FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Trustable oracles towards trustable blockchains

Pedro Duarte da Costa

WORKING VERSION

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Filipe Figueiredo Correia

Second Supervisor: Hugo Sereno Ferreira

June 20, 2019

Trustable oracles towards trustable blockchains

Pedro Duarte da Costa

Mestrado Integrado em Engenharia Informática e Computação

June 20, 2019

Abstract

Resumo

Acknowledgements

Pedro Duarte da Costa

"If I have seen further it is by standing on the shoulders of Giants."

Isaac Newton

Contents

List of Figures

LIST OF FIGURES

List of Tables

Abbreviations

SLR	Systematic Literature Review

Chapter 1

Introduction

Once more, a technological revolution sparked in a not-yet-ready world. Just as the Internet invention brought us closer together and opened an unlimited virtual world of possibilities so does blockchain. The technology is still in its early development days and many different proposals are being worked on to improve its performance and scalability. Akin to the dotcom boom, a plethora of blockchain projects live on more expectations than results but ultimately blockchain could resolve the Internet's failed promise. To understand what is blockchain and why it is necessary we need to comprehend the social background around the time of its release. The Internet promised of a peer-to-peer connected world, however, financial incentives and technological challenges led to a centralized and non-privacy advocated virtual world. The increasing general concern regarding the privacy of personal information and the meddling of third parties in everyday online actions allied with the financial crash of 2008 lead to a new technological and social breakthrough.

Satoshi Nakamoto's introduced Bitcoin, in 2009 [Nak09], and revolutionized money and currency, setting the first example of a digital asset which has no backing or intrinsic value and more importantly no centralized issuer or controller. In order to require no third party to verify each transaction and prevent double-spending, he introduced a distributed ledger mechanism now known as Blockchain.

1.1 Blockchain

Blockchain is a tool for distributed consensus, in a byzantine fault-tolerant approach, without requiring to trust in centralized parties. In this ledger, transactions are recorded in an ongoing chain, creating an immutable public record that cannot be changed without redoing the proof-of-work. Anyone can become a node and leave and rejoin the network. Having incentives to work on the CPU intensive proof-of-work, extending the chain, and so, for as long as the majority of nodes are trustworthy, the longest and honest chain will thrive. The proof-of-work used on

Bitcoin is HashCash, proposed in 1997 by Adam Back, is a cryptographic hash-based proof-of-work algorithm that requires a selectable amount of work to compute, but the proof can be verified efficiently. Nodes can easily verify that a block is valid and that some effort was put in its creation. The proof-of-work difficulty can increase and decrease depending on the network size and capability, creating on average a block every 10 minutes, like a heartbeat.

In simpler terms, transactions are grouped in blocks and for each block there is a mathematical challenge (proof-of-work) which requires time and computational resources to be solved, guaranteeing that some effort is put into solving the challenge and therefore making it extremely hard to quickly manufacture false blocks. Each block has a hash, a signature, of the previous block linking all blocks in a single chain. Nodes always work on the longest chain, so as long as the majority of the nodes are honest and work in building correct blocks, which means they don't have double entries and transactions are legitimate, the biggest chain will grow and remain a trusted and distributed ledger.

Leveraging Blockchain, Bitcoin requires no personal information to exchange value, anyone can join the network and no central authority is needed. This opens an unlimited world of new scenarios for the use of blockchain.

1.2 Smart Contracts

In 2015, Ethereum [Gav14] was launched as an alternative protocol for building decentralized applications called smart contracts. Introduced as applications that run on the blockchain, smart contracts are self-verifying, self-executing and immutable contracts whose terms are directly written in lines of code which persist on the blockchain, promising to replace real-world contracts. Contracts are the building blocks of our identity, economy and society. They enforce agreements between multiple parties and ensure trust in the compliance of the rules of the agreement but traditional contracts lack automation and decentralization. Smart Contracts provide the ability to execute tamper-proof digital agreements, which are considered highly secure and highly reliable.

Smart contracts have a wide range of use cases. For example, they can be used in Supply Chains and Logistics. Smart contracts allow tracking product movement from the factory to the store shelves. Each intermediary signs a step of the contract which then the final consumer can analyse and have the guarantee of the origin of the product.

1.3 The Smart Contract Connectivity Problem

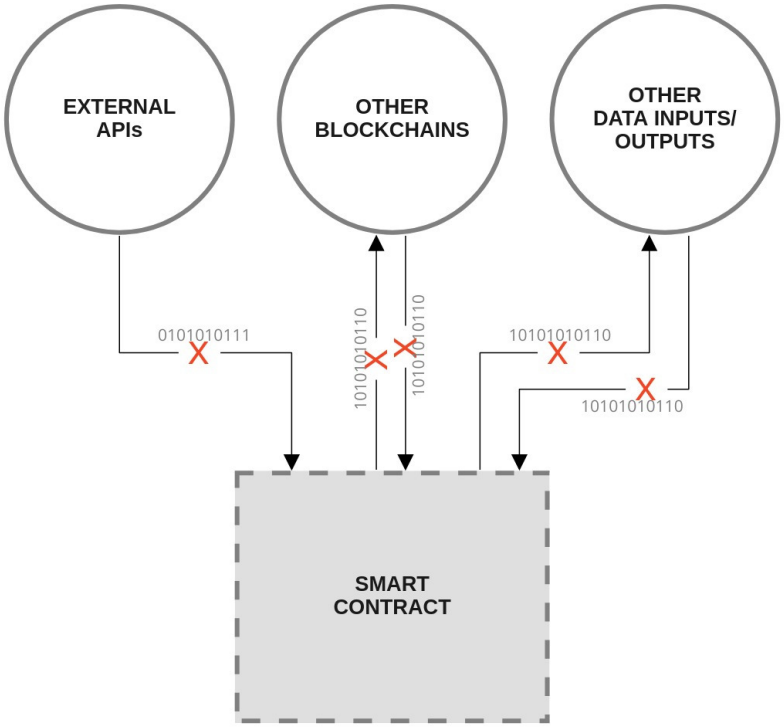

Figure 1.1: Smart contract connectivity problem.

The Ethereum blockchain is designed to be entirely deterministic [Gav14], meaning that if someone downloads the whole network history and replays it they should always end up with the same state. Bearing this in mind, smart contracts cannot directly query URLs for certain information since everyone must be able to independently validate the outcome of running a given contract making it impossible to guarantee that everyone would retrieve the same information since the internet is non-deterministic and changes over time. Determinism is necessary so that nodes can come to a consensus. In order for smart contracts to gain traction, they need access information of the real world, outside of the blockchain. For example, the current price of the US dollar. However smart contracts cannot directly query the internet for information due to the non-deterministic nature of the internet. Meaning that the information retrieved at some point in time cannot be entrusted to be available or equal in another point in the future, which may result in different states when validating smart contracts by querying the internet in different moments. Oracles solve the non-deterministic problem, of querying the internet, by inputting external information on the blockchain through a transaction making sure that the blockchain contains all the information required to verify itself.

1.4 Smart Contracts Space and Computation Limits

Another problem for smart contracts is performing long and costly operations in terms of computation and space. Several platforms are implementing smart contracts, also called DAPPs, Distributed Applications, namely Ethereum and EOS [Blo18], among others.

On the Ethereum platform, smart contracts pay "Gas" to run. "Gas" is a unit that measures the amount of computation effort that certain operations require to execute. "Gas" is basically the fees paid to the network in order to execute an operation. Therefore, the longer the application runs the more "Gas" the smart contract as to pay.

EOS, on the opposite of Ethereum, works on an ownership model whereby users own and are entitled to the use of resources in proportion to their stake. Basically, instead of paying transaction fees, the owner who holds N tokens is entitled to N*k transactions. While Ethereum rents out computational power on the network, EOS gives ownership of the resources in accordance with the amount of EOS held. The mentioned resources are RAM, corresponding to the used state on the network, CPU measuring the average consumption of computing resources and NET which measures used bandwidth. With increasing prices of EOS tokens, staking these resources becomes very costly.

All in all, either for users of smart contracts or the teams deploying them, keeping smart contracts efficient and performing a non-costly operation is the key. Nonetheless, sometimes applications require costly operations and outsourcing them to an oracle outside of the blockchain is the answer.

1.5 Oracles as a Solution

The solution to the smart contract connectivity problem and to outsourcing computation from the blockchain is the use of a secure blockchain middle-ware, mentioned before as, an oracle. Oracles can query data from APIs, data feeds, other blockchains or perform their own calculations and input that data on the smart contract. This way the blockchain has all the necessary information to verify the result of running a smart contract, and will always produce the same result, independently of the point in time in which that verification runs.

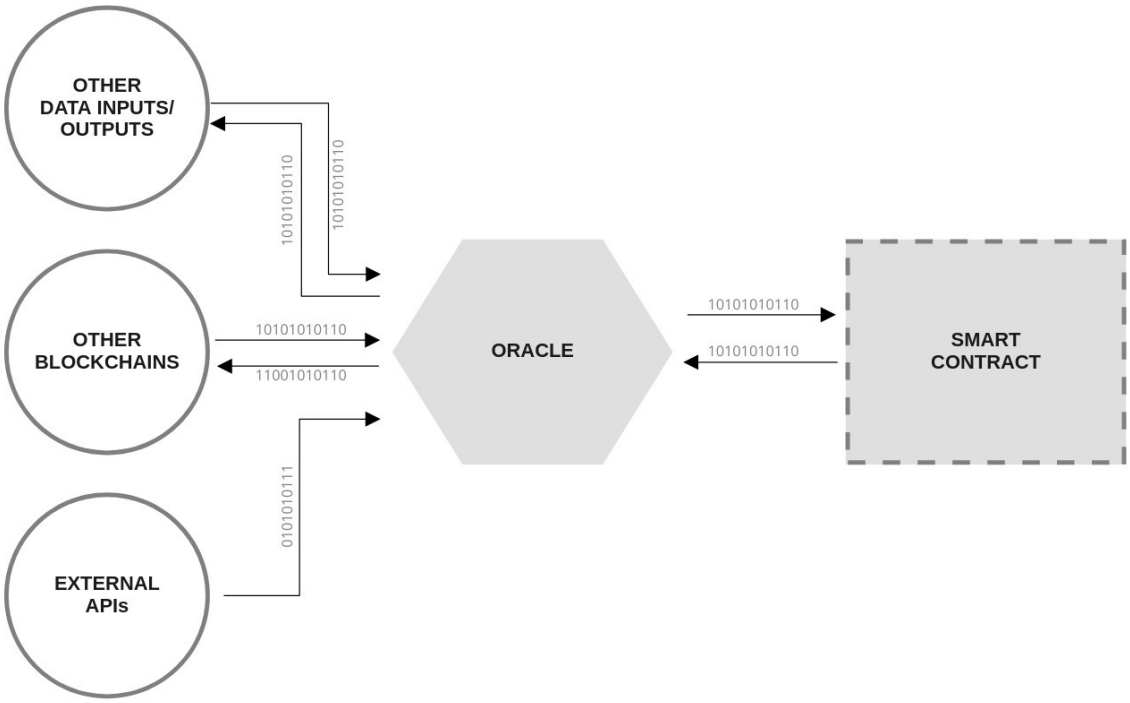

Figure 1.2: Oracle integration.

1.6 Authenticity Proofs

Authenticity proofs, are cryptographic proofs commonly used by oracles in order to prove their honest behaviour. By generating some cryptographic document that can later be used to prove that the oracle actually saw the information that it relayed or computed. In Chapter 3 I take a closer look to the existing proofs.

1.7 Motivation and Objectives

The research hereby exposed was proposed by Takai, a blockchain start-up born in Porto, Portugal with the purpose to be the first blockchain open innovation platform. Sponsored by Bright Pixel, an innovation hub and venture investment house, which supports promising startups in their early years. Taikai is building a platform that connects talent and entrepreneurs with the challenges of the corporate players, through the power of the sharing economy and blockchain trust.

The growing interest in blockchain technology and especially in the potential of Smart Contracts together with the lack of research on trustable oracles creates a gap in the general adoption of blockchain by business and governments.

The proposed objectives for this work are as follows:

- Defining the requirements for oracle trust;
- Understanding blockchain oracles behaviour;

- In-depth analysis of existing Authenticity Proofs;
- Define several oracle architectures in terms of trust;
- Oracle implementation and analysis.

1.8 Document Structure

Additionally to the Introduction, this document contains seven more chapters.

In Chapter 2, the author analyse the state-of-the-art in terms of blockchain oracles. Initially by performing a systematic literature review, to capture existing academic work on the field and later I expose some more work detailed by companies and individuals in their projects' whitepapers.

In Chapter 3, the author deep-dive on existing authenticity proofs, detailing how they work, what they can achieve and their limitations.

Chapter 4 exposes the problem statement underlying this dissertation and exposes a set of forces that are meant to be achieved.

Chapter 5 looks at different oracle architectures and their different approaches for achieving trust, assigning a context to be solved by each one and the resulting context of its application.

Chapter 6 describes the implementation of the last architecture, creating a simple and effective boilerplate that can be leveraged for a wide range of oracle usage scenarios.

In Chapter 7 the author validates each architecture against the forces described in the problem statement, as well as the implementation in its ability to achieve its goal.

Finally in Chapter 8, the other concludes on its contributions, details possible future work and describes some of the challenges the author faced during the dissertation.

Chapter 2

Blockchain Oracles

The topic of blockchain oracles is still unexplored territory mostly investigated by start-up companies and individuals thriving to solve a new problem. Therefore, research related to oracles is scarcely found on peer-reviewed publications but, nonetheless, is invaluable in such an early phase of the technology. Consequently, the state of the art cannot be complete without reviewing the work developed by the academia and also by start-ups, enterprises, governments and individuals.

2.1 Literature Review

To get an overview of academic research a systematic literature review was performed. It's main components and finding are described in this section.

A literature review allows scholars not to step on each other's shoes but to climb on each other's shoulders, meaning, not duplicating existing research and find the gaps and strive to discover something new. To conduct a non-biased, methodical and reproducible review, to the extent that a human can, it is necessary to clarify and identify at the beginning of the research its methodology, what are the data sources and what is the selection criteria.

The goal of this literature review is to get a sense of the corpus of existing works on the topic of blockchain oracles, and the directions and extent to which previous research has rendered significant results.

2.1.1 Research Questions

First of all and to guide the focus of the research, the following research questions were defined:

- RQ1: What kind of blockchain oracles have been proposed?
- RQ2: What are the research trends on blockchain oracles?

RQ1, analyses the scope of existing blockchain oracles. The methodologies and technologies used, so as to understand how the oracle problem is tackled.

RQ2, tries to identify the direction that is proving to be the most effective. Analysing past solutions that never made it into production and solutions currently adopted.

2.1.2 Search Strategy and Data-sources

Figure 2.1, depicts the predefined review strategy used in order to achieve such a goal and maintain unbiased, transparent and reproducible research. These steps are inspired on the guidelines for performing a systematic review by Kitchenham et al., 2007 [KKC07].

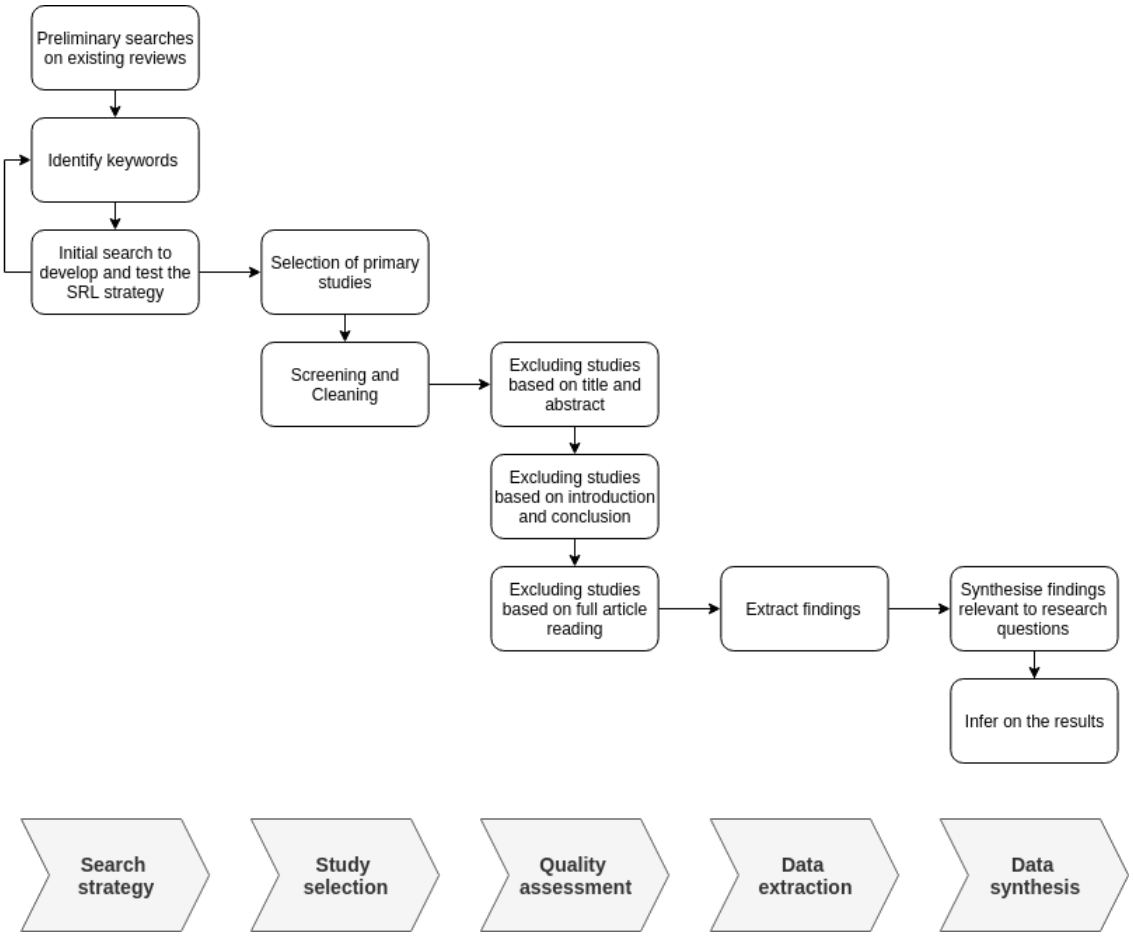

Figure 2.1: Review strategy.

The first step, **Search Strategy**, compromises a preliminary search on several databases trying to optimize the query that best fits the research questions. After identifying the set of keywords that best describe the problem a full query is built and tested. Finally, we define and describe the steps to of the systematic review.

Once a satisfactory query is achieved, we proceed to the next step, **Study selection**, here we aggregate the studies from all databases and in the *Screening and cleaning* phase we remove papers written in other languages or duplicated.

Next, in the **Quality assessment** step we iteratively exclude papers that do not answers to any of the research questions. Initially analysing only the title, and alter the abstract and so on until a full read of the article seems worth it to take conclusions and respond to que research queries.

This leads to the **Data extraction** step, in which we take and summarize the findings after reading each paper.

So that later, in the **Data synthesis** step, we can summarize all the findings, infer some conclusions and answer the research questions.

Having defined the strategy for the systematic review and after testing some keywords on several databases, the author selected the following four electronic databases to query for relevant information:

- ACM Digital Library
- IEEE Xplore
- Scopus
- Google Scholar

The defined search query was the following:

```
(("blockchain" OR "block chain" OR "block-chain") AND ("oracles" OR "oracle" OR "middle-ware" OR "middleware" OR "middle ware" OR "datafeed" OR "data feed" OR "data-feed"))
```

This search query was used to comprise all the possible ways of referring to blockchain and oracles. Some scholars have investigated the oracle issue by simply calling them a middleware or data-feed since oracles can either be used as an intermediary that relays data or as the source of the data.

The search was performed on the 5th of February 2019 and revealed the results presented in Table 2.1.

Database	**Filters**	**Results**
ACM Digital Library	Title, abstract and keywords	34
IEEE Xplore	Title, abstract and index terms	24
Scopus	Title, abstract and keywords	57
Google Scholar	Title	8
Total		**123**

Table 2.1: Number of results and applied filters per database

Since the concept of smart contracts on the blockchain was only introduced in 2015, with the introduction of the Ethereum blockchain, only results after 2015 were considered, also, all

duplicated papers were removed. Analysing the initial search results per year, in Figure 2.2, we can infer the growing popularity of oracle-related academic research. The year 2019 only comprises work done in the month of January since the search was performed at the beginning of February.

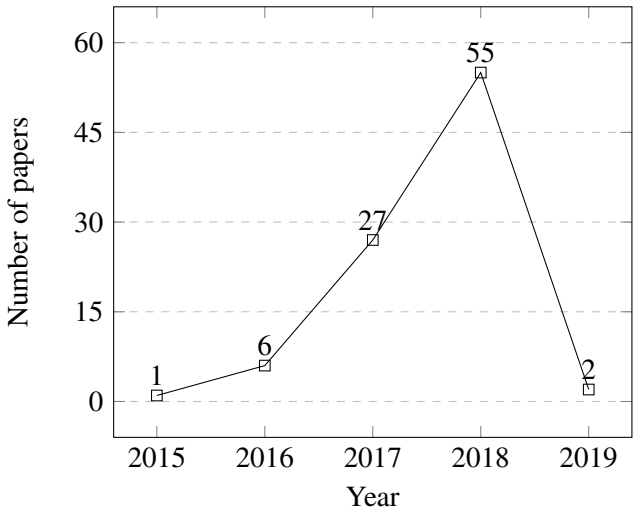

Figure 2.2: Resulting papers from search distributed per year

2.1.3 Study Selection and Quality Assessment

The process of exclusion is depicted in Figure 2.3 and all the information regarding the papers and in which phase they were excluded is transparently presented in Appendix A.

The study selection process initially started with a pool of 123 papers from the previously stated online databases. As described on Figure 2.1, the selection and quality assessment compromised four stages:

- Stage 1: Screening and cleaning duplicated articles or articles that were not in English.
- Stage 2: Exclusion by carefully reading the title but most importantly the abstract. After this stage, only 13 of the 91 non-duplicated papers were either describing specific trustable oracle implementations or mentioning the use of oracles.
- Stage 3: Analysing the introduction and conclusions in order to remove papers which do not describe an implementation of a trustable oracle or a protocol to overcome the trust in oracles.
- Stage 4: Full article reading to assess if the final bucket of articles answers the research questions.

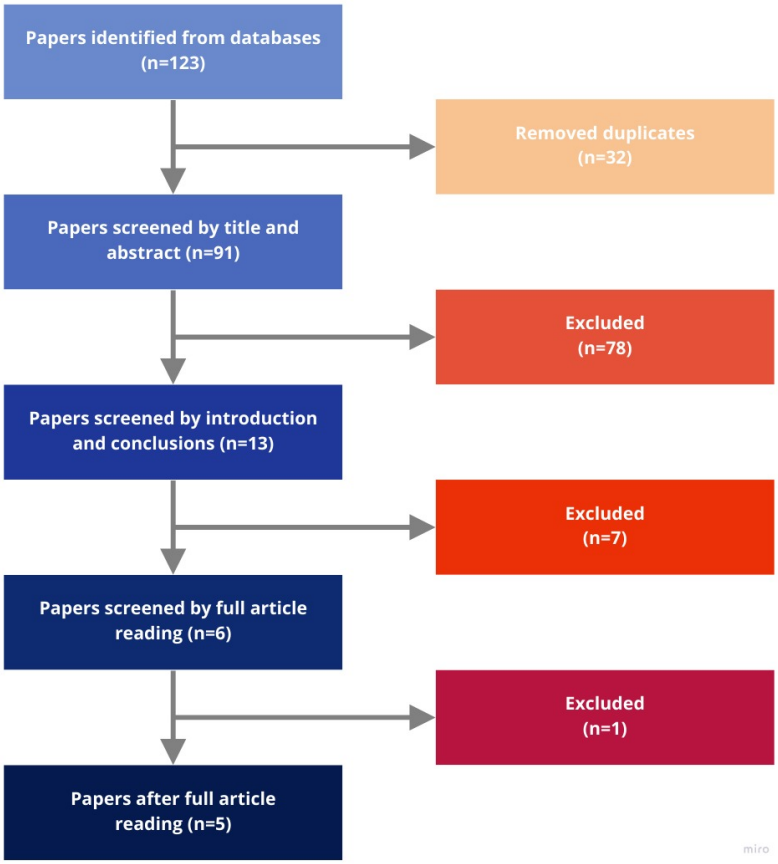

Figure 2.3: Screening stages.

2.1.4 Data extraction and Data Synthesis

The following process resulted in three articles and two theses that approach varying problems in implementing and guaranteeing trust in oracles.

Town Crier (TC) [ZCC+16], leverages trusted hardware, specifically Intel SGX[1], to scrape HTTPS-enabled websites and serve source-authenticated data to smart contracts. TC architecture, depicted on Figure 2.4 [2], involves a TC contract on the blockchain that receives requests from a client contract and communicates those request to a TC server which runs a SGX-protected process to retrieves an answer from a data source through an HTTPS connection. TEE prevent even the operating system of the server from peeking into the enclave or modifying its behavior, while use of TLS prevents tampering or eavesdropping on communications on the network.

[1]Intel Corporation. Intel® Software Guard Extensions SDK. https://software.intel.com/en-us/sgx-sdk, 2019
[2]Image taken from: https://town-crier.readthedocs.io/en/latest/how_tc_works.html

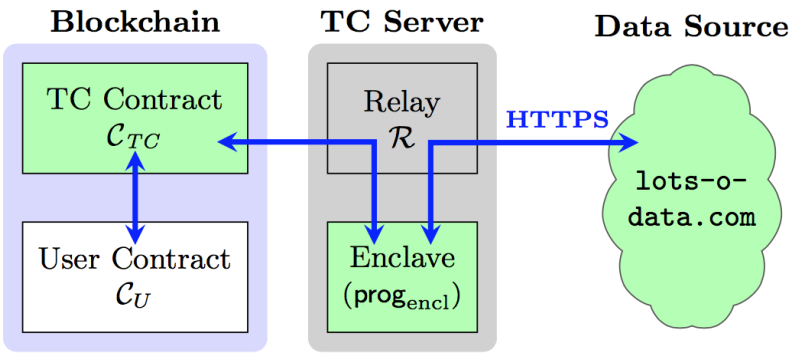

Figure 2.4: Town crier high level view.

Astraea [ABV+18a], depicted on Figure 2.5 [3], proposes a decentralized oracle network with submitters, voters and certifiers, in which voters play a low-risk game and certifies a high-risk game with associated resources. Using an monetary incentive structure as a means to keep the players honest.

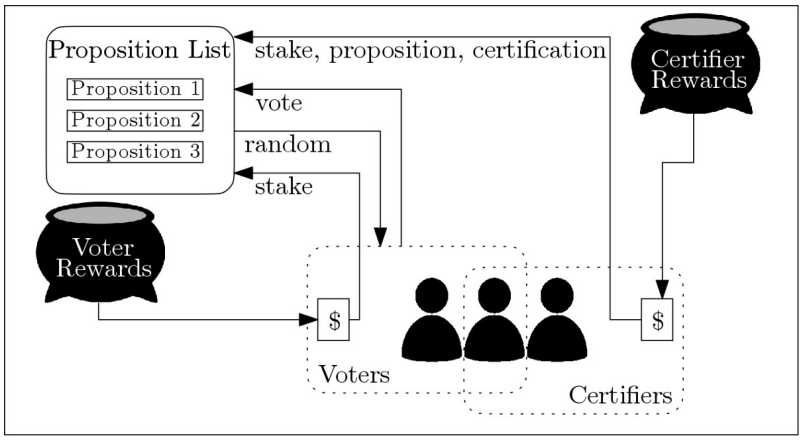

Figure 2.5: High-level overview of Astraea's architecture.

Gilroy Gordon [Gor17] proposes a protocol for oracle sensor data authenticity and integrity to IoT devices network with low computational resources. Using sets of public and private keys to authenticate that the oracle sensor data actually was originated by that oracle even if the information needs to pass by several oracles before being consumed by the application.

Francisco Monroy [Mon18] defines a gambling protocol based on incentives and assuming that every entity involved has the objective to maximize their profit. The protocol overcomes the trust in a single Oracle by polling a network of 7 oracles from a large network of available oracles, they will then stake their money on a specific bet and only receive their investment back if the

[3] Image taken from: https://blockchain.ieee.org/technicalbriefs/march-2019/astraea-a-decentralized-blockchain-oracle

majority of the oracles vote in the same winner. Creating, therefore, incentives for Oracle good behaviour.

J. Eberhardt [EH18] does not propose a specific method but analyses existing solutions and defines a systematic classification for existing trustable off-chain computation oracles. The authors identify the following off-chain computation oracles approaches:

- *Verifiable off-chain Computation*, a technique where a prover executes a computation and then publishes the result including a cryptographic proof attesting the computation's correctness to the blockchain. An on-chain verifier then verifies the proof and persists the result in case of success. Identified existing solutions are zkSNARKs, Bulletproofs and zkSTARKs. zkSNARKs require a setup phase which is more expensive than naive execution. After the setup, however, proof size and verification complexity are extremely small and independent of circuit complexity. This amortization makes zkSNARKs especially efficient for computations executed repeatedly, which is usually the case for off-chain state transitions. While zkSTARKs and Bulletproofs require no setup, proof size and verification complexity grow with circuit complexity, which limits applicability.

- *Secure Multiparty Computation*, SMPCs, enable a set of nodes to compute functions on secret data in a way that none of the nodes ever has access to the data in its entirety. Identifies Enigma [Tam18], which proposes a privacy-preserving decentralized computation platform based on multiple parties where a blockchain stores a publicly verifiable audit trail. However, current SMPC protocols add too much overhead for them to be practical. Hence, Enigma now relies on Trusted Execution Environments.

- *Enclave-based Computation*, EbC, relying on Trusted Execution Environments (TEE) to execute computations off-chain. Identified existing solutions are Enigma and Ekiden [CZK+18] which present two different implementations of EbCs. In Enigma, programs can either be executed on-chain or in enclaves that are distributed across a separate off-chain network. An Enigma-specific scripting language allows developers to mark objects as private and hence, enforce off-chain computation. In contrast to Enigma, Ekiden does not allow on-chain computation but instead, the blockchain is solely used as persistent state storage.

- *Incentive-driven Off-chain Computation*, IOC, relies on incentive mechanisms applied to motivate off-chain computation and guarantee computational correctness. IOCs inherit two critical design issues: (1) keep verifiers motivated to validate solutions and (2) reduce computational effort for the on-chain judge. The paper identifies TrueBit [TR17], as the first IOC implementation, proposing solutions for both challenges. As verifiers would stop validating if solvers only published correct solutions, TrueBit enforces solvers to provide erroneous solutions from time to time and offers a reward to the verifiers for finding them.

2.2 Commercial Products and Projects

This search, on the contrary of the systematic one explained before, cannot be described in a systematic way, since the source of the information is spread on whitepapers and startup companies' documentation pages which cannot be guaranteed to be available and consulted on a systematic way.

To search for existing commercial products and projects, Google, a search engine and Medium, a platform for blog posting used widely by developers and the start-up community, were used as a means to find new projects or solutions for the oracle trust problem. Using these two tools a lot of projects were found trying to solve the oracle trust problem and are solely documented on white-papers or on the companies' website documentation page. This kind of literature cannot be found in peer-reviewed databases, but can nonetheless provide invaluable information and is therefore worth being analysed.

The results of this search revealed a wide range of projects and protocols with varying degrees of decentralization or authenticity. A short explanation of each will be detailed here:

- Oraclize.it [Ora], provides Authenticity Proofs for the data it fetches guaranteeing that the original data-source is genuine and untampered and can even make use of several data sources in order the gather trustable data, but its centralized model does not guarantee an always available service.

- ChainLink[EJN17], describes a decentralized network of oracles that can query multiple sources in order to avoid dependency of a sole oracle which can be prone to fail and also to gather knowledge from multiple sources to obtain a more reliable result. ChainLink is also considering implementing, in the future, authenticity proofs and make use of trusted hardware, as of now it requires users to trust in the ChainLink nodes to behave correctly.

- SchellingCoin [Vit14] protocol incentivizes a decentralized network of oracles to perform computation by rewarding participants who submit results that are closest to the median of all submitted results in a commit-reveal process.

- TrueBit [TR17], introduces a system of solvers and verifiers. Solvers are compensated for performing computation and verifiers are compensated for detecting errors in solutions submitted by solvers.

2.3 Summary

Summing up, this research highlighted two main types of oracles. The first is **Data-Carrier oracles**, whose main purpose is relaying query results from a trusted data source to a smart contract. The second is **Computation Oracles**, which not only relay query results but also perform the relevant computation themselves. Computation oracles can be used as building blocks to construct off-chain computation markets. A summary of the results is described in Table 2.2.

Name	Type	Distributed Network	Achieves trust through
Town Crier	Hardware-based	No	Trusted hardware signed attestations
Astraea	Consensus-based	Yes	Network with submitters, voters and certifier
[Gor17]	Software-based	Yes	Sets of public and private keys
[Mon18]	Consensus-based	Yes	Gambling protocol based on incentives
TrueBit	Consensus-based	Yes	System of solvers and verifiers
Oraclize.it	Software-based	No	TLSNotary, Android Proof
ChainLink	Consensus-based / Software-based	Yes	Query multiple sources
SchellingCoin	Consensus-based	Yes	Incentive based

Table 2.2: Summary of oracle projects/research.

Summing up, this research highlighted three main types of oracles. The first is **Software-based oracles**, which try to prove their honest behaviour through the use of software-based authenticity proofs. These, mostly take advantage of some features of TLS to prove that the data they are relaying is the actually provided data. The second type is **Hardware-based oracles**. These leverage specific hardware, TEE, to securely separate the environment running the oracle code from the operating system and other applications to achieve higher guarantees on untampered code execution. They may even provide authenticity proofs regarding that the query actually came from a legit TEE. Lastly, **Consensus-based oracles**, which require a network of peers working together to achieve higher redundancy, having several peers querying the data and even in some cases peers performing the role of the verifier. This last approach largely depends on the existence of such a network and requires the use of monetary incentives to keep the networking running.

Table 2.2, summarises the found existing projects and answers the first research question 2.1.1.

2.4 Conclusions

Two main conclusions arise from both academic and non-academic research, and answer the second research question 2.1.1.

First of all, there is a clear lack of academic research on the topic of creating trustable oracles. This is mostly likely due to the specificity of the problem and that blockchain related technology is usually paved by start ups and enthusiasts and not yet addressed in universities curricular plans.

Secondly, even though the main research on trustable oracles is being pursued by startups or sole developers all the existing projects seem to be blockchain specific or in very early phases and not yet ready to be generally adopted.

Chapter 3

Authenticity Proofs

In this chapter, the author takes a deep dive into existing authenticity mechanisms, in terms of their applicability and limitations.

In the context of today's Web, we are accustomed to trusting that a certain website or data is originated from the expected source due to the general adoption of the HTTPS protocol. An extension of the HTTP protocol which creates an encrypted and authenticated channel between the client and the web-server providing the requested information. Then it becomes a matter of whether we trust the source or not, but no doubts are raised as for the channel through which we received it.

Unfortunately, in the context of blockchain, the most used, available and trusted protocols do not have a direct way of communicating with HTTPS enabled services and therefore obtain authenticated data. This creates the need for a trusted service to input that information, but trusting in a third-party service requires it to provide irrefutable proof of its honesty.

Oracles, currently resort to two main techniques to prove their honesty. (1) Authenticity proofs, which is a software or hardware generated cryptographic proof during or after an execution that can later be used to prove the integrity and honesty of the execution or of the provided data. (2) Trusted Execution Environments (TEE) which add another layer of security by isolating the application code from the environment in which it ran, and may also provide cryptographic proof of their honest behaviour.

3.1 Trusted Execution Environment (TEE)

A Trusted Execution Environment is a secure computational environment that is strongly isolated from the main operating system. It provides application isolation, integrity and memory confidentiality. Sensitive data is stored, processed and protected from the main operating system or network. This isolation is accomplished through software and hardware-enforced mechanism. TEE runs a small operating system which exposes a minimal interface to the running application

and therefore reduces the attack surface. Advanced TEE embeds unique identities that allow to verify the device authenticity and can be used to generate proofs of the device honest execution.

Examples of TEEs are Intel Software Guard Extensions (SGX) [1] and ARM Trustzone-based Secure Elements [2], the latter is commonly found on smartphones. Another example is Trusty [3], a secure Operating System (OS) that provides a TEE for Android. It is isolated from the rest of the system by both hardware and software. Trusty's isolation protects it from malicious apps installed by the user and potential vulnerabilities that may be discovered in Android.

3.2 Authenticity Proofs Mechanisms

Several authenticity mechanisms have been developed and, as described in the state-of-the-art revision, most oracles as a service providers use authenticity proofs to prove their honest behaviour. However, these proofs are not infallible and the details or their implementation are not always transparent or do not provide the disclosed level of trust. I will deep dive on the most common proofs and discuss their implementation and applicability.

3.2.1 TLSNotary

TLSNotary is a mechanism for independently audited HTTPS (Hyper Text Transfer Protocol Secure) sessions. Allowing clients to provide evidence to a third party auditor that certain web traffic occurred between himself and the server. This mechanism takes leverage of the TLS (Transport Layer Security) [4] handshake protocol to create an irrefutable proof, as long as the auditor trusts the server's public key, by splitting the TLS master secret [5] between three parties: the server, the auditee and the auditor.

The algorithm allows an auditor to verify some part of a session by withholding a small part of the secret data used to set up the HTTPS connection while allowing the client to conduct an HTTPS session normally. The auditor never fully possesses, at any time, any of the session keys and therefore cannot decrypt any sensitive information and can only verify that certain traffic did occur.

3.2.1.1 How it works

TLSNotary modifies the TLS handshake protocol on the client side by levering some properties of TLS 1.0 and 1.1. More specifically the pseudorandom function (PRF) used in the TLS 1.0 RFC 2246.

[1] More information on Intel SGX can be found here: https://software.intel.com/en-us/sgx/sdk

[2] ARM whitepaper on ARM Trustzone-based TEE can be found here: https://www.arm.com/files/pdf/TrustZone-and-FIDO-white-paper.pdf

[3] More information on Trusty can be found here: https://source.android.com/security/trusty

[4] Information on the TLS protocol can be found here: https://www.ietf.org/rfc/rfc2246.txt

[5] The master secret is used, when generating keys and MAC secrets, as an entropy source, and the random values provide unencrypted salt material and IVs for exportable ciphers.

$$PRF(secret, label, seed) = P_MD5(S1, label + seed) \otimes P_SHA - 1(S2, label + seed) \quad (3.1)$$

This function compromises two secrets, S1 and S2. The auditor and auditee will independently generate random bytes of data, S1 and S2, respectively.

The auditee applies P_MD5 to S1, generating 48 bytes:

$$H_1 = H_{1,1} \parallel H_{1,2} \quad (3.2)$$

The auditor applies P_SHA-1 to S2, generating 48 bytes:

$$H_2 = H_{2,1} \parallel H_{2,2} \quad (3.3)$$

The auditor and auditee then exchange H21 and H12 allowing each other to construct different halves of the master secret, M2 and M1, respectively.

$$M_2 = H_{1,2} \parallel H_{2,2} \quad (3.4)$$

$$M_1 = H_{2,1} \parallel H_{1,1} \quad (3.5)$$

The auditee and auditor calculate X and Y, respectively.

$$X = P\_MD5(M_1) \quad (3.6)$$

$$Y = P\_SHA - 1(M_2) \quad (3.7)$$

The auditor sends sufficient bytes from Y to the auditee so that it can compute the necessary encryption keys and client mac key to send the request to the server.

Then the server response is received, but not decrypted, and the network traffic is logged and a hash of the traffic is computed and set to the auditor as commitment.

Only then, does the auditor send the remaining bytes of Y to the auditee and this allows him to calculate the server mac key and safely execute the regular TLS decryption and authentication steps.

This complex sequence of calculations prevents the auditee from creating a fake version of the post-handshake traffic from the server since he did not have in his possession the server-mac-write-secret to decrypt and authenticate the initially requested data.

A more detailed flow and explanation can be consulted in the TLSNotary white-paper [TLS14].

3.2.1.2 Limitations

TLSNotary provides some capabilities to attest TLS connections but comes with several limitations. Firstly, TLSNotary supports only TLS 1.0 or 1.1, the properties mentioned before are not present in TLS 1.2 and 1.3 and the former are considered less secure versions of TLS. Secondly, TLSNotary depends on RSA Key exchange, which does not provide forward secrecy. Thirdly, TLSNotary uses MD5 and SHA-1 functions, which are now considered deprecated. Finally and

most importantly, TLSNotary requires trusting in a third party in most of its implementations, such as in Oraclize [Ora], and being an interactive proof there is no way to verify the TLSNotary proof unless you were performing the role of the auditor during the retrieval. Oraclize, runs an auditor node on Amazon Web Services (AWS), claiming that this implementation is secure as far as AWS is trusted, simply moving the trust to a larger central entity. It also only allows the existence of one auditor in which we must trust, TLSNotary will not be a suitable solution if more than one auditor is required.

3.2.1.3 Conclusions

The TLSNotary proof is promising due to be software based and is, as of this moment, the most spoken of authenticity proof. However, it's applicability is increasingly getting limited due to the deployment of new TLS versions and the assurances provided by the proof current implementations, which simply move the trust to a bigger entity. Therefore, it should not be considered a reliable authentication method for future implementations.

3.2.2 Android Proof

In the oracle context, the Android Proof [6] results from Oraclize research and development efforts. It takes leverage of SafetyNet software attestation and Android Hardware Attestation to provision a secure and auditable environment to fetch authenticated data.

3.2.2.1 SafetyNet Attestation

SafetyNet [7], developed by Google, is an API service that allows assessing the Android device in which an app is running on. It provides a cryptographically-signed attestation, assessing the integrity of the device, looking at the software and hardware environment for integrity issues. By returning an SHA-256 hash of the application that called the SafetyNet API it allows assessing if the application running on the device has not been tampered with by comparing the application SHA-256 hash with its publicly available and distributed open-source code.

3.2.2.2 Android Hardware Attestation

Since Android Nougat [8], developers are able to generate an hardware attestation object with details regarding the device unique key stored in the Android Hardware KeyStore [9]. The attestation object is signed by a special attestation key kept on the device and the root certificate used by that key is a known Google certificate. This guarantees that the hardware running the code has not been tampered with.

[6] Documented explanation of the proof can be found here: https://provable.xyz/papers/android_proof-rev2.pdf
[7] Further described on the google developer page: https://developer.android.com/training/safetynet/attestation.html
[8] https://www.android.com/versions/nougat-7-0/
[9] Further explanation can be found here: https://source.android.com/security/keystore/

3.2.2.3 How it works

The application running on the Android device, on its first run, creates a NIST-256p key pair, containing the Hardware Attestation Object to prove the integrity of the key, using the Android Hardware KeyStore.

When a request is sent to the Android device, it starts an HTTPS connection and the entire HTTP response is retrieved. The response's SHA256 hash is signed using the hardware attested key pair created on the application start. A call to SafetyNet API is then issued to attest the SHA-256 hash of the application package running on the device, which should be open-source and public available and distributed, guaranteeing the application integrity and therefore that no change has occurred on the HTTP response before it was signed and its hash used in the SafetyNet request.

SafetyNet then returns an attestation response in the JSON Web Signature format (JWS) that guarantees the integrity of the application running, the integrity of the system in which the application ran and that both the HTTPS request and the signing process using the initially created and attested key has taken place in the application issuing the SafetyNet request.

The SafetyNet JWS response and the HTTP response is sent back for off-chain verification and validation.

3.2.2.4 Conclusions

The Android proof is a far more complex and in-depth authenticity proof in comparison to TLSNotary. It provides strong guarantees of software and hardware integrity as well as of the requested data. Nonetheless, it relies on a centralized authority, Google, to develop a secure Trusted Execution Environment (TEE), used by Android to generate private keys, and to maintain SafetyNet security sophisticated enough to offer good guarantees of the device and application integrity. A bottleneck in this approach can be the required use of a physical Android device, limiting the scalability context of this approach, but nonetheless, as long as Google is trustworthy it is a very secure model.

3.2.3 Ledger Proof

The ledger proof is based on the use of a specific trusted environment, the Ledger Nano S [10] and Ledger Blue [11], developed by a French company to secure crypto assets safely. This device provides an attestation for its authenticity and code integrity.

3.2.3.1 How it works

This device implements several layers of hardware and software to prove the security of its execution. These devices run specific software, BOLOS [12] (Blockchain Open Ledger Operating

[10] https://shop.ledger.com/products/ledger-nano-s
[11] https://shop.ledger.com/products/ledger-blue
[12] Specific information regarding BOLOS can be found here: https://ledger.readthedocs.io/en/0/bolos/index.html

System), which has an SDK that enables developers to build application which can be installed on the hardware. BOLOS exposes a set of kernel-level APIs which allows running secure cryptographic operations as well as attest the device and the code running on it. The later is very useful as it allows to run code in a secure manner and provides an attestation for the code. An application can ask the kernel to produce a signed hash of the application binary code. A special attesting key is used in this process and is safely controlled by the kernel, away from attacks attempts by any application code. With this, the ledger proof leverages both the device attesting and code attesting features to prove that the applications are running on a TEE of a ledger device.

3.2.3.2 Conclusions

Currently, the ledger proof is used by Oraclize to provide true random data to a smart contract. But its use can be extended to other computation operations that may require to run outside of the blockchain as long as there is support in terms of computational and memory capacity by the ledger device. The device also lacks a direct connection to the internet and therefore cannot be used to query data from the internet.

3.2.4 TLS-N

TLS-N [RWG+17a], is the first privacy preserving TLS extension that is efficient and most importantly provides non-repudiation [13]. TLS-N does not require the use of a third-party or any trusted hardware and is an extension to the TLS 1.3 protocol. In comparison to other implementations such as TLSNotary which rely on deprecated versions, is up to date to the current technologies. It guarantees non-repudiation, not only in a single TLS message exchange but also in a conversation compromising several messages. It allows, with an additional computation overhead, to obfuscate certain parts of the conversation (such as passwords or other sensitive information) while keeping its trust model intact.

In the TLS-N model, there is no need to trust in a single auditor, such as in TLSNotary, since the proofs are non-interactive and can be inspected by anyone, at any point in time, without having to trust in a single auditor honesty.

3.2.4.1 How it works

TLs-N requires the web server (generator) and the client (requester) to have both support for the protocol.

Initially, both generator and requester establish a TLS connection and negotiate the TLS-N parameters in the handshake. The generator stores the state of the conversation which comprises a hash value incorporating all previous records, an ordering vector and the time stamp from the start of the session.

[13] Non-repudiation refers to a state of affairs in which the authenticity of something cannot be challenged, meaning that there is absolute guarantee that something happened the way that it is stated.

The protocol ends when the requester sends a request for evidence. The evidence is composed of a window of the exchanged ordered messages signed by the generator. The window begins right after the handshake, this prevents Content Omission Attacks, depicted on Figure 3.1.

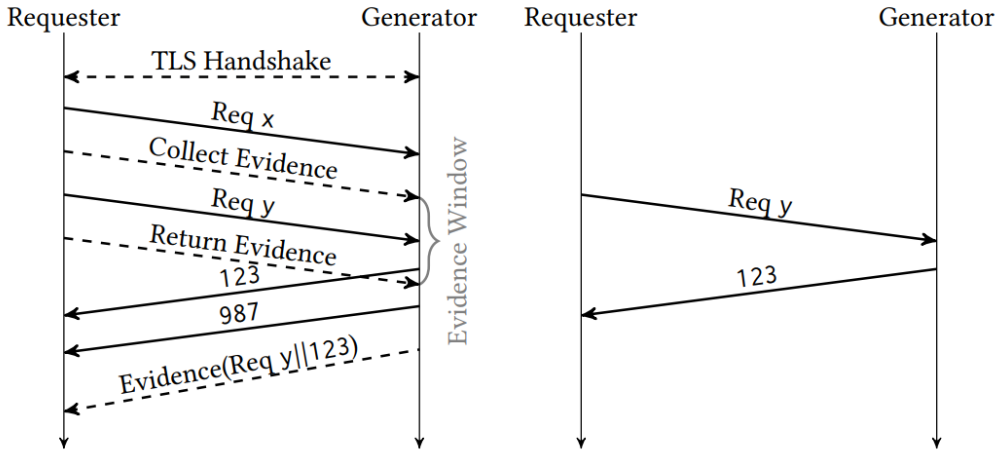

Figure 3.1: Content Omission Attack - The left figure shows the original and the right figure the signed conversation.
Image extracted from [RWG+17b]

In this situation the evidence collection only starts after the first request is done, and another request is asked right after (this one, inside the window collection, Req y on the image) and the response for the first request is assumed to be the response to the second one. Only this two messages are stored in the evidence window, since context is missing in the signed conversation, the response 123 appears to belong to request y which is incorrect. Therefore, TLS-N always starts right after the TLS handshake. The correct TLS-N flow is presented on Figure 3.2.

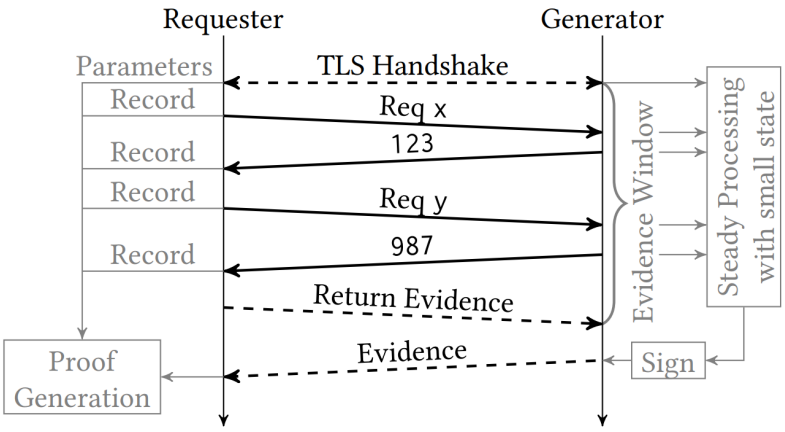

Figure 3.2: Simplied Overview of TLS-N.
Image extracted from [RWG+17b]

To generate a small proof independently of the number of messages, TLS-N uses merkle trees [Mer79] to create a chain of messages' hashes and then returns only the last hash, which to be created requires all the previous hashes. This ensures a small storage overhead per TLS session.

3.2.4.2 Conclusions

TLS-N was designed with the oracle trust problem in mind, the generated proof is small enough to be evaluated on-chain on a smart-contract. The only drawback is that the smart contracts cannot verify TLS signatures based on the web-PKI (public-key infrastructure) and therefore the contract must have the generator public key.

TLS-N is, therefore, a promising solution to the oracle trust problem being the only major drawback requiring the data providers to adopt the TLS-N protocol.

3.2.5 Town Crier

Town Crier [ZCC+16] (TC) authenticates data-feeds through the use of trusted hardware, more specifically Intel SGX enabled CPUs. By levearing intel SGX, and as long as this system is trusted, there's is no need to trust the environment in which TC is running since it the TEE guarantees the isolation of the TC implementation.

Thanks to its use of SGX and various innovations in its end-to-end design, Town Crier offers several properties that other oracles cannot achieve:

1. **Authenticity guarantee**: There's no need to trust any particular service provider(s) in order to trust Town Crier data. (You need only believe that SGX is properly implemented.)

2. **Succinct replies**: Town Crier can prune target website replies in a trustworthy way to provide short responses to queries. It does not need to relay verbose website responses. Such succinctness is important in Ethereum, for instance, where message length determines transaction costs.

3. **Confidential queries**: Town Crier can handle secret query data in a trustworthy way. This feature makes TC far more powerful and flexible than conventional oracles.

3.2.5.1 How it works

Intel's Software Guard Extensions (SGX) is a set of new instructions that confer hardware protections on user-level code. SGX enables process execution in a protected address space known as an enclave. The enclave protects the confidentiality and integrity of the process from certain forms of hardware attack and other software on the same host, including the operating system.

Upon request, the enclave can generate a proof, usually called attestation, signed by and hardware protected key that can be used to prove that a certain software was run in a legit TEE.

Proof	Requires specific hardware	Can query the Web	Future-proof	Currently in production
TLSNotary		X		X
Android Proof	X	X	X	X
Ledger Proof	X		X	X
TLS-N		X	X	
Town Crier	X	X	X	X

Table 3.1: Summary of authenticity proofs

The flow of a TC requests, depicted in Figure 3.3, starts with a User Contract creating a datagram request to the TC Contract. The TC Contract, is a simple smart-contract on the blockchain that accepts client requests and to whom the TC server listens to for new requests.

The TC Server, listens to events on the TC Contract and queries the requested data-source. The enclave takes care of signing the operation where as the relay is a simple networking interface. The answer is signed and returned to the TC Contract to later relay it to the User Contract.

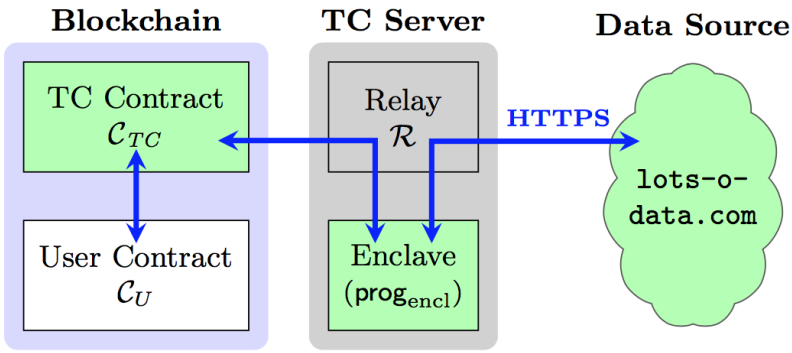

Figure 3.3: Town crier Architecture.

3.2.5.2 Conclusions

Town Crier, presents an effective solution to tackle trust issues in the oracle operation. It does require the use of proofs as well as its later analysis for good behaviour and specific hardware.

In scenarios, where there is a higher requirements for trust in the oracle behaviour and cost/performance is not a problem, TC is a viable and more secure solution than software-based proofs.

3.3 Summary

Table 3.1 summarizes the previous analysed proofs regarding the necessities of specific hardware, the possibility of their usage to not be deprecated in the next following years, if whether they can query the web and are already be used in production.

Chapter 4

Problem Statement

Smart contracts power a decentralized world of automation and trust-less commitments. Companies, groups and individuals are able to automate tasks and contracts but in the current ecosystem, smart contracts are still very much limited to the information available in the blockchain. Therefore, connecting with the outside world requires a trusted authority to input in the blockchain the required information upon request from the smart contract. This trusted authority is generally called an oracle.

As explained before, the deterministic nature of blockchain does not allow smart contracts to directly query a data-feed for information. In this context, oracles help connecting smart contracts to the world outside of the blockchain. The problem here is to trust the oracle service to not behave maliciously and undermine the trust provided by the blockchain consensus mechanisms. Blockchain technology can be trusted to behave correctly even in byzantine environments, but the oracle service does not abide by the same rules and therefore some workings must be put into action to ensure the oracle's response credibility.

As seen in Chapter 2, current solutions to the oracle problem use complex techniques to achieve a certain desired level of trust. Some use complicated trusted hardware others incentive-based mechanisms or authenticity proofs, but neither of these are simple and fully trusted approaches and they add extra complexity from the developer side, as detailed on Chapter 3.

The oracle problem is neither simple nor has a single solution, but its importance in powering greater applications for smart contracts is undeniable. Their need arises from the following three factors.

- **Smart contract empowerment** - Providing smart contracts with trustable information from outside of the blockchain is decisive to gain general adoption and practicality.

- **Cost optimization** - Blockchain operations tend to be quite expensive, therefore, the oracle solution should introduce a lower overhead cost as possible.

- **Keeping trust standards** - As blockchain technology creates a trust-less environment, oracles should as well keep up with the level of trust in their functioning.

4.1 Proposal

With this thesis, the author intends to lay the foundations for the development and architecture of trustable oracle systems that will power several smart contract use cases.

The author believes that by describing, in a trust-guided manner, multiple architectures and examples where they are being applied or possible use cases not yet documented creates a guided model that helps future cases to have a systematic approach to which architecture will fit the best. The architecture and design of blockchain oracles is still very much unexplored territory, specially in terms of academia research but also in the industry, and therefore this thesis approaches it broadly and investigates possible approaches and their trade-offs so that later studies can be developed on the specifics of each architecture.

Furthermore, in Chapter 6, the author presents a possible implementation of a self-hosted oracle. After analysing the state of the art in oracle development and the specifics of used authenticity proofs, the author believes that the best way to achieve trust in an oracle is to deploy one instead of relying on a third-party. The described approach, when in comparison to deployed solutions in the industry reduces operations costs, increases trust and empowers the contract with purpose built oracles. The author will demonstrate that deploying and oracle, can be more trivial than at first seems, and that trust in its operation is directly the trust in one's code and no more measures (authenticity proofs) are need. This measures usually add a considerable extra cost and constrain the developer.

4.2 Desiderata

This section describes several forces that separately or in combination drive the design of oracle architectures and implementations. Defining this forces will help validate each architecture/implementation in what forces they are able to accomplish in their different scenarios.

Fast time-to-market Not having to assemble a team or allocate resources into a developing a new product which will only serve as component of the main product being developed.

Keeping trust standards The company focus is not the development and security of the oracle service and may not have enough resources to keep the oracle as secure and reliable as the underlying blockchain.

Data-feed fault tolerance Ensuring that a contract can follow through even if a data provider is down by querying another provider.

Data veracity Querying several data sources guarantees a higher trust on the veracity of the data by not allowing a single service to be the owner o the truth.

Lower smart contract costs Checking authenticity proofs leads to higher contract deployment costs, as the proof can be long and computationally expensive. Striving to build simple oracle smart contracts will lead to reduced costs.

Lower oracle complexity Dealing with authenticity proofs and implementing the necessary mechanisms verify them requires a higher and deeper knowledge of their implementation mechanisms and underlying cryptography.

Oracle decentralization Connecting a smart contract to data through a single node, creates the problem that smart contracts intend to avoid, a single point of failure. With a single oracle, a smart contract is only as reliable as that one oracle.

Oracle ownership decentralization Having one party control the oracle network centralizes the power to manipulate all the contracts relying on the information provided by that network of oracles.

4.3 Conclusions

This project aims to pave the way for oracle and smart contract development. It does not try to come up with a new authenticity proof which adds extra complexity for the common smart contract developer, but instead guide the developer to a solution accordingly to the problem necessities. As well as, providing a simple but yet effective implementation of a self-hosted oracle so as to have a simple skeleton to which the developer can iterate upon and adapt to the specific smart contract needs.

Chapter 5

Trustable Oracles

At this point, a definition, of what trust in an oracle is, seems appropriate. Trust has a lot of meanings, depending on the needs of all the parties involved. I will model several levels of trust and the requirements and fallacies of each model as well as its application and drawbacks.

Starting from an absolute trust scenario, in this model, the end user, being the smart contract which receives information provided by the oracle, has complete assurances from both the veracity of the data provided by the data source, as well as, undeniable proof that the oracle did not tamper with the relayed information. This scenario points out two main points of failure, either maliciously or unintentionally.

The first component which can be faulty or compromised is the data source. Assuring that the information provided is correct does not have a straightforward answer. What correct means is open to interpretation. For example, if the data source is an IoT sensor, which is prone to failures, being correct is relative. The sensor needs to be perfectly calibrated and accurate. In this case, using several sensors and averaging its values or removing outliers would solve its correctness. Another example could be an API that returns the current value of the EUR in USD. In this scenario, a party that would benefit from a higher conversion than the real one could coerce or attack the data-source into providing a favourable value. The answer here can also be using several data sources. Another solution would be to use a highly trusted entity such as the European Central Bank (ECB) which can be a lot harder to coerce or attack and having a signature from the ECB that backs the provided data. Choosing what type of data-source to use has a huge impact on the trust fullness of the provided data not to mention architecture centralization when using a source such as the ECB. All in all, the end user will have to understand the requirements and level of trust necessary.

The second, and most relevant for analysis, is the oracle service used. Oracles are a necessary part of the process since the other option would be having the data providers adapting to the blockchain which does not seem to be a realistic option at the moment. Therefore we must trust an oracle or a group of oracles. Two main options are available, either trusting a third-party oracle or

self-deploying an oracle. In the first scenario, three variables take part in the level of trust. Firstly he third-party oracle, if paid for, has the monetary incentive to be honest, since a bad record of dishonesty would have the service losing credibility and therefore clients. Secondly, by using proofs the oracle can establish its legitimacy, as long as, the proofs can undoubtedly be trusted and verifiable by the smart contract, I will later analyse in depth this issue. Finally, oracle execution transparency by using open-source code and having means for being audited. Additionally, to guaranteeing single oracle integrity, it may be in the interest of the user to use several oracles either to provide service availability or to increase trust by combining the result from different oracle services.

5.1 Oracle Architectures

Having analysed what trust means, it is evident that no short definition is appropriate and that it depends on the stakeholder beliefs. Hence, several architectural models for what a trustable system arise. Varying in decentralization and complexity. Each model satisfies different requirements, such as performance, security and decentralization. In this section, I will describe several possible architectures and point out use cases and compromises for each model.

5.2 Oracle as a Service w/ Single Data Feed.

5.2.1 Context

Connecting smart contracts with information provided by data-feeds, which do not, by themselves, input the required information on the blockchain requires the use of a trusted oracle. Developing and maintaining a oracle may be prohibitive in terms of cost (if assembling a team was needed) and desired time to market. Outsourcing such service would be desirable in this context, it may not be in the interest of the company to specialize in the secure development of oracles.

5.2.2 Example

A smart contract developer needs to obtain information from an API source without having the trouble to develop and launch its on oracle whilst having some guarantees of the origin of the information. Mainly, this scenario is focusing on fast-time to market, untampered data and cost is not a problem.

5.2.3 Problem

How can a non-blockchain company keep up with the fast pace of industry while maintaining trust in its services? It is critical to be able to quickly build a smart contract and connect it with the needed information. How can a company do so, without allocating human resources into to the development of yet another service and simply focus on its business logic?

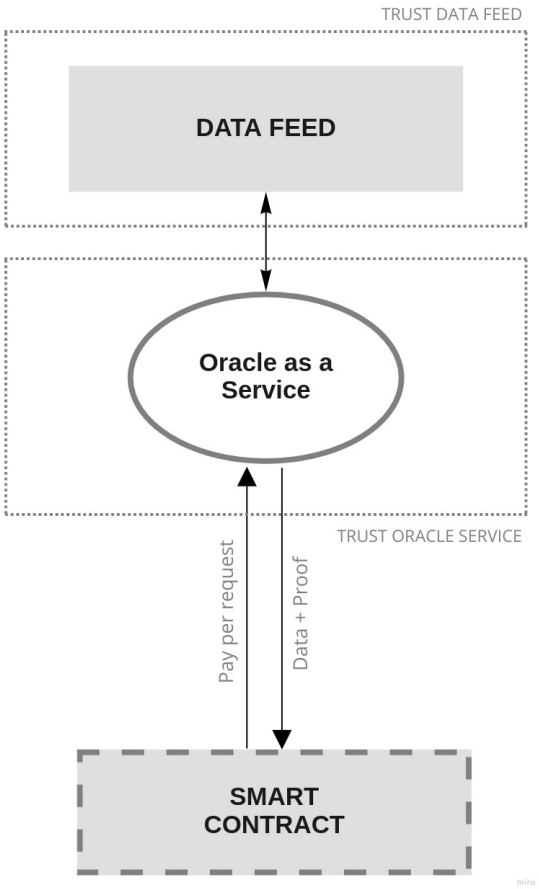

Figure 5.1: Oracle as a Service w/ Single Data Feed.

5.2.4 Forces

- **Fast time-to-market** - Quickly deploy a product without the overhead of focusing on oracle functionality and security.
- **Trust** - The oracle should guarantee as much as possible the same level of trust in its functioning as the underlying blockchain technology.

5.2.5 Solution

Oracle as a service, come as a quick and efficient solution for fast moving companies and individuals. Providing easy integration between a smart contract and a data-feed by means of specific function calls and/or libraries. Theses services are per-request fee-based and can be cheaper comparing to assembling a team dedicated to the development and maintenance of an oracle. The fee-based system increases the trust in the service as being honest is crucial to their business model. Additionally, this services usually provide authenticity proofs which serve as another layer of trust in the service. In the Chapter 4 I deep dive on the subject of the proofs.

5.2.6 Example Resolved

5.2.7 Resulting Context

This solution results in an architecture that compromises two points of trust. The first being the data-feed itself. No guarantees are given that the data provided is reliable and the smart contract owner must, therefore, to the best of his knowledge, select a data-feed in which, by the operator size or record of good behaviour, he can trust.

The second point of failure is the oracle service itself. Although smart contracts, in the resulting context, have access to the information from the outside, that is only possible due to the use of a third party to honestly relay the data. In this architecture, if the oracle simply relays the data, then no trust model can be achieved as the oracle good behaviour is not tested against. As this would not be a feasible architecture the existing services provide authenticity proofs to guarantee, to a certain level, their honest behaviour. The problem here is on how are these proofs generated, can they be verified on-chain or only off-chain and who is making, or providing, the verification tools. In Chapter 4 I deep dive on these questions and techniques. Another reason to trust in the service can be the monetary incentive for good behaviour. By paying the oracle for each request, that becomes the oracle service business model, an extensive record of good behaviour is crucial for business prosperity and therefore a good enough incentive for honestly conveying the requested data. In this context, if the authenticity proofs provide enough assurances for the smart contract creator and he trusts in the selected data-feed to provide the required data, then this model can satisfy its needs in terms of trust, as well as, performance since it only queries one data-feed and uses only one oracle. By not having any consensus mechanism an exchanging the least amount of messages it can both achieve greater performance and a lower cost. But this lower cost and higher performance architecture by itself is prone to failure due to lack of decentralization and does not guarantee service availability which could lead to a failure in the smart contract to obtain the requested information.

5.2.8 Known Uses

5.3 Oracle as a Service w/ Multiple Data Feeds.

5.3.1 Context

This scenario iterates on the previous one but focuses on data veracity. Sometimes an answer to a contract request cannot be truly accepted unless several sources confirm it. Either because it is unwise to trust in a single identity or because there might not be a single true answer but only an answer that is accepted by a selected majority.

5.3.2 Example

5.3.3 Problem

The previous architecture specified a single point of failure on the data-source layer. A contract with high requirements in terms of availability cannot rely on using a single data-source, as doing so would void the contract when the service providing that data is down or taken down. In terms of trust, certain contracts may also require that several services provide an answer and then have a consensus between all the received answers. This cannot be achieved by querying a single source and therefore the oracle service must be able to query several sources and either define the resulting answer or provide all the responses to the smart contract and let the smart contract resolve to a final answer.

5.3.4 Forces

- **Availability** - Higher fault-tolerance guarantees in the data provider.
- **Trust** - Higher trust on the veracity of the queried data..

5.3.5 Solution

The oracle service should have a mechanism to query several data-sources during a specified timeframe. And have a predefined consensus mechanism that would require to have m of n data-sources providing the possible answers and reduce them to a final answer to the smart contract.

5.3.6 Example Resolved

5.3.7 Resulting Context

In this context, the layer of trust regarding the data-feed is almost eliminated by having the ability to choose from several data providers and therefore not relying on a source of truth. It also provides a higher system availability, as the oracle/smart contract can have some degree of redundancy in the data providers selection.

5.3.8 Known Uses

5.4 Single-Party Self Hosted Oracle.

5.4.1 Context

Although the use of Oracles as a Service allows for a lower product time to market by not having to take care of the development, maintenance and deployment of the oracle service it usually leads to less flexibility in the oracle design, vendor lock-in and fees charged by the vendor. If the product requirements do not allow for the specified challenges or the trust levels required by the contract are more than what the oracle vendor can provide it may be a solution to deploy its own oracle.

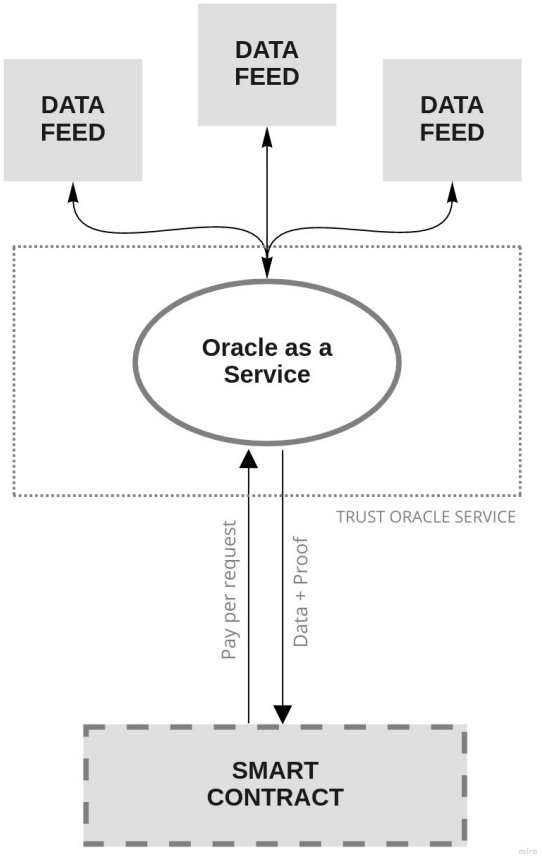

Figure 5.2: Oracle as a Service w/ Multiple Data Feeds.

A company with its own developing team capable of allocating resources for the development of the oracle or a single developer who does not want to incur in the oracle vendor fees will benefit from their own deployment in terms of cost and most importantly in regard to trusting the oracle behaviour.

Instead of fast-time to market the main focus here is trusting the oracle provider to behave correctly. The smart contract output will only affect a single party or multiple parties which are non competing and therefore trust someone to run the oracle. In such scenario, costs can be reduces, with increasing developer workload, by not using any authenticity proofs and the oracle can be further customised to handle a specific contract requirement.

5.4.2 Example

5.4.3 Problem

Currently, oracle behaviour is neither easy to check nor fully transparent and trustable. As seen in Chapter 4, verifying oracles authenticity proofs sometimes cannot be done on-chain, resulting in a contract being executed with an incorrect proof which is only later verified but the contract is irreversible adding not much to oracle trustability except the ability to cancel future contracts. Proofs

also add complexity to the smart contract code which will result in slower contract development and more importantly in higher contract costs. Most blockchains charge contracts by either CPU, memory and network use, or even all of these, and therefore receiving the proof and verifying it on-chain will increase the cost of running a contract.

5.4.4 Forces

- **Trust** - Higher trust requirements than those provided by the authenticity proofs.
- **Cost optimization** - Checking authenticity proofs leads to higher contract deployment costs, as the proof can be long and computationally expensive. The use of authenticity proofs also increases personal cost, as it requires higher knowledge of the proofs workings and of cryptography.

5.4.5 Solution

A solution to trusting an oracle service is to deploy our own oracle service. Surely, doing so incurs in technical expenses for programming, deploying and maintaining the oracle, however, does not require to trust in a third party but only on our ability to maintain the necessary level of security in our own oracle. Additionally, it will free the smart contract owner from the fees charged by the oracle provider and allow for further flexibility in adapting the oracle to new sources of information. Furthermore, it will also lead to simpler and cheaper smart contracts by not requiring the use of authenticity proofs in regards to the oracle behaviour, as the developer knows exactly what the oracle is running under the hood.

5.4.6 Example Resolved

5.4.7 Resulting Context

With this solution we almost remove the second layer of trust, trusting in the oracle service. Nonetheless, we move the trust to the developer ability in coding a secure and reliable oracle. The main benefit is not requiring to have the overhead expense of using, understanding and verifying the authenticity proofs required for a trustable use of Oracles as a service.

5.4.8 Known Uses

5.5 Multi-Party Self Hosted Oracle.

5.5.1 Context

In some cases, competing parties may rely on a smart contract to keep track of some value with interest to them, therefore, it may be a requirement that several of these parties take part in the process of providing the data to the smart contract. It may also be the case, that if a single oracle is the source of truth of a smart contract, then the easiest way to attack the smart contract is by

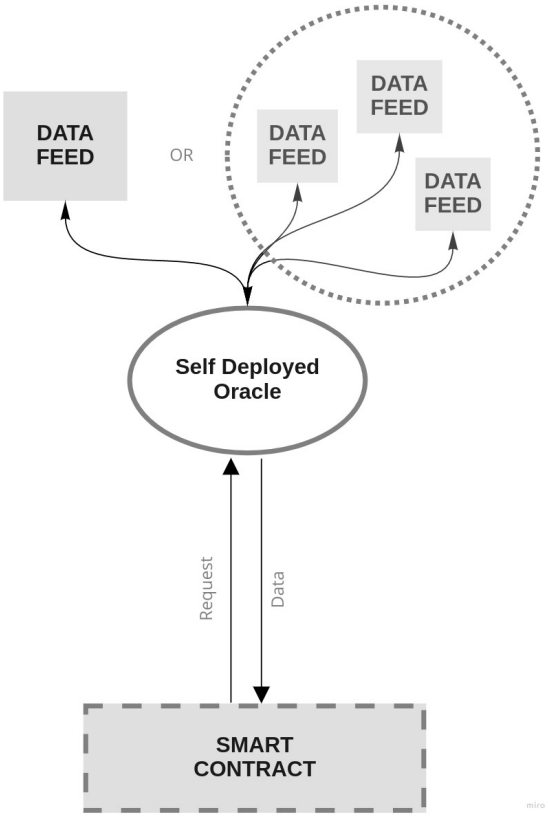

Figure 5.3: Single-Party Self Hosted Oracle.

attacking the central point of failure, the oracle. In both of these cases, the oracle singularity needs to be tackled.

5.5.2 Example

5.5.3 Problem

This context raises two problems, oracle consensus and availability. Whoever owns the oracle providing the data to the smart contract holds the smart contract and therefore can influence the execution of the contract, in which several competing parties rely upon. In terms of availability, a single oracle creates a single point of failure in case of an attack or system failure.

5.5.4 Forces

- **Availability** - Oracle decentralization eliminates the oracle service as a single point of failure and possible denial-of-service attacks.
- **Trust** - Oracle ownership decentralization brings increased trust in the result of the smart contract, as it requires multiple distinct entities to deploy an oracle and have their say in the final data provided by the oracle to the smart contract.

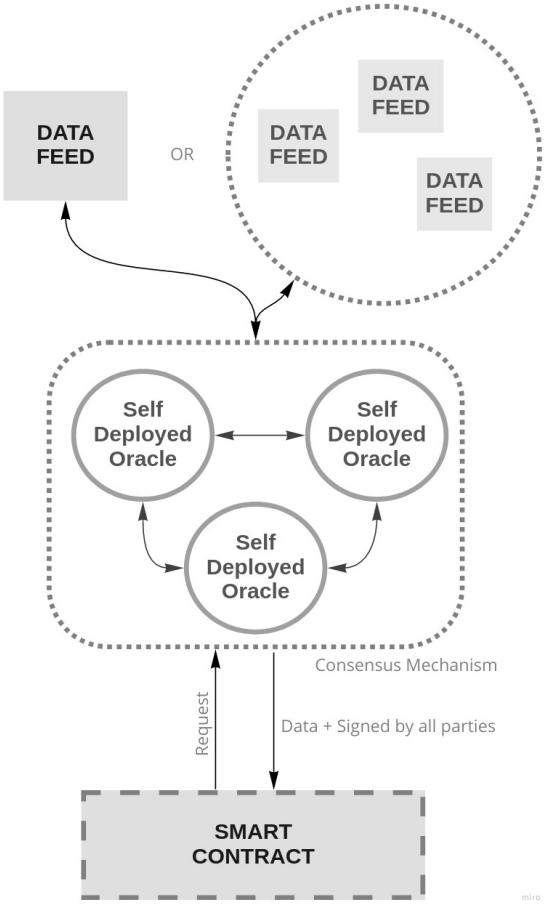

Figure 5.4: Multi-Party Self Hosted Oracle.

5.5.5 Solution

The most beneficial and simple solution, here, is having each interested party launching their own oracle and having all oracles communicating between themselves with a mechanism for consensus. The consensus mechanism would vary from case to case, and from how critical the smart contract solution is. To increase the level of trust in each party, each node would sign their response and be able to launch only one node. With this, once one of the nodes had collected all the signatures than it would provide the contract with the requested information. Also, a party would not be able to gain control over the network of oracles by launching more nodes than the remaining stakeholders. However, the consensus algorithm should never require that all nodes provide a response since that would again create a weak network in which by tacking down one oracle the whole system would fail.

5.5.6 Example Resolved

5.5.7 Resulting Context

With this context, we bring the same trust level given by blockchain technology to the oracle service. Resulting in a decentralized network with no single party running it and every stakeholder has the same weight in providing the data. This context, however, is only suitable for previously defined user groups, with an agreed minimum necessary quorum for consensus and known public keys of all nodes.

In a community context, this approach is not suitable since nodes would be able to join and leave making it harder to achieve a predefined consensus. Involved parties would be able to launch more than one node, resulting in some parties being able to take over the minimum consensus quorum and overpower the network unless some proof-of-work mechanism is implemented. This would also result in a context of wisdom of the crowd, in which the most effective way of controlling a correct answer would be by implementing some incentives mechanism such as [ABV+18b]. The problem around incentives is that they do not guarantee that, in edge cases, with enough incentives, the network will provide a wrong answer if justifiable. Although, as far as the author is concerned, no other mechanism is available when dealing with wisdom-of-the-crowd information.

5.5.8 Known Uses

5.6 Summary and Conclusions

The described patterns represent different trust level requirements and forces. Each resolve a specific issue and may create another. When the trust requirements increase so does the gap from idea to market and development costs. Each architecture involves trading cost and flexibility with trust.

Figure 5.5 depicts a possible simple flow of thought when choosing the previous defined pattern that better fits a specific need.

First the decision maker must look at the smart contract needs and decide if the level of trust and audibility provided by an oracle service is sufficient. If so, then can he trust the data source or is there a need for several data providers? Leaving two patterns, 5.2 and 5.3. If he cannot trust any existing oracle service, either because the existing proofs are to expensive to verify, or cannot be verified in the contract or just don't provide the necessary audibility, among others, whatever the reason he needs to think if he has the, either monetary and human, to build his own oracle service. If not, and with increasing costs due to per-request fees, he may choose to use multiple oracle services and then perform some consensus mechanism on the smart contract. If he can then build and maintain its own oracle service he must aks him self the question, Who will use this oracle? How many different and maybe competing parties rely on the smart contract to which the oracle will provide data. If there is only one stakeholder of the smart contract and he runs the oracle, then a perfect system of trust is achieved since outside the blockchain he controls every part of the process, resulting in the pattern 5.4. However if a smart contract has several stakeholders then, no

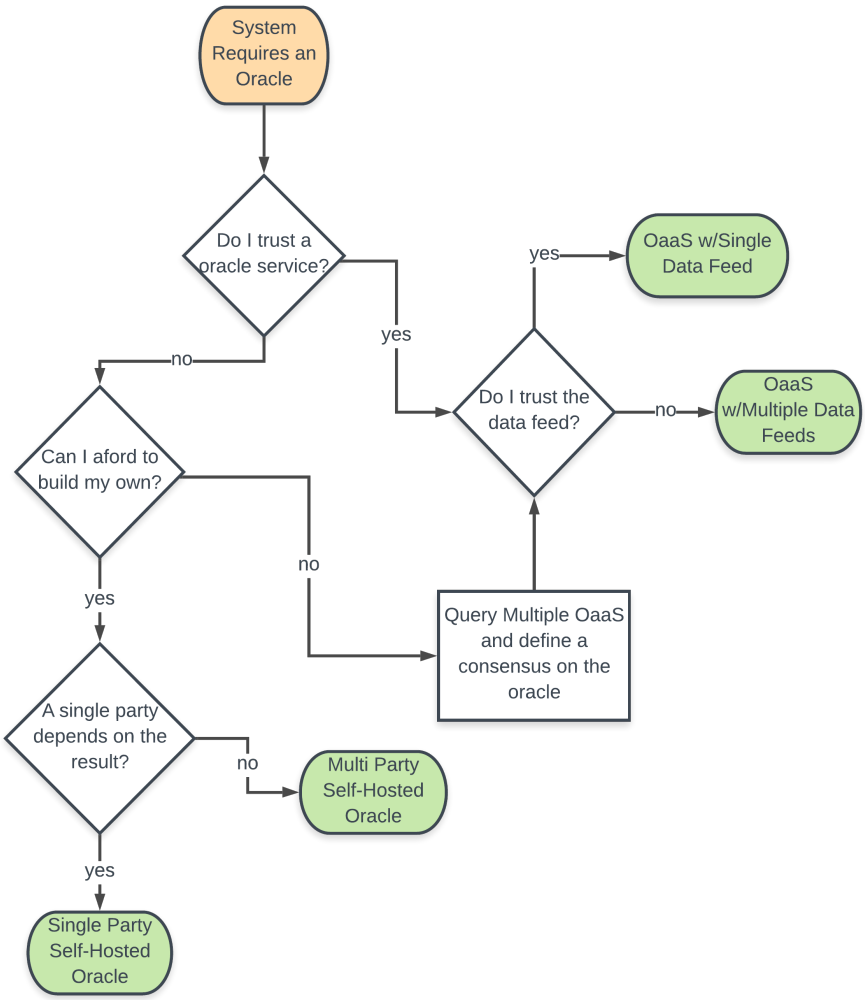

Figure 5.5: Process for choosing the architecture of a blockchain oracle.

single party should control the oracle and there must be a mechanism to deploy several oracles to power the smart contract while achieving consensus outside of the blockchain and only providing the smart contract with the final result. This reduces smart contract costs while allowing every stakeholder to have a say in the data provided to the smart contract, pattern 5.5.

Chapter 6

Self-hosted Oracle Implementation

In this Chapter I present a possible implementation of a multi-purpose self-hosted oracle. Multi-purpose since it will be able to query a requested API and return a specific value from the answer of that API, allowing to be used by several contracts which require different information and different sources. Self-hosted, as its code is available for anyone to copy and use for their own purpose and not having to rely on and oracle-as-a-service product by deploying their own version of the oracle.

As far as the author as searched, at the moment there is no clear explanation on how to implement your own oracle and therefore on how to power smart-contracts to query the web. Creating, therefore, a need for such a clear and detailed explanation as it will be presented in this section.

In principle, the described oracle is intended to be used by single entities or competing parties. Meaning, that it requires a list of predefined oracles and a predefined minimum quorum. Therefore, is not open to a community in which oracles can leave and join the network. The rationale behind this decision is that if it were to be open to a community the decision power in the final result would be dependent on who could launch the most oracles, solving this issue would require the use of strategies, such as, proof-of-work which would become a different issue that the one the author is trying to solve. In this setup, competing parties which may not trust each other, would be able to power their contracts by having each party launching one oracle, and therefore having all the same power of decision. Has the list of the oracles address is in the open on the oracle smart contract, there is no way for a party to cheat in their voting power.

6.1 Oracle Overview

The oracle compromises two main components, the on-chain oracle and the off-chain oracle. Figure 6.1 depicts the general architecture and a simplified version of the messages exchanged.

The on-chain oracle is a smart contract that functions as a bridge between a client smart contract that needs information from the web and the oracle service that will query the web. This oracle has a whitelist of oracle addresses which are trusted by the oracle to query the web and has

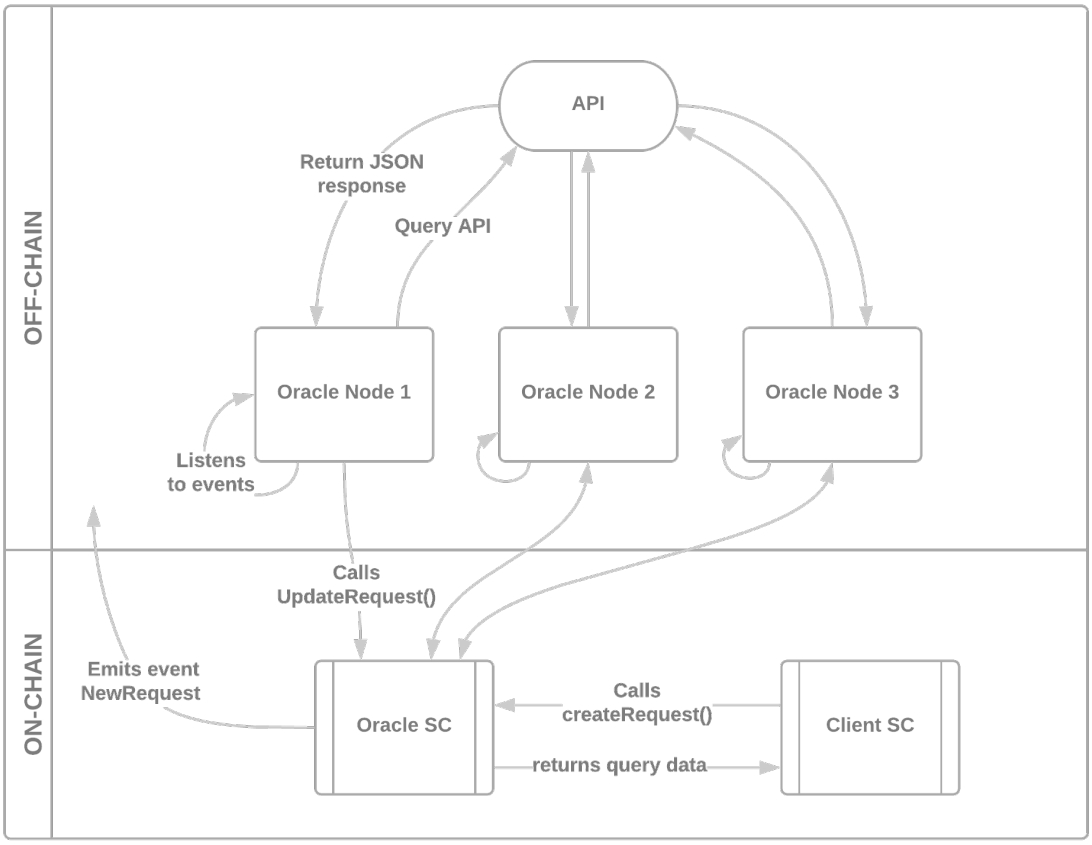

Figure 6.1: Self-hosted architecture.

the necessary functions to create events that will trigger API calls and reach a consensus and the necessary data structures to store the requests and the agreed answer.

The off-chain oracle, or oracles, are services that continuously listen to specific events emitted by the oracle smart contract. Upon listening to a *NewRequest* event query the specified API and key and return a single value to the smart contract by means of a new transaction.

This architecture allows for the use of several oracle nodes and the use of minimum voting quorums to achieve higher levels of trust, include more parties or increase service availability. However, the higher the number of oracles the higher the cost per transaction. Table 6.2 shows the cost of each query in euro using different numbers of oracles[1].

[1]Each test was composed of 110 requests using the same settings (Gas Price of 20 Gwei) except for the number of oracles. The result shown is the average cost of each request.

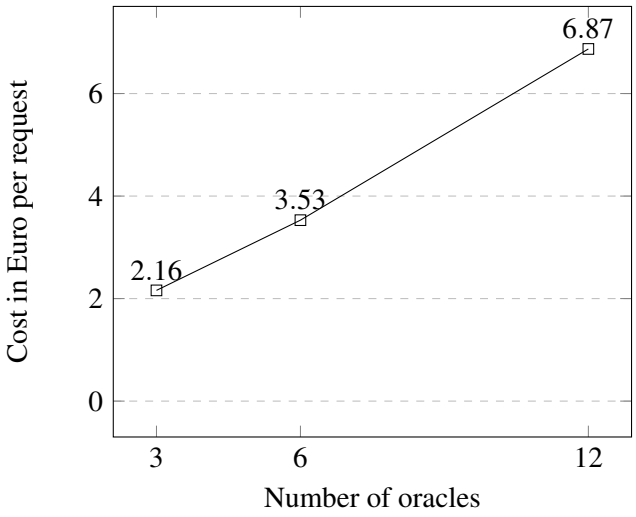

Figure 6.2: Cost per query using a consensus of 2/3,
Queried Google Finance on the 22th of May, 2019.

6.2 Component analysis

6.2.1 On-Chain Oracle

The on-chain oracle is a smart contract that has an array which stores the requests made to the contract. Hard-coded in the contract is the predefined minimum quorum, which is the minimum number of equal answers needed to trust in the declaration of a final result. This minimum quorum will be used in all requests to the contract. Also hard-coded are the white-listed addresses of oracles that the contract will accept transactions to update requests.

Having the addresses and minimum quorum hard-coded on the oracle smart contract is not a software anti-pattern but rather an imposition on the contract terms. Doing so allows all parties who depend on this oracle to previously know that the oracle will always query those addresses and therefore they cannot be later altered for the benefit of one or more parties.

The code for the on-chain oracle can be found in the appendix B and due to its small size can be easily interpreted. Nonetheless, an explanation of its logic is detailed bellow.

6.2.1.1 Creating a request

Initially the request structure 6.2.1.1 only contains the URL which will be queried by the off-chain oracle and the attribute to return in the json API response.

```
1  struct Request {
2    uint id;                            //request id
3    string urlToQuery;                  //API url
4    string attributeToFetch;            //json attribute (key) to retrieve in the
     response
```

```
5     string agreedValue;                  //value from key
6     mapping(uint => string) anwers;      //answers provided by the oracles
7     mapping(address => uint) quorum;     //oracles which will query the answer (1=
      oracle hasn't voted, 2=oracle has voted)
8   }
```

A client smart contract calls the public function *createRequest* passing the url to query and the attribute from the api response that it needs to retrieve. This will add a new request to the array of requests in the oracle smart contract, initializing the list of trusted off-chain oracles addresses, the quorum. This list is composed of the addresses of the accounts which are trusted to add their input by creating transactions to the on-chain oracle contract.

The mapping of each address to an unsigned int is initialized at one, due to the fact that by default a mapping contains all possible addresses initialized at zero. By marking an address at one we explicitly set the trusted addresses so that later we can filter messages whose sender was previously marked with one.

Finally, an the *NewRequest* event 6.2.1.1 is emitted so as to alert the off-chain oracles of the existence of a new request.

```
1     event NewRequest (
2         uint id,
3         string urlToQuery,
4         string attributeToFetch
5     );
```

6.2.1.2 Reaching consensus

Each off-chain oracle, upon listening to the *NewRequest* event will query the specified API and call the *updateRequest* function on the on-chain oracle contract passing the id of the request and the value it retrieved from the API. The calling of the function is done by means of inputing a new transaction on the blockchain addressed to the smart contract, this will make the requested information available on the blockchain to be used by the contract.

The on-chain oracle contract, will first check if the transaction came from the whitelisted oracle addresses and if so, mark, that for this specific request this oracle has inputed his answer. Then it will save the answer on the list of answers for that request and count how many answers are on the list that match the current answer. If the count matches the minimum quorum set on the contract, the oracle contract will set a final agreed value for that request, meaning that at least a specified minimum number of oracles have provided the same answer and so it can be trusted to be the correct answer. This will emit a *UpdatedRequest* event 6.2.1.2 that will alert who ever made that request that an agreement was reached on its answer.

```
    event UpdatedRequest (
        uint id,
        string urlToQuery,
        string attributeToFetch,
        string agreedValue
    );
```

6.2.2 Off-Chain Oracle

The off-chain oracle is a service that continuously listens to the events emitted by the on-chain oracle contract, more specifically to the *NewRequest* event.

For this proof-of-concept the author used a node.js service, detailed in Appendix C, which upon new requests queries a specified API and returns a value to the smart contract by means of a transaction.

In order to create theses transactions we connect to the Ethereum blockchain, using web3.js[2], an Ethereum JavaScript API, and configure the accounts which will add the transactions with the answers to the requests to the smart contract. These accounts are the ones whose account addresses are specified on the white-listed list of addresses on the on-chain oracle. The detailed code can be found on Appendix D.

The off-chain oracle is very versatile, as it can be written in any language, that is supported by the Ethereum API. It can be worked upon to easily integrate new APIs or further logic and features without requiring any changes to the smart contract, as long as it respects the contract callback requirements.

6.3 Summary and Conclusions

This oracle implementation, although simple, is a versatile proof-of-concept which can already be applied to multiple smart-contracts in different scenarios. Since it allows to query any supported API and to choose which field the client smart contract needs. Limited only in accepting requests to APIs which return an answer in JSON format and also to only one value, but those limitations are only from the developer perspective and can easily be removed. This allows to achieve a desired simplicity that does not constrain the current model to a single use case. Rather, the code base can be used to add new features accordingly to the needs of each case without requiring big and breaking changes, being therefore a great starting point tackling the biggest problems when creating an oracle from scratch.

Having explained the versatility of this approach we have to consider the cost and benefits of being self deployed. Firstly, by using a self-deployed oracle, no extra-fees[3] are charged by a third

[2] https://web3js.readthedocs.io/en/1.0/
[3] https://docs.oraclize.it/#pricing

party. To be clear, the cost of making a request to an oracle-as-a-service provider is composed of making the call to the oracle smart-contract, meaning its execution, plus an extra-fee for using the service. Secondly, trust is maintained since we know exactly the code that is being executed, even in an environment of competing parties, since all parties can launch an oracle and have the same say in its final result. Thirdly, cost can be easily optimized, since cost in a smart-contract is proportional to the amount of executed code, the developer has full access to edit and improve the on-chain oracle contract and therefore can improve the cost of each transaction/answer from the oracle to the oracle contract. Whereas in a third-party service the on-chain oracle contract code is managed by the service to whom we pay. Since, third-parties usually use authenticity proofs as evidence of their honesty, those proofs will add an extra cost to the transaction. Nonetheless, there are some possible extra costs not taken into account in this analysis which are relate to the maintenance and deployment of the off-chain oracle, something that is taken care by the third-party oracle provider. However, this off-chain oracle can be also be self deployed or even if deployed on a cloud provider the cost is insignificant in comparison to the cost of each request made to the on-chain oracle as demonstrated in Figure 6.2.

All in all, this Chapter alongside the corresponding Appendixes, B C D, present a simple but effective way of deploying a oracle platform, on-chain and off-chain, that easily abd cost effectively can support and further empower existing and future smart-contract scenarios with trust in mind.

Chapter 7

Validation

In this Chapter, the author seeks to demonstrate how the architectures and prototype detailed in Chapters 5 and 6 provide answers to the desiderata stated in Chapter 4.

Initially, the author compares the defined architectures with existing solutions and how broadly they describe all possible scenarios.

Then, the implemented solution in comparison to the state of the art, as well as its applicability, use case scenarios and limitations.

7.1 Oracle Architectures

In this section, the author will validate each architecture against the forces defined previously in the desiderata, in Section 4.2.

7.1.1 Oracle as a Service w/ Single Data Feed

This architecture achieves a **fast time-to-market**, allowing the developer of the smart-contract to quickly retrieve data from the web by plugin-in an oracle-as-a-service solution. Thus, not requiring any study and development and maintenance of an oracle service, quickly satisfying the contract data needs.

In terms of **keeping trust standards**, this type of architectures usually involves the use of trusted hardware or authenticity proofs to achieve such end. There are some caveats associated with some of them, such as the most widely used one, TLSNotary, but others such as the Android Proof and the use of trusted hardware may suffice to guarantee trust in the oracle service if their implementation and proofs are readily and transparently available. This proofs are managed by the third-party provider and therefore are not an overhead in achieving a fast time-to-market.

7.1.2 Oracle as a Service w/ Multiple Data Feeds

Iterating on the previous architecture, this one upholds the same characteristics of the previous one, **fast time-to-market** and **keeping trust standards**.

However, in this scenario, we don't fully trust the data-source. Trust here, has two components. Trusting the data-source availability (uptime) and trusting the veracity of the data provided. By querying multiple data-feeds and setting a quorum both problems are solved. If some data providers fail to answer the request and the minimum quorum upholds the contract can still perform its duties. And if we cannot trust a single party to provide the data, as doing so will give full power to the outcome of the contract to that single data provider, the problem is solved by having a minimum quorum that can provide the same answer. This way, an outlier in the selected data provider is discarded. Guaranteeing both **data-feed fault tolerance** and **data veracity**.

7.1.3 Single-Party Self Hosted Oracle

This scenario takes a different approach, from the two previous ones, in terms of where to place the trust. In the previous scenario, due to the use of a third-party, it is required that some sort of proof is generated in order to guarantee the good behaviour of the third party. However, these proofs do add extra overhead to each request in terms of memory and computation used. In a scenario where the smart contract execution will only affect a single user, the owner of the client smart contract, and he is able to develop and maintain its own oracle, then it will become a more viable approach to use a self-hosted oracle approach than to use a third-party oracle as a service. Since it is own code that is being executed and he controls the environment of execution he can have full trust in the oracle behaviour, not depending therefore on any extra mechanism.

In a scenario wherein a set of predefined parties are stakeholders in the smart-contract execution, it is possible to skip the use of authenticity proofs by having all the parties contributing to the oracle execution. With such approach, we get to **keep the trust standards** of the execution of the smart-contract as a whole.

However, this approach introduces increases the time-to-market as it requires to develop and maintain the oracle service.

Additionally, it reduces the costs of the oracle operation by removing the extra fee from each query. Also, as there is no need for authenticity proofs, the contract code becomes simpler and the storage costs decrease as well leading to **lower smart contract costs**. In fact, due to being purpose-built, these contracts should have less executable code in comparison to contracts that need to fulfil a bigger list of requisites, **lower oracle complexity**, and therefore become cheaper contracts.

7.1.4 Multi-Party Self Hosted Oracle

Iterating on the previous scenario, this architecture addresses the issue of having several parties depending on the execution of the client-smart contract. Here a predefined set of parties are stakeholders in the smart-contract execution, it is, therefore, possible to skip the use of authenticity

proofs by having all the parties contributing to the oracle execution. Each party can deploy their own off-chain oracle and the on-chain oracle smart contract will serve as the contract to which all parties accept to abide by. Having all the parties identified in the contract, for example with the account address, then only they can make transactions that will be accepted in the smart contract. With such approach we get to **keep the trust standards** of the execution of the smart-contract as a whole, as well as, **oracle decentralization** and **oracle ownership decentralization**, by having each party contributing to the final result by launching their own oracle and having the same say in the final result. Having higher oracle availability and trust in its behaviour by having multiple stakeholders performing the same job. It also allows for **data-feed fault tolerance** and **data veracity** if this quorum of oracles query different data-sources or being the source of information.

7.1.5 Conclusions

Having addressed these architectures in terms of a predefined set of values, that are expected to be upheld in the operation of an oracle, it is possible to understand how each architecture addresses different concerns and the implied use cases. Table 7.1 summarises the architectural forces according to each architecture.

	OaaS w/SDF 7.1.1	OaaS w/MDF 7.1.2	SP-SHO 7.1.3	MP-SHO 7.1.4
Fast time-to-market	X	X		
Keeping trust standards	X	X	X	X
Data-feed fault tolerance		X		X
Data veracity		X		X
Lower smart contract costs			X	X
Lower oracle complexity			X	X
Oracle decentralization				X
Oracle ownership decentralization				X

Table 7.1: Summary of architecture forces

As the Table 7.1 visually points out, no architecture can satisfy all requirements, which can be easily understood. For example, in a self-hosted scenario it is always impractical to achieve **fast time-to-market** as it is necessary to develop, study and deploy the whole solution.

Also, **lower smart contract costs** requires further analysis. The author, considered that the self-hosted option is cheaper comparing to the use of a third-party provider by considering only the costs of running the oracle and not the costs of paying developers and infrastructure. Still on the topic of costs, the architecture MP-SHO 7.1.4 can only be assumed to have lower costs than the as a service approach since it does not require complex contracts or paying fees. But it might be more expensive than the architecture OaaS w/SDF 7.1.1 if the number of oracles deployed is

too large in comparison to the use of a single oracle in the latter mentioned architecture. As for each oracle, the cost of operations increases, as depicted on Figure 6.2.

Finally, analysing the factor **keeping trust standards** all architectures checked this boxes, as oracles-as-a-service have at their disposal authenticity proofs, such as TLS-N and Town Crier, that can provide a trustable enough proofs for certain scenarios. It might not be the ones that their are using now, but looking from an architecture standpoint they can achieve this as well.

7.2 Self-hosted Oracle Implementation

The author proposes three main characteristics of its implementation comparing to the current existing oracle-as-a-service solutions. These are reduced costs, higher trust and higher contract empowerment.

7.2.1 Reduced costs

In this context, cost per query compromises multiple dimensions. Firstly, the cost of querying the oracle and inputting the result in the contract which corresponds to the execution of the contract code and is therefore directly related to the amount of code that needs to run. Secondly, underlying fees imposed by the third-party service. And finally, a smaller cost but still worth mentioning, the cost of the off-chain oracle service that will query the API.

Analysing the first one, the contract executing cost paid by the caller, that is not much that the developer of the smart contract can do to optimize this since it is fully managed by the third-party service. Hence, on a self-deployed oracle, the cost can be further optimized by modelling a single purpose oracle for the smart contract needs, which will inherently run less code due to its simplicity. Also, on a self-hosted oracle there is no need to add the over-head of authenticity proofs which either verified on-chain or partially stored off-chain lead to higher transaction costs.

Secondly, existing services are for-profit companies and therefore require an extra-payment for their service. Oraclize.it adds an extra fee paid in dollars, depicted on Table 7.2, that depends on the data source and authenticity proof used. Chainlink requires that every request is paid using its token LINK whose value depends on the current market price. In a self-deployed oracle approach, none of these fees are present leading therefore to lower costs.

		Proof type			
Datasource	Base price	None	TLSNotary	Android	Ledger
URL	0.01$	+0.0$	+0.04$	+0.04$	N/A
WolframAlpha	0.03$	+0.0$	N/A	N/A	N/A
IPFS	0.01$	+0.0$	N/A	N/A	N/A
random	0.05$	+0.0$	N/A	N/A	+0.0$
computation	0.50$	+0.0$	+0.04$	+0.04$	N/A

Table 7.2: Oraclize fees in USD

Finally, in a self-deployed oracle scenario, there are inherent costs of running the off-chain oracle which, in the oracle-as-a-service scenario these are taken care of by a third-party service. Although the cost per transaction of the service depends on the platform in which the service will be deployed it can be assumed that in comparison to the fees or, even more, to the cost of executing the smart contract code this cost is risible. Solutions such as AWS Lambda [1] that offer 1M requests and 400,000 GB-SECONDS of compute time per month for free in their free tier [2], and even in a scaling scenario each request costs $0.0000002. Therefore, this cost is not considered throughout this dissertation due to its small size in comparison with the previous analysed ones.

7.2.2 Higher trust

Trust is defined as having complete certainty that the provided answer is corrected or was indeed the one provided by the API. In the self-hosted oracle scenario, both can be achieved. The first proposition, that the answer is correct, can be maximized by using multiple oracles and a quorum so that multiple sources can confirm the requested result. With minor alterations, the oracle could receive more than one URL and maximize even more the trust in the result by querying multiple sources. The second, that the API actually return that value is achieved since the off-chain oracle is fully controlled by the parties interested in the result of the smart-contract and therefore know exactly the code being executed.

This approach, comparing to the state-of-the-art found solutions, although simple provides higher guarantees that the smart contract will receive the desired answer. In the currently existing solutions, trust is ultimately achieved through the use of authenticity proofs, which, as analysed in Chapter 3, do not provide the necessary guarantees. Either by not being able to be analysed on the chain contract and can only be later inspected. And also, their implementations can be dubious, as they are being managed by a third party and always require to trust in a higher entity such as the service where they are being deployed.

[1] More information about the lambda service can be found here: https://aws.amazon.com/lambda/

[2] The pricing for lambda was queried here: https://aws.amazon.com/lambda/pricing/ on the 29th of May 2019.

7.2.3 Higher contract empowerment

The presented implementation provides a great starting point to be worked upon and tailored to specific contract needs. At the moment, can already work with any JSON API and therefore be used in a wide range of applications. Being able to use and tailor, with minimum effort, the existing boilerplate and adapt to its needs. When using third-party services there's no such flexibility. Even in terms of which blockchain you can deploy the contract, some services are not yet available on some blockchains' mainnet, and features available are totally dependent on the oracle service provider.

7.2.4 Conclusions

This implementation intends to be a simple, cost-effective boilerplate that can already service a range of different contracts by allowing to query any JSON API and accomplishes exactly that purpose. Furthermore, implement already an important feature, allowing the outcome of each request made from a client smart contract to the oracle smart contract to require a minimum quorum of off-chain oracles responses. This way increasing the infrastructure availability and decentralize its ownership.

With this in mind, this implementation fully accomplishes the objectives proposed by the author.

Chapter 8

Conclusions and Future Work

This chapter reflects on the entire work of this dissertation. Looking at the contributions of the work performed, difficulties and possible paths for improvement and work that can still be done.

8.1 Difficulties

One of the main difficulties around this topic is the complexity of the underlying system, the blockchain. In order to first understand the purpose of oracles, it is required to understand in depth the underlying blockchain. Each blockchain presents a different paradigm, programming language and challenges, and although the oracle problem usually transverses all of them or at least the most used ones, it still requires some adaptation. For this specific work, the author chose the Ethereum for its huge adoption and smart contract features but others, such as the EOS blockchain could have been used.

The other main difficulty is around the oracle topic itself. There's not a lot of documentation around it, especially in the blockchains documentation pages. The only information found is on a few papers and two companies which seem to dominate the market of oracles-as-a-service and are not so transparent in the actual implementation or proof verification.

In the end, in terms of oracle implementation, the author learned from the combined knowledge of a few articles from independent developers who tried to share their knowledge on how they addressed their oracle implementation.

8.2 Contributions

With this dissertation, the author intends to pave the way for the development of secure oracle architectures and implementations.

The author tries to demystify the existing authenticity proofs, explaining their inner workings and limitations so future developers can be more aware of their usage by the third party oracle providers.

The author also presents several architectures and explains their points of trust and how to address each of those points. Also the weaknesses and strengths of each architecture so future developers can have a guided approach to choose the best model for their use case.

Finally, due to the lack of documentation around building a self-hosted oracle the author provides a simple boilerplate that can be easily used and even worked upon and tailor to the needs of the smart contract. The author even adds in this implementation the necessary logic for having the oracle being used and deployed by multiple competing parties allowing them to trust in the oracle execution even if they don't trust the other involved parties. This implementation, however, is directed for the Ethereum blockchain, but the author believes that its logic can be ported to other blockchains effortlessly.

8.3 Future Work

In terms of future work, it would be interesting to analyse use cases of the presented architectures. As of now, the author could not find examples to fit all of them and therefore, the author was not confident enough to call them patterns. This work would also help in the validation of the assertions made about each architecture.

In terms of the implementation of a self-hosted oracle, the author would like to see some work done in terms of designing a self-hosted that could be applied to a community problem. This means, to a scenario where the stakeholders of the oracle execution are not predefined and can enter or leave the network, more similar to the way blockchain works. Due to time limitations, the author could not think of a feasible way of achieving this without using some kind of proof-of-work, similarly to the blockchain.

8.4 Conclusions

All in all, this dissertation allowed the author to gain a broader knowledge of cryptography, consensus mechanisms and blockchain technology. The author also, truly believes that this dissertation is paving the way for future applications of smart-contracts and blockchain technology. And will help developers who may be struggling using oracles and, unnecessarily, will recur to third-party providers having to support extra costs in their product.

References

[ABV+18a] John Adler, Ryan Berryhill, Andreas Veneris, Zissis Poulos, Neil Veira, and Anastasia Kastania. Astraea: A Decentralized Blockchain Oracle. aug 2018.

[ABV+18b] John Adler, Ryan Berryhill, Andreas Veneris, Zissis Poulos, Neil Veira, and Anastasia Kastania. ASTRAEA: A Decentralized Blockchain Oracle. Technical report, 2018.

[Blo18] Block.one. EOS.IO Technical White Paper v2, 2018.

[CZK+18] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah M. Johnson, Ari Juels, Andrew K. Miller, and Dawn Xiaodong Song. Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contract Execution. *undefined*, 2018.

[EH18] Jacob Eberhardt and Jonathan Heiss. Off-chaining Models and Approaches to Off-chain Computations. In *Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers - SERIAL'18*, pages 7–12, New York, New York, USA, 2018. ACM Press.

[EJN17] Steve Ellis, Ari Juels, and Sergey Nazarov. ChainLink A Decentralized Oracle Network. Technical report, 2017.

[Gav14] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. Technical report, Ethereum, 2014.

[Gor17] Gilroy Gordon. Provenance and authentication of oracle sensor data with block chain lightweight wireless network authentication scheme for constrained oracle sensors. 2017.

[KKC07] B. Kitchenham, B. Kitchenham, and S Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. 2007.

[Mer79] Method of providing digital signatures. sep 1979.

[Mon18] Francisco Javier Andrés Montoto Monroy. Bitcoin gambling using distributed oracles in the blockchain. 2018.

[Nak09] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. Technical report, Bitcoin, 2009.

[Ora] A Scalable Architecture for On-Demand Untrusted Delivery of Entropy. Technical report, Oraclize.

[RWG+17a] Hubert Ritzdorf, Karl Wüst, Arthur Gervais, Guillaume Felley, and Srdjan Capkun. TLS-N: Non-repudiation over TLS Enabling - Ubiquitous ContentSigning for Disintermediation. *IACR Cryptology ePrint Archive*, 2017(578), 2017.

[RWG+17b] Hubert Ritzdorf, Karl Wüst, Arthur Gervais, Guillaume Felley, and Srdjan Capkun. TLS-N: Non-repudiation over TLS Enabling - Ubiquitous ContentSigning for Disintermediation. *IACR Cryptology ePrint Archive*, 2017(578), 2017.

[Tam18] Andrew Tam. Secret Voting Smart Contract with Enigma: A Walkthrough, 2018.

[TLS14] TLSnotary-a mechanism for independently audited https sessions. Technical report, TLSnotary, 2014.

[TR17] Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. Technical report, 2017.

[Vit14] Vitalik Buterin. SchellingCoin: A Minimal-Trust Universal Data Feed, 2014.

[ZCC+16] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town Crier: An Authenticated Data Feed for Smart Contracts. Technical report, 2016.

Appendix A

SLR Screening Stages

3rd screen	2nd screen	1st Screen	Remove Duplicates	Source	Year	Title	Authors
			Duplicate	ACM	2016	Weaver: A High-performance, Transactional Graph Database Based on Refinable Timestamps	Ayush Dubey and Greg D. Hill and Robert Escriva and Emin Gün Sirer
			Duplicate	ACM	2016	Town Crier: An Authenticated Data Feed for Smart Contracts	Fan Zhang and Ethan Cecchetti and Kyle Croman and Ari Juels and Elaine Shi
			Duplicate	ACM	2016	Proof of Luck: An Efficient Blockchain Consensus Protocol	Mitar Milutinovic and Warren He and Howard Wu and Maxinder Kanwal
			Duplicate	ACM	2017	PlaTIBART: A Platform for Transactive IoT Blockchain Applications with Repeatable Testing	Michael A. Walker and Abhishek Dubey and Aron Laszka and Douglas C. Schmidt
			Duplicate	ACM	2018	Ouroboros Genesis: Composable Proof-of-Stake Blockchains with Dynamic Availability	Christian Badertscher and Peter Gaži and Aggelos Kiayias and Alexander Russell and Vassilis Zikas
			Duplicate	ACM	2017	On the Design of Communication and Transaction Anonymity in Blockchain-based Transactive Microgrids	Jonatan Bergquist and Aron Laszka and Monika Sturm and Abhishek Dubey
			Duplicate	ACM	2017	FruitChains: A Fair Blockchain	Rafael Pass and Elaine Shi
			Duplicate	ACM	2018	ContractFuzzer: Fuzzing Smart Contracts for Vulnerability Detection	Bo Jiang and Ye Liu and W. K. Chan

Table A.1 continued from previous page

3rd screen	2nd screen	1st Screen	Remove Duplicates	Source	Year	Title	Authors
			Duplicate	ACM	2016	Bringing Secure Bitcoin Transactions to Your Smartphone	Davide Frey and Marc X. Makkes and Pierre-Louis Roman and François Taïani and Spyros Voulgaris
			Duplicate	ACM	2017	Blackchain: Scalability for Resource-constrained Accountable Vehicle-to-x Communication	Rens W. van der Heijden and Felix Engelmann and David Mödinger and Franziska Schönig and Frank Kargl
			Duplicate	ACM	2017	A General Framework for Blockchain Analytics	Massimo Bartoletti and Stefano Lande and Livio Pompianu and Andrea Bracciali
			Duplicate	ACM	2017	EPBC: Efficient Public Blockchain Client for Lightweight Users	Lei Xu and Lin Chen and Zhimin Gao and Shouhuai Xu and Weidong Shi
			Duplicate	ACM	2016	Blockchains and the Logic of Accountability: Keynote Address	Maurice Herlihy and Mark Moir
			Duplicate	ACM	2017	A Byzantine Fault-tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform	Alysson Bessani and João Sousa and Marko Vukolić
			Duplicate	IEEE	2018	Zero-Trust Hierarchical Management in IoT	M. Samaniego; R. Deters
			Duplicate	IEEE	2018	Secure Pub-Sub: Blockchain-Based Fair Payment With Reputation for Reliable Cyber Physical Systems	Y. Zhao; Y. Li; Q. Mu; B. Yang; Y. Yu

Table A.1 continued from previous page

3rd screen	2nd screen	1st Screen	Remove Duplicates	Source	Year	Title	Authors
			Duplicate	IEEE	2018	Secure Attribute-Based Signature Scheme With Multiple Authorities for Blockchain in Electronic Health Records Systems	R. Guo; H. Shi; Q. Zhao; D. Zheng
			Duplicate	IEEE	2018	Privacy Improvement Architecture for IoT	E. Kak; R. Orji; J. Pry; K. Sofranko; R. Lomotey; R. Deters
			Duplicate	IEEE	2018	Distributed Solar Self-Consumption and Blockchain Solar Energy Exchanges on the Public Grid Within an Energy Community	C. Plaza; J. Gil; F. de Chezelles; K. A. Strang
			Duplicate	IEEE	2018	Confidential Business Process Execution on Blockchain	B. Carminati; C. Rondanini; E. Ferrari
			Duplicate	IEEE	2018	ChainFS: Blockchain-Secured Cloud Storage	Y. Tang; Q. Zou; J. Chen; K. Li; C. A. Kamhoua; K. Kwiat; L. Njilla
			Duplicate	IEEE	2018	Blockchain-Based IoT-Cloud Authorization and Delegation	N. Tapas; G. Merlino; F. Longo
			Duplicate	IEEE	2017	Blockchain world - Do you need a blockchain? This chart will tell you if the technology can solve your problem	M. E. Peck
			Duplicate	IEEE	2018	Blockchain as a Platform for Secure Inter-Organizational Business Processes	B. Carminati; E. Ferrari; C. Rondanini
			Duplicate	IEEE	2018	Analysis of Security in Blockchain: Case Study in 51%-Attack Detecting	C. Ye; G. Li; H. Cai; Y. Gu; A. Fukuda

Table A.1 continued from previous page

3rd screen	2nd screen	1st Screen	Remove Duplicates	Source	Year	Title	Authors
			Duplicate	IEEE	2018	An ID-Based Linearly Homomorphic Signature Scheme and Its Application in Blockchain	Q. Lin; H. Yan; Z. Huang; W. Chen; J. Shen; Y. Tang
			Duplicate	IEEE	2019	A New Lattice-Based Signature Scheme in Post-Quantum Blockchain Network	C. Li; X. Chen; Y. Chen; Y. Hou; J. Li
			Duplicate	Scopus	2017	Towards an economic analysis of routing in payment channel networks	Engelmann, F., Kopp, H., Kargl, F., Glaser, F., Weinhardt, C.
			Duplicate	Scopus	2018	13th EAI International Conference on Security and Privacy in Communication Networks, SecureComm 2017	[No author name available]
			Duplicate	Scopus	2017	VIBES: Fast blockchain simulations for large-scale peer-to-peer networks	Stoykov, L., Zhang, K., Jacobsen, H.-A.
			Duplicate	Scopus	2017	HyperPubSub: a decentralized, permissioned, publish/subscribe service using blockchains	Zupan, N., Zhang, K., Jacobsen, H.-A.
			Duplicate	Scopus	2018	Blockchain as a platform for secure inter-organizational business processes	Carminati, B., Ferrari, E., Rondanini, C.
		-		ACM	2017	Towards an Economic Analysis of Routing in Payment Channel Networks	Felix Engelmann and Henning Kopp and Frank Kargl and Florian Glaser and Christof Weinhardt
		-		ACM	2017	VIBES: Fast Blockchain Simulations for Large-scale Peer-to-peer Networks: Demo	Lyubomir Stoykov and Kaiwen Zhang and Hans-Arno Jacobsen
		-		ACM	2018	StreamChain: Do Blockchains Need Blocks?	Zsolt István and Alessandro Sorniotti and Marko Vukolić

Table A.1 continued from previous page

3rd screen	2nd screen	1st Screen	Remove Duplicates	Source	Year	Title	Authors
		-		ACM	2018	Sol2Js: Translating Solidity Contracts into Javascript for Hyperledger Fabric	Muhammad Ahmad Zafar and Falak Sher and Muhammad Umar Janjua and Salman Baset
		-		ACM	2018	Scaling Byzantine Consensus: A Broad Analysis	Christian Berger and Hans P. Reiser
		-		ACM	2018	Resource Fairness and Prioritization of Transactions in Permissioned Blockchain Systems (Industry Track)	Seep Goel and Abhishek Singh and Rachit Garg and Mudit Verma and Praveen Jayachandran
		-		ACM	2018	Powering Software Sustainability with Blockchain	Omar Badreddin
		-		ACM	2017	Hyperpubsub: A Decentralized, Permissioned, Publish/Subscribe Service Using Blockchains: Demo	Nejc Zupan and Kaiwen Zhang and Hans-Arno Jacobsen
		-		ACM	2017	How Blockchains Can Help Legal Metrology	Wilson S. Melo,Jr and Alysson Bessani and Luiz F. R. C. Carmo
		-		ACM	2018	eVIBES: Configurable and Interactive Ethereum Blockchain Simulation Framework	Aditya Deshpande and Pezhman Nasirifard and Hans-Arno Jacobsen
		-		ACM	2018	EVA: Fair and Auditable Electric Vehicle Charging Service Using Blockchain	Jelena Pajic and José Rivera and Kaiwen Zhang and Hans-Arno Jacobsen
		-		ACM	2018	Deconstructing Blockchains: Concepts, Systems, and Insights	Kaiwen Zhang and Roman Vitenberg and Hans-Arno Jacobsen

Table A.1 continued from previous page

3rd screen	2nd screen	1st Screen	Remove Duplicates	Source	Year	Title	Authors
		-		ACM	2018	CIDDS: A Configurable and Distributed DAG-based Distributed Ledger Simulation Framework	Mohamed Riswan Abdul Lathif and Pezhman Nasirifard and Hans-Arno Jacobsen
		-		ACM	2018	Blockchains for Business Process Management - Challenges and Opportunities	
		-		ACM	2018	Blockchain Landscape and AI Renaissance: The Bright Path Forward	Hans-Arno Jacobsen and Mohammad Sadoghi and Mohammad Hossein Tabatabaei and Roman Vitenberg and Kaiwen Zhang
		-		ACM	2018	A Federated Low-Power WAN for the Internet of Things	Mehdi Bezahaf and Gaëtan Cathelain and Tony Ducrocq
		-		ACM	2018	Authenticated Modular Maps in Haskell	Victor Cacciari Miraldo and Harold Carr and Alex Kogan and Mark Moir and Maurice Herlihy
		-		ACM	2018	Attack and Vulnerability Simulation Framework for Bitcoin-like Blockchain Technologies	Fabian Schüssler and Pezhman Nasirifard and Hans-Arno Jacobsen
		-		Google Shoolar	2017	Blockchain Oracles–Einsatz der Blockchain-Technologie für Offline-Anwendungen	A Hoppe
		-		Google Shoolar	2018	Blockchain Coupled Oracle Fusion	D Satpathy
		-		Google Shoolar	2018	Blockchain and Consensus from Proofs of Work without Random Oracles	JA Garay, A Kiayias, G Panagiotakos

Table A.1 continued from previous page

3rd screen	2nd screen	1st Screen	Remove Duplicates	Source	Year	Title	Authors
		-		Google Shoolar	2018	Blockchain across Oracle: Understand the details and implications of the Blockchain for Oracle developers and customers	R van Mölken
		-		IEEE	2018	Understanding Blockchain Technology: The Costs and Benefits of Decentralization	
		-		IEEE	2018	Towards Application Portability on Blockchains	K. Shudo; R. Kanda; K. Saito
		-		IEEE	2017	Secure one-time biometrie tokens for non-repudiable multi-party transactions	K. Nandakumar; N. Ratha; S. Pankanti; S. Darnell
		-		IEEE	2017	Multiclouds in an Enterprise – a Love-Hate Relationship	M. Yousif
		-		IEEE	2019	Leveraging the Capabilities of Industry 4.0 for Improving Energy Efficiency in Smart Factories	N. Mohamed; J. Al-Jaroodi; S. Lazarova-Molnar
		-		IEEE	2017	Fostering consumers' energy market through smart contracts	I. Kounelis; G. Steri; R. Giuliani; D. Geneiatakis; R. Neisse; I. Nai-Fovino
		-		IEEE	2018	ChainMOB: Mobility Analytics on Blockchain	B. Nasrulin; M. Muzammal; Q. Qu
		-		IEEE	2016	Blockchains and the logic of accountability	M. Herlihy; M. Moir
		-		IEEE	2018	Blockchain Based Security Framework for IoT Implementations	K. N. Krishnan; R. Jenu; T. Joseph; M. L. Silpa
		-		IEEE	2018	Blockchain Based Vehicular Data Management	R. Sharma; S. Chakraborty
		-		Scopus	2016	Weaver: A high-performance, transactional graph database based on refinable timestamps	Dubey, A., Hill, G.D., Sirer, E.G., Escriva, R.

Table A.1 continued from previous page

3rd screen	2nd screen	1st Screen	Remove Duplicates	Source	Year	Title	Authors
		-		Scopus	2018	Towards a smart contract-based, decentralized, public-key infrastructure	Patsonakis, C., Samari, K., Roussopoulos, M., Kiayias, A.
		-		Scopus	2016	SysTEX 2016 - 1st Workshop on System Software for Trusted Execution, colocated with ACM/IFIP/USENIX Middleware 2016	[No author name available]
		-		Scopus	2018	Systematic performance evaluation using component-in-the-loop approach	Kocsis, I., Klenik, A., Pataricza, A., Telek, M., Deé, F., Cseh, D.
		-		Scopus	2018	Synchronized aggregate signatures from the RSA assumption	Hohenberger, S., Waters, B.
		-		Scopus	2018	Simple proofs of sequential work	Cohen, B., Pietrzak, K.
		-		Scopus	2017	SERIAL 2017 - 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers, Colocated with ACM/IFIP/USENIX Middleware 2017 Conference	[No author name available]
		-		Scopus	2018	Security of the blockchain against long delay attack	Wei, P., Yuan, Q., Zheng, Y.
		-		Scopus	2018	Secure Pub-Sub: Blockchain-Based Fair Payment with Reputation for Reliable Cyber Physical Systems	Zhao, Y., Li, Y., Mu, Q., Yang, B., Yu, Y.
		-		Scopus	2018	Secure Attribute-Based Signature Scheme with Multiple Authorities for Blockchain in Electronic Health Records Systems	Guo, R., Shi, H., Zhao, Q., Zheng, D.

Table A.1 continued from previous page

3rd screen	2nd screen	1st Screen	Remove Duplicates	Source	Year	Title	Authors
		-		Scopus	2017	RingCT 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency Monero	Sun, S.-F., Au, M.H., Liu, J.K., Yuen, T.H.
		-		Scopus	2016	Proof of Luck: An efficient blockchain consensus protocol	Milutinovic, M., He, W., Wu, H., Kanwal, M.
		-		Scopus	2018	Privacy improvement architecture for IoT	Kak, E., Orji, R., Pry, J., Sofranko, K., Lomotey, R.K., Deters, R.
		-		Scopus	2017	PlaTIBART: A Platform for Transactive IoT blockchain applications with repeatable testing	Walker, M.A., Dubey, A., Laszka, A., Schmidt, D.C.
		-		Scopus	2017	Overcoming Cryptographic Impossibility Results Using Blockchains	Goyal, R., Goyal, V.
		-		Scopus	2018	Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain	David, B., Gaži, P., Kiayias, A., Russell, A.
		-		Scopus	2017	On the design of communication and transaction anonymity in blockchain-based transactive microgrids	Bergquist, J., Laszka, A., Sturm, M., Dubey, A.
		-		Scopus	2017	Middleware 2017 - Proceedings of the 2017 Middleware Posters and Demos 2017: Proceedings of the Posters and Demos Session of the 18th International Middleware Conference	[No author name available]

Table A.1 continued from previous page

3rd screen	2nd screen	1st Screen	Remove Duplicates	Source	Year	Title	Authors
		-		Scopus	2017	M4IoT 2017 - Proceedings of the 2017 Workshop on Middleware and Applications for the Internet of Things 4th Edition and 2nd Federated Event with the MoTA Workshop, Part of Middleware 2017 Conference	[No author name available]
		-		Scopus	2018	IoTBDS 2018 - Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security	[No author name available]
		-		Scopus	2018	Introducing the new paradigm of Social Dispersed Computing: Applications, Technologies and Challenges	García-Valls, M., Dubey, A., Botti, V.
		-		Scopus	2017	FruitChains: A fair blockchain	Pass, R., Shi, E.
		-		Scopus	2017	EPBC: Efficient Public Blockchain Client for Lightweight Users	Xu, L., Chen, L., Gao, Z., Xu, S., Shi, W.
		-		Scopus	2018	Distributed Solar Self-Consumption and Blockchain Solar Energy Exchanges on the Public Grid Within an Energy Community	Plaza, C., Gil, J., De Chezelles, F., Strang, K.A.
		-		Scopus	2018	Designing blockchain-based SIEM 3.0 system	Miloslavskaya, N.
		-		Scopus	2018	ChainFS: Blockchain-Secured Cloud Storage	Tang, Y., Zou, Q., Chen, J., Li, K., Kamhoua, C.A., Kwiat, K., Njilla, L.
		-		Scopus	2016	Bringing secure Bitcoin transactions to your smartphone	Frey, D., Makkes, M.X., Roman, P.-L., Taïani, F., Voulgaris, S.

Table A.1 continued from previous page

3rd screen	2nd screen	1st Screen	Remove Duplicates	Source	Year	Title	Authors
		-		Scopus	2015	Blockchain-based model for social transactions processing	Sarr, I., Naacke, H., Gueye, I.
		-		Scopus	2018	Blockchain-Based IoT-cloud authorization and delegation	Tapas, N., Merlino, G., Longo, F.
		-		Scopus	2017	Blockchain world - Do you need a blockchain? This chart will tell you if the technology can solve your problem	Peck, M.E.
		-		Scopus	2017	Blackchain: Scalability for resource-constrained accountable vehicle-to-x communication	Van Der Heijden, R.W., Engelmann, F., Mödinger, D., Schönig, F., Kargl, F.
		-		Scopus	2017	Beyond hellman's time-memory trade-offs with applications to proofs of space	Abusalah, H., Alwen, J., Cohen, B., Khilko, D., Pietrzak, K., Reyzin, L.
		-		Scopus	2017	Analysis of the blockchain protocol in asyn-chronous networks	Pass, R., Seeman, L., Shelat, A.
		-		Scopus	2018	Analysis of security in blockchain: Case study in 51%-attack detecting	Ye, C., Li, G., Cai, H., Gu, Y., Fukuda, A.
		-		Scopus	2018	An integrated platform for the Internet of Things based on an open source ecosystem	Li, Y.Q.
		-		Scopus	2018	An ID-Based Linearly Homomorphic Signature Scheme and Its Application in Blockchain	Lin, Q., Yan, H., Huang, Z., Chen, W., Shen, J., Tang, Y.
		-		Scopus	2019	A New Lattice-Based Signature Scheme in Post-Quantum Blockchain Network	Li, C.-Y., Chen, X.-B., Chen, Y.-L., Hou, Y.-Y., Li, J.

Table A.1 continued from previous page

3rd screen	2nd screen	1st Screen	Remove Duplicates	Source	Year	Title	Authors
		-		Scopus	2017	A general framework for blockchain analytics	Bartoletti, M., Lande, S., Pompianu, L., Bracciali, A.
		-		Scopus	2018	A critical look at cryptogovernance of the real world: Challenges for spatial representation and uncertainty on the blockchain	Adams, B., Tomko, M.
		-		Scopus	2017	A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform (Short Paper)	Bessani, A., Sousa, J., Vukolić, M.
		-		Scopus	2017	4th International Conference on Future Data and Security Engineering, FDSE 2017	[No author name available]
		-		Scopus	2018	3rd International Conference on Internet of Things, ICIOT 2018 Held as Part of the Services Conference Federation, SCF 2018	[No author name available]
		-		Scopus	2017	36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT 2017	[No author name available]
		-		Scopus	2018	21 - Bringing down the complexity: Fast composable protocols for card games without secret state	David, B., Dowsley, R., Larangeira, M.
		-		Scopus	2018	13th EAI International Conference on Security and Privacy in Communication Networks, SecureComm 2017	[No author name available]

Table A.1 continued from previous page

3rd screen	2nd screen	1st Screen	Remove Duplicates	Source	Year	Title	Authors
		-		Scopus	2017	11th International Conference on Provable Security, ProvSec 2017	[No author name available]
	-	pass		ACM	2018	Towards Solving the Data Availability Problem for Sharded Ethereum	Daniel Sel and Kaiwen Zhang and Hans-Arno Jacobsen
	-	pass		Google Shoolar	2018	Trusted agent blockchain oracle	MD Jackson
	-	pass		IEEE	2018	Towards Distributed SLA Management with Smart Contracts and Blockchain	R. B. Uriarte; R. de Nicola; K. Kritikos
	-	pass		Scopus	2018	Zero-trust hierarchical management in IoT	Samaniego, M., Deters, R.
	-	pass		Scopus	2018	The interface between blockchain and the real world	Damjan, M.
	-	pass		Scopus	2018	Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability	Badertscher, C., Gaži, P., Kiayias, A., Russell, A., Zikas, V.
	-	pass		Scopus	2018	ContractFuzzer: Fuzzing smart contracts for vulnerability detection	Jiang, B., Liu, Y., Chan, W.K.
-	pass	pass		Scopus	2018	Confidential Business Process Execution on Blockchain	Carminati, B., Rondanini, C., Ferrari, E.
pass	pass	pass		ACM	2018	Off-chaining Models and Approaches to Off-chain Computations	Jacob Eberhardt and Jonathan Heiss
pass	pass	pass		Google Shoolar	2018	Astraea: A decentralized blockchain oracle	J Adler, R Berryhill, A Veneris, Z Poulos, N Veira...

Table A.1 continued from previous page

3rd screen	2nd screen	1st Screen	Remove Duplicates	Source	Year	Title	Authors
pass	pass	pass		Google Shoolar	2017	Provenance and authentication of oracle sensor data with block chain lightweight wireless network authentication scheme for constrained oracle sensors	G Gordon
pass	pass	pass		Google Shoolar	2018	Bitcoin gambling using distributed oracles in the blockchain	FJA Montoto Monroy
pass	pass	pass		Scopus	2016	Town crier: An authenticated data feed for smart contracts	Zhang, F., Cecchetti, E., Croman, K., Juels, A., Shi, E.

Appendix B

On-Chain Oracle Code

```
1   pragma solidity >=0.4.21 <0.6.0;
2
3   contract Oracle {
4     Request[] requests; //list of requests made to the contract
5     uint currentId = 0; //increasing request id
6     uint minQuorum = 2; //minimum number of responses to receive before declaring
       final result
7     uint totalOracleCount = 3; // Hardcoded oracle count
8
9     // defines a general api request
10    struct Request {
11      uint id;                            //request id
12      string urlToQuery;                  //API url
13      string attributeToFetch;            //json attribute (key) to retrieve in
      the response
14      string agreedValue;                 //value from key
15      mapping(uint => string) anwers;     //answers provided by the oracles
16      mapping(address => uint) quorum;    //oracles which will query the answer
      (1=oracle hasn't voted, 2=oracle has voted)
17    }
18
19    //event that triggers oracle outside of the blockchain
20    event NewRequest (
21      uint id,
22      string urlToQuery,
23      string attributeToFetch
24    );
25
26    //triggered when there's a consensus on the final result
27    event UpdatedRequest (
28      uint id,
29      string urlToQuery,
30      string attributeToFetch,
```

```
31          string agreedValue
32      );
33
34      function createRequest (
35          string memory _urlToQuery,
36          string memory _attributeToFetch
37      )
38      public
39      {
40          uint lenght = requests.push(Request(currentId, _urlToQuery,
    _attributeToFetch, ""));
41          Request storage r = requests[lenght-1];
42
43          // Hardcoded oracles address
44          r.quorum[address(0x6c2339b46F41a06f09CA0051ddAD54D1e582bA77)] = 1;
45          r.quorum[address(0xb5346CF224c02186606e5f89EACC21eC25398077)] = 1;
46          r.quorum[address(0xa2997F1CA363D11a0a35bB1Ac0Ff7849bc13e914)] = 1;
47
48          // launch an event to be detected by oracle outside of blockchain
49          emit NewRequest (
50              currentId,
51              _urlToQuery,
52              _attributeToFetch
53          );
54
55          // increase request id
56          currentId++;
57      }
58
59      //called by the oracle to record its answer
60      function updateRequest (
61          uint _id,
62          string memory _valueRetrieved
63      ) public {
64
65          Request storage currRequest = requests[_id];
66
67          //check if oracle is in the list of trusted oracles
68          //and if the oracle hasn't voted yet
69          if(currRequest.quorum[address(msg.sender)] == 1){
70
71              //marking that this address has voted
72              currRequest.quorum[msg.sender] = 2;
73
74              //iterate through "array" of answers until a position if free and save
    the retrieved value
75              uint tmpI = 0;
76              bool found = false;
77              while(!found) {
```

```
78              //find first empty slot
79              if(bytes(currRequest.anwers[tmpI]).length == 0){
80                found = true;
81                currRequest.anwers[tmpI] = _valueRetrieved;
82              }
83              tmpI++;
84            }
85
86            uint currentQuorum = 0;
87
88            //iterate through oracle list and check if enough oracles(minimum quorum)
89            //have voted the same answer has the current one
90            for(uint i = 0; i < totalOracleCount; i++){
91              bytes memory a = bytes(currRequest.anwers[i]);
92              bytes memory b = bytes(_valueRetrieved);
93
94              if(keccak256(a) == keccak256(b)){
95                currentQuorum++;
96                if(currentQuorum >= minQuorum){
97                  currRequest.agreedValue = _valueRetrieved;
98                  emit UpdatedRequest (
99                    currRequest.id,
100                   currRequest.urlToQuery,
101                   currRequest.attributeToFetch,
102                   currRequest.agreedValue
103                 );
104               }
105             }
106           }
107         }
108       }
109     }
```

Appendix C

Off-Chain Oracle Code

```
require("dotenv").config();

import request from "request-promise-native";

import {
    updateRequest,
    newRequest
} from "./ethereum";

const start = () => {

    newRequest((error, result) => {

        let options = {
            uri: result.args.urlToQuery,
            json: true
        };

        request(options)
            .then(parseData(result))
            .then(updateRequest)
            .catch(error);
    });
};

const parseData = result => (body) => {
    return new Promise((resolve, reject) => {
        let id, valueRetrieved;
        try {
            id = result.args.id;
            valueRetrieved = (body[result.args.attributeToFetch] || 0).toString
();
        } catch (error) {
```

```
33                  reject(error);
34                  return;
35              }
36              resolve({
37                  id,
38                  valueRetrieved
39              });
40          });
41      };
42
43      export default start;
```

Appendix D

Off-chain ethereum connection - ethereum.js

```
require("dotenv").config();

import Web3 from "web3";


const web3 = new Web3(new Web3.providers.HttpProvider(process.env.
    WEB3_PROVIDER_ADDRESS));
const abi = JSON.parse(process.env.ABI);
const address = process.env.CONTRACT_ADDRESS;
const contract = web3.eth.contract(abi).at(address);

const account = () => {
  return new Promise((resolve, reject) => {
    web3.eth.getAccounts((err, accounts) => {
      if (err === null) {
        resolve(accounts[process.env.ACCOUNT_NUMBER]);
      } else {
        reject(err);
      }
    });
  });
};

export const updateRequest = ({
  id,
  valueRetrieved
}) => {
  return new Promise((resolve, reject) => {
    account().then(account => {
      contract.updateRequest(id, valueRetrieved, {
        from: account,
```

```
        gas: 60000000
      }, (err, res) => {
        if (err === null) {
          resolve(res);
        } else {
          reject(err);
        }
      });
    }).catch(error => reject(error));
  });
};

export const createRequest = ({
  urlToQuery,
  attributeToFetch
}) => {
  return new Promise((resolve, reject) => {
    account().then(account => {
      contract.createRequest(urlToQuery, attributeToFetch, {
        from: account,
        gas: 60000000
      }, (err, res) => {
        if (err === null) {
          resolve(res);
        } else {
          reject(err);
        }
      });
    }).catch(error => reject(error));
  });
};

export const newRequest = (callback) => {
  contract.NewRequest((error, result) => callback(error, result));
};

export const updatedRequest = (callback) => {
  contract.UpdatedRequest((error, result) => callback(error, result));
};
```