementation[28]
- Geth is the implementation of Ethereum protocol in Go language [29]
- Serpent is Python version of Ethereum[30]
- Solidity is JavaScript implementation

The code written in any of above mentioned language is compiled into EVM bytecode.

3.2.8 Merkle Patricia trees

Unlikely from Bitcoin, Ethereum uses more complex hashing structure called "Merkle Patricia trees".

Merkle Patricia are a combination of Merkle's scheme of hashing and Patricia (Radix) tree structure, providing a tree that that has cryptographically authenticated data structure that can store key-value bind data. (Xie, 2015) It is very efficient for insert, deletes and especially lookups which is very important for integrity checks made by nodes.

3.2.9 Differences from other block chains

Block propagation time is 17 seconds unlikely from Bitcoin's 10 minutes. The block structure is also more complex as following:

Block header

- Transactions
- State – contracts state
- Receipts – hash of state after contract execution

[28] https://github.com/ethereum/ethereumj

[29] https://ethereum.github.io/go-ethereum/

[30] https://github.com/ethereum/serpent

3.2.10 Frontier release

Although, the development begun in December 2013 and was granted by open-source community more than 31,000 bitcoins (by that time roughly 18 million USD), up until 2015 were available only development versions.

The first release of software was named *Frontier* and its release version was published on 29th of January, 2015. Following by Genesis block on 30th of January. Since then, Ethereum network is live. (Ethereum GmbH, 2015)

3.3 Other similar platforms

3.3.1 Blockstream

Blockstream[31] is a platform that enhances creation of crypto currencies, smart contracts and open assets with inbuilt interaction to other systems without central authority like Bitcoin Block chain. Simply said, it is an extension to other block chains that are designed to solve specific issue, in bitcoins ecosystem it is currency.

"Blockstream's core area of innovation is sidechains, a technology focused on improving on the block chain, the most powerful public utility for distributed trust systems." (Blockstream, Inc, 2016)

3.3.2 Eris project

Eris project is in the community considered a "fork of Ethereum pattern". The platform enhances the block chain for building smart contracts. Eris is open-source project, but provides subscription level access to the world's first smart contract libraries focused on specific industries. By providing high-quality, tested smart contract primitives and templates, we greatly reduce the complexity of creating smart contract applications. (Eris industries, Inc, 2016)

3.3.3 Maidsafe

Maidsafe is a long-term project started in 2006 that takes focus on securing whole internet, it is more than a block chain platform. The currency of Maidsafe is called SAFEcoin

[31] https://blockstream.com/

(abbr. Secure Access for Everyone). Maidsafe network uses the available space of participating nodes that are paid for providing their storage space.

"Over time, the SAFE vault on your computer will start to fill up with network data, and as a consequence your virtual wallet will automatically start receiving safecoin. You can use the safecoin you've earned to pay for other services on the network, or convert them to another currency via a SAFE currency exchange. (..) MaidSafe's aim is to provide privacy, security and freedom to everyone on the planet. This has been our unwavering ambition since we started on this journey in 2006, and it remains our driving force today. (MaidSafe.net Limited, 2016)

3.4 Potential

From my studies of Ethereum technology, I humbly predict that platform can boost development and transformation of financial sector as we know it today – centrally controlled banking.

Ethereum will mostly be used as a storage for data that are necessary to be kept available for a lifetime. The platform also allows most secured world-wide available storage.

The success can be proved by the skyrocketing of ETH price within last months. Based on Kraken.com ETH/EUR chart[32] since 1st January 2016 the price increased from 1 EUR per ETH to 10 EUR for ETH (at the time of writing 5th of March 2016).

[32] https://dwq4do82y8xi7.cloudfront.net/widgetembed/?symbol=ETHEUR&interval=D&symboledit=1&toolbarbg=f1f3f6&hideideas=1&studies=&theme=White&style=1&timezone=exchange

4. Practical usage of Ethereum Block chain

The following part will be dedicated to practical usage of Ethereum block chain demonstrated by dapp for voting. Similar platform for voting was suggested by Dominik Schiener and also used as an example in Solidity programming language reference (Solidity, 2016). Fur purpose of this thesis more than a simple voting mechanism will be provided. The author suggests more enhanced platform for managing eligible voters through privilege definitions, extended vote options, moreover results will be permanently stored within the Ethereum Block chain. Unlikely from Schiener's first implementation.

"There is currently one smart contract coded in Solidity that is used for placing a poll into the Block chain and for casting the votes. For display purposes on the website and making it easy to vote, I'm also storing poll information, votes and the related Ethereum accounts in MongoDB collections." (Schiener, 2015)

4.1 PureVote - Decentralized Voting Application

The purpose of this application is to introduce, test and evaluate available functions of Ethereum platform and also Solidity language that will be used for writing the smart contract itself.

Dapp shall serve purpose of decentralized voting system where any or specific addresses controlled by the network peers can participate in voting on a *Poll* controlled by smart contract. The participation is only limited to sending publicly defined commands defined within the smart contract and paying the transaction cost. The commands sent in the transaction invoke particular parts of the contract's source code stored on the block chain and modifies the current state (storage) of the contract. As the transaction is being processed by the miners, modified state is permanently stored on the block chain and is practically impossible to be reversed, because total computing power put into the verification would have to be computed again by every single node in the network.

The application is designed to serve only one poll voting per smart contract. Nonetheless, the functions of Ethereum enables smart contracts to create and manage other smart contracts, thus it is possible to design master voting smart contract for creating new contracts for each poll.

The obligation of payment for broadcasting transaction that contains choice being voted by the node lays on the participants themselves. Each of the participants pays the transaction cost for their vote. Nonetheless, it is technically possible to let the contract pay for casted votes as it allows to possess of ETH. But this becomes impractical because the cost for executing the cost is zero for any peer in the network. This could be easily exploited by other peers if the list of eligible voters wouldn't be properly defined.

The application consists of a single file named purevote.sol. Contains contract named PureVote with below defined classes and methods.

The source code of application is available online at https://github.com/mareksip/PureVote

4.1.1 Prerequisites

The application is written in Solidity language for its similar syntax to JavaScript that author is already familiar with. For writing the contract online Solidity editor called Solidity Browser[33] was used. Solidity Browser has interface already prepared for publishing the contract right after its competition. Nonetheless, PureVote will be published using Ethereum Mist Wallet. Solidity Browser also checks for internal integrity of source code, which is really useful as currently there are not many easily available testing environments for smart contracts.

Furthermore it was necessary to have ETH cryptocurrency as a form of payment to miners for deploying the smart contract on the network and for payment of fees for its execution. Kraken[34] was chosen as it is only exchange allowing users to deposit EUR with SEPA bank transfer. Other exchanging offering ETH trades do the transaction right between the cryptocurrency Bitcoin and ETH.

On below address 1 ETH withdrawal request was made on Kraken.com website.

[33] http://chriseth.github.io/browser-solidity/

[34] www.kraken.com

0x1ff080d4c538d36160a8c3a9338a949dcd558f1d

0.995 ETH have been received from Kraken.com with transaction[35]:

0x2867c6d97635549cd780ac54cd1140c8156d720a36a5913a806a945e7d80ed95

Furthermore, for convenience some software for managing funds and contract related operation was needed. The only available at the time of writing was Mist pre-release software version 0.3.9[36]. Mist manages private keys and also offers extended interface for publishing smart contract and invoking its functions.

4.1.2 Classes

1) Poll – Poll object having attributes of header text, deadline, total votes casted, deadline status and its creator.
2) Voter – eligible addresses allowed to vote, if not specified any address can participate in voting. Voters can have different voting weight, but for contract simplicity static value 1 is used.
3) Option – values that can be voted by participants

4.1.3 Events

Perception of events is slightly different from standard programming languages. Events in Ethereum are built for efficient change storage that can facilitate parsing block chain data without downloading whole block chain e.g. for SPV nodes.

1) NewVote event lets SPV nodes react efficiently on changes

[35] https://etherchain.org/tx/0x2867c6d97635549cd780ac54cd1140c8156d720a36a5913a806a945e7d80ed95

[36] https://github.com/ethereum/mist/releases/tag/0.3.9

4.1.4 Methods

1) NewPoll – Creates new Poll object with certain name, UNIX timestamp deadline, poll options and optional eligible addresses
2) Vote – Checks if eligible voters are defined, if so the address is validated for voting. Otherwise anyone can cast a vote. Furthermore, the function checks if Poll deadline is not passed. Finally, casts a vote from senders address.
3) winningProposal – Poll votes are computed and option with most casted votes is declared as a constant
4) terminate – Sets Poll status to closed. This can only be performed by the Poll creator.
5) remove – Contract can be deleted from block chain using *suicide* function. This can be performed only by the creator of the Poll.

4.1.5 Publishing contract

By the time of publishing current gas cost *is 0.04545455 ETH* per million gas and all gas cost calculations will be made with this value.

The contract was created by putting solidity source code to *New Contract* function in Mist Browser.

The publishing cost exactly 0.0290845 ETH and has been published within a few seconds to connected peers. Contract address[37] is:

0x1a77eb194c6b85ff55eda03088492ced99ffda86

Total used gas for propagating contract was: 639,859.

4.1.6 Executing contract functions

Now the contract is published, any of its functions can be invoked by propagating new transaction with above mentioned destination address.

[37] http://etherscan.io/address/0x1a77eb194c6b85ff55eda03088492ced99ffda86

Out of the box, after publishing a contract, Mist offers interface for executing the contract functions. This feature can be noticed on the right panel *Write to Contract*. By selecting desired function, the function parameters are loaded accordingly to the list below. Furthermore, the user has the option to send additional ETH to the contract. This is not required because transaction cost is paid separately. The contract balance is displayed on top of the panel, indicating currently disposed ETH balance.

Lastly, for successful code execution it is necessary to choose the Mist wallet account from which the transaction fee will be deducted from.

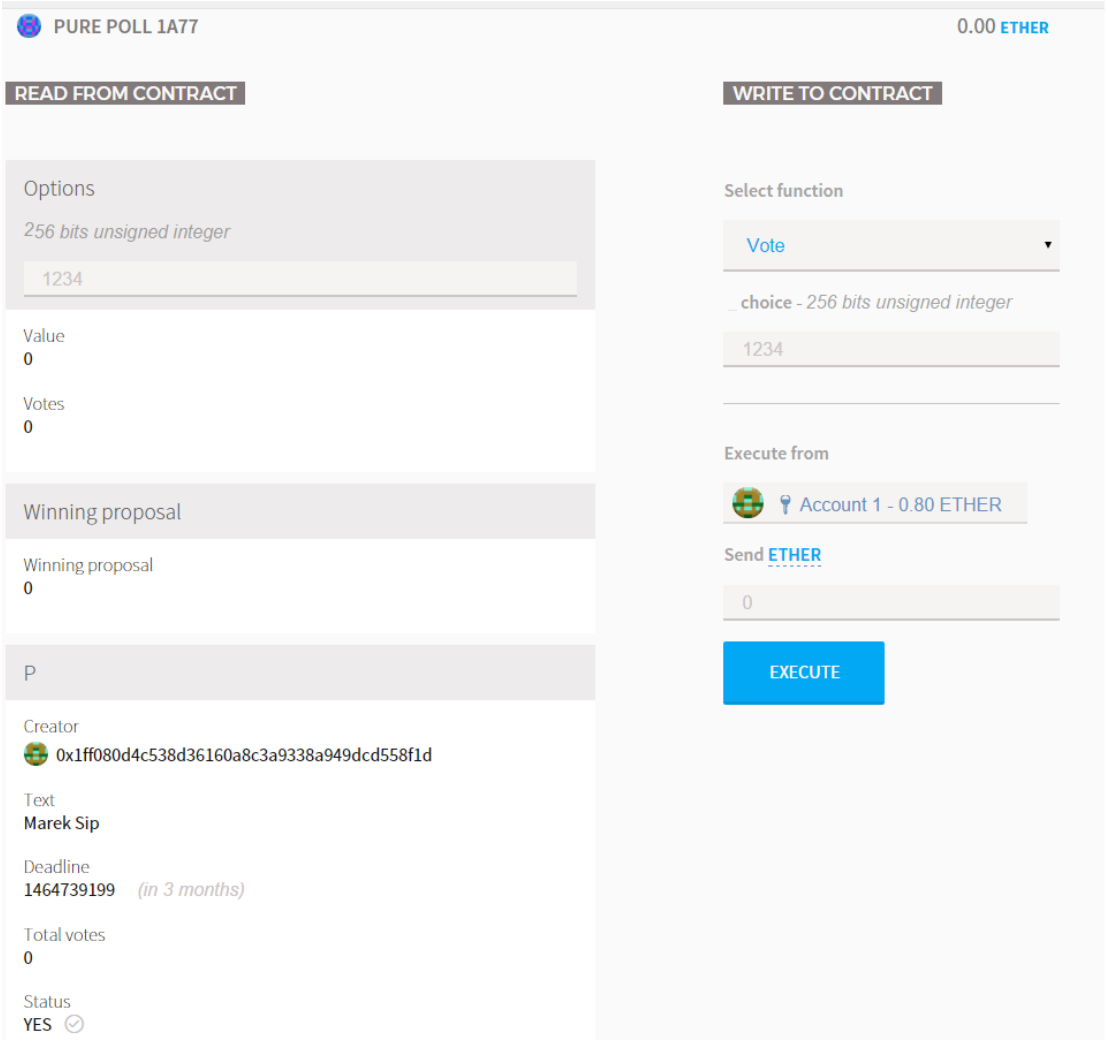


Figure 5: Mist interface after publishing contract

Firstly, it is necessary to start new poll. The Poll will be called "*Marek Sip*" with deadline timestamp: *1464739199* set to 31.5.2016 23.59.59, with array of options to vote "Failed" or "Passed". In order to have the code variables efficient, values are represented as numbers *[0, 1]*.

Contracts *NewPoll* function was successfully raised with transaction:

0x7ffa4a289f6da73f55421494f5694b3f59b435d5593fed796182432c63310c0d

The cost paid for code execution was: 0.00777741 ETH for 171,103 used gas.

Lastly, it is necessary to cast a vote for one of these options. Which is achieved by selecting *Vote* function and filling choice value "0" that represents value "Failed".

This was done on transaction:

0x0b583fc41cdb798c130bbdc19f5137ed99b514e41144db7cf5af5e875009f4b6

The code execution cost 0.00285518 with 62,814 used gas. The data sent with the transaction are following. First eight bytes are identification of the transaction and last byte is the function parameter, in this case represented by 0 for voting value "Failed".

0x**0121b93f**0000000000000000000000000000000000000000000000000000000000000**0**

Lastly, it is possible to close the voting with *Terminate* function that was raised on transaction:

0xcf5b63e3296e89744210d9d0f3fd36feffaa2fdea2986a9293cc89a4a3578b62

The code execution cost 0.00060605 and took 13,333 used gas.

4.1.7 Gas usage summary and cost

Used rate for EUR calculations

By the time of publishing (6th March 2016) according to Kraken.com 1 ETH is 9.55 EUR. 1 EUR is approx. 27 CZK according to CNB (Česká Národní Banka, 2016). The table below shows how costly to execute were particular functions of PureVote. Certainly, most expensive is to publish the contract itself. First transaction for publishing contains whole smart contract source code. Miners have to be paid for verifying and publishing each byte of this code. For executing the contract, transaction sent from initiator contains only identification of method to be invoked, and its required attributes.

Action	Gas used	Cost ETH	Cost EUR	Cost CZK
Creating contract	639,859	0.0290845	0.27	7.29
Creating Poll	171,103	0.00777741	0.07	1.89
Casting Vote	62,814	0.00285518	0.03	0.81
Terminate Poll	13.333	0.00060605	0.005	0.13
Total:	**887.109**	**0.04032314**	**0.37**	**10.12**

Table 5: Contract testing fee overview

4.1.8 Confirmations

Transaction requesting the network for contract publishing or invoking functions are broadcasted to the connected nodes immediately. Nonetheless, the transactions credibility is increasing with the amount of blocks that have been added on top of it.

As explained in the chapter Bitcoin Block (3.1.2) each block references the previous as its validation. In Bitcoin it is considered impossible for transaction to be reverted (and double spent) after more than 6 blocks are added on top of the block that included given transaction. (Bitcoin Wikipedia community, 2016).

Ethereum Mist client default settings is waiting for 13 confirmations before considering the transaction valid. This makes it at the rate 17 seconds per block almost 4 minutes. Each inclusion to block is considered as confirmation, because miners publishing new blocks are referencing to previous block which they verified.

4.1.9 Evaluating results

The application has been successfully deployed into Ethereum network, its function tested and results permanently stored on Ethereum Block chain.

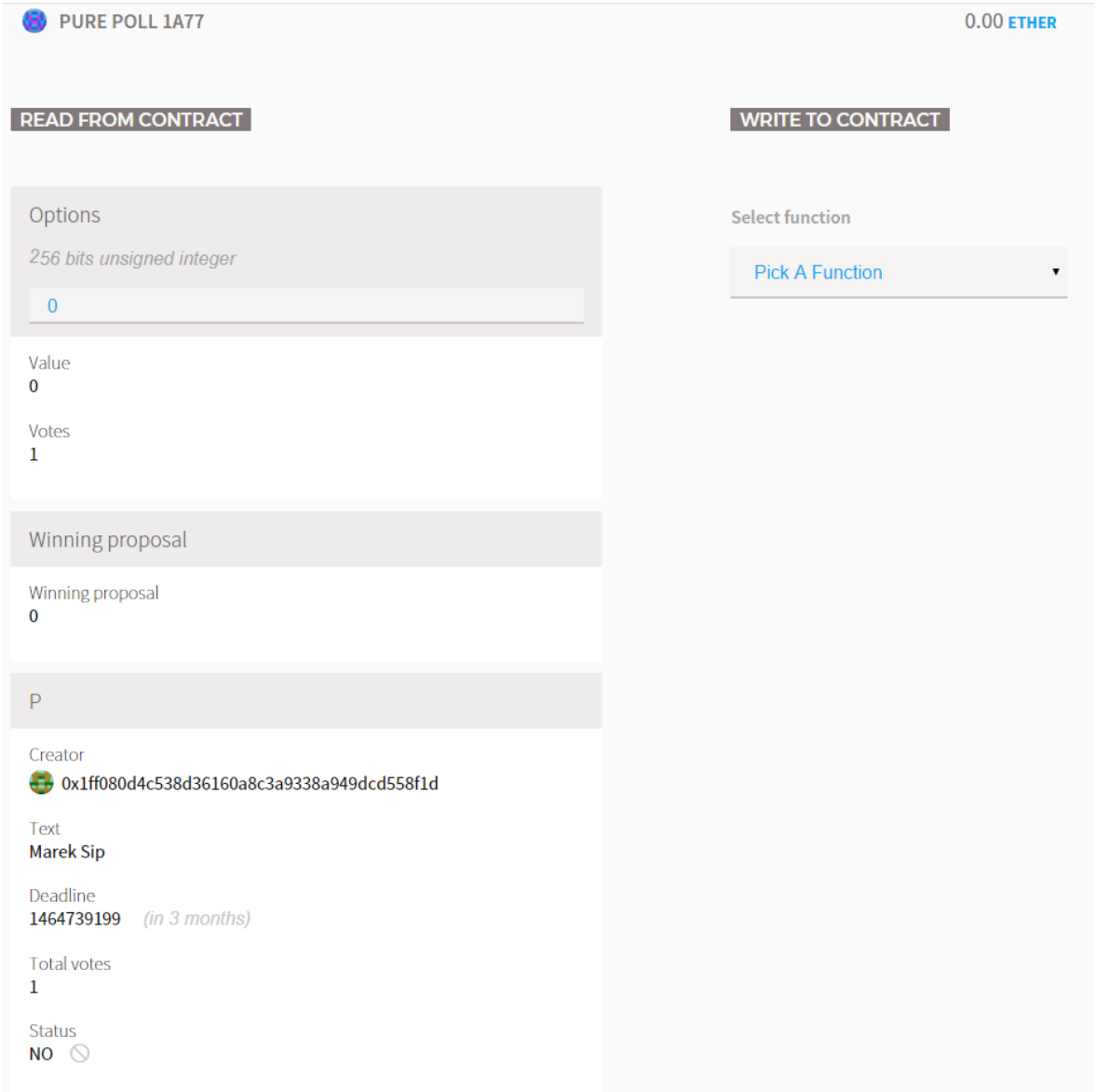
Currently, author sees huge obstacle for using Ethereum platform, which lays in difficulty of obtaining ETH units. The units can be obtained either by mining or purchased for fiat or cryptocurrency like Bitcoin. Purchasing for ordinary currency was chosen for this practice for convenience. Nonetheless, it still took more than 2 days, until CZK were transferred from author's bank to Kraken.com account and exchanged for ETH units.

Once ETH units are available, the publishing and execution of smart contract, thanks to tools like Mist is really smooth.

Below figure displays User Interface of Mist browsing current state of the smart contract after execution of Terminate function.

Left panel *Read from the Contract* indicates that the *Option 0* has value 0 and one vote has been casted for that it. Below, the value of Winning proposal is displayed. The *P* represents the object of *Poll* that stores attributes about the contract creator, Poll name, its expiration and the current status which indicates if voting is open.

Figure 6: Contract state after termination of Poll



5. Used environment

Acer Aspire E 15

Intel CORE i5-5200U

NVIDIA GEFORCE 840M with 2GB Dedicated VRAM

1 GB DDR3 L Memory

1000 GB HDD

Raspberry Pi 2 Model B

ARM Cortex-A7 quad-code processor

VideoCore IV

1GB LPDDR2 SDRAM

6. Discussion and Results

Ethereum allows anyone to distribute decentralized applications and opens floodgates of decentralized application development. This is huge improvement after 6 years from introduction of block chain technology. Ethereum block chain is multi-purpose foundation layer for any possible system.

The platform has been successfully tested through voting smart contract application PureVote. The application was deployed on Ethereum block chain and its function invoked by several transaction.

The concept of PureVote allows to prove that certain activity occurred on a specific time. This activity cannot be reversed without re-computing the PoW that miners used to verify and propagate the transactions, and is stored technically forever[38].

Proposed application proves that Ethereum is currently stable to handle similar and also much more complex applications than PureVote.

At the current rate ETH/EUR the propagation and execution of smart contract in the similar complexity costs 0.3 EUR (0.04 ETH) which is cheaper than renting centrally managed cloud solutions.

The key question here is what would be the price of gas when ETH/EUR rate reaches 100 EUR. As platform author Vitalik Buterin[39] assures it depends solely on miners which transaction include or not based on the gas price.

Similar application based on block chain are currently being designed for Estonian government as a solution for e-residency[40]. Similarly, Ukraine government is looking to implement block chain for voting[41].

---

[38] Whole blockchain PoW would have to be remade. Currently 1st Bitcoin block mined in 1st January 2009 have not been rewritten.

[39] https://www.reddit.com/r/ethereum/comments/499a7w/gas_prices_are_already_kinda_down_to_20_shannon/

[40] http://www.coindesk.com/nasdaq-shareholder-voting-estonia-blockchain/

[41] https://bitcoinmagazine.com/articles/ukraine-government-plans-to-trial-ethereum-blockchain-based-election-platform-1455641691

The key benefit from block chain based application is decentralized nature in peer-to-peer network, making it impossible to be restricted by third party. Once deployed, it can only be modified by the author. Whereas, centrally controlled solutions are more vulnerable for attackers, making the provider single point of failure where attacker or even the service provider have ability to intrude to the users content.

In Ethereum network, user contracts and data, even though they are published to the network, are cryptographically secured and can only be controlled by the user.

From what can be currently seen, the community projects range is from financial application to democracy platforms like voting. A whole bank can be implemented within 40 lines of code[42], domain name registration by 10 lines[43].

Lack of middle man enhances credibility of key financial and democracy systems with no-risk of being intruded by the middle man.

---

[42] https://blog.ethereum.org/2015/12/07/ethereum-in-practice-part-3-how-to-build-your-own-transparent-bank-on-the-blockchain/

[43] https://github.com/ethereum/go-ethereum/wiki/Mutan-0.6-Examples

7. Conclusion

The goal of analysing block chain timestamped databases with accent to 6 year of development was provided within the chapter of Account for technology (3), Moreover, an overview of practical usage of Ethereum block chain was made in chapter Practical usage of Ethereum Block chain (4) with key benefits for its users and society in general.

Ethereum takes decentralized computing onto broader scale than single-purpose currency platform Bitcoin. Bitcoin was first application to introduce usage of timestamp database called block chain. Using block chain the nodes could easily synchronize and validate data. The key achievement was that untrusted nodes could achieve consensus without any centrally trusted entity.

Until the invention of Ethereum, many forks of Bitcoin protocol were made to decentralize systems like domain name registration. Ethereum Inc, designed generally applicable block chain platform with Turing-complete programming language that lets any type of application to be deployed on top of it.

The usage of block chain technology is currently being implemented throughout European state parliaments. Ukraine parliament decided to use block chain based voting system as a transparent mechanism for passing government proposals[44]. Similar solution is being tested in Estonian Stock Market[45]. According to Great Britain's Government for Office Science the technology could prove to have the capacity to deliver a new kind of trust to a wide range of services (UK: Government for Office Science, 2015).

Thanks to Ethereum it is now possible to build any application trustworthy and transparently.

---

[44] https://bitcoinmagazine.com/articles/ukraine-government-plans-to-trial-ethereum-blockchain-based-election-platform-1455641691

[45] http://www.coindesk.com/nasdaq-shareholder-voting-estonia-blockchain/

8. Bibliography

ANTONOPOULOS, A. M. (2014). *Mastering Bitcoin* (First edition ed.). O'Reilly Media.

Bitcoin Wikipedia community. (2016). *Bitcoin Wiki*. Retrieved from Bitcoin Wiki: https://en.bitcoin.it

Blockchain Ltd. (2016). *Difficulty*. Retrieved from Blockchain.info: https://blockchain.info/charts/difficulty

Blockstream, Inc. (2016). Retrieved from Blockstream: https://www.blockstream.com/

Buterin, V. (2014, 8). *Ethereum Github wiki*. Retrieved from Ethereum White Paper: A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM: https://github.com/ethereum/wiki/wiki/White-Paper

Česká Národní Banka. (2016). *Kurzy devizového trhu.* Retrieved from Česká Národní Banka: http://www.cnb.cz/cs/financni_trhy/devizovy_trh/kurzy_devizoveho_trhu/denni_kurz.jsp

Eris industries, Inc. (2016). *Products*. Retrieved from Eris Indistries: https://erisindustries.com/products/

Ethereum Development - Github. (2015). Retrieved from Ethereum Development Tutorial: https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial

Ethereum GmbH. (2015). *Ethereum Frontier Guide.* Retrieved from Gitbooks.io: https://www.gitbook.com/book/ethereum/frontier-guide/details

ETHEREUM SWITZERLAND GMBH. (2015, 8 28). *Developer documentation.* Retrieved from ETHEREUM.org: https://github.com/ethereum/wiki/wiki

ETHEREUM SWITZERLAND GMBH. (2016). *Go-ethereum developer documentation*. Retrieved from Github: Ethereum/go-ethereum: https://github.com/ethereum/go-ethereum/wiki

Harding, D. A. (2015, 8 28). *Developer documentation.* Retrieved from BITCOIN: https://bitcoin.org/en/developer-documentation

Lin, C. (2003, 6 22). *Charles Lin: Computer organization*. Retrieved from Understanding : https://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Mips/stack.html

MaidSafe.net Limited. (2016). *The New Decentralized Internet*. Retrieved from MaidSafe: http://maidsafe.net

Marek Palatinus, P. R. (2014, 4 24). *BIP 44: Multi-Account Hierarchy for Deterministic wallets*. Retrieved from Github.com Bitcoin Improvement proposals: https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki

Merkle, R. C. (1979). *A digital signature based on a conventional encryption function.* Retrieved from Barkeley.edu: http://www.eecs.berkeley.edu/~raluca/cs261-f15/readings/merkle.pdf

Nakamoto, S. (2008, October 1). *Bitcoin: A Peer-to-Peer Electronic Cash System.* Retrieved from http://p2pfoundation.ning.com/: https://bitcoin.org/bitcoin.pdf

Schiener, D. (2015, 10 28). *PublicVotes: Ethereum-based Voting Application*. Retrieved from Medium.com: https://medium.com/@DomSchiener/publicvotes-ethereum-based-voting-application-3b691488b926#.sqrsm4uvu

Solidity. (2016). Retrieved from Solidity language documentation: http://solidity.readthedocs.org/en/latest/index.html

UK: Government for Office Science. (2015). Distributed Ledger Technology:. Retrieved from https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf

Wood, D. G. (2013, 11). *Gavin Wood*. Retrieved from Ethereum: A SECURE DECENTRALIZED GENERALISED TRANSACIION LEDGER, FINAL DRAFT - UNDER REVIEW.

Xie, J. (2015). *Patricia Tree*. Retrieved from Ethereum: https://github.com/ethereum/wiki/wiki/Patricia-Tree

9. Appendix

9.1 Acronyms and used abbreviations

EVM – Ethereum Virtual Machine, environment for executing source code

SHA256 – Secure Hash Algorithm

Merkle root – result of hashing through Merkle structure

BTC/Bitcoin – One bitcoin, unit of Bitcoin network

ETH – One Ethereum, crypto currency of Ethereum platform

Wei – $10(^{-18})$ of ETH

Szabo – $10(10^{-6})$ of ETH

Ethereum – Block chain based platform for smart contracts

Dapp – Decentralized application, smart contract

API – Application Programming Interface

UTXO – Unspent Transaction Output, request send transaction "to"

ECDSA – Elliptic Curve Digital Signature Algorithm

Gas – fuel of Ethereum platform needed as a payment for EVM executions

PoW – Proof of Work, computational power released to publish block

SPV – Simple Payment Verification – abbreviation used for nodes that are storing only merkle roots and not downloading whole block chain.

10. Supplements

```
contract PurePoll{

    //define Poll attributes and state variables
    struct Poll{
        //address of creator
        address creator;
        //name of the poll
        string text;
        //not required contract expiration UNIX timestamp
        //expires when eth runs out
        uint deadline;
        //number of casted votes
        uint totalVotes;
        //checking if Poll ended
        bool status;
    }
    //Elegible addresses that can vote
    struct Voter{
        address addr;
        bool voted;
        uint weight;
    }
    //options to be voted
    struct Option{
        //name of option
        uint value;
        //number of casted votes
        uint votes;
    }
    Option[] public options;
    Voter[] public voters;

    Poll public p;

     // event tracking of all votes
  event NewVote(uint votechoice);

    //initiator function that stores the necessary poll information
  function NewPoll(string _text, uint[] _options, address[] _voters, uint _deadline) {
    p.creator = msg.sender;
    p.text = _text;
    p.deadline = _deadline;
    p.status = true;
    p.totalVotes = 0;
```

```
45
46      // Add each option to contract options
47      for (uint i = 0; i < _options.length; i++){
48          // Option({}) creates a temporary Option object
49          // options.push(...) appends _option to contract options
50              options.push(Option({
51              value: _options[i],
52              votes: 0
53              }));
54      }
55      for (uint x = 0; x < _voters.length; x++){
56          // Option({}) creates a temporary Option object
57          // options.push(...) appends _option to contract options
58              voters.push(Voter({
59              addr: _voters[x],
60              voted: false,
61              weight: 1
62              }));
63              //TODO: assign correct votes
64      }
65  }
66      //function for user vote. input is a string choice
67  function vote(uint _choice) returns (bool) {
68      //now = alias for block.timestamp
69      if(now > p.deadline){
70          p.status = false;
71          return false;
72      }
73
74      //if (msg.sender != p.creator || p.status != true) {
75      //  return false;
76      //}
77
78      uint voteWeight = 1; //default weight is 1
79
80      if(voters.length > 0){
81          //Poll requires authentication
82          bool verified = false;
83          for(uint i = 0; i < voters.length; i++){
84              //loopin through elegible voters
85              if(msg.sender == voters[i].addr){
86                  //address corresponds verify if havent voted
87                  if(voters[i].voted == false){
88                      verified = true;
89                      //assign voting power
90                      voteWeight = voters[i].weight;
```

```
                }

            }
        }
        if(!verified){
            //only specific addresses are allowed to vote
            //senders address was not found  or
            //address already casted a vote
            return false;
        }

    }

    for(uint x = 0; x < options.length; x++){
            //loopin through options
            if(_choice == options[x].value){
                //vote casted
                options[x].votes += voteWeight;
                p.totalVotes += 1;
            }
            else{
                //choice was not found
                return false;
            }
    }
    NewVote(_choice);

    return true;
  }
    /// @dev Computes the winning proposal taking all
    /// previous votes into account.
    function winningProposal() constant
            returns (uint winningProposal)
    {
        uint winningVoteCount = 0;
        for (uint o = 0; o < options.length; o++)
        {
            if (options[o].votes > winningVoteCount)
            {
                winningVoteCount = options[o].votes;
                winningProposal = o;
            }
        }
    }

  //only creator can end the poll
  function terminate() returns (bool) {
    if (msg.sender == p.creator) {
        p.status = false;
        return true;
    }
    return false;
  }

  //only creator can delete the contract
  function remove() returns (bool) {
    if (msg.sender == p.creator) {
      suicide(p.creator);
      return true;
    }
    return false;
  }
}
```