Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

DEPARTMENT OF HUMANITIES, SOCIAL AND POLITICAL SCIENCES

BIOTS Hackathon Winter 2018

Democracy 2.0

A Decentralized Autonomous Organisation

Martin Blapp	blappm@student.ethz.ch
Claudio Ferrari	ferraric@student.ethz.ch
Jonas Hagemann	hagjonas@student.ethz.ch
Philippe Hasler	phasler@student.ethz.ch
Manuel Schmuck	schmucma@student.ethz.ch
Kaan Sentürk	skaan@student.ethz.ch
Adrian Soos	asoos@student.ethz.ch

Zürich, 18 April 2018

All contributors contributed equally to this report.

STATEMENTS

This project report was written as part of the spring 2018 course "Blockchain And the Internet of Things (851-0591-01L)" run by M. Dapp, S. Klauser, and D. Helbing.

The software code, which is part of this report, is open source and available at `github.com/skaan/HugsForBugs`.

ABSTRACT

Voting fraud and manipulation has always existed in democratic systems. One explanation for this is that voting is a highly centralized process. Hence, blockchain technology has been proposed as a possibility to overcome said problem due to its ability to decentralize processes. As part of this project we implemented a voting system and a proposal system on the Ethereum blockchain using smart contracts. On the one hand, the voting system allows a managing entity to create a ballot by issuing tokens that are used by the eligible voters to cast their vote. On the other hand, the proposal system allows a voter to create a proposal for a ballot. Once this proposal has reached sufficient support from other voters, the managing entity creates the ballot.

To counteract a lack of awareness and knowledge that is often present among voters, the concept of a liquid democracy was also implemented. In a liquid democracy, voters are able to pass on their voting right to another eligible voter of their choice. Additionally, a small library was built with which it is possible to configure a custom token, and deploy it to a selected Ethereum network.

CONTENTS

CHAPTER

1 INTRODUCTION

Since the birth of democracy electoral fraud has posed a serious threat for the democratic world. In recent years, there have been many cases of suspected manipulation during elections. For example, during the recent elections in Venezuela election results might have been manipulated by at least 1 million votes according to the Washington Post. Unfortunately, Venezuela is not the only country where electoral fraud is a concern and the manipulation of votes is widespread. [1]

There are several different methods of counting votes. Common methods are so-called optical scans, using a direct-recording electronic (DRE) voting machine or punch cards. An optical scan works similar to a standardised test. A voter casts his ballot and a machine scans the ballot and counts the vote. DRE machines are designed to directly register a votes that are entered by the voters. The punch card method is mostly obsolete and consists of punching a hole into a card – the votes are either counted by hand or scanned and recorded electronically. [2]

While said methods have been in use for many years, they all have a common flaw. Voting is highly centralised and it is difficult to ensure that votes are counted properly. Once a ballot has been cast citizens have no more control over the election process. It is impossible to guarantee that votes that have been cast are not manipulated. Consequently, centralisation is at the heart of the problem of the current voting system.

Blockchain technology lends itself to the decentralisation of centralised processes. It is disrupting many industries and has the potential to disrupt the current central voting system that has been in place for decades. This work aims to show how blockchain technology can be used to decentralise and innovate the current system making it more efficient and secure in the process.

1.1. Ethereum Smart Contracts

Ethereum is platform to run decentralized applications. Smart contracts are the key element of Ethereum. In them any algorithm can be encoded. Smart contracts can carry arbitrary state and can perform any arbitrary computations. They are even able to call other smart contracts. This gives the scripting facilities of Ethereum tremendous flexibility.

An important aspect of how smart contracts work in Ethereum is that they have their own address in the blockchain. In other words, contract code is not carried inside each transaction that makes use of it. This would quickly become unwieldy. Instead, a node can create a special transaction that assigns an address to a contract. This transaction can also run code at the moment of creation. After this initial transaction, the contract becomes forever a part of the blockchain and its address never changes. Whenever a node wants to call any of the methods defined by the contract, it can send a message to the address for the contract, specifying data as input and the method that must be called. The contract will run as part of the creation of newer blocks up to the gas limit or completion. Contract methods can return a value or store data. This data is part of the state of the blockchain.

CHAPTER

2 CHALLENGE AND SOLUTION

2.1. Challenge

Before diving into the solution, a closer examination of the challenges is necessary. In order for the current legacy voting system to work, there needs to be a trusted central authority. Every time a case of electoral fraud is made public the trust in the system shrinks undermining the legitimacy of the democratic process. As Smartmatic CEO Antonio Mugica says, “Confidence – from the first ballot cast to the final result – is the bedrock of democracy.” [3] It is frightening to think that one in five Americans does not believe that results of the past election were correctly tabulated. This has caused the participation in elections to drop drastically and has created a threat to democracy.

Another challenge for the present system is its rigidity. Currently, there are two main systems of democracy, namely the direct system and the representative system. On the one hand, all citizens in the direct system have an equal say in all matters – as it is the case in Switzerland. The system of direct democracy assumes that the citizens choosing to vote are informed about the issue at hand. In a representative system, on the other hand, citizens choose a representative to stand for them in parliament. This system is used in Germany and assumes that the representatives have the citizens’ best interest at heart. Unfortunately, both assumptions only hold rarely. Citizens are often uninformed and politicians are often faced with a conflict of interest between the interest of the citizens and the interest of lobbyists. In theory, a mélange of the systems exists and has come to be known as a so-called liquid democracy.

A liquid democracy allows voters to pass on their voting right to a person of their choice if they feel that they are not capable of making a decision, e.g. due to being not sufficiently informed. This would also allow every person to petition changes to existing laws under the condition that enough people agree with the person's views and pass on their right to vote to said person. Due to the complexity of a liquid democracy the system has not been implemented in practice yet. [4]

According to [5], an ideal voting system must fulfil the following conditions: secrecy, verifiability, integrity and resistance. This means that a voter must be able to cast his vote in secrecy, while at the same time being able to track and check the vote count at all times. Furthermore, the system should correctly track votes and allow the user to change his own vote even once it has been cast.

2.2. Idea

All of the challenges discussed could be solved through the creation of a blockchain-based voting system. To this end the "Democracy 2.0 Token" has been created. The token would allow the tokenisation of voting rights and lead to a decentralisation of the voting system. During elections all registered voters would receive a token, which represents the right to vote. The process of voting itself can be considered a type of transaction in the proposed system, where one token can be traded in for a vote. In order to complete a transaction a private key is necessary which is only known to the voter himself. Transactions or votes are, however, not stored centrally, but as blocks on the blockchain.

The tokenisation of votes would ensure absolute secrecy as the vote of each citizen is recorded as a transaction with a corresponding number while the voter remains anonymous. At the same time, since the blockchain is stored on all devices corresponding to the network, vote counts can be checked continuously. Additionally, it would be difficult if not impossible manipulate votes, as these changes would immediately become visible for all on the blockchain. As a further step, we would like the user to be able to change his vote, in case his opinion on the subject changes. We also implemented a functionality of transferring votes which would pave the way to a liquid democracy.

The properties of the blockchain allow the design of a voting system that meets all four described conditions of an ideal voting system.

The following diagram (Figure 2.1) depicts the solution design of the voting process that was implemented by our group.

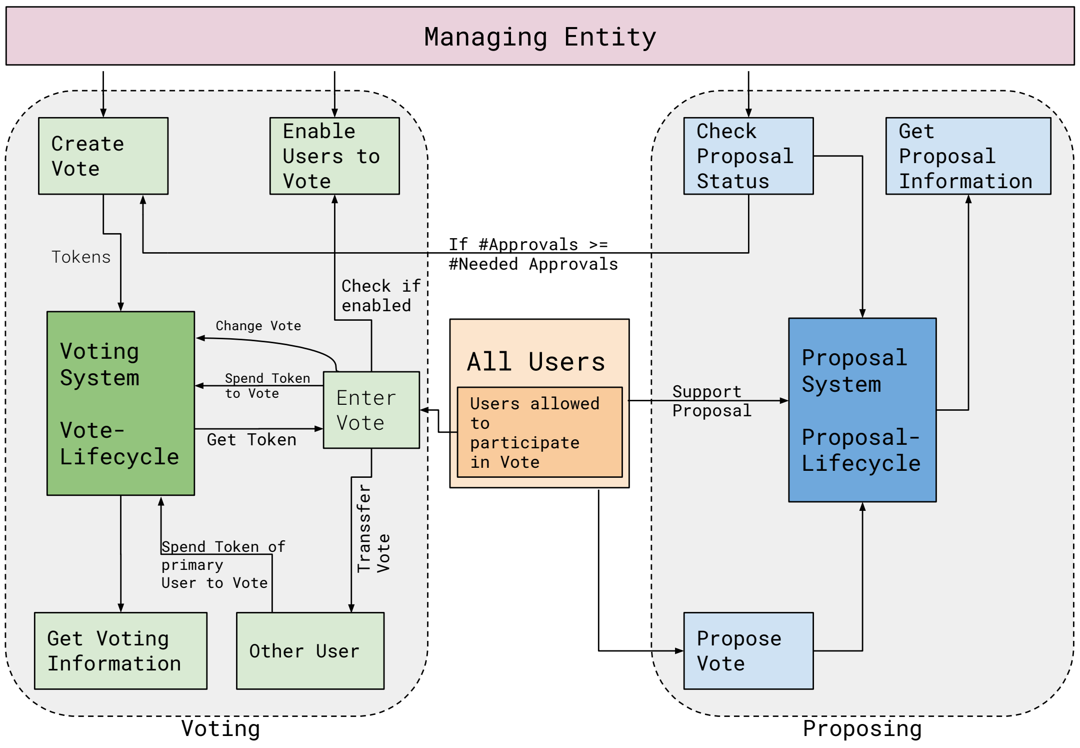

Figure 2.1.: An overview of the voting process and its functionalities. A managing entity creates the voting system. All users can create and support proposals. If a proposal is backed by a sufficient amount of people, it is put to a vote. Users can also enter a vote if they are eligible. Upon becoming eligible, the users receive a token which represents their voting right. The token can then be used to cast a vote or it can be transferred to another user (this is also known as liquid democracy). A more detailed description can be found in 3.1 and 3.2.

CHAPTER

3 SOLUTION DESIGN

3.1. Technical Implementation of the Voting System

The voting system represents a pool of voters with a *Managing Entity*. That could for example be all eligible voters in Switzerland, but also in a different use case the administration board of a large company. The *Managing Entity* organises this pool of voters. The voters' count is the total number of eligible voters in the system. In the `Voting` array every vote that has been entered into the *Voting System* is saved.

Specifically, for every vote the number of *Users* that are allowed to vote, the number of votes in favour, the number of votes against, the amount of *Ether* that a *User* receives for voting, the `endTime` of the vote (an array containing the address of an eligible voter) and every *Users*' votes are saved.

Only the *Managing Entity* is capable of starting a vote which it does using the `createVote` function. When calling this function, a small amount of *Ether* needs to be sent by the caller so that the *Users* can be compensated for the *Gas* they spend to vote. Along with a new vote a *Token* is created. The number of *Tokens* that are generated has to match the number of eligible voters. Additionally, it needs to be specified how long the ballot should be open and if the `transferVote` function should be enabled. Using that function a *User* can transfer his/her voting rights to another trusted *User*.

Once a vote is created, a *User* can request a *Token* to vote using the `enterVote` function. If *User A* called `transferVote` to *User B*, *User B* can also call the function `voteFor` and cast a vote for *User A*. Since calling these functions requires *Users* to spend *Gas*, they are paid back part of the *Ether* that the *Managing Entity* put into the *Voting System* when creating the vote. To further incentivise people to vote, the compensation could also exceed the initially paid *Gas* price. If, at a later time, a *User* decides to change his/her vote, either because of a change of mind or because the *User* doesn't agree with what the "trusted" person voted, it is possible to change the vote by using the `changeVote` function. It is crucial to mention that only the original owner of the *Token* can call `changeVote` which implies that even if the *Token* (and along with it the vote) was transferred, it is still the original *User* who's in authority. Besides that, it is to be noted that *Gas* has to be paid when calling the `changeVote` function which could be seen as an incentive for people not to change their vote multiple times and waste computational resources in the process.

When the deadline for a ballot has passed, all functions are disabled. Following that, the *Managing Entity* can call the function `finalizeVoting` to evaluate the results and `clearBalance` to recover any unused *Ether* from the smart contract. Otherwise this money would be stuck in that contract.

3.1.1. Voting Fraud

At first glance, it seems that a certain amount of trust in the *Managing Entity* is needed to create a fair voting. However, this is not entirely true. Under a set of assumptions it is possible to prove that potential fraud, for example by a malicious manager, could be detected.

Assumptions

The following assumptions are made: a *User* can prove that he/she is an eligible voter, the total number of eligible voters is known, a *User* will complain if not granted a *Token* for a vote, and, lastly, the total number of *Tokens* that were created for a vote are known.

Proving that a person is eligible to vote is rather simple for countries like Switzerland for example. In different circumstances, however, this might be more difficult and prone to manipulation. The estimation of the total number of eligible voters could also be problematic since another entity has to be trusted which might be closely connected to the *Managing Entity* (e.g. a government institution responsible for managing all the data on the country's citizens).

Knowing the total number of *Tokens* that are created is a rather easy task, since this information is stored on the blockchain. Lastly, the assumption that people would complain if they are denied their vote becomes more reasonable considering the fact that to manipulate a vote, it would be necessary to deny a substantial amount of voters their vote in order to make a difference. Hence, the chances that this would not be detected decreases substantially.

Proof

The proof has two parts. At first, we prove that if an eligible voter is denied his/her vote, the fraud is detected. As a second part, we prove that if an address (human or smart contract) can vote that is not eligible, it will be detected. Those two facts ensure that only eligible voters can vote and that no ineligible voter can vote. Thus, manipulation and fraud will always be detected.

If an eligible voter does not get a *Token*, then, by assumption, the voter will complain. Additionally, since all the information is stored on the blockchain the voter can prove that he/she did not get a *Token*. We also assumed that such a claim could be validated by the person proving that he/she is in fact eligible to vote. Hence, no person could claim to not having received a *Token*.

Accordingly, we now know that all eligible voters will get a *Token*, or else the fraud will be detected. If the entity distributing the *Tokens* now issues additional *Tokens* to an unauthorised party, the total number of *Tokens* issued will exceed the number of eligible votes. We assumed that this information is available. Therefore, this type of fraud will also be detected.

Consequences

Consequently, if we can find a way for these assumptions to hold, we do not need to trust a *Managing Entity* to have trust in the voting system.

Weighted Voting

Another concept we discussed as part of this project was to give voters a fixed amount of *Tokens* at the beginning of a specific time period in which all upcoming votings are already known. Voters would then be able to weight their votes according to which vote they have most interest in. For example, every voter would get 100 *Tokens* for the next 10 polls. Alice decides that she only cares about the first one, so she puts all her 100 *Token* to use by voting 100 in the first poll. After that she cannot vote anymore on the last 9.

Our implementation would be able to handle such a feature with just a few minor adjustments. For instance, currently it is possible for a single person to cast multiple votes in one ballot and to later change the vote.

3.2. Technical Implementation Proposal System

The Proposal system allows the *User* to Propose a new idea for a vote. Thus, the `createProposal` function can be called by any *User* in the voter list, differing to the implementation of `createVote` which can only be called by the *Managing Entity*.

All proposals are saved in a proposal array. For every proposal the following information is saved: the number of users needed that approve the proposal in order for the proposal to be eligible to become a vote, the number of approvals by users, the `endTime` of the proposal, and an array containing all the voters in which we track if the voter approved the proposal already.

In its current implementation the *User* can define the required number of approvals and the endTime by creating the proposal. For a future improvement of the system, this will be predefined by the *Managing Entity*.

`supportProposal` is used by users to support a specific proposal, and can only be done once per proposal per user. Note that in comparison to the *Voting System*, the *Proposal System* does not use *Tokens*. Instead it tracks the approvals in an array. All users are eligible to approve a proposal, and do not have to be enabled as `Voter` in the *Voting System*.

`createVotingForProposal` must be run by the *Managing Entity*. It checks if the `endTime` has not yet passed, and if the proposal gathered sufficient approvals from the users. Following that, it will call the `createVoting` function from the *Voting System* to create a vote. For this it also takes the duration of the vote, and if the transfer of votes is allowed as input.

In order to be able to call the `createVoting`-Function from the *Voting System*, the `vSystem` variable must point to the *Voting System*.

Additionally, the *Proposal System* contains several informative functions which are self-explanatory: `getProposalCount`, `checkProposalStatus`, `showProposalApproval`, `showProposalDeadline`, `showProposalApprovalNeeded`.

3.3. Technical Implementation Token System

The Token System is managed by the Token contract we generate on each voting. When a manager decides to open a new voting, the smart contract of our voting system deploys a new token contract with an `initialSupply`, `tokenName` and `tokenSymbol`. The initial supply is defined by the number of voters already registered in the voting system during the creation of the voting. Currently, the initial supply is equal to the total supply. In the future, we are considering increasing the total supply after the creation of the contract. This could be the case if a voter becomes eligible to vote after the creation of the voting.

The token name is always “VoteCoin” and the symbol always “VTC”. Additionally, it would be possible to also define individual names and symbols for each voting. These values would then be given as parameters to the createVoting function.

The voting system manages each vote from the voters by checking first the amount of available tokens in the Token contract of the corresponding voting. Since we used a standardized ERC20 Token the functions and variables in the code are well explained in [6]. The only thing that we changed is that we added a sender address parameter to most functions. This was done because the calling contract is always the voting system, but we want to transfer tokens to and from actual voters. And since we do not want them to call the token standard directly, we added a layer of indirection that allows the voting system to call token functions in the name of a user.

3.4. Testing

To test the *Voting System*, we defined about 50 functionality tests. The description of these tests is written in the `testcases` file.[1]

To reproduce the tests in Remix, the following steps have to be taken:

1. Import all the files in the folder `voting-system/contracts` to Remix.
2. To create the *Voting System*:
   a) Increase the Gas-limit (i.e. to 30'000'000).
   b) Click on the *Create* Button while having `VotingSystem` selected.[2]
3. To create the *Proposal System*:
   a) Copy the address of the *Voting System* and set the `address_vsystem` field to the address.
   b) Now Click on the Create Button while having the ProposalSystem selected. Also use the *Managing Entity* for this step.
4. *Tokens* get created automatically, for debugging reasons here a short guide how to open the matching contract:
   a) A `Vote` can only be created if at least one `User` is enabled. Thus first enable a `Voter` by adding the `User`-address in the `enableVoting` function.
   b) Create a vote via the `createVoting` function.

[1]As test environment we used Remix. Remix allows to run Solidity Code in a browser-based integrated development environment (IDE).
[2]The *User* used during the creation defines the *Managing Entity*.

c) The array `votings` now contains the vote, to get the information use the matching index for the voting-function. If it was the first created vote, the index is 0.

d) The address field contains the token address. Copy this address, put it in the field *Load contract at address* and click *At Address* to load the token system of that vote.

CHAPTER

4 EVALUATION

4.1. Limitations of the current Implementation & Possible Improvements

The current system is the result of a two-day Coding challenge. Due to this time-constraint, there are several important functionalities missing, and certain problems need yet to be solved.

User Authentication/Verification In its current implementation it is assumed that each Voter has a wallet-address, and this address is known to the *Managing Entity*.

For a real implementation this address must be generated in a secure and easy to use way, and verified by certain means to make sure the *Managing Entity* knows the relationship between wallet-address and voter. We already implemented a function disableVoter(address voter, uint index) which disables a voter from voting in a certain vote. However, we chose to disable this function because it would give the *Managing Entity* means to manipulate the vote. Solutions would also need to take problems like loss/stealing of private key and how to disable/removing voters into account.

No Authentication for Petitions The current implementation allows any voter to start a proposal and approve other proposals. In certain democracies this is only allowed for citizens that are eligible to vote. Such a feature is currently not implemented, but can be implemented easily.

Cleanup/Lifecycle of Votes/Proposals Currently, new votes/proposals get added to respective arrays. When a vote/proposal is over, the vote/proposal just stays in the array and is no longer needed. At the moment, it is also possible to create several identical votes, or from one proposal create several votes once the proposal gathered sufficient approval. Here, additional functionality is needed.

Accessibility of Internet for Users We assumed all the voters have internet access. To drop this assumption, we would need a more thorough analysis of possible solutions.

CHAPTER 5 CONCLUSION

During the short timeframe of BIOTS 2018 we created a Proof-of-Concept of a Voting System based on Blockchain. Using the Blockchain via Solidity-Ethereum proved to be a very useful tool to implement a distributed Voting System. We believe such an implementation of a Voting System allows for a high level of trust and certain prevention of Voting Fraud.

We successfully implemented the following features:

- A token based decentralized binary Voting System
- A proposal system which allows user to submit and support proposals
- Liquid Democracy: the possibility to allow somebody else to vote with your vote on a subjects.
- Revote: voter can change previously submitted vote.

In a time where Democratic Processes are more and more under attack, we believe Blockchain is a great opportunity to provide a safe voting system, and to increase the trust of the voters in the voting process.

Using the Solidity-Language based on the Ethereum-Blockchain allowed us to implement Features with relative ease.

Additionally to our work on the Voting System we created a Javascript-Framework that lets you create standardized ERC20 Token, without any technical Knowledge. While this Framework was not used for the Voting-System we think this is a valuable contribution to the community.

All the Code created by the Project we provide publicly via Github-Repository, and we encourage anybody to freely use our Code as they please and use this idea to help democratize decision making.

APPENDIX

A ADDITIONAL MATERIAL

A.1. Token Creator Library

Since there already exists a standardized ERC20 Token in Solidity, we decided to build a Node Package Module (NPM) for automatic configuration, creation and deployment of Tokens in Javascript. One option would have been to use multiple modules and patching part of them. However, this would have affected the simplicity of the FrontEnd. Hence, we built a small library with which it is possible to configure a custom token, build and compile the Solidity code and deploy it to a selected Ethereum network.

As a first step, the NPM module is downloaded via

```
npm install -save eth-token-creator
```

to the current Javascript project. This installs the module and all of its dependencies in the `node_modules` folder, where it can be required and used for a project. The dependencies this module mainly uses is `solc` and `web3`.

- ***solc*** is a widely used module that has Javascript bindings for the Solidity compiler. We configured its code for the token in order to be able to create a custom Token with various functionalities. Currently, however, only small payment and traffic rules that can be added are supported.

- ***web3*** is a Javascript API for Ethereum networks.

After requiring a configuration object can be defined that is passed to the module. Necessary fields are: `name`, `symbol`, `initialSupply` and `gaz`. Using additional options it is possible to arrange the ability of adding and clearing Tokens depending on payments or other valued inputs from sensors. A drawback of this approach is that currently anyone could give the sensor data according to the functions of the Token. By calling other functions of the module a provider for the *web3* part of the module can be set and one can define define which network one wants to use and which credentials. Following that, the Solidity code that was generated after the configuration is compiled, and the interface, the ABI of the Solidity contract are extracted for future use with *web3*. By deploying the contract using *web3* and the defined provider a *web3* wrapper of an active contract on an ethereum network is created with which one gains access to the functionality that is defined in the compiled Solidity code.

We will continue to maintain the module and build more functionality. As we only had a very small amount of time for this project, we could not build a full token creation Javascript API that is able to handle all use cases. However, we have managed to create a small library that can be used for deploying contracts without the need of programming any Solidity code. Currently, we have daily download rates between 20-30 installs and we continue working on new features to improve the transfer system of the tokens created by the module.

BIBLIOGRAPHY

[1] Anthony Faiola. *Venezuela election results 'manipulated' by at least 1 million votes, polling company says.* 2 August 2017. URL: https://www.washingtonpost.com/world/the_americas/venezuela-election-results-manipulated-by-at-least-1-million-votes-ballot-company-says/2017/08/02/0ce9025c-778c-11e7-8c17-533c52b2f014_story.html (visited on 15/02/2018).

[2] Drew DeSilver. *On Election Day, most voters use electronic or optical-scan ballots.* 8 November 2016. URL: http://www.pewresearch.org/fact-tank/2016/11/08/on-election-day-most-voters-use-electronic-or-optical-scan-ballots/ (visited on 15/02/2018).

[3] *New Study: 2016 voters and poll workers see improved technology as key to restoring trust in U.S. voting system.* URL: http://www.smartmatic.com/news/article/new-study-2016-voters-and-poll-workers-see-improved-technology-as-key-to-restoring-trust-in-us-voting-system/ (visited on 15/02/2018).

[4] *What are liquid, direct and representative democracy?* 12 December 2013. URL: http://www.demsoc.org/2013/12/12/what-is-liquid-democracy/ (visited on 15/02/2018).

[5] *Democracy Earth Foundation.* URL: https://www.democracy.earth (visited on 15/02/2018).

[6] *ERC20 Token Standard*. URL: https://theethereum.wiki/w/index.php/ERC20_Token_Standard (visited on 15/02/2018).