OraclesLink: An architecture for secure oracle usage

Benedikt Berger*, Stefan Huber*, Simon Pfeifhofer§
*University of applied sciences Kufstein
Kufstein, Austria
Email : benedikt.berger@fh-kufstein.ac.at, stefan.huber@fh-kufstein.ac.at
§Distributed Ledger
Innsbruck, Austria
Email : simon.pfeifhofer@dlteam.io

***Abstract*—Smart contracts encode critical application logic for realizing digital agreements in a tamper-proof form. Blockchains guarantee that smart contracts cannot be altered after the first deployment and that the execution is strictly followed. Smart contracts can only operate on data available on-chain. Oracles are bridging the gap between on-chain and off-chain data. Oracles introduce a wide range of security risks, which were already exploited in publicly known hacks. In this paper OraclesLink is proposed, which is a secure and developer-friendly architecture for using oracles within smart contracts. The goal of the architecture is to eliminate single points of failure and single sources of truth through distribution. In order to demonstrate feasibility, a proof-of-concept implementation is provided.**

***Index Terms*—Blockchain, Oracles, Ethereum, Smart Contracts**

I. Introduction

Blockchain technology has gained increased attention over the last years, with its adoption and usage predicted to grow significantly in the nearby future. A major pitfall of blockchain networks is their inability to communicate in a standardized way with the world outside their own network environment. Therefore connecting on-chain data with off-chain data is a serious problem that must be solved before blockchain systems can be widely implemented for their proposed use cases.

Blockchain applications are implemented with smart contracts. Smart contracts are executed on-chain and can operate only on the managed on-chain data in the context of transactions. Smart contracts, at least on the Ethereum blockchain, have no native means to access off-chain data sources. However, an off-chain component can act as a bridge between the on-chain environment and transfer external data on the blockchain. Such a component is commonly called an oracle. The problem describing how those oracles can best fulfil the necessary tasks has become known as the oracle problem.

Oracles introduce a wide range of security risks, as important properties of blockchains, i.e. the auditability and provenance of data, cannot be applied to off-chain data sources. Publicly known hacks, such as the attacks on trading platforms bZx[1] and Synthetix[2] were caused by oracle attacks. The main reason for the success of the attacks was a single point of failure within the smart contract and oracle implementations. This allowed the attackers to target a specific component which had critical influence over the implemented oracle solution to deliver sensitive price data.

In this paper we propose OraclesLink, an architecture which provides a secure and developer-friendly way to apply an oracle design pattern in smart contract development. Additionally, a proof-of-concept implementation is provided for an exemplary use case. With the proposed approach, security risks such as the ones mentioned above could be mitigated. As a smart contract developer, several aspects of an oracle implementation can be influenced to eliminate single points of failure and single sources of truth. The proposed architecture aims to distribute the involved parties in the on-chaining of external data. To attain this objective, it relies on multiple oracle service providers which each deliver data from multiple off-chain data sources.

This paper is organized as follows: In Section II a short introduction to the concept of oracles in blockchains is given. Section III gives an overview of the related research. The proposed architecture is described in Section IV and the prototypical implementation, on the Ethereum blockchain, is described in Section V. Furthermore the architecture is evaluated in Section VI. A discussion including limitations of the proposed architecture is given in Section VII. Finally, the paper is concluded in Section VIII.

II. Background

Smart contracts encode critical application logic for realizing digital agreements in a tamper-proof form. The alteration of the smart contract logic and the interference of the smart contract execution is secured by the strong guarantees a blockchain offers. These security guarantees restrict the smart contract to operate only on data which is available on-chain.

It must be noted that for many useful applications, access to off-chain data is an important requirement for a smart contract. Therefore the concept of oracles was introduced. Oracles are bridging the gap between on-chain and off-chain data.

Fig. 1 shows a typical interaction sequence between a blockchain and an external data source mediated by an oracle. A smart contract is initializing (1) the data flow by requesting external data from an oracle. The oracle queries (2) the respective data source and receives (3) the response containing the requested data. Finally, the oracle puts the external data on the blockchain (4) and thus the data is available to the smart contract.

[1]https://bit.ly/2NycZYM
[2]https://bit.ly/2A4voJL

978-1-7281-8370-1/20/$31.00 ©2020 IEEE

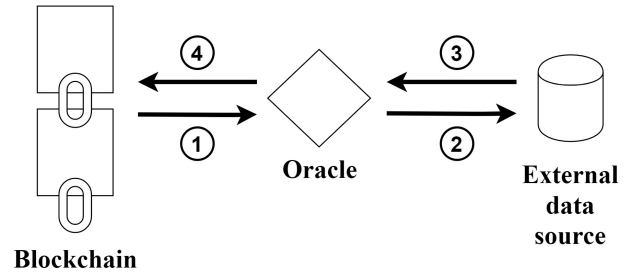

Fig. 1. On-chain to off-chain communication flow with an Oracle.

The responsibility for reliably and securely providing the external data into the tamper-proof blockchain environment relies on the oracle architecture. Heiss et al. [1] introduced key requirements for a holistic evaluation of such oracle architectures. The requirements are summarized in the following:

- Truthfulness: Oracles and data providers must be attributable and also accountable, i.e. depending on their actions they can be penalized or rewarded.
- Safety: It must be guaranteed that the obtained data originates from the specified data source (authenticity) and that it has not been changed (integrity).
- Liveness: It must be guaranteed that the data source is available at any time and that there is unrestricted access to the data.

These key requirements were carefully considered while designing the OraclesLink architecture and also while implementing the respective prototype.

III. Related Work

Several approaches have been proposed, which operate on the oracle problem from a game-theoretical point of view. Astraea [2] is an oracle design that decides the truth value of binary propositions. The mechanism is based on different user groups which vote on the truthfulness of the respective propositions. Users are motivated by economic incentives to provide correct answers. Shintaku [3] provides an extension to Astraea with the aim of a complete end-to-end oracle decentralization. Another protocol was proposed by Merlini et al. [4] following a similar approach. Voters are presented with antithetic questions, the convergence or divergence of the answers are used by the oracle to determine the correctness of the answers.

In [5] a system called Town Crier was introduced. Town Crier offers a Trusted Execution Environment, by using Intel's Software Guard Extensions. The system collects data from HTTPS-enabled websites and publishes the data on the Ethereum blockchain as input for respective smart contracts.

TLSNotary [6] allows a client to prove to a third party that certain data has been obtained from a server within a TLS session. The third party must trust the server's public key. E.g., the oracle provider Provable[3] uses TLSNotary as an authenticity proof for its services.

Wang et al. [7] propose an oracle implementation scheme that supports multiple websites as data sources, which are crawled and aggregated by the oracle service.

An empirical analysis of smart contracts deployed on the Ethereum blockchain was done in [8]. The study showed that oracles were mainly used by games and in financial smart contracts. Another study [9] compared the capabilities of available code analysis tools to detect common vulnerabilities in smart contracts. The study revealed that no tool can identify vulnerabilities in relation to oracles, although smart contracts involving oracles have the highest severity level.

Al-Breiki et al. [10] compare existing oracle architectures and explain how decentralized oracle networks could resolve the issue of having a single point of failure.

IV. Architecture

This section elaborates on the proposed OraclesLink architecture. The architecture aims to provide a reusable solution for smart contract developers which allows connecting to off-chain data sources while ensuring the previously mentioned requirements truthfulness, safety and liveness. In order to accomplish this goal, OraclesLink builds on the fundamental concept of distribution. As a result of this distributed architecture approach there arise several parameters to configure the level of distribution. Note that more distribution not only increases security but also costs. Consequently, the architecture comes with the ability to flexibly configure those parameters to fine-tune the level of security for each specific use case - if necessary.

The OraclesLink architecture is composed of various components which are involved in the process:

- Consumer Contract: Smart contract which implements the business logic for the use case and also uses OraclesLink to integrate external data.
- Oracles Linker: Acts as the bridge between the Consumer Contract and the other components. The Oracles Linker coordinates the workflow. A smart contract developer is using the interface of this component to integrate oracles.
- Random Oracles Provider: Provides the non-deterministic selection of oracles.
- Oracles Store: Store for the available oracles with the associated metadata (such as the ranking of oracles).
- Oracle: The third-party Oracle nodes, which accept and fulfil requests.

The interaction between all the involved components is depicted in Fig. 2. An interaction is always initiated by the Consumer Contract. The contract uses the Oracles Linker to build and send OraclesLinks - requests to multiple external data sources processed by multiple oracles. To accomplish this, the Oracles Linker selects oracles at random through the Random Oracles Provider, sends out requests to the selected oracles, and processes their responses. The Random

[3]https://provable.xyz

Oracles Provider selects the oracles from the Oracles Store in a non-deterministic way. In the following the components Oracles Linker, Random Oracles Provider and Oracle Store are described in more detail.

A. Oracles Linker

The Oracles Linker provides the interface, that consuming smart contracts interact with, to create OraclesLinks. This includes the parametrization of security requirements set by the consumer contract. Furthermore, it organizes the selection of oracles and the communication with the selected oracles. Finally, it also includes the logic to receive oracle responses and to combine the responses into a single result once the respective security requirements are met.

Security requirements are based on oracle levels. Oracles are assigned a certain level which aims to best categorize oracles by how well they fulfil key oracle characteristics [1]. This level is a ranking based on reputation-, validation- and trust score following the proposed trust score for peer-to-peer networks by Li et al. [11]. Nodes rate each other after a task has been executed. Similar reputation and validation services were also described in the Chainlink whitepaper [12].

For the oracle selection process, it must be ensured that requests are distributed fairly across all available oracles. Guaranteeing security and reliability for requests has the highest priority. Although, low-ranked oracles should be allowed to participate in the selection process, as low-ranked oracles can increase their ranking only by participation. It must be assured that low-ranked oracles do not receive enough power to confound results.

In general, the selection process must aim to be able to handle the highest possible number of parties with malicious intent without confounding the results. A measurement to attain this objective is that the majority of selected oracles get assigned a level that ensures a high degree of fulfilment of key oracle characteristics.

B. Random Oracles Provider

The Random Oracles Provider provides a set of random, duplicate-free numbers which uniquely identify oracles that support the desired operation and meet given security requirements (encoded by the oracle level). This process depends on random number generation, a functionality which is currently not available within the deterministic nature of blockchain environments. Thus, an off-chain service must be used to provide true randomness.

C. Oracles Store

Oracles that are included in the selection process are stored on-chain through the Oracles Store smart contract. This contract must use a fair, transparent, and distributed process for the storing, removing or updating of oracles, based on the concept of distributed governance. We propose a round-based process, which allows whitelisted addresses to vote for executing updates in the Oracles Store contract.

The distributed governance is limited to whitelisted addresses, which are allowed to vote for changes. Changes are applied once a certain threshold is reached, which defines the minimum amount of voters required. As part of the distributed governance, voters can propose changes to the state of the Oracles Store such as adding or removing a whitelisted address or altering the minimum voters' threshold. Note that the first whitelisted address is the address deploying the contract.

Whitelisted addresses can furthermore propose changes to the on-chain oracle data set. For this, the whitelisted addresses run an off-chain component which aggregates oracle data and calculates their score to assign each oracle to a certain level.

This process is concluded in rounds as shown in Fig. 3. Round-based processing allows to ensure a certain frequency in which the whitelisted off-chain components provide information. Off-chain components can propose to start a round e.g. because they observed changes that must be reported or because a given amount of time has passed since the last round. Rounds are started when the minimum voters' threshold is reached. Off-chain components listen to an event, which is fired when a round is started. When this happens, they retrieve the oracle data from the third party sources, e.g. listing services, and report this up-to-date information in the form of change proposals to the Oracles Store. This guarantees that all whitelisted addresses provide up-to-date data and that all reported changes work on the same set of data obtained from external sources. Once all change proposals have been reported, whitelisted addresses can suggest to end a round, which again depends on the minimum voters' threshold. When a round is ended, the Oracles Store coordinates the proposed changes and applies the ones which fulfil the minimum voters' threshold.

V. Prototype

This section outlines the implementation of the exploratory prototype based on the architecture described in the preceding section IV. The Consumer Contract implements an exemplary showcase for simulating inquiry handling for frost insurance. Frost insurance is commonly used in agriculture when crops are at risk for temperatures falling below freezing, especially relevant for fruit and wine growers [4]. The Consumer Contract fetches the current air temperature through the implemented prototype of OraclesLink. Automatic insurance payout could be triggered if the air temperature is below freezing.

The prototype is focused on the Ethereum blockchain network and uses Chainlink as the oracle service. Chainlink operates and provides a decentralized oracle network service.

The smart contracts are coded in Solidity, the primary language of choice for Ethereum [13]. The tools of the Truffle Suite[5] are used for local smart contract development. The Truffle Suite provides, amongst other functionalities, a local Ethereum blockchain environment, testing capabilities and support for interaction with deployed smart contracts through a command line interface or a script-based configuration. The implemented smart contracts furthermore rely on well established library implementations for common features, such as

[4]https://www.climate.axa/products/frost

[5]https://www.trufflesuite.com

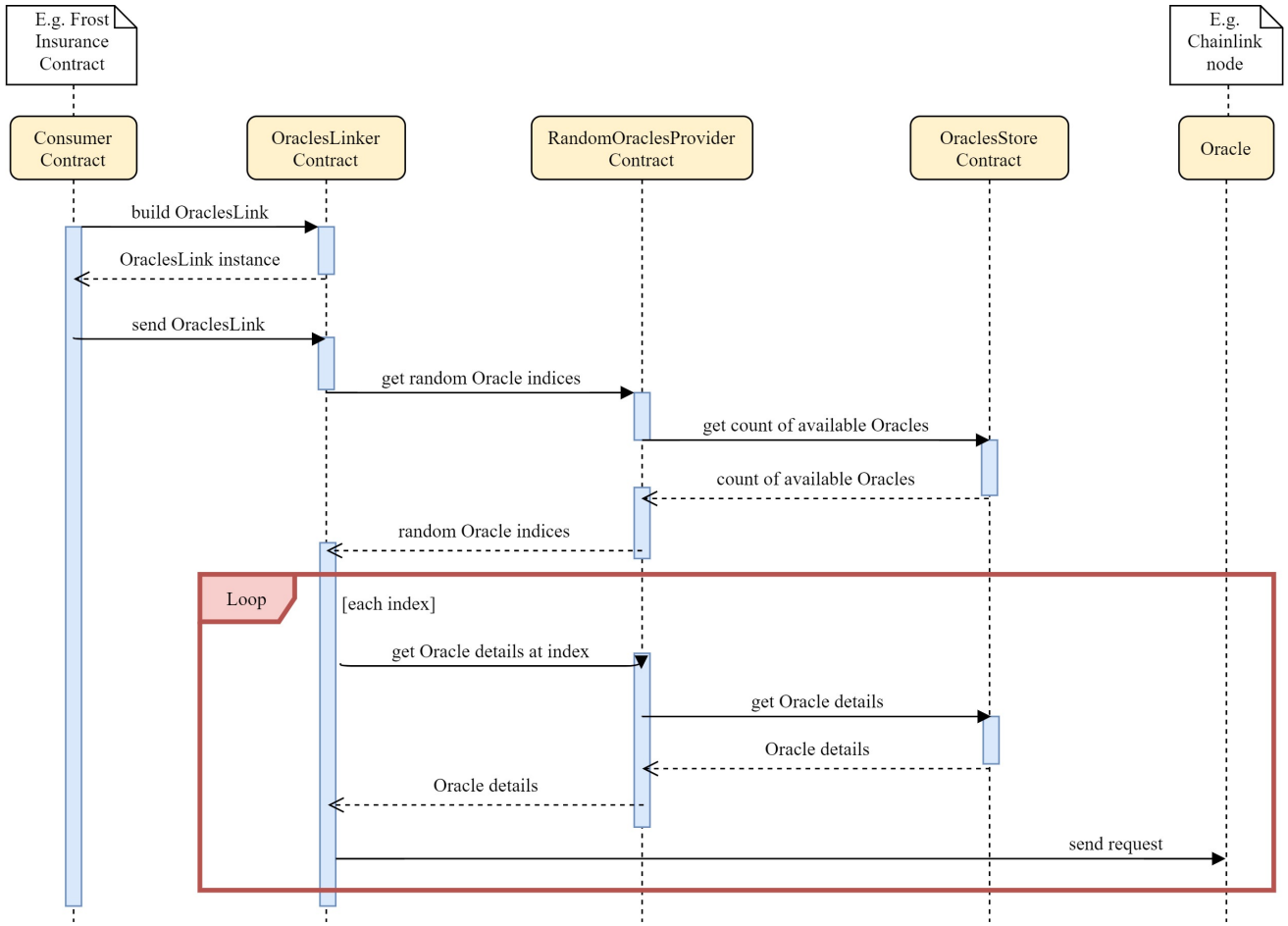

Fig. 2. Sequence diagram showing the involvement of the on-chain components for the handling of a distributed oracle request (OraclesLink).

the ownership of a contract. Used third-party implementations include the Ownable and SafeMath library by OpenZeppelin[6] as well as the Median and an adapted version of the Whitelisting implementations offered by Chainlink.

Finally, all smart contracts are deployed to the Ropsten Ethereum test network with the help of Truffle and the Ethereum node as a service provider infura.io[7]. The code of the deployed smart contracts is verified on the Etherscan Ropsten blockchain explorer.

For a detailed inspection, the source code is publicly available within a git-repository[8]. The README includes links to the deployment of the application, the smart contracts, and oracle explorers.

A. Consumer Contract implementation

The Consumer Contract demonstrates the usage of the implemented solution. The required code in a consuming smart contract is shown in Fig. 4. First, an OraclesLink object is built and enriched with information regarding the external sources. Security requirements are defined and finally, the OraclesLink is sent out. The aggregated answer is received in the fulfil callback method. Due to the dependency on the Chainlink oracle network, the consumer smart contract must own enough LINK tokens to pay the rewards for the oracle nodes. LINK is a token used to pay for services within the Chainlink network.

The showcase includes a web application for a generic interaction with the smart contract. To interact with the Ethereum blockchain, the web application relies on the web3.js library [14]. The web application allows any Ethereum account with some Ropsten test Ether to trigger the on-chain logic which creates an OraclesLink. The web application is deployed in the form of a decentralized application on the InterPlanetary File system [15].

B. Oracles Linker Implementation

The code for the Oracles Linker fulfils the main objective to provide a simple interface which allows creating distributed off-chain requests. It implements the security requirements and the aggregation of answers by calculating the median on the numeric values once those requirements are met. Strongly deviating responses from individual oracles do not affect the aggregation method median as much as e.g. an arithmetic mean. Furthermore, it allows the consumer contract to tweak security requirements.

[6]https://github.com/OpenZeppelin/openzeppelin-contracts
[7]https://infura.io
[8]https://github.com/bergben/OraclesLinker

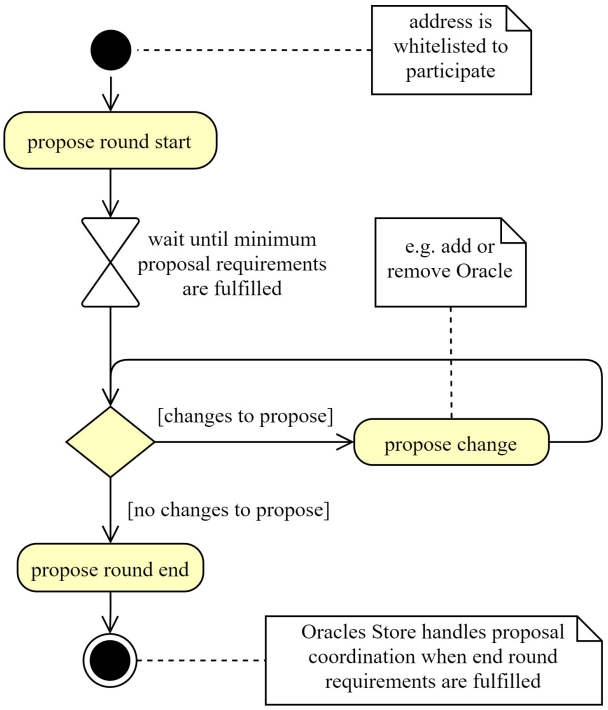

Fig. 3. Activity diagram displaying the change proposal process in rounds through whitelisted address for the Oracles Store smart contract.

```
contract ConsumerContract is OraclesLinker {
    // […] code ommitted
    // Method creating an OraclesLink
    function createInquiry() public returns
    (bytes32 oraclesLinkId) {
        // […] code omitted: ensure LINK balance
        OraclesLinkInt256.Request memory oraclesLink
         = buildOraclesLinkInt256(5, 4);
        oraclesLink.addSource(
            "http://api.weather.example/data...",
            "main.temp", 100
        );
        // […] code omitted: repeat oraclesLink.addSource(…)
        // for each of the 5 external data sources
        oraclesLink.setSecurityLevel(
            OraclesLink.SecurityLevel.Default
        );
        oraclesLinkId = sendOraclesLinkInt256(oraclesLink);
        emit InquiryCreated(oraclesLinkId, msg.sender);
        return oraclesLinkId;
    }
    // Callback method triggered with aggregated answer
    function fulfillOraclesLinkInt256(
        bytes32 _oraclesLinkId,
        int256 _answer
    ) internal override {
        emit InquiryFulfilled(_oraclesLinkId, _answer);
        if (_answer < 0) {
            // valid inquiry -> trigger payout
        }
    }
}
```

Fig. 4. Usage of the implemented solution within a consumer smart contract.

A limitation of the Chainlink infrastructure requires an adjustment to the aggregation process. Chainlink oracle nodes set a default maximum gas consumption limit for their callback transaction of 500 thousand units[9]. As a result, an out of gas exception is encountered when all the required aggregation processes are done within the final transaction that fulfils all OraclesLink requirements. Instead, it is necessary to split this computation and aggregate the answers for each external source as soon as enough answers for this source have been received.

C. Random Oracle Provider implementation

True randomness through an off-chain component is not implemented for the scope of this proof-of-concept. An on-chain version for random number calculation through hashing is used instead. It uses global variables, parameters and a seed hash combined with a nonce to generate a hash which is transformed into a random number. Note that this should be replaced with a true randomness provider in a production setting.

D. Oracles Store Implementation

The implementation of the Oracles Store includes the distributed, round-based change proposal process as suggested in chapter IV-C. While a round is active, whitelisted participants can propose changes such as adding and removing oracles or jobs. Chainlink requests are handled as jobs. Each job works sequentially through a list of tasks, defined in a job specification. This prototype focuses on the job specification that allows fetching a numeric value from an external data source, which is fed back to the on-chain environment in the form of a Solidity int256 data type value.

The distributed governance functionality, which allows whitelisted addresses to vote for alterations to the smart contract settings, is substituted with a simpler version to limit the scope of the implementation. The implemented version instead allows the owner of the smart contract to tweak settings such as changing the threshold for the minimum proposal supporters required. In a real world scenario, the distributed governance should be strictly implemented.

To simulate how oracle data is fed to the Oracles Store, a script is executed within the environment provided by Truffle. This script simulates round-based proposals with one participating entity and feeds a list of oracles to the contract. This list is populated manually for the scope of this proof-of-concept work. Oracle data is retrieved from a Chainlink listing service[10] for the Ropsten Ethereum network. The oracle nodes are categorized into three levels, based on the number of job runs that a given oracle has executed. A more sophisticated ranking algorithm is part of future work.

VI. Evaluation

The architecture and the respective implementation distributes requests to fairly at random selected oracles for each external source defined by the consuming contract. Thus, it fulfils the goal to eliminate both a single point of failure and

[9]https://docs.chain.link/docs/configuration-variables
[10]https://market.link

a single source of truth [10] in the application of the oracle design pattern. Consequently, the security hacks mentioned in Section I could have been circumvented by the proposed architecture. For evaluating the trustworthiness of the proposed architecture the stated oracle requirements truthfulness, safety and liveness [1] were selected as evaluation criteria and discussed in the following.

A. Truthfulness

In the proposed architecture the selection of oracles is based on non-deterministic randomness combined with a given oracle level based on a ranking. The non-deterministic selection reduces the ability of an oracle to time a malicious answer. Furthermore, oracles which provide provably altered or malicious data can be penalized by downgrading their ranking or by a complete elimination from the Oracles Store.

Additionally, the answers from the oracles are aggregated. The aggregation is not generalizable and needs to be selected according to the respective use case. In the given prototype the weather data is gathered from 5 different sources and the median is used as an aggregation function. The aggregation is intended to eliminate malicious answers from oracles.

B. Safety

The component that implements the necessary logic to fulfil safety are essentially the oracles. OraclesLink can therefore only indirectly influence the safety criteria through the oracle selection process (i.e., selecting oracles based on the preferred safety requirement of the use case). The prototype uses the decentralized oracle network provided by Chainlink. While there are centralized oracle services available which fulfil safety, solving this for a decentralized oracle network is a complex task. As a result, approaches which are meant to solve safety for Chainlink oracles, such as trusted computation, are not in place yet. Note that as soon as Chainlink enables such functionality, it is automatically included in the implemented prototype.

C. Liveness

Distribution, given that the network has a reasonable number of participants, provides a reliable oracle service with scalable guarantees of no downtime. This leads to a scalable fulfilment of the key characteristic liveness through the number of participants.

VII. Discussion

In this paper an architecture for using distributed oracles was proposed. The feasibility was demonstrated by implementing a prototype for an exemplary frost insurance use case. The proposed architecture manages the complexity of ensuring security in the usage of oracles for smart contract developers. This eliminates the necessity for reimplementations to solve the same reoccurring problem. Providing a design pattern or architecture for a common problem has proven to be effective in other areas relevant for security, such as identity management. In the following several limitations and improvements to the proposed architecture are discussed.

For a real world scenario, the implemented on-chain random number generator should be replaced with a true randomness provider like the Chainlink Verifiable Random Function[11]. Note, that a solution using the Chainlink Verifiable Random Function would have to rely on manually whitelisted oracle nodes. Thus, it is advisable to use an on-chain provider for random number generation with sufficient randomness e.g. available in Ethereum 2.0[12].

The oracle ranking procedure used to populate the Oracles Store with oracles data is deliberately kept as simple in the implemented prototype. It relies on one third party listing service to provide the data, conclusively introducing a single source of truth. The prototype does enable a more sophisticated data writing process based on distribution. Future work should aim to implement an off-chain component which ranks oracles by aggregating data and criteria from multiple listing services. This off-chain component can then be run by whitelisted participants. This process might require payment for the participating parties. Thus, an integration of the mentioned off-chain component for oracle ranking into the infrastructure of existing distributed oracle networks could be desirable.

An improvement to the on-chaining of data managed by the Oracles Store would be possible as follows: Instead of allowing whitelisted parties to directly propose changes, whitelisted addresses could be limited to only propose a refresh of the data, e.g. by watching the listing services. The Oracles Store contract itself then retrieves the data from the external source through the solution proposed in the prototype by triggering an OraclesLink. The Oracles Store then compares the received data with the stored on-chain data and applies necessary changes. Note that this is currently not possible due to technical limitations. Firstly, cost and scaling prevent heavy computation such as comparing a whole result set to on-chain data. Secondly, a strict limitation of Solidity is, that calls to on-chain methods cannot contain lengthy JSON data. Other blockchain environments and future scaling might support the described solution.

An unexpected limitation is encountered regarding the availability of Chainlink oracle nodes listed for the Ropsten test network on the used listing service. Many oracle nodes listed are not actually available to fulfil requests. To prevent such a situation a proper incentive model must be in place. The automatic oracle selection implemented in the prototype increases the incentive to run an oracle node in a decentralized oracle network because it enables the fair distribution of rewards for the fulfilment of requests. Conclusively, it also increases the degree of distribution of the network, which is a valuable metric for a decentralized network.

Scaling is a problem that blockchain technology is trying to solve. This problem is also encountered in the implemented prototype. The transaction that creates an OraclesLink for the showcase e.g. consumes about 5 million gas. The current limit

[11] https://docs.chain.link/docs/chainlink-vrf

[12] https://blocking.net/7155/ethereum-2-0-randomness

on Ropsten is roughly 7 million gas units, the limit on the Ethereum main network was raised recently to 12.5 million. It should be noted that optimizing for cost and scaling was not a focal point in this work.

A major problem of current blockchain technology and decentralized oracles is the handling of confidential data. The URLs for the sources in the exemplary Consumer Contract must contain the keys for authentication with the external data sources in cleartext.

A custom solution to the above mentioned limitations on scaling and handling of confidential data is possible by using trusted computation environments. The combination of blockchain technology with trusted computation has shown to be promising [16] and is an avenue for future research in the context of the proposed architecture.

Another threat to the requirement safety which is yet to be solved in the usage of decentralized oracles in general is Freeloading. Freeloading describes the process where one oracle observes the answers given to the on-chain environment by other oracle nodes. The cheating oracle could then provide the same answer as the observed answer given by another oracle. This way, the cheating oracle could potentially avoid cost resulting from requests to a third party data source [12]. The proposed Oracle Store contract involves the functionality to remove oracles or degrade their ranking if Freeloading is identified.

VIII. Conclusion

Connecting the on-chain environment of blockchain networks to off-chain components is critical for the integration of blockchain solutions in current digital infrastructures. Oracles act as a bridge between those two environments. The usage of oracles introduces unsafe components to the otherwise secure blockchain network. Elimination of both a single point of failure and a single source of truth can be used to avoid potential attack vectors. This objective can be attained through distribution.

An architecture is proposed and evaluated through the implementation of a prototype. The architecture enables smart contract developers to apply the oracle design pattern in a way, that includes distribution across all involved components. Although the prototype fulfils the desired goals, future work and advancement of blockchain technology as a whole is required for a better fit into a real-world scenario. Issues which need to be tackled are scaling, cost and handling of confidential data.

Future work should develop a ranking algorithm that can be used for determining the oracle level for the selection process. Future implementations should also aim to support multiple decentralized oracle networks. Furthermore, solutions should be explored to resolve the scaling and privacy issues, e.g. by off-chaining certain functionality, such as storing credentials for accessing external data sources.

References

[1] J. Heiss, J. Eberhardt, and S. Tai, "From oracles to trustworthy data on-chaining systems," in *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2019, pp. 496–503.

[2] J. Adler, R. Berryhill, A. Veneris, Z. Poulos, N. Veira, and A. Kastania, "Astraea: A decentralized blockchain oracle," 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)*. IEEE, 2018, pp. 1145–1152.

[3] R. Kamiya, "Shintaku: An end-to-end-decentralized general-purpose blockchain oracle system," 2018. [Online]. Available: https://gitlab.com/shintaku-group/paper/raw/master/shintaku.pdf

[4] M. Merlini, N. Veira, R. Berryhill, and A. Veneris, "On public decentralized ledger oracles via a paired-question protocol," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2019, pp. 337–344.

[5] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," Cryptology ePrint Archive, Report 2016/168, 2016, https://eprint.iacr.org/2016/168.

[6] TLSNotary, "Tlsnotary - a mechanism for independentlyaudited https sessions." [Online]. Available: https://tlsnotary.org/TLSNotary.pdf

[7] S. Wang, H. Lu, X. Sun, Y. Yuan, and F. Wang, "A novel blockchain oracle implementation scheme based on application specific knowledge engines," in *2019 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, 2019, pp. 258–262.

[8] M. Bartoletti and L. Pompianu, "An empirical analysis of smart contracts: platforms, applications, and design patterns," in *International conference on financial cryptography and data security*. Springer, 2017, pp. 494–509.

[9] A. Mense and M. Flatscher, "Security vulnerabilities in ethereum smart contracts," in *Proceedings of the 20th International Conference on Information Integration and Web-Based Applications & Services*, ser. iiWAS2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 375–380. [Online]. Available: https://doi.org/10.1145/3282373.3282419

[10] H. Al-Breiki, M. H. U. Rehman, K. Salah, and D. Svetinovic, "Trustworthy blockchain oracles: Review, comparison, and open research challenges," *IEEE Access*, vol. 8, pp. 85 675–85 685, 2020.

[11] L. Y.-J. D. Ya-Fei, "Research on trust mechanism for peer-to-peer network [j]," *Chinese Journal of Computers*, vol. 3, 2010.

[12] S. Ellis, "A decentralized oracle network steve ellis, ari juels, and sergey nazarov," 2017. [Online]. Available: https://link.smartcontract.com/whitepaper

[13] R. M. Parizi, A. Dehghantanha *et al.*, "Smart contract programming languages on blockchains: an empirical evaluation of usability and security," in *International Conference on Blockchain*. Springer, 2018, pp. 75–91.

[14] W.-M. Lee, *Using the web3.js APIs*. Berkeley, CA: Apress, 2019, pp. 169–198. [Online]. Available: https://doi.org/10.1007/978-1-4842-5086-0_8

[15] J. Benet, "IPFS - Content Addressed, Versioned, P2P File System," *arXiv e-prints*, p. arXiv:1407.3561, Jul. 2014.

[16] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *2019 IEEE European Symposium on Security and Privacy (EuroS P)*, 2019, pp. 185–200.