Vilnius university
Institute of Data Science and Digital Technologies
L I T H U A N I A

INFORMATICS (N009)

CREATION OF BLOCKCHAIN-BASED APPLICATION MODEL FOR DECENTRALISED ELECTRICAL ENERGY EXCHANGE

Paulius Danielius

October 2020

Technical Report DMSTI-DS-N009-20-20

VU Institute of Data Science and Digital Technologies, Akademijos str. 4, Vilnius
LT-08412, Lithuania
www.mii.lt

Abstract

This report presents research methods, review of blockchain platforms with smart contract functionality, theoretical and empirical research. A model of smart contract for electricity exchange on Ethereum platform is proposed. An experiment for testing vulnerabilities of Ethereum smart contracts of open source projects related to electricity sector is conducted and summary of experimental results is presented together with directions for further research.

Keywords: Blockchain, Ethereum, Smart contract, electricity trading.

Contents

1 Introduction

Following current trends of energy systems' transformation towards sustainable and clean energy and increasing deployment of renewable energy sources, more households are acquiring independent power generation capacities and thus becoming prosumers. This encourages fostering community capabilities of sustaining themselves by sharing excess of produced energy among community members and reducing the need and usage of electricity provided by conventional centralized systems.

The challenge lies in finding effective technological solution which enables automated electricity sharing/trading over distributed network in *decentralized*, *transparent* and *trustful* way.

Blockchain technology is explored as potentially suitable platform for distributed energy market and number of blockchain research and industrial projects and startups are seeking solutions for the energy industry [2].

Smart contracts executed on certain public and private blockchain platforms like *Ethereum* provide *Turing complete* programming environment and are of particular interest as they enable to create sophisticated and varied applications. Their resistance to vulnerabilities is the key factor for functioning and stability of the power infrastructure.

This work is devoted to the investigation of features of 2nd generation blockchain platforms (i.e. smart contract capable), their suitability for distributed electricity trading application, vulnerability issues and is organized in following way:

Section 2 describes research methods used for theoretical and empirical research. Section 3 presents review of blockchain platforms. Section 4 presents a model of Ethereum blockchain smart contract for electricity exchange. Section 5 presents an empirical research of vulnerabilities of existing Ethereum blockchain smart contracts from projects within energy sector as well as excess gas consumption, as one of vulnerability categories, analysis.

2 Research methods

Literature review and comparative analysis were used for reviewing 2nd generation blockchain platforms and evaluating their suitability for decentralised electricity trading application. Literature review and analysis were also used for preparation of experiment – for selection of testing tools and smart contract test set from blockchain-based projects related to the energy sector.

Conceptualization and modelling were used for building smart contract model for peer-to-peer electricity exchange on Ethereum blockchain platform and creation of model smart contract for experiment.

Experimental analysis was used to check and evaluate vulnerabilities of smart contract test set and to evaluate extra gas consumption of the modelled smart contract.

3 Review of blockchain platforms

For implementation of customised business logic on the blockchain, the platform has to have smart contract functionality. Such blockchain platforms are called 2nd generation blockchains. When considering a smart contract enabled blockchain platform suitability for a specific use case many aspects should be accounted for.

- Industry focus – the role which a platform is intended to occupy in the industry: is it a general-purpose cross-industry platform designed to build various types

of applications on top of it, or providing financial services, or certain similarly more focused role.

- Consensus protocol which is a crucial component for members of distributed system to reach consensus on its true state at the certain point in time. Different consensus protocols ensure different levels of security and different performance and scalability capabilities.
- Programming language for smart contract development – is it language specific to certain blockchain platform or well-known widely used programming language.
- Permissioned or permissionless platform – does it allow every user reading and updating blockchain state or just selected, identified users.
- Transaction throughput – what is blockchain platform capability to perform certain number of transactions per second.
- Native currency – does a blockchain platform employs native cryptocurrency.
- Architecture of the platform which can be highly modular, configurable, allow for changeable consensus, or have certain limitations which could hugely impact its performance.

Table 1 provides characteristics [33], [34], [30], [38], [35], [29], [31], [37], [36] of the most popular 2nd generation blockchain platforms.

Table 1. Characteristics of 2nd gen. blockchain platforms.

BC platform	Industry focus	Consensus protocol	Programming language	Permissioned/ permissionless	Transactions per second	Native currency
Ethereum	Cross-industry	PoW	Solidity	permissionless	~15	+
Hyperledger Fabric	Cross-industry	pluggable	Go, Java, JavaScript	permissioned	>1000	-
Corda	Cross-industry	RAFT, BFT	Java, Kotlin	permissioned	>100	-
Tendermint	General purpose	BFT	Any via ABCI	permissionless	>10000	-
NEM	Services-oriented	Proof of Importance	Java and C++ via API	permisionless	>100	+
Cardano	Cross-industry	Ouroboros (PoS)	Solidity, Plutus	permissionless	>1000	+
EOS	Cross-industry	delegated PoS	C++	permissioned	>1 Million	+
Stellar	Financial services	SCA (type of FBA)	Multilang. via API	based on proliferated trust	>1000	+
NEO	Smart Economy	Delegated BFT	Multilang. via plug-ins	permissioned	>1000	+

Considering these various characteristics as well as the size of active developer community and user base, time of successful operation, vulnerabilities history, choosing of blockchain platform for a particular scenario is not a trivial task.

One of the critical characteristics for transaction-intensive application is network throughput – i.e. transactions per second a blockchain can pull off. While comparing blockchain platforms' performance in registering simple monetary transactions, provided they have native currency, may be easier (although information provided by [21] shows significant discrepancies between claimed and registered performance), transaction throughput of smart contracts implementing more sophisticated business

logic in general should be tested on the basis of specific scenarios either by simulation or by exploring production environment.
Consensus protocols such as Proof of Work (PoW) provide more secure environment as it requires more than half (51%) of hash rate to take over the network, while *Byzantine fault tolerance* (BFT) can operate only with less than one third of malicious/faulty nodes. There are modifications of BFT or *Proof of Stake* (PoS) protocols which increase security. However, PoW protocol requires enormous amount of computational power and in consequence is one of major limiting factors for network performance and scalability. The PoS protocol in this regard is much more lightweight and ensures adequate level of security (by slashing staked wealth in case of malicious actions), but network with PoS may hinder acceptance of users if all they want is using provided services without staking (i.e. effectively freezing) their monetary assets.
Blockchains with native cryptocurrency may facilitate the implementation of trading mechanism through the use of smart contracts but may have additional vulnerabilities related to malicious activities directed to take ownership of coins. For blockchain platforms without native currency, tokens may be implemented through smart contracts which could be specifically tailored to the specific application scenario.
The ability to use general-purpose widely-known programming languages for smart contract development is a huge advantage as compared to blockchain platform-specific language, however special measures must be deployed to ensure deterministic outcome of transactions accounting for non-deterministic nature of general-purpose languages code.
Some of the blockchain platforms have specific disadvantages like NEM, which, while being the most secure, according to numerous experts, and highly scalable, uses off-blockchain smart contract code, which makes it less decentralized, or Stellar, which only allows for simple, non-Turing complete smart contracts such as ICOs.

Ethereum blockchain platform remains one of the best known and widely used platforms for smart contract deployment despite the major drawback – the network is already operating at 100% capacity and this is the limiting factor on the number of transactions per second it is capable to perform. To expand network capabilities, improve scalability, increase security and reduce energy consumption due to PoW consensus protocol, Ethereum platform is going to get the massive upgrade. Ethereum 2.0 (Eth2) will change the consensus protocol to the Proof of Stake and introduce shard chains which will allow for parallel transaction processing. This shift to the new version will be performed in several phases with the launch of the main chain, registering validators and stakes on the first phase expected to happen in 2020, introducing shard chains and eventually moving entire Ethereum user base on the new network in subsequent phases. The entire upgrade is expected to be achieved in about two years [32].
Hyperledger Fabric is of particular interest being the first blockchain platform to introduce the *execute-order-validate* architecture which separates the transaction flow into three steps: execution, ordering and validation which may be run on separate entities in the system. In contrast, all previous permissionless and permissioned blockchain systems use *order-execute* architecture which means that transactions are ordered first using a consensus protocol, and then executed on all peers sequentially in the same order which seriously limits the effective throughput [3].

4 Proposed model of a smart contract for electricity exchange on Ethereum blockchain

For smart contract development we used methodology described in [6], appropriate for industrial-strength blockchain application design and implementation.
The process involves three stages:

- analysis stage
- design stage
- implementation stage

In the **analysis stage** we analyzed the requirements, identified involved entities, their roles and types of interaction (**Fig. 1**) for the deployment and usage of the electricity exchange smart contract.

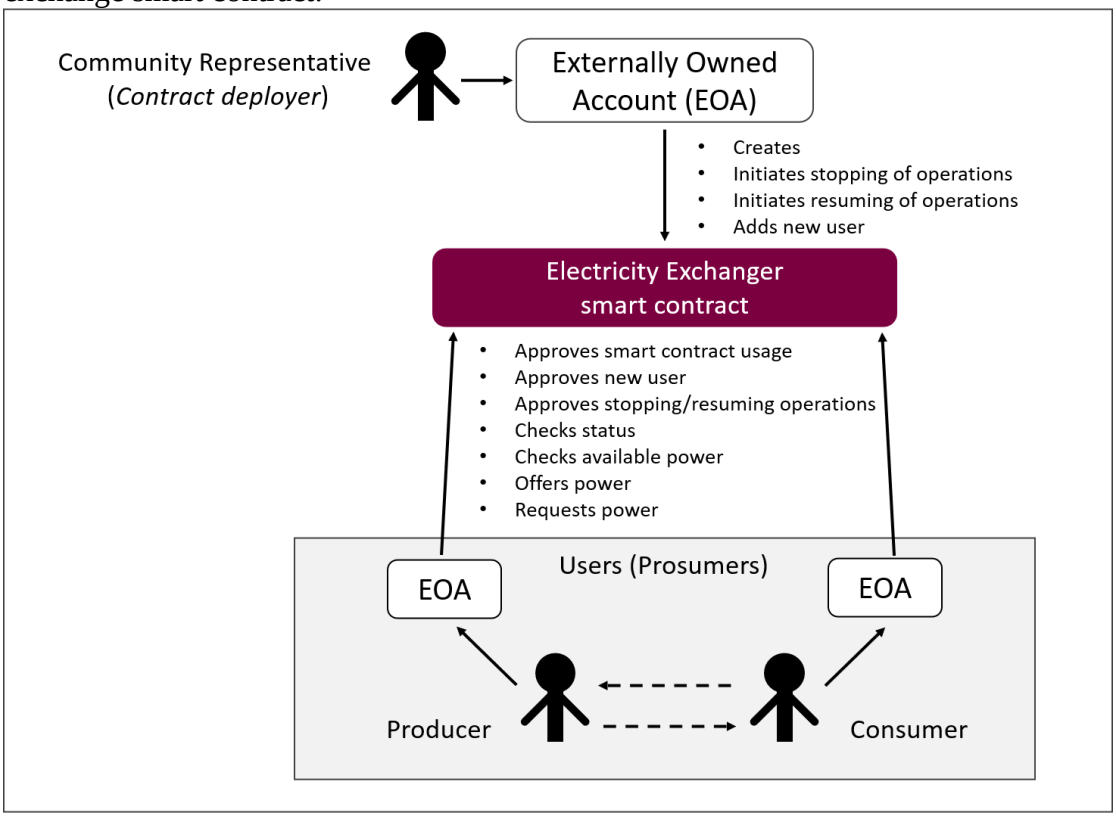

Fig. 1. Analysis stage of the smart contract.

Community representative deploys smart contract on the blockchain along with initial information: addresses of contract users accounts, approval threshold (percentage of users needed for approval) and minimal power offer required.
Community members approve contract by using multisignature scheme and when approval threshold is reached, the contract is activated. Users, who have power surplus can make offer to share that power and users who are in the need of power can make requests to consume it. When power request transaction is recorded on the blockchain, *releasePower* event is dispatched which may act as activating signal for smart device to release power to the consumer.
Additional users may be added at a later time. Smart contract can be stopped and resumed. Initiation of these actions is carried out by community representative. New users and the changing of contract state are also approved by members using multisignature scheme.

In the **design stage** entities' attributes were modelled as state variables and interactions between entities as functions.
We used following **design patterns** to circumvent certain common security vulnerabilities of smart contracts like reentrancy and unauthorized actions [28], [16]:

- *finite state machine* - for ensuring finite number of states contract could take, and to restrict certain function to certain states, the states are presented in **Fig. 2**;
- *access restriction* – for ensuring that only certain users (contract owner) or users with certain status (users, who are approved by other members) are allowed to use specific functions;
- *locking* – for ensuring that state variables could not be altered by other users, when one of them is invoking a function which changes values of such variables.

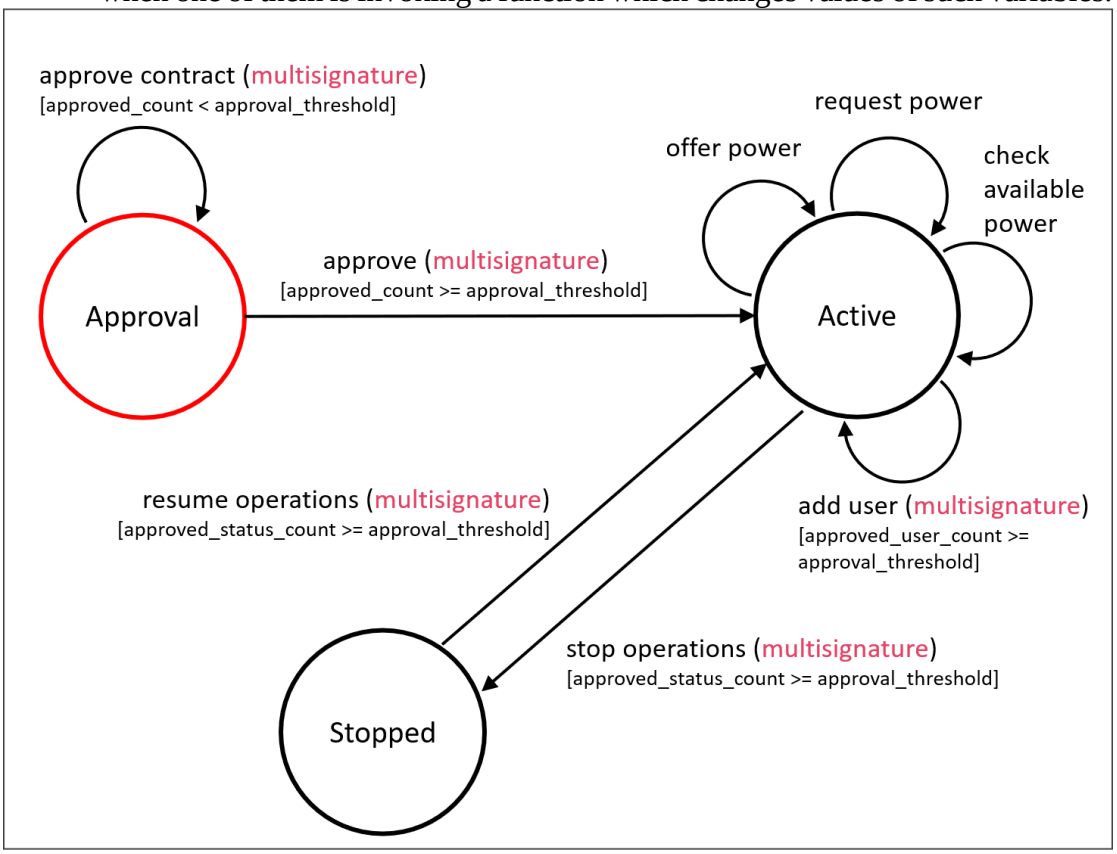

Fig. 2. State diagram of the smart contract.

In the **implementation stage** we implemented smart contract based on the state variables and functions identified in the design stage (**Fig. 3**). For implementation we used high-level *Solidity* language.
By designing this smart contract, the assumption was made that all prosumers have the capacity to accumulate produced energy by employing batteries and there is no increasing excess of power over period of time. This allows for exchanging discrete units of power (as measured in kWh for example) and consume all produced energy inside the community.

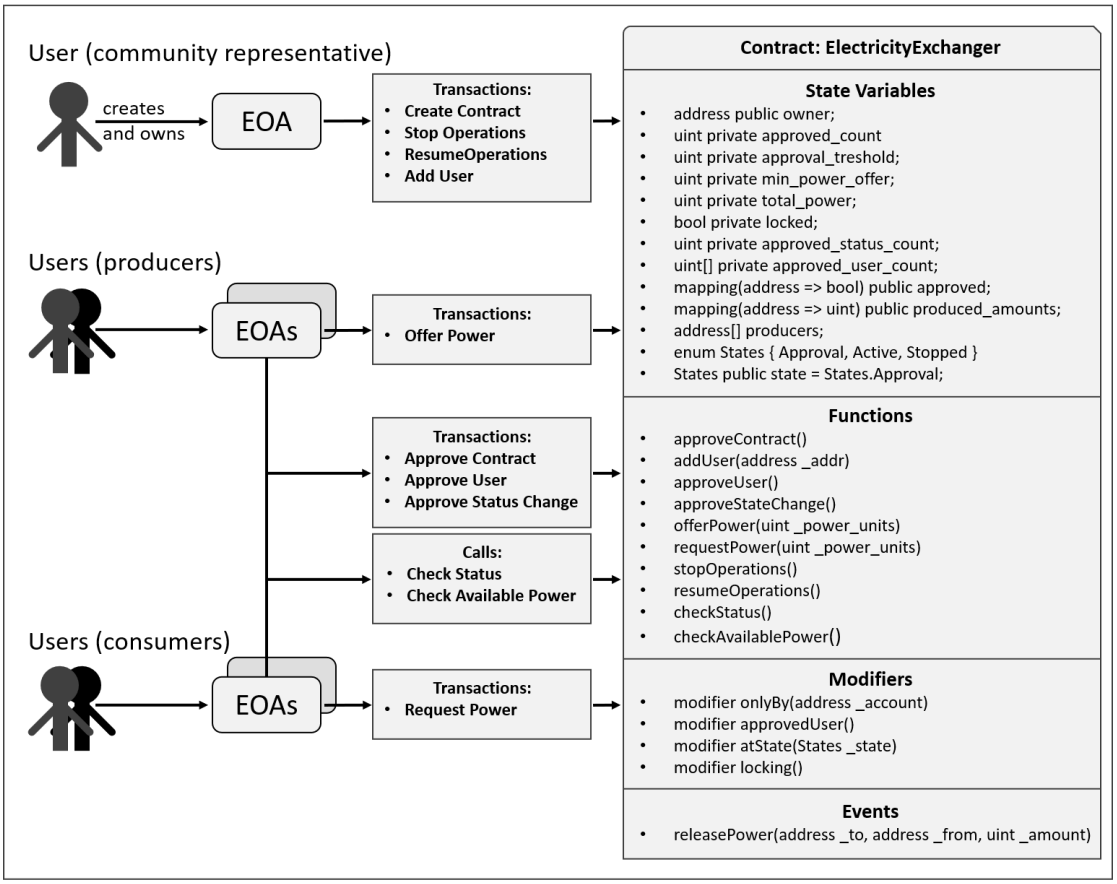

Fig. 3. Implementation stage of the smart contract.

Further directions for refinement of this model would be incorporating appropriate energy trading mechanism to expand the model for continuous power exchange scheme without the need of power accumulation capacities.

5 Empirical research of smart contract vulnerabilities within Ethereum network

Issues related to the security of information systems are particularly significant for business environments due to the importance and value of resources controlled by these systems. They are also an important field of research for the scientific community. While the security of traditional information systems is a fairly well-developed matter, the emergence of new technologies, the use of which has not been fully tested, creates a further need for in-depth research and analysis. In particular, the presented research is focused on the *Ethereum* blockchain and smart contracts technologies employed to manage energy infrastructure.

In a recent couple of years, i.e. since the popularization of the idea of smart contracts, a whole series of works on the security of this type of software were produced. More and more potential vulnerabilities and threats are also known. The importance of this topic has increased when the first successful attacks started [25]. Those attacks made the public aware that the significant real value is at stake.

Much theoretical and practical work has been done since then to ensure increased safety. An example of practical actions includes improvements in the *Ethereum Virtual Machine* (EVM) construction and changes to the *Solidity* language. Nevertheless, the research area remains open. New threats will certainly be discovered. This is due, for

example, to the fact that the blockchain infrastructure and programming languages in which smart contracts are created are currently being intensively developed. It is also worth noting that new blockchain platforms offering the possibility of performing logical operations in conjunction with transactions, especially Turing-complete, are constantly emerging [19].

5.1 Research workflow

The presented research is based on the three-factor approach. First, a critical literature review is done to identify potential threats and blockchain-based projects related to the energy sector. Second, the evidence from the experimental analysis gathered and organized using third-party tools. The test set of applications results from the first part of the study. Third, a theoretical experiment in the computing economy field is provided. The experimental setting is the effect of an analysis and conclusions drawn from the second part of the research.

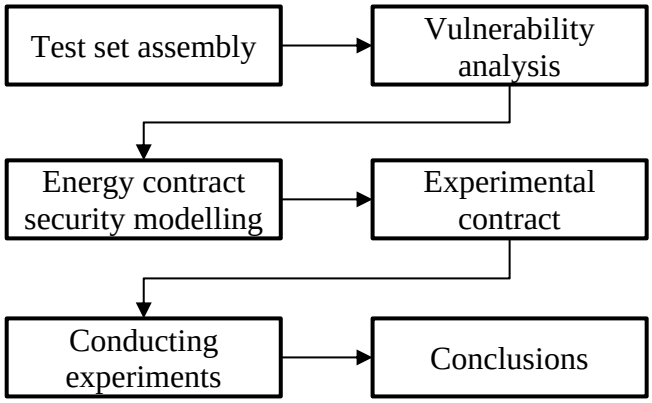

Fig. 4. The research workflow.

The ordered steps of the study are shown on **Fig. 4**. In this research, we focused on smart contracts' vulnerabilities. The filtered set of selected contracts represent only the cases of electricity sector applications. Their characteristics are described in detail in Section 5.5.1.

5.2 Initiatives for Peer-to-Peer energy markets

In this section a short summary of the idea and the most notable projects related to creation of a P2P (peer-to-peer) energy markets are given.

A smart grid is a utility grid that includes a variety of elements such as smart meters, smart appliances, multiple inputs, and output facilities with the ability to balance itself both at the micro and macro scale. The future of a utility grid involves the process of creating a hybrid with the original infrastructure. It means that the basic grid is enriched with the means of exchanging information between grid nodes, as well as the nodes themselves must have at least the functionality to monitor their internal state.

Early ideas of connecting the smart grids with blockchain infrastructure have emerged in 2016 and were developed a year later [15], [18]. Some small projects have been carried out so far, but scholars omit the impact of blockchain architecture on grid security. In our opinion, the research on the implication of blockchain security characteristics applied to the smart grid solutions is an important challenge.

A very extensive analysis of energy sector blockchain projects has been presented in [22]. Two related solutions are offered in [13] that are named Pando and Exergy. Pando enables utilities and retailers to introduce an energy marketplace. All the members of the community can trade energy. The exchange allows achieving of consumers' goals as well as maximizes community energy utility. The application is simply deployable, highly extensible, and flexible.
Another offering [12] of the blockchain-based open-source and customizable energy exchange. The company develops the D3E energy exchange engine which may be used by communities and other platforms. The exchange is designed with an emphasis on supporting environmental protection and green energy production.

5.3 Threats

A resemblance with the *Internet of Things* (IoT) exists regarding the safety of energy applications. Some of the home appliances that allow measuring or controlling power consumption are treated as specific IoT components. Two types of issues are crucial both for IoT and smart grids which are security and privacy issues [11].

5.4 Related works

Currently, only a very limited number of works is devoted to the security of electricity sector-related blockchain applications. The early papers that deal with the issue of smart contracts security are dated back in the year 2016. Then, the first attempts to formally verify the soundness of the code were proposed [7]. Additionally, a survey on Ethereum attacks has been conducted [5]. It was an important milestone as on the one hand, it raised the problem of the growing number of security incidents related to the network and on the other hand, it showed that the whole blockchain construction deals with the whole typology of potential vulnerabilities.
In 2017 another incident after DAO Hack took place. It was the Parity wallet security-hole. It was analyzed among others by [23]. [8] makes the gas cost analysis of running smart contracts. They identify 7 gas-costly patterns and group them into 2 categories. They also propose and develop GASPER, a new tool for automatically locating gas-costly patterns by analyzing smart contracts' bytecodes. Their results show that some of the patterns are widespread within the smart contracts population. The issue of consuming gas is vital as it leads most of the developers to the introduction of the code neglecting the quality and security requirements.
In [24] blockchain is leveraged to a platform that facilitates trustworthy data provenance collection, verification, and management. The system utilizes smart contracts and is secure as long as the majority of the participants are honest. The last condition is the general requirement for any blockchain setting. The paper of [9] presents a novel way to permit miners and validators to execute smart contracts in parallel. It shows that a speedup of 1.33x can be obtained for miners and 1.69x for validators with just three concurrent threads. This kind of research introduces major innovation to smart contract engines. On the other hand, the decentralized and parallelized platforms are to be even more vulnerable, especially on the account of more complicated code and unpredictable effects.
[1] is a systematic study of main topics addressed in conducted researches (metanalysis) that are related to smart contracts. It shows that there are four main streams of topics. These key issues are codifying, security, privacy, and performance issues. The texts that dealt with these groups of problems consisted of 67% of analyzed papers. This result is vital as it highlights the crucial aspects of smart contract safety.

The problem of information privacy in the context of blockchain and smart contracts is also tackled in the [17] text.

5.5 Vulnerabilities analysis results

In this section the analysis of available *Solidity* source code of projects created to work within the energy sector is presented. The general characteristics of the test set is given. The specific results of vulnerabilities detected are presented as well.

5.5.1 Test set and tools

The test set included 6 applications together with 60 smart contracts. These are all the thematically related projects identified on the basis of related works and projects review that were available as open-source. The overall number of lines of code is 4 978 and the source lines of code (SLOC) equivalent totals 4 121. All the source codes were generated between January 2018 and November 2019. The details about the test set items are presented in **Table 2**.

The work of [4] presented the survey of existing tools for supporting the development of secure smart contracts and analysis of smart contracts. As there are quite a number of different tools with various capabilities, the tools we have chosen were based on public availability, and the ability to detect differentiated security issues. They were positively assessed and have advantages that include a rich library of known vulnerabilities, threats categorization, as well as an associated knowledge base.

Table 2. Test set characteristics.

Application Name	Latest version date	Number of files	Number of contracts	External libraries
Carbos	08/04/2019	6	6	0
SunContract	27/11/2019	2	8	0
Grid+	20/12/2018	7	7	0
GoGreenContract	19/02/2019	1	2	0
WePowerNetwork	28/01/2018	19	22	1
HivePower	18/05/2018	17	15	2
TOTAL		52	60	3

5.5.2 SmartCheck

In the preliminary research we acquired all analysis tools described in the literature. We rejected these that were outdated or out-of-order. Out of the three working tools the *SmartCheck* provides the most advanced results.

The SmartCheck [26] tool has been used to extensively test the smart contracts for potential threats. The tool is capable of indicating an impressive list of known vulnerabilities from numerous categories. It also warns about improper programming practices resulting in poor code quality. It is available online which makes the analysis a straightforward task. This tool is described in detail in [27].

The statistics obtained from the performed analysis are collected in **Table 3**. The table also shows the severity levels of the detected code malpractices. The total errors column shows the number of problems identified by the tool. The errors per SLOC is the result of our own calculations based on the total errors values. The severity categorization comes from the analysis tool with little downgrade adjustment of less critical malpractices.

Table 3. SmartCheck analysis results.

Application Name	Total errors	Errors per SLOC	Severity 1	Severity 2	Severity 3
Carbos	22	0,06	21	1	0
SunContract	156	0,43	154	2	0
Grid+	216	0,38	216	0	0
GoGreenContract	13	0,10	13	0	0
WePowerNetwork	208	0,11	203	4	1
HivePower	90	0,11	89	0	1
TOTAL	705		696	7	2

In **Fig. 5** the relations between the number of code lines and the number of detected malpractices are presented. Markers on the graph represent singled code files. The data reflects the information presented in **Table 3**. It is shown with more details because in the figure the files are the errors aggregation level instead of the whole projects. The graph reveals that frequently there is a visible linear relation between the length of the contract code and potential programmer mistakes. The inclinations of the lines are different for particular applications. It may reflect different backgrounds, experiences, and levels of programmers as well as expose their coding habits and style. These habits and style approaches may not always be best suited for the Solidity and the way of recommended contract construction.

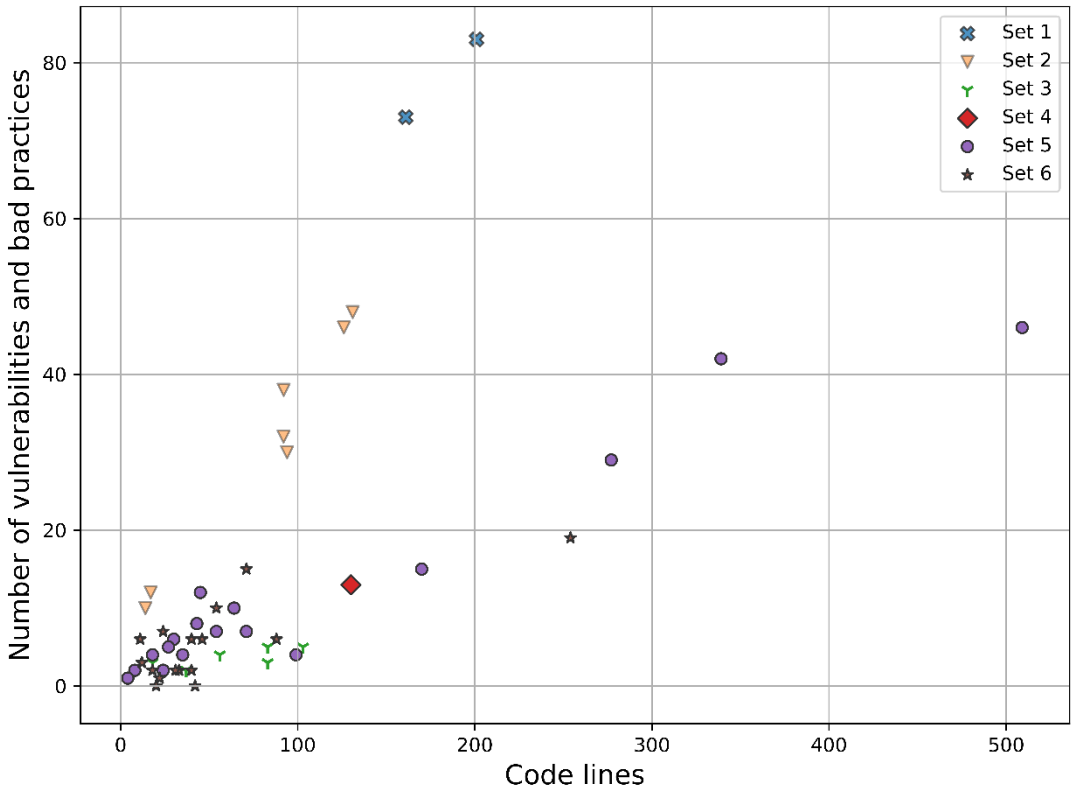

Fig. 5. The relation between the number of code lines and the number of detected flaws (SmartCheck).

The major part of detected issues, marked as low severity level (1), is related to bad coding practices and is easily remedied by following SmartCheck knowledge base recommendations.

The high (3) and middle (2) severity level vulnerability categories, as well as low-level issues (per SmartCheck) which are related to increased consumption of gas, as this is an important aspect for efficient electricity trading mechanism implementation, are presented in **Fig. 6**.

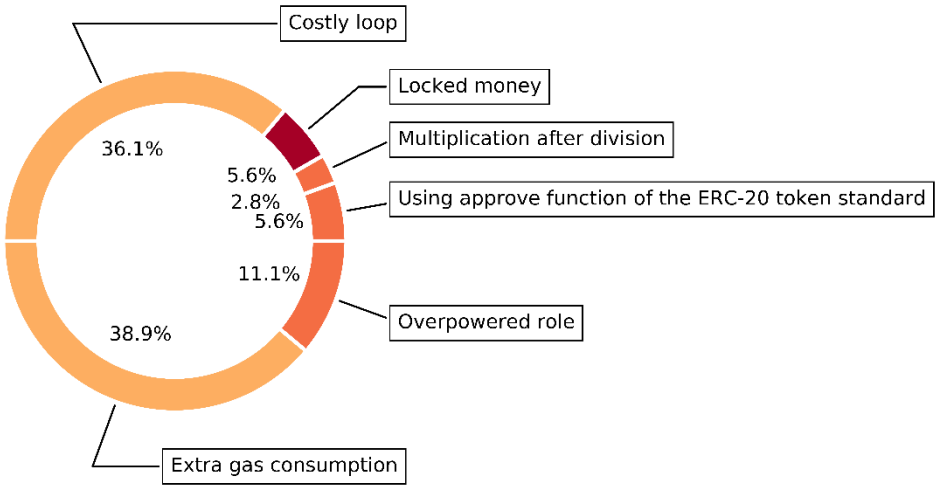

Fig. 6. Frequency of high-middle severity and low severity level vulnerabilities with increased gas consumption as detected per SmartCheck.

5.5.3 Other tools and conclusions from the empirical contracts testing

Maian [20] is another publicly available tool that may be used to analyze the security of Solidity contracts. Unlike the SmartCheck it is not accessible online and it is no longer maintained. It is able to recognize three types of code vulnerabilities. The three types of threats are related to the potential behavior of the contracts in the event of some extreme situations. These types are *greediness*, *suicidality*, and *prodigality*.

Although the tool is no longer actively developed its code is open-sourced. As part of our study, we managed to fix the tool code and arrange the inconsistencies related to the changes in the Solidity as well as Ethereum Virtual Machine revisions. Moreover, we extended the functionality to automatically detect the Solidity version in order to use the appropriate compiler (solc).

The results of the tool are uniform yet disappointing. Throughout the analysis of the whole set mentioned in Section 5.5.1 the behavior of only one contract has been identified as *greedy* (Grid+). No other vulnerabilities were discovered.

A similar situation is with the **Oyente** [14] tool. The trials with this tool brought no significant results.

We managed to make several important insights resulting from the analysis of results obtained during the process described in Section 5.5.2. The main points include:

the catalog of statistically most significant vulnerabilities evolves due to common knowledge, EVM improvements, and programmers' maturity.

As a consequence, the severity of the security vulnerabilities and programming errors change in time due to such factors as programmers' consciousness of the threats, general skills, and knowledge level as well as the development stage of the Ethereum network.

Aside from solving the major vulnerability issues (marked as severity levels 2 and 3), which were not frequent cases, the other most relevant cases for improvement are with code using loops with arrays or mapping types and in consequence resulting in the unpredictable expenditure of gas.

Contracts with loops do have another aspect of risk to consider, and this is especially important for P2P electricity trading in auctions: running such contracts potentially might cost huge amounts of gas, however, transactions with high gas expenditure are less likely to be picked by miners and included in a block, and this certainly could hinder smooth operation of an auction.
In consequence, the provisioning and reception of the energy might be perturbed which may be dangerous both to the consumers and the infrastructure itself.
The identified aspects allow the construction of another experiment that will be described in Section 5.6.

5.6 Electricity trading contracts security and EVM gas consumption

5.6.1 Experiment description

A special smart contract was developed for the experiment. The creation of a model smart contract is a rational step as the experiment was to give the explanation for a general category of typical energy sector smart contracts. None of the contracts from the test set exemplified the expected properties of the whole category. Additionally, working on a specifically designed source code allows for greater flexibility in conducting experiments. The designed smart contract has several features to resemble the simplest operations of electricity market auction. This means looping through records with energy amounts and its prices provided for trading and updating energy amount values. Cryptocurrency transactions were not considered.
Although loops are generally advised to be avoided in smart contract code altogether, practical auction implementation is hardly achievable without using them. Besides, such design allows measuring gas consumption with a certain amount of iterations and estimate the influence of recommended fixes for two vulnerabilities "extra gas consumption" and "costly loop" on gas cost. This, in turn, allows comparing obtained values with average transaction gas expenditure with the aim to estimate how many participant records could be operated upon with reasonable gas consumption.
The average gas cost per transaction was calculated based on *bitinfo.com* real-time data (**Table 4**) and block gas limit on the Ethereum network which is very close to 10 000 000.
The following data structures were used for keeping market participant data: array of participant addresses which will be used to iterate through all participant records and mapping from addresses to *struct* type elements with energy amount and energy price values. Test data was generated for a number of participant records using integer numbers as a basis for conversion to address type and for *amount* and *price* values.

Table 4. Average gas expenditure per transaction on the Ethereum network (bitinfocharts.com, 2020-04-28).

Transactions avg. per hour	Blocks avg. per hour	Transactions avg. per block	Gas avg. per transaction
35 505	271	131	76 327

5.6.2 Obtained results

Within the experiment, 2 types of test were conducted:

1. Estimating gas expenditure of reading operations, where 3 values are read for each participant in each loop iteration: one value from address array and two values, amount and price, from corresponding struct elements.
2. Same as above with the addition of updating amount value. Iteration count was chosen for reaching gas expenditure close to average transaction gas expenditure. A transaction basis fee of 21 000 gas is included in each measurement.

The test results are presented in **Fig. 7** and **Fig. 8**.

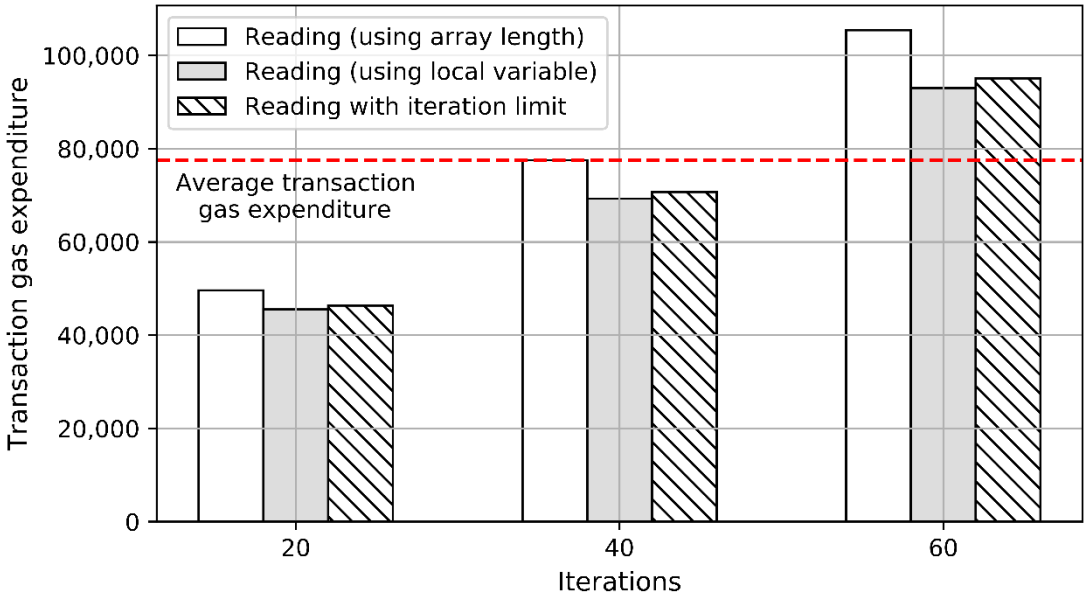

Fig. 7. Gas expenditure of reading operations.

Fig. 7 displays the gas expenditure of reading operations. Three variants of stopping the loop were used:

1. based on array length value – bad practice and "extra gas consumption" category by SmartCheck,
2. based on local variable value to which array length value was copied (recommendation of how to improve over the first variant),
3. iteration limit set through function parameter as additional loop end condition; limiting loop iterations to known number is the recommendation of how to safeguard against the "costly loop" vulnerability category by SmartCheck.

It is evident from the obtained results, that checking of local variable value in the loop condition does decrease gas expenditure as compared to checking array length value, and saving accumulates with more iterations. Checking for iteration limit as an additional loop condition only slightly increases gas expenditure as compared to less costly options from the first and the second.

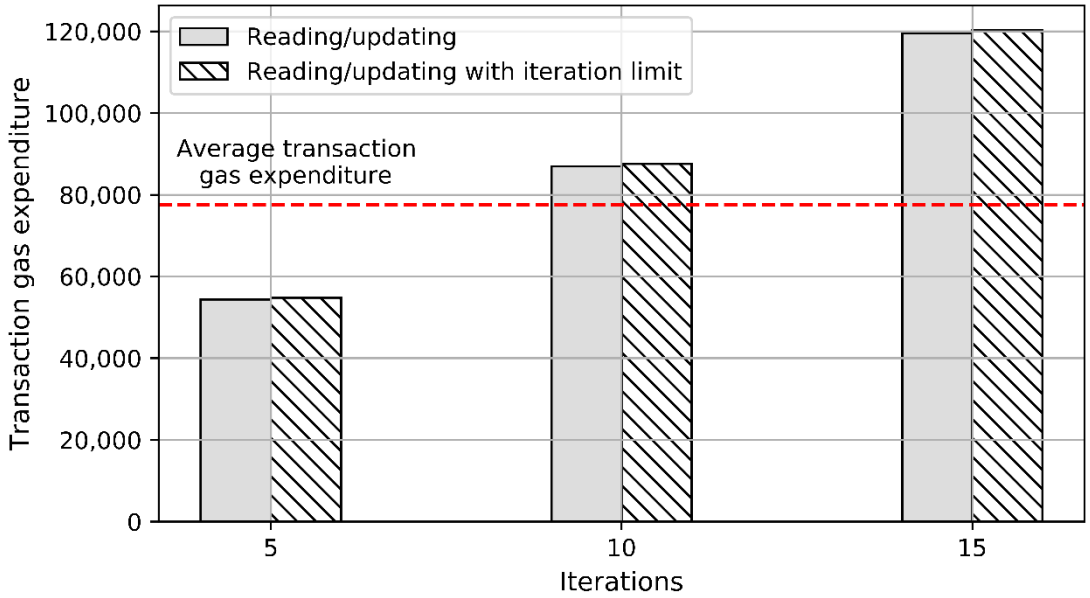

Fig. 8. Gas expenditure of reading and updating operations.

Reading and updating records' values was done with two variants of stopping the loop: known number of iterations by checking local variable value and adding additional loop condition to limit iteration number. The results are represented in **Fig. 8**. The difference in the gas expenditure of both variants is negligible. However, for reaching average transaction gas expenditure, less than 10 reading and updating operation iterations can be performed as compared to performing only reading operations. In case there is a need for more operations in each loop iteration, the gas expenditure would be accordingly higher which means average transaction gas expenditure would be reached with an even lower number of iterations.

5.7 Summary of experimental research and further directions

Checking for Ethereum smart contract vulnerabilities and risks is not a trivial task, because there are few publicly available tools and most of them have very limited or very specific capabilities.

In this work, we explored three tools, SmartCheck, Oyente, and Mayan. By testing 60 smart contracts of 6 projects dedicated to the electricity sector, 706 vulnerability issues, and coding malpractices were found, 705 of them were detected by SmartCheck, 1 by Mayan, while Oyente bore no results. In addition, SmartCheck provided a three-level severity classification of flaws. As the major part of issues were low level, easily fixable, and were more as bad coding practice than the actual threat they were not included for further consideration. The results allowed creating a model contract source code that was used to estimate gas usage of two cases from low severity level (as categorized by SmartCheck), which concern high gas expenditure.

Developers should be aware of gas usage when designing smart contracts for Ethereum platform with special care when writing the code, where using loops is inevitable, and follow the recommended practice, because not having control of loop iteration number may result in gas overuse.

Using smart contracts for electricity auction trading is limited on the Ethereum blockchain network in terms of a feasible number of participating members. From experiment results, we can conclude, that electricity auction would be more suitable for trading between a small number of agents (~10) than between prosumers, whose

number would be certainly far greater, in which case transaction gas expenditure can reach levels which may not be attractive for miners to include in their blocks. Consequently, auction performance may not be in line with expectancy, and some more costly in terms of gas usage transactions may stall leading to unpredictable negative effects in the grid. Another important aspect is that the number of independent electricity providers for EU counties is rising due to the liberalization of the electricity market. For example, in Lithuania, it is 4 (households) and 11 (B2B) operators [10]. These numbers correspond well to the mentioned above number of agents.

After the Ethereum platform upgrade to Ethereum 2.0 is finished a new evaluation of Ethereum blockchain for electricity auction trading will be needed, as network capacities and rules of game may change significantly.

For evaluation of electricity auction trading on a blockchain with better performance characteristics the Hyperledger Fabric platform will be used. Additional argument for selecting this platform is it's different *execute-order-validate* architecture as described in Section 3. As Hyperledger Fabric is a highly configurable modular platform, the impact of various configuration settings like block size, number of ordering nodes etc. on performance characteristics like throughput and latency will be tested.

6 References

1. Alharby, M., van Moorsel, A.: A systematic mapping study on current research topics in smart contracts. Int. J. Comput. Sci. Inf. Technol. 9, 5, 151–164 (2017).
2. Andoni, M. et al.: Blockchain technology in the energy sector: A systematic review of challenges and opportunities. Renew. Sustain. Energy Rev. 100, November 2018, 143–174 (2019). https://doi.org/10.1016/j.rser.2018.10.014.
3. Androulaki, E. et al.: Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In: Proceedings of the 13th EuroSys Conference, EuroSys 2018. (2018). https://doi.org/10.1145/3190508.3190538.
4. Di Angelo, M., Salzer, G.: A survey of tools for analyzing ethereum smart contracts. In: 2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON). (2019).
5. Atzei, N. et al.: A survey of attacks on Ethereum smart contracts. IACR Cryptol. ePrint Arch. 2016, 1007 (2016).
6. Bahga, A., Madisetti, V.: Blockchain Applications: A Hands-on Approach. VPT, 2017.
7. Bhargavan, K. et al.: Formal verification of smart contracts: Short paper. In: PLAS 2016 - Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security, co-located with CCS 2016. (2016). https://doi.org/10.1145/2993600.2993611.
8. Chen, T. et al.: Under-optimized smart contracts devour your money. In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER). pp. 442–446 (2017).
9. Dickerson, T. et al.: Adding concurrency to smart contracts. In: Proceedings of the ACM Symposium on Principles of Distributed Computing. pp. 303–312 (2017).
10. ESO: Lithuania Power Market Outlook, http://www.eso.lt/lt/verslui/elektra_99/informacija-elektros-ir-duju-rinkos-dalyviams/elektros-ir-duju-tiekejai.html, last accessed 2020/07/02.
11. Eze, K.G. et al.: Internet of Things and Blockchain Integration: Use Cases and Implementation Challenges. Presented at the (2019).

https://doi.org/10.1007/978-3-030-36691-9_25.
12. Grid Singularity: Grid Singularity, https://gridsingularity.com/.
13. Llo3energy: Llo3energy, https://lo3energy.com/.
14. Luu, L. et al.: Making Smart Contracts Smarter. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 254–269 ACM, New York, NY, USA (2016). https://doi.org/10.1145/2976749.2978309.
15. Mannaro, Katiuscia; Pinna, Andrea; Marchesi, M.: Crypto-trading: Blockchain-oriented energy market. In: 2017 AEIT International Annual Conference. pp. 1–5 (2017).
16. Mavridou, A., Laszka, A.: Designing Secure Ethereum Smart Contracts: A Finite State Machine Based Approach. Presented at the (2018). https://doi.org/10.1007/978-3-662-58387-6_28.
17. McCorry, P. et al.: A smart contract for boardroom voting with maximum voter privacy. In: International Conference on Financial Cryptography and Data Security. pp. 357–375 (2017).
18. Mengelkamp, Esther; Notheisen, Benedikt; Beer, Carolin; Dauer, David; Weinhardt, C.: A blockchain-based smart grid: towards sustainable local energy markets. Comput. Sci. Res. Dev. 33, 207–214 (2018). https://doi.org/10.1007/s00450-017-0360-9.
19. Neo: Neo Cryptocurrency, https://neo.org/.
20. Nikolić, I. et al.: Finding the greedy, prodigal, and suicidal contracts at scale. In: Proceedings of the 34th Annual Computer Security Applications Conference. pp. 653–663 (2018).
21. O'Neal, S.: Who Scales It Best? Inside Blockchains' Ongoing Transactions-Per-Second Race, https://cointelegraph.com/news/who-scales-it-best-inside-blockchains-ongoing-transactions-per-second-race, last accessed 2020/10/10.
22. Pichler, M. et al.: Decentralized Energy Networks Based on Blockchain: Background, Overview and Concept Discussion. Presented at the (2019). https://doi.org/10.1007/978-3-030-04849-5_22.
23. Qureshi, H.: A hacker stole $31 M of Ether-how it happened, and what it means for Ethereum. Free. org, Jul 20, 2017. (2017).
24. Ramachandran, A. et al.: Using blockchain and smart contracts for secure data provenance management. arXiv Prepr. arXiv1709.10000. (2017).
25. Santos, Francisco; Kostakis, V.: The DAO: a million dollar lesson in blockchain governance. Sch. Bus. Governance, Ragnar Nurkse Dep. Innov. Gov. (2018).
26. SmartCheck: SmartCheck, https://tool.smartdec.net/, last accessed 2020/07/02.
27. Tikhomirov, S. et al.: Smartcheck: Static analysis of ethereum smart contracts. In: 2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). pp. 9–16 (2018).
28. Wöhrer, M., Zdun, U.: Design patterns for smart contracts in the ethereum ecosystem. In: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). pp. 1513–1520 (2018).
29. About the Cardano Network, https://docs.cardano.org/en/latest/explore-cardano/cardano-network.html, last accessed 2020/10/19.
30. Corda Documentation, https://docs.corda.net/docs/corda-os/4.6.html, last accessed 2020/10/19.
31. EOSIO Manuals, https://developers.eos.io/welcome/latest/manuals/index, last

accessed 2020/10/19.
32. Ethereum 2.0 (Eth2), https://ethereum.org/en/eth2/, last accessed 2020/10/19.
33. Ethereum development documentation, https://ethereum.org/en/developers/docs/, last accessed 2020/10/10.
34. Hyperledger Fabric, https://hyperledger-fabric.readthedocs.io/en/release-2.2/, last accessed 2020/10/19.
35. NEM, https://docs.nem.io/en, last accessed 2020/10/19.
36. Neo White Paper, https://docs.neo.org/docs/en-us/basic/whitepaper.html, last accessed 2020/10/19.
37. Stellar DOCS, https://developers.stellar.org/docs/start/introduction/, last accessed 2020/10/19.
38. Tendermint Core, https://docs.tendermint.com/master/, last accessed 2020/10/19.

Appendix Nr. 1.

Used abbreviations:

BFT – Byzantine Fault-Tolerance
EVM – Ethereum Virtual Machine
ICO – Initial Coin Offering
PoS – Proof of Stake
PoW – Proof of Work