CS 6675 – Advanced Internet Computing Systems

Course Project, Pascal Wissmann

PiTrust – A blockchain-based trust network (Checkin 2)

Abstract

In a world of global connections between people, most of them having never seen each other, it is hard to gain (but often easy to lose) trust. While centralized trust databases are already in place and serve well for particular purposes such as eBay or Uber, these are still designed for very particular purposes and are fully controlled by a central party while not being independently verifiable by any peer.

In this project, I propose a system to quantify the trustworthiness or expertise of an entity in a particular topic. The system will be based on blockchain technology in order to keep an non-revokable ledger on ratings and therefore expertise of the entity, which shall facilitate honesty across all participating parties and provide a high level of reliability and availability of the system. In order to achieve those goals, I will leverage the capabilities of an existing blockchain, which will be enhanced by a so-called smart contract.

This project is based on my submission for assignment M4 [1] and will therefore by nature be congruent to a certain extent.

Introduction

In order to retain a very particular use-case rather than a generic proposal for a "trust network", I consider the following real-world problem: A new employee joins a company of multiple thousands of colleagues. He does not know anybody except his direct colleagues yet, but has a very specific question, for example about export regulations for a particular product of this company. The "classic" way will be to ask his direct colleague, who may know somebody, who knows somebody, who has an idea about that topic. However, the knowledge of this colleague may be outdated, or he may even have left the company at that time. Additionally, that particular colleague may not provide the optimal answer since, even though he will certainly have some knowledge about that topic, others maybe know better about it. Furthermore, asking oneself through a large organization step-by-step may be a tough and time-consuming challenge. This scenario may then also easily be scaled to global public expertise network.

A centralized capability database may partially solve that issue, but these often come with a main issue: The ratings are either based on a self-estimation or on judgement of the employee's direct supervisor. Both will again lead to poor ratings with at most "local optimums".

So, the idea is that employees - or more generally: peers - rate each other's knowledge rather than being restricted to self or supervisory estimation. Furthermore, the system shall not be based on a static "1-to-5 stars" rating for particular domains, where people will tend to

overestimate their own or others knowledge[1]. Therefore, I chose a *transaction*-based system, in which a peer can rate the knowledge of another peer based on a particular action, for example answering a question or otherwise supporting on a specific topic. As opposed to a pure *scoring*-based system, this will allow more advanced techniques for data mining and competency evaluation as the *reasoning* behind an actual score can in the future be evaluated with more sophisticated algorithms than the one that is developed in this project. This may then also include advanced machine learning algorithms or correlations to other sources such as social media, which are not taken into account in the basic implementation. However, this is what is commonly called *off-chain analysis* and not considered to be part of this project.

This paper will begin with an initial need-finding which includes research on prior work, as well as the evaluation of the basic building blocks of existing work compared to the preliminary planning for this project. This will then be followed by a proposal for the baseline design, which will also include a brief explanation of the prototype implementation for the required smart contract and insights on the actual design process and a first evaluation of the performance metrics for the implementation. In the next chapter "Redesign through Refinements", I will propose major changes to the very first prototype, which aim on improving the resiliency and performance of the smart contract in daily usage. The actual effectiveness of the implementation will be planned and executed in the next two chapters by modelling and simulating a group of users using the system. The project whitepaper will finally be concluded with some additional remarks on what I learned throughout the project and how the PiTrust system could further be extended for future and more advanced use cases.

Initial Need-finding Analysis and Preliminary Evaluation

As briefly described in the previous chapter, the main goal of this work is to propose a system, which objectively consolidates ratings of peers about other peer's knowledge of a particular domain and enable to find experts registered to the network.

Extended research in the cryptocurrency space revealed that with Braintrust [2], a token with a comparable use-case, connecting freelancers to employers, already exists. However, in comparison to the proposed PiTrust token, the profiles are again based on self-estimation while the entire ecosystem is more focussed on the matchmaking, settlement, and monetarization of the liaised assignments while this work more focus on building and storing an objective metric for competencies. As the system proposed shall not only contain a globally acknowledged trust-score for a particular peer (and knowledge-domain), but also the reason for that score.

In principle, this may be implemented in a central database and that may indeed by sufficient for a permissioned use-case such as a company-internal knowledge database. However, this would also be prone to misuse and/or compromise, for example by malicious administrators, and is not applicable to another, more broad use-case, where the knowledge network shall be traceable and evidential for everybody. Furthermore, a central solution will not scale very well with a large number of users and would require additional (decentral) infrastructure.

[1] This effect is certainly heavily influenced by rating systems such as Amazon, where everything below 5 stars (the absolute maximum) is considered to be a complaint.

A decentralized system, such as EigenTrust [3] may generally serve the purpose of a permissionless scenario, i.e., anybody may join and use the system without explicit registration, but it does not allow any traceback, *why* the score of a particular user is especially low or high as any trust-building or destroying action is immediately reflected in the peer's trust score while the reason will not be retained by the network. Also, other work proposed such as [4] and [5] elaborate on how to calculate a trust score, but do not refer to retaining the reasoning behind the score. Retaining a ledger of transactions, which cannot be changed anymore after their commitment, however, is the core functionality of a blockchain such as Bitcoin [6] or Ethereum [7]. The choice of the blockchain to be used will be a major part of the Baseline Design.

In order to consider the implementation to be successful, five aspects have to be sufficiently accommodated, in descending order of importance:

- Smart contract security,
- Rating value,
- Transaction performance,
- Usability,
- PiToken *tokenomics*

In fact the smart contract security has to be considered of the highest importance, as a potential hack on it may render either, all ratings as well as the token itself, completely worthless. However, this will not be the center of this homework as the security analysis of a smart contract would very well exceed the scope of this work.

The rating value, i.e., the objective correctness of the calculated ratings, however, depicts a major part of the design evaluation as this is a cornerstone of the actual idea, even relatively independent of the question, if it is a centralized, decentralized or blockchain based implementation.

Third, the transaction performance takes an important part of the baseline design and later evaluation as there are huge differences in the use of different blockchains as well as how the smart contract itself will be implemented. This will also be a major task within the refinement chapter.

The usability of a blockchain based solution stands and falls with the integration on any form of wallet to be used. As an easiest approach, the user interface is based on classic username/password websites, where the credentials are used to compute the necessary cryptographic keys for the blockchain interface. However, this would a) make password changes complicated and b) not be compliant with the clear goal to have a *decentralized* ledger of transactions. Furthermore, many people will be wanting to keep their cryptographic keys on a software or even hardware wallet such as Ledger or Trezor making such an approach infeasible. So, the approach will be to make use of a well-known software wallets such as Metamask[2], which also allows the usage of additional hardware wallets for more security-demanding users while retaining a simple interface for the "broad masses".

The last factor of success for the implementation of a blockchain based system are the *tokenomics* of the underlying utility token. In a nutshell, this is a mixture of decisions on the token's total supply, how it will be initially distributed across different parties (founders, community fund, public sale etc.) and how the count of tokens will evaluate over time by

[2] https://metamask.io/

creation/mining of additional tokens or *burn* of tokens. In addition to these changes on the total supply, the concept will also comprise an idea, how the tokens shall actually be used, e.g., as a reward for writing honest feedback and, of course, as a reward for providing a high level of knowledge to the users on the PiTrust platform, so that it will be adapted not only as a trading good like other crypto tokens, but as something which can also be *used*.

Baseline Design Method and Measurement Results

The baseline to use a smart contract for implementing PiTrust was already made as a consequence of the requirement for decentral and secure storage of the rating ledger. A purely transaction-storing blockchain, such as Bitcoin, will not be sufficient for the implementation, which shall not only comprise the transaction (i.e., ratings), but also enable calculations out of these, which will discussed in-depth at a later stage. Actually starting with the blockchain to be used seems a little bit "upside-down", but doing so was simply necessary to not get lost between all options as this will be the first smart contract to be implemented by myself. So, I will start with comparison of several smart contract enabled blockchains:

	Transactions/s	Gas fees	SC language
Ethereum	25	18$	Solidity, Vyper
Binance Smart Chain	160	0,33$	Solidity, Vyper
Cardano	250	0,50$	Marlowe, Plutus
Algorand	1000	0,00$	TEAL (Python)
Terra	10000	0,01$	Rust
Solana	50000	0,00$	Rust, C, C++
Polygon	65000	0,01$	Solidity, Vyper
Waves	100	0,00$	Ride
EOS	4000	0,00$	C++
Tezos	40	0,01$	Michelson

Most numbers for Transactions/s taken from [8], all other sources are consolidated in the extended reference list as they would totally blow up the regular reference list. Even though it looks as if Ethereum is the worst decision to program a token for, I deliberately chose to do so. The reasons are the following:

- Ethereum is by far the most anticipated blockchain, right behind Bitcoin, which does not enable any smart contract implementation, therefore the platform is proven and established
- Due to the same reason, the community support for programming smart contracts for Ethereum is by far larger as any other community, making finding help much easier
- Smart contracts written for Ethereum can be migrated easily to work on the BSC or Polygon as well

While I am personally more used to program in Python, which would encourage to use either Vyper or TEAL for programming the required smart contract, I chose to rather use Solidity as the community using Solidity is of magnitudes larger than any other smart contract programming language, leading to much more available documentation including a full-blown IDE called Remix[3]

[3] https://remix-project.org/

In order to ease the very first iteration of the implementation, each *block* of the blockchain only consists of exactly one transaction. For a real-world implementation, one would surely include more transactions into one block in order to reduce the required overhead and network load. The basic design of storage of the transactions is depicted in Figure 1 with Transaction 0 being the "genesis transaction" which did not contain any relevant transaction. Storing this information on the (Ethereum) blockchain is the maybe easiest smart contract to be implemented and serves as the starting point for the solution's implementation.

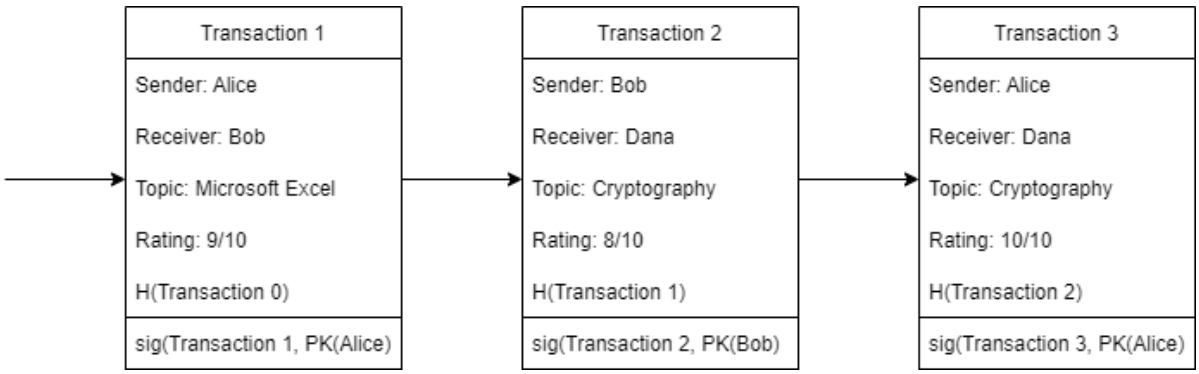

Figure 1: Basic transaction design

Since the PiTrust smart contract is written based on an existing blockchain, it does not even have to deal with the actual signatures and chaining, but can just save the actual rating (Sender, Receiver, Topic, Rating). Or more precise: An updated array of all ratings. Brief additional research revealed that this was also evaluated prior to this work, even though the scope was completely different. [9] This however will lead to an excess cost of storing the entire array after each transaction of $13.82/KB already 1 ½ years ago [10] so that the programmatically easy solution had to be abandoned and going back to the initial idea of storing only the *one* actual transaction/rating on the blockchain and "rolling them up" every now and then to reflect the actual rating of a person's knowledge based on past transactions as it was also proposed in the prior assignment. [10] However, in fact the above mentioned cost prohibit to store any more data on the blockchain as needed at all. Therefore the rollup-mechanism as proposed in the prior assignment has to be revised and beside storing the actual rating, the new rating will be immediately calculated and stored on the blockchain.

As a first estimation, the competency of a user may be calculated by his average rating divided by the overall average rating in that respective area. This can easily be calculated by looking up the current rating of the user r_{old}, weighting his score by the ratings he got so far (n), adding the scaled new rating r and divide the result by n+1:

$$r_{new} = \frac{(r_{old} \times n) + r}{n + 1}$$

New users (and fields) will be added to the ledger upon their first occurrence. This may lead to newly joined users to join the system likely with a maximum score of 10/10, which will not be sustainable for a longer period as this very first rating may also be a self-estimation of the skill (where the "sender" and "receiver" will be the same person). However, this behaviour will be traceable on the blockchain ledger, so there will be an, at least, social incentive for honest behaviour.

The baseline design is still incomplete for two particular reasons, which may not be entirely important in a more or less trusted environment such as a corporate network, but render it unusable in an untrusted, public, environment:

1. The system may be exploited by creating fake accounts to boost (or downvote) an account.
2. Even if issue 1 is solved, the system is still prone to honest but incorrect up- (or down-) rating people without professional reasoning (as stated above; someone who can create a pivot table is not an Excel guru).

Even though newly joined user's ratings do barely contribute to the actual score of another user, just flooding the system with new users and use them to rate (positively or negatively) a particular user may still lead to a significant undesirable effect. In order to prevent this, the *PiTrust token* comes into play. The token acts as kind of currency to *pay* to be able to rate another user. So, the primary idea is, that new users/accounts start without any balance in their wallets and need to earn tokens by *getting* rated (or *buy* them). By granting r-1 tokens to a user receiving a rating *r* (out of 10 points) and burning 1 token[4] of the user's wallet upon his vote. With that approach, it becomes impossible to create sustainable circles of ratings, especially not for *down*rating. Depending on the actual (fiat) price of the token this would make exploiting the system quite costly. Furthermore, this behavior may easily be discovered by analyzing the blockchain and my then be penalized, either socially or algorithmically.

However, why shall anybody be willing to *pay* for their own contribution? In a permissioned system, every user could receive a fixed number of tokes each month (for example 10), each being equivalent to one vote. In a permissionless system however, this will not work as this would even incentivize creating lots of fake accounts to gather "voting power" faster.

In order to prevent "hoarding" of tokens, there shall be a decay implemented, removing a token (including fractions) 1 month after it has been transferred to the wallet, except for those which have been initially bought for fiat money. As an incentive for not letting the gained tokens void but rather spend them for ratings, the number of ratings done by a particular wallet address may additionally be tracked and rewarded by, for example, using it as an additional factor for incoming votes and not only granting the initially proposed $\frac{r-1}{10}$ tokens, an additional $\frac{(r-1)\ x\ \#votes}{\#total\ votes}$ so that using the actual voting power will act as some kind of long-term investment, especially for highly capable experts. Furthermore, it is imaginable to prevent the token to be traded back into fiat currency (or other cryptocurrencies) as it is *designed* to be spent and not as an investment object. However, due to the nature of cryptocurrencies, this approach will likely fail and making buying the token completely uninteresting for any user.

During development it turned out that it is not possible to keep track on-chain, when a token was redeemed due to it fungible nature. However, the decay is very easy to implement on a block chain by a very simple and well-known mechanism: inflation. Even though a stable or even deflationary supply is usually desirable for a cryptocurrency this is not the case for the *PiTrust* token. This implemented by minting an additional 20% of tokens to the so-called funding wallet every year. The function *inflation()*, which checks, if a year is over will simply be called upon every registered rating. This in fact means that the inflation will not be continuous, but marks big steps in the total supply (Figure 2) and may also not happen *exactly* one year after the last step (or the initial deployment), but rather on the next transaction *after* a year has passed by.

[4] The common wording for transferring tokens/coins to a non-usable address, i.e., where the private key is unknown. In the initial implementation, it will be sent to a "funding wallet" owned by the smart contracts creator.

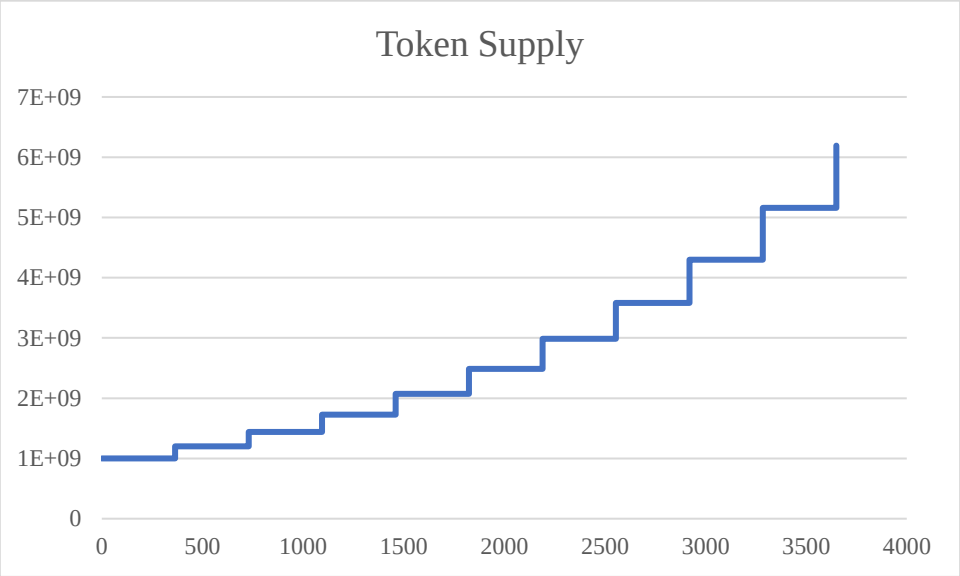

Figure 2: Token Supply for PiTrust token over time (days)

After having implemented the smart contract as described, this will actually be the solution as it was initially proposed in assignment M4, excluding what was meant to be an "additional consideration" (see next chapter) and a user interface. However, the main challenge of this project's part was not the refinement of the actual proposal but rather its real implementation, which came with its own challenges as I personally did not have any prior knowledge of how to program smart contracts resulting in a very steep, but interesting, learning curve.

Redesign through Refinements: ~3 pages

Refinement to ERC20 token

While the basic implementation works quite well in a local simulation, it is lacking one major property: It is not yet adhering to the most important standard for smart contracts on the Ethereum blockchain, ERC-20 [11]. Refactoring the token to be compliant to that standard makes it a) tradeable, b) verifiable and c) migratable to other blockchains. Therefore the following functions and events have been properly implemented (Source: [11]):

```
function name() public view returns (string)
function symbol() public view returns (string)
function decimals() public view returns (uint8)
function totalSupply() public view returns (uint256)
function balanceOf(address _owner) public view returns (uint256 balance)
function transfer(address _to, uint256 _value) public returns (bool success)
function transferFrom(address _from, address _to, uint256 _value) public returns (bool success)
function approve(address _spender, uint256 _value) public returns (bool success)
function allowance(address _owner, address _spender) public view returns (uint256 remaining)
event Transfer(address indexed _from, address indexed _to, uint256 _value)
event Approval(address indexed _owner, address indexed _spender, uint256 _value)
```

Most of the functions are quite self-explanatory (or can be be looked up in the above mentioned source) while events can be considered as callback functions, which can be subscribed to by any internal and external program (e.g. a blockchain explorer) to be notified upon the particular event. The most notable semantic change to the code has been the implementation of the *approve* and *allowance* functions, which are designed to limit the impact of a smart contracts behaviour on an entities wallet. In this case, the approval was important to be given to either the *PiTrust* smart contract for actual usage of the tokens as

well as the possibility to grant this access to cryptocurrency exchanges such as Uniswap[5] to enable the onramping with the token, i.e. enabling people to trade other tokens such as USDT for PiTrust tokens as well as enabling any interaction with standardized off-chain systems such as Web3 frontends.

In order to maintain compliance to the standard, even for functions which will not be explicitly implemented, a well-known framework called OpenZeppelin[6] is available and will be used. Furthermore it will support the development of a basic frontend at a later stage. This also boiled down the code of the very first prototype from 113 lines to 86 while providing all necessary interfaces and additional security features according to the OpenZeppelin website.

Optimization on rating weights

However, the basic implementation will not solve the issue of the "average" case where many people upvote mediocre knowledge. One way to solve this issue will be to weight people's vote either on their overall voting behaviour or based on their own rating on a particular topic assuming that a person who is rated high by a person who himself has a good rating on a topic (i.e., is an expert) will be more trustworthy than a person who is appreciated by persons without any knowledge on the respective area. The formular out of the original homework

$$R_A = \frac{1}{11(n+1)} \sum_B R_A^B (R_B + 1)$$

with R_A being the rating of person A, R_A^B being the rating of person B for person A and n being the total number of persons, who voted within a particular area of knowledge at all, shall be applied. *n* is the count of *previous* votes. The rating R_B needs to be scaled up by one so that also votes of users without knowledge in an area or who have newly joined the system (i.e., having an own rating of zero), are also considered, even though heavily scaled down compared to already known experts. This approach is comparable to the EigenTrust system [3] even though in the current implementation, weights are assigned statically and will not reflect increasing or decreasing weights of voters over time reflecting their expertise to change over time.

The formula however cannot be calculated in a closed form, because given a time *t*, the information about any ratings R_A^B prior to t are not available anymore. Nevertheless, the same result can be achieved by calculating

$$R_A^t = \frac{(R_A^{t-1} \times n_A) + (R_A^B \times (R_B + 1))}{11\ (n_A + 1)}$$

with a R_A^{t-1} being the rating of an entity prior to the newest vote. The overall design is depicted in Figure 3. It must furthermore be mentioned that the processing of one transaction does not correspond to a block on the Ethereum/Polygon blockchain, but the *results* of the process or, more likely, multiple subsequent processes will be stored in one block.

[5] https://uniswap.org/
[6] https://openzeppelin.com/

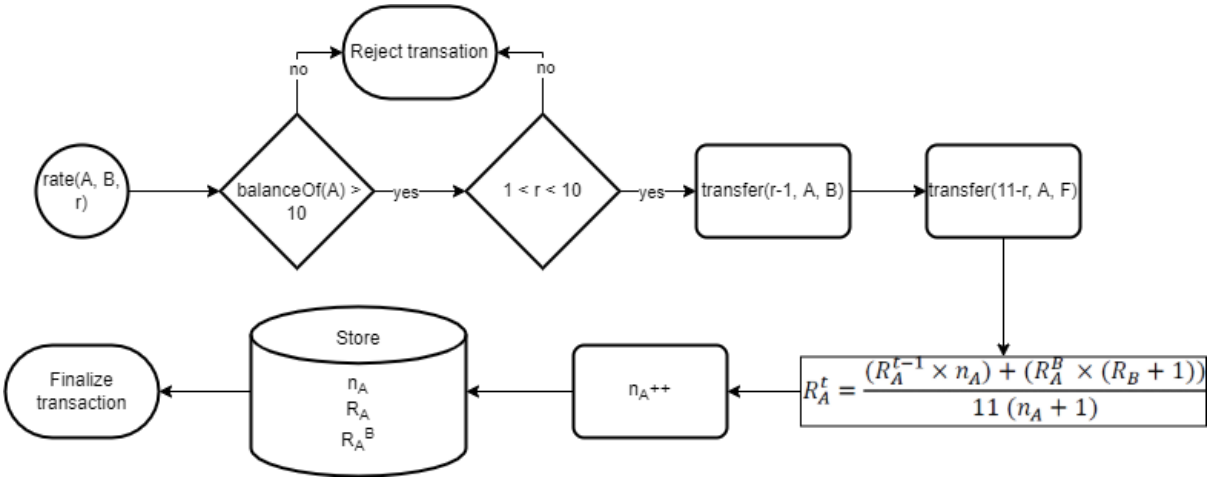

Figure 3: Process of a rating transaction

Frontend Development

Furthermore, the system is yet lacking a frontend. While this project's focus lies more on the smart contract functionality, at least a basic frontend which is able to interact with the smart contract will be required. This includes the following tasks:

- Create a rating for a known address and a particular field of expertise
- Query the current ratings (all fields) of a known address
- Get the top 10 experts for a given field of expertise
- Query the balance of a wallet address
- Add 1000 PiTrust tokens to a given address (for test purposes only, will be removed)

TODO: Frontend development

Evaluation Plan: ~2 pages

TBD

Tokenomics
Web Interface
Blockchain deployment (Testnet)
Gas Fees
Tradability
Blockchain deployment (Mainnet)

Evaluation Execution and Results: ~3 pages

TBD

Concluding Remarks: ~1 page

TBD

References

[1] P. Wissmann, *CS 6675 Assignment M4,* 2022.

[2] T. B. T. Foundation, „Braintrust: The Decentralized Talent Network," September 2021. [Online]. Available: https://www.usebraintrust.com/whitepaper. [Zugriff am 3 March 2022].

[3] M. T. S. H. G.-M. Sepandar D. Kamvar, „The Eigentrust algorithm for reputation management in P2P networks," in *Proceedings of the 12th international conference on World Wide Web,* New York, NY, United States, 2003.

[4] C. J. Zhao Yuhong, „A P2P trust model based on trust factor and feedback aggregation," in *3rd International Conference on Electronic Information Technology and Computer Engineering (EITCE),* 2019, pp. 214-219.

[5] B. B. Y. L. P. A. Yuhui Zhong, „A Computational Dynamic Trust Model for User Authorization," *IEEE Transactions on Dependable and Secure Computing,* Bd. 12, Nr. 1, pp. 1-15, 2015.

[6] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System,* 2008.

[7] V. Buterin, *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform,* 2014.

[8] Aleph Zero Blog, „What Is The Fastest Blockchain And Why? Analysis of 43 Blockchains," 4 January 2021. [Online]. Available: https://alephzero.org/blog/what-is-the-fastest-blockchain-and-why-analysis-of-43-blockchains/. [Zugriff am 30 March 2023].

[9] C. M. B. M. G. Gamze Gürsoy, „Using Ethereum blockchain to store and query pharmacogenomics data via smart contracts," *BMC Medical Genomics,* Bd. 13, Nr. 74, 2020.

[10] N. Feuerstein, „StackExchange: What is the cost to store 1KB, 10KB, 100KB worth of data into the ethereum blockchain?," 23 July 2020. [Online]. Available: https://ethereum.stackexchange.com/questions/872/what-is-the-cost-to-store-1kb-10kb-100kb-worth-of-data-into-the-ethereum-block. [Zugriff am 05 April 2022].

[11] P. W. a. more, „Ethereum.org, ERC-20 Standard," 03 December 2021. [Online]. Available: https://ethereum.org/en/developers/docs/standards/tokens/erc-20/. [Zugriff am 07 April 2022].

[12] T.-W. U. B. Z. G. M. L. Nguyen B. Truong, „Strengthening the Blockchain-Based Internet of Value with Trust," in *IEEE International Conference on Communications (ICC),* Kansas City, MO, USA, 2018.